Compressing Encrypted Data Over Small Fields

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

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

Abstract. A recent work of Fleischhacker, Larsen, and Simkin (Eurocrypt 2023) shows how to efficiently compress encrypted sparse vectors. Subsequently, Fleischhacker, Larsen, Obremski, and Simkin (Eprint 2023) improve upon their work and provide more efficient constructions solving the same problem. Being able to efficiently compress such vectors is very useful in a variety of applications, such as private information retrieval, searchable encryption, and oblivious message retrieval. Concretely, assume one is given a vector $(m_1, \ldots, m_n)$ with at most t distinct indices $i \in [n]$, such that $m_i \neq 0$ and assume $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ is an additively homomorphic encryption scheme. The authors show that one can compress $(\mathsf{Enc}(k, m_1), \ldots, \mathsf{Enc}(k, m_n))$, *without* knowing the secret key k, into a digest with size dependent on the upper bound on the sparsity t, but not on the total vector length n. Unfortunately, all existing constructions either only work for encryption schemes that have sufficiently large plaintext spaces or require additional encrypted auxiliary information about the plaintext vector.
In this work, we show how to generalize the results of Fleischhacker, Larsen, and Simkin to encryption schemes with arbitrarily small plaintext spaces. Our construction follows the same general ideas laid out in previous works but modifies them in a way that allows compressing the encrypted vectors correctly with high probability, independently of the size of the plaintext space.

1 Introduction

A recent work by Fleischhacker, Larsen, and Simkin [FLS23] showed how to efficiently compress encrypted sparse vectors. More concretely, consider a vector $(m_1, \ldots, m_n)$ with at most t distinct indices $i \in [n]$ such that $m_i \neq 0$ and let $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ be an additively homomorphic encryption scheme. The authors showed that one can compress $(\mathsf{Enc}(k, m_1), \ldots, \mathsf{Enc}(k, m_n))$, *without* knowing the secret key k, into a digest with size dependent on the upper bound on the sparsity t, but not on the total vector length n.

Being able to efficiently compress such encrypted sparse vectors is very useful, as this problem appears in a variety of applications that work on encrypted data. In the oblivious message retrieval setting [LT22], clients can send each other encrypted messages via an untrusted intermediary server, which acts as communication channel. To hide which messages are intended for which client, the receiving clients could always just naively download all messages from the server and check which ones are intended for them, but this would be highly inefficient in terms of communication complexity. Oblivious message retrieval protocols allow for achieving the same functionality with much smaller communication complexity, when an upper bound for the number of messages expected by the the receiving client is known. In the encrypted search setting [YSK+13, LLN15, CKK16, CKL15, AFS18, CDG+21], a client stores an encrypted database on an untrusted server. Later on, the client may want to search for a keyword and retrieve all entries form the database that match it, without revealing the keyword or the entries that were matching. In the batch private information retrieval setting [IKOS04, ACLS18, MR23], a client retrieves a subset of the entries of a database held by an untrusted server, without revealing which entries the client wants.

⋆ 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

In all of the above applications, many of the state-of-the-art protocols [LT22, CDG+21, MR23] follow the same two-step approach. First, the client sends an encrypted hint to the server, which allows the server to *obliviously* compute an encrypted vector, where all irrelevant database entries are replaced by encryptions of zero and all relevant entries are just encrypted. Next, this vector is obliviously compressed by the server under the assumption that it is sparse and sent back to the client, who can then use the secret decryption key to obtain the desired results. Improving compression algorithms for encrypted sparse vectors directly benefits all of these application.

The compression algorithm of [FLS23] based on invertible bloom lookup tables (IBLTs) [GM11] as well as the improvements to that algorithm from a follow-up work by Fleischhacker, Larsen, Obremski, and Simkin [FLOS23] only allow for compressing encrypted vectors if the message space of the corresponding encryption scheme is sufficiently large. More concretely, they require the message space to be a finite field $\mathbb{F}_q$ with $q \geq 2^{\Omega(\kappa \log \kappa + \log(n))}$, where κ is an error parameter that ensures that the compression algorithm is correct with probability roughly at least $1 - 2^{-\kappa}$. In particular, their compression algorithm[4] is not applicable to encrypted vectors, where the encryption scheme has a small plaintext space.

1.1 Our Contribution

In this work, we extend the results of Fleischhacker, Larsen, and Simkin [FLS23], along with the improvements from [FLOS23], to encryption schemes with arbitrarily small message spaces. For a vector encrypted with an additively homomorphic encryption scheme with message space $\mathbb{F}_q$ for an arbitrary prime power q, the compression algorithm produces a digest that is $\tilde{O}\left(\xi \cdot \frac{\kappa t + \kappa^2}{\log q}\right)$ bits large[5], where ξ is the bit length of a single ciphertext. For small plaintext spaces, our result presents the currently best known compression algorithm for encrypted sparse vectors. For sufficiently large plaintext spaces, our result degenerates to the algorithm from [FLS23] with the improvements from [FLOS23]. We note that [FLOS23] only provided a brief discussion of how their work can be used to improve the algorithm of [FLS23]. Our work on the other hand, provides full formal descriptions of the compression algorithm, along with our extension for small plaintext spaces and full proofs.

Just like the work of Fleischhacker, Larsen, and Simkin, our compression algorithm has highly efficient compression and decoding procedures that only rely on homomorphic additions and homomorphic multiplications by constants, in particular no homomorphic multiplications of encrypted values are needed. This is desirable, even if the encryption scheme would support homomorphic multiplications, as these operations are usually significantly more expensive computationally.

1.2 Technical Overview

To understand why the previous constructions of [FLS23] requires a large plaintext space, let us recall it from a high level point of view. Overly simplifying and ignoring many important details, their digest is effectively composed of two matrices M and K, where initially each matrix entry is set to zero. During compression of a vector of ciphertexts $(c_1, \ldots, c_n)$, each c_i is assigned to some number of random cells. If c_i is assigned to some cell (i, j), then c_i is added to $M[i, j]$ and $c_i \cdot \tau_i$ is added to $K[i, j]$, where τ_i is some "efficiently recognizable" pseudorandom value. When compression is done, some of the cells in M will be zero, some will contain exactly one encrypted message, and some will contain the encrypted sum of messages.

To decode the digest, the encrypted entries in M and K are first decrypted and then a peeling process begins. At each step of this process, the decoding procedure tries to find a cell (i, j), where $K[i, j]/M[i, j]$ is one of those efficiently recognizable pseudorandom values. Let us assume there is a function $\mathsf{Peelable}$ with $\mathsf{Peelable}(K[i, j]/M[i, j]) = 1$, when cell (i, j) contains one of those recognizable values and zero otherwise. If such a cell (i, j) with $\mathsf{Peelable}(K[i, j]/M[i, j]) = 1$ is found, we assume that this cell contained exactly one

[4] The work of Fleischhacker, Larsen, and Simkin [FLS23] provides a second algorithm based on sparse polynomial interpolation, but the decoding complexity of that algorithm is $\Omega(\sqrt{n})$ and it also does not work for arbitrarily small plaintext spaces. In this work, we exclusively focus on the more efficient IBLT based solutions.

[5] The soft-O notation $\tilde{O}(\cdot)$ ignores log factors in κ and n.

entry, we obtain the message $M[i,j]$, we additionally remove that message from all the other cells in M and K in which it was put during compression and continue with our peeling process until both M and K are the zero matrices again.

The correctness analysis of their decoding procedure crucially relies on the fact that with overwhelming probability $\mathsf{Peelable}(K[i,j]/M[i,j]) = 1$ only happens, if the matrix cell (i,j) indeed only contained one entry. To argue that the bad event of $\mathsf{Peelable}(K[i,j]/M[i,j]) = 1$ and yet cell (i,j) contains more than one item does not happen, their analysis requires that the efficiently recognizable pseudorandom values are sampled from a large space. Since this space must necessarily be a subset of the plaintext space, the plaintext space must therefore be large.

The main idea behind our modified compression algorithm for small plaintext spaces is to basically perform a parallel repetition of these checks. We note that even when the plaintext space is small, the check in their construction for whether a matrix cell contains exactly one message will produce the correct result with some at least constant probability. Now instead of the value τ_i being a scalar, we define it to be a vector $\tau_i := (\tau_{i,1}, \ldots, \tau_{i,\eta})$ for some η and we define $\tau_i \cdot c_i := (c_i \cdot \tau_{i,1}, \ldots, c_i \cdot \tau_{i,\eta})$. During decoding we will apply the function $\mathsf{Peelable}$ to each vector component and we will exploit the fact that the function may accidentally return an incorrect one, but will never return zero incorrectly. If all individual checks for a cell return one, then we assume the cell contains exactly one entry with overwhelming probability. If even a single check fails, then we can conclude with certainty that the cell contains more than one entry. By choosing η sufficiently large, we can then drive down the probability of all checks simultaneously returning an incorrect one, which would be required for the decoding algorithm to perform an incorrect peeling step. We conclude our technical overview by remarking that we presented a gross oversimplification of how the compression algorithm actually works here. Our compression algorithm in the following sections will be heavily based on the specific algorithm presented Fleischhacker, Larsen, Obremski, and Simkin [FLOS23].

1.3 Related Work

The task of compressing encrypted, but structured data has been studied in several works prior to ours. Johnson, Wagner, and Ramchandran [JWR04] showed that one-time pad encryptions of messages from a source with bounded entropy can be compressed without knowledge of the encryption key through the use of Slepian-Wolf coding [SW73]. This result was then extended to block ciphers using certain chaining modes by Klinc et al. [KHJ+09]. Neither of those two results is applicable to our setting.

In compressed sensing [Don06, CRT06, GI10], the goal is to construct a matrix A such that one can recover a sparse vector x from a vector Ax of small dimension. Fleischhacker, Larsen, and Simkin [FLS23] presented a construction based on sparse polynomial interpolation, which is closely related to techniques that are used in the context of compressed sensing. These approaches tend to suffer from high computational complexities during both compression and decoding.

The compression algorithms for encrypted data that were constructed by Choi et al. [CDG+21] are not applicable to ciphertexts with small plaintext spaces. The work of Liu and Tromer [LT22] proposes compression algorithms that work for arbitrarily small plaintext spaces. Their computational compression cost is $\Omega(n \cdot t)$ and their decoding cost is $\Omega(t^3)$, whereas our computational costs are $O\left(\frac{n \cdot (\log t + \kappa)^2}{\log q}\right)$ during compression and $\tilde{O}\left(\frac{\kappa t + \kappa^2}{\log q}\right)$ during decoding. In the case of the plaintext space being $\{0,1\}$, their construction slightly outperforms ours. As the plaintext space gets bigger, our compression rate improves, whereas theirs stays the same and eventually our solution becomes preferable over theirs. Most importantly, however, their compression algorithm requires an auxiliary input in addition to the encrypted sparse vector and without it, they can not compress. If our compression algorithm were to be given the exact same auxiliary information, our compression would improve in a straightforward manner and outperform their solution in terms of compression rate for all plaintext spaces.

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 $x \in X$. We write $[n]$ to denote the set $\{0, \ldots, n-1\}$. For a vector $\boldsymbol{v} \in X^n$ and $i \in [n]$, 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. For a set X^n, we use the scissor operator $\not\Join(X^n) := \{(x_1, \ldots, x_n) \in X^n \mid x_i \neq x_j \ \forall i, j \in [n]\}$ 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 to compute an encryption of $f(\boldsymbol{m})$ given only the description of f and an encryption of $\boldsymbol{m}$. Throughout the paper, we assume that functions are represented as circuits composed of addition and multiplication gates. We recall the formal definition of a homomorphic encryption scheme, closely following the notation of [FLS23].

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 $m \in \mathcal{M} \cup \{\perp\}$.*

Throughout the paper it is assumed that the ciphertext size is fixed and does not increase when applying 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 $M'[i, j] = \mathsf{Dec}(\mathsf{sk}, C[i, j])$. This also applies recursively when, for instance, decrypting a vector of matrices of ciphertexts. Let $\mathcal{E}$ be an additively homomorphic encryption scheme with message space $\mathcal{M} = \mathbb{F}_q$ for some prime power q. 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. We naturally extend these notions to vectors, i.e. for two vectors $\boldsymbol{c}, \boldsymbol{c}' \in \mathcal{C}^n$ we denote $\boldsymbol{c} \boxplus \boldsymbol{c}' = (c_0 \boxplus c'_0, \ldots, c_n \boxplus c'_n)^\intercal$ and for a vector $\boldsymbol{\alpha} \in \mathcal{M}^n$ we denote $\boldsymbol{\alpha} \boxdot c = (\alpha_0 \boxdot c, \ldots, \alpha_0 \boxdot c)^\intercal$. 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 $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{Gen}(1^\lambda)$. The unique secret key is denoted as $\mathsf{Gen}^{-1}(\mathsf{pk})$ and we stress that the function $\mathsf{Gen}^{-1}(\cdot)$ does not need to be efficiently computable.

We recall the definition of ciphertexts valid relative to a class of circuits and of a ciphertext compression scheme from [FLS23].

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 $\perp \notin \mathsf{Dec}(\mathsf{Gen}^{-1}(\mathsf{pk}), \boldsymbol{c})$ 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.*

Definition 4 (Ciphertext Compression Scheme). *Let $\mathcal{E} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Eval}, \mathsf{Dec})$ 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.

Just like the construction of [FLS23], the construction described in Section 3 works for ciphertext vectors of low Hamming weight which allow for the homomorphic evaluation of inner product functions. The relevant definitions, taken verbatim from [FLS23] are recalled in the following.

Definition 5 (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*

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

Definition 6 ($\mathcal{Z}_{\mathsf{ip}}$-Valid Low Hamming Weight Ciphertext Vectors). *Let $\mathcal{E} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Eval}, \mathsf{Dec})$ 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 $\mathcal{F}^{\mathsf{ip}}_t := \{\mathcal{F}^{\mathsf{ip}}_{t,\mathsf{pk}} \mid \mathsf{pk} \in \mathcal{P}\}$.

2.2 Stacked 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[6] and the list operation recovers all currently present elements with high probability, if not too many elements are present.

An improved version of IBLTs, called stacked IBLTs, was recently introduced by Fleischhacker, Larsen, Obremski, and Simkin [FLOS23]. The data structure, when instantiated such that List correctly recovers t elements with probability $1 - 2^{-\kappa}$, consists of a series of $\log t$ consecutively smaller triples of count, key, and value matrices $\boldsymbol{C}$,$\boldsymbol{K}$,$\boldsymbol{V}$. For each row in each triple of matrices it further requires an $O(\mathrm{loglog}\, t + \kappa)$-wise independent hash functions $h_{i,j}$ mapping to $[\gamma_{i,j}]$, where $\gamma_{i,j}$ is the length of the jth row in the ith triple of matrices. . Initially all values in all matrices are set to zero. To insert a key-value pair (d, x) into the data structure, we locate the cells $\boldsymbol{C}_i[j, h_{i,j}(d)]$, $\boldsymbol{K}_i[j, h_{i,j}(d)]$ and $\boldsymbol{V}_i[j, h_{i,j}(d)]$ and add 1 to each counter, d to each key sum and x to each value sum for all values of i and j. To remove an element,the inverse operations are performed. To list all elements we start with the first and largest $\boldsymbol{C}, \boldsymbol{K}, \boldsymbol{V}$, search for any 1 entries in $\boldsymbol{C}$ and add the corresponding key-value pairs to the output list. We then continue with the next smallest $\boldsymbol{C}, \boldsymbol{K}, \boldsymbol{V}$, remove all pairs found so far from it, and continue looking for 1 entries. This continues until we have iterated over all triples of matrices. Fleischhacker, Larsen, Obremski, and Simkin [FLOS23] showed that this list procedure will fail with probability at most $2^{-\kappa}$ over the choice of the hash functions.

We simplify the description of stacked IBLTs a bit here, as we will not require multiple insertions or delete operations. So instead we describe stacked IBLTs with just a single shot encoding operation and a matching decoding operation in Figure 1.

The following corollary follows immediately from [FLOS23, Theorem 1] by replacing the $O(\kappa + \mathrm{loglog}\, n)$-wise independent hash functions with truly random hash functions.

Corollary 7 ([FLOS23]). *Let $\boldsymbol{h}$ be a vector of vectors of truly random hash functions with appropriate output domains. Then for any $S \in \mathbb{F}_q \times \mathbb{F}_q$ with $|S| \leq t$ and such that for all $(i, m), (i', m') \in S$, $i \neq i'$, it holds that*

$$\Pr[\mathsf{Decode}(\mathsf{Encode}(S, \boldsymbol{h}), \boldsymbol{h}) = S] \geq 1 - 2^{\kappa}$$

where the probability is taken over the random choice of $\boldsymbol{h}$.

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

$\mathsf{Encode}(S, \boldsymbol{h})$	$\mathsf{Decode}((F_0, \ldots, F_{\lceil \log t \rceil - 1}), \boldsymbol{h})$
for $0 \leq i < \log t - \log \tau$	$S' := \emptyset$
$\quad F_i := \mathsf{BasicEnc}(1, \lceil Ct2^{-i} \rceil, S, \boldsymbol{h}_i)$	**for** $0 \leq i < \lceil \log t \rceil$
for $0 \leq i < \log \tau$	$\quad F_i := \mathsf{BasicDel}(F_i, S', \boldsymbol{h}_i)$
$\quad i' := \lfloor \log t - \log \tau \rfloor$	$\quad S' := S' \cup \mathsf{BasicDec}(F_i, \boldsymbol{h}_i)$
$\quad F_{i'} := \mathsf{BasicEnc}(2^i, \lceil C\tau 2^{-i} \rceil, S, \boldsymbol{h}_{i'})$	**return** S'
return $(F_0, \ldots, F_{\lceil \log t \rceil - 1})$	

Fig. 1. The stacked IBLT of [FLOS23], slightly simplified as we only require one-shot encoding and decoding instead of successive updates. It uses the basic filter specified in Figure 2 as a building block and requires a vector of vectors of hash functions with the appropriate output domains. We have $\tau = C_0\kappa$ for a sufficiently large constant $C_0 > 0$ and $C = 8e$.

2.3 Pseudorandom Functions with Variable Codomains

The construction presented in Section 3 relies on a pseudorandom function that needs to be able to produce outputs from variable codomains. We define such a variant of PRFs here.

Definition 8 (Pseudorandom Function with Variable Codomain). *An efficiently computable function* $\mathsf{PRF} : \{0,1\}^\lambda \times \mathbb{N} \times \{0,1\}^* \to \mathbb{N}$ *is a pseudorandom function with variable codomain, if it satisfies the following properties.*

1. *For any* $s \in \{0,1\}^\lambda$*, any* $\gamma \in \mathbb{N}$ *with* $\log \gamma = \mathsf{poly}(\lambda)$*, and any* $x \in \{0,1\}^*$*, it holds that* $\mathsf{PRF}(s, \gamma, x) \in [\gamma]$.
2. *Let* $\mathcal{G}$ *be the set of all functions* $g : \mathbb{N} \times \{0,1\}^* \to \mathbb{N}$ *such that for all* $\gamma \in \mathbb{N}$ *and all* $x \in \{0,1\}^*$ *it holds that* $g(\gamma, x) \in [\gamma]$*. For all PPT adversaries* $\mathcal{A}$ *it holds that*

$$|\Pr[\mathcal{A}^{\mathsf{PRF}(s,\cdot,\cdot)}(1^\lambda) = 1] - \Pr[\mathcal{A}^{g(\cdot,\cdot)}(1^\lambda) = 1]| \leq \mathsf{negl}(\lambda)$$

where the probabilities are taken over the uniform choice of $s \in \{0,1\}^\lambda$ *and* $g \in \mathcal{G}$ *respectively.*

While this funky definition of a PRF is helpful to us as an abstraction, such PRFs are luckily existentially equivalent to regular PRFs. To see this, consider a regular PRF $\mathsf{PRF}' : \{0,1\}^\lambda \times \{0,1\}^* \to \{0,1\}^\lambda$. We can construct a PRF with variable codomain PRF as follows. On input (s, γ, x) first compute $s' := \mathsf{PRF}'(s, \gamma)$. This step gives us (computationally) independent keys for the PRF evaluations for different output domains. Then compute $y' := \mathsf{PRF}'(s', x)$, this already gives us a pseudorandom value however it's from the wrong domain. We can now stretch y' to a sufficient length using a pseudorandom generator and finally reduce it modulo γ to get a pseudorandom value in $[\gamma]$. A simple hybrid argument can be used to establish pseudorandomness.

2.4 Wunderbar Pseudorandom Vectors over $K \in \mathbb{F}_q^\eta$

As in the original construction, the ciphertext compression scheme relies on wunderbar pseudorandom vectors. The construction requires that the the universe K over which the wunderbar pseudorandom vector operates is "large enough". In [FLS23] this was achieved by requiring that the field the encryption scheme operates on is large. Here we show how the same can be achieved by instead defining $K \subseteq \mathbb{F}_q^\eta$ for an arbitrarily small q and large enough η. We first recall the definition of a a wunderbar pseudorandom vector taken verbatim from [FLS23].

Definition 9. *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*

$\mathsf{Sample}(1^\lambda, 1^n)$**:** *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*.*

$\mathsf{BasicEnc}(\rho, \gamma, S, \boldsymbol{h})$

$\boldsymbol{M} := 0^{\rho \times \gamma}$
$\boldsymbol{K} := 0^{\rho \times \gamma}$
$\boldsymbol{C} := 0^{\rho \times \gamma}$
foreach $(d, m) \in S$
  foreach $i \in [\rho]$
    $j := h_i(d)$
    $\boldsymbol{M}[i, j] := \boldsymbol{M}[i, j] + m$
    $\boldsymbol{K}[i, j] := \boldsymbol{K}[i, j] + d$
    $\boldsymbol{C}[i, j] := \boldsymbol{C}[i, j] + 1$
return $(\boldsymbol{M}, \boldsymbol{K}, \boldsymbol{C})$

$\mathsf{BasicDel}((\boldsymbol{K}, \boldsymbol{M}, \boldsymbol{C}), \tilde{S}, \boldsymbol{h})$

foreach $(d, m) \in \tilde{S}$
  foreach $i \in [\rho]$
    $j := h_i(d)$
    $\boldsymbol{M}[i, j] := \boldsymbol{M}[i, j] - m$
    $\boldsymbol{K}[i, j] := \boldsymbol{K}[i, j] - d$
    $\boldsymbol{C}[i, j] := \boldsymbol{C}[i, j] - 1$
return $(\boldsymbol{M}, \boldsymbol{K}, \boldsymbol{M})$

$\mathsf{BasicDec}((\boldsymbol{K}, \boldsymbol{M}, \boldsymbol{C}), \boldsymbol{h})$

$S' := \emptyset$
for $(i, j) \in [\rho] \times [\gamma]$
  if $\boldsymbol{C}[i, j] = 1$
    $S' := S' \cup \{(\boldsymbol{K}[i, j], \boldsymbol{M}[i, j])\}$
return S'

Fig. 2. The basic IBLT of [FLOS23], slightly simplified as we only require one-shot encoding and decoding instead of successive updates. The basic filter requires a number of rows ρ, a number of columns γ as well as a vector of hash functions $\boldsymbol{h} \in \{h : \mathbb{F}_q \to [\gamma]\}^\rho$.

$\mathsf{Entry}(s, i)$**:** *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 $s \leftarrow \mathsf{Sample}(1^\lambda, 1^n)$ 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 $\mathcal{A}$*

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

Fleischhacker, Larsen, and Simkin [FLS23] construct a wunderbar pseudorandom vector for $K \subseteq \mathbb{F}_q$ from a pseudorandom permutation. The construction essentially just takes a PRP over $\mathbb{F}_2^\lambda$ and uses an efficiently computable and invertible injective function to map values from $\mathbb{F}_2^\lambda$ to $\mathbb{F}_q$ and back. The construction is easily generalized for $K \subseteq S$ for any set S as long as there exists an efficiently computable and invertible injective function from $\mathbb{F}_2^\lambda$ to S.

The new construction requires $K \subseteq \mathbb{F}_q^\eta$ for some η and such that $|K| > \alpha$ for some given lower bound α. We specify the required injective function in the following.

Let $\mathsf{decomp}_p : \mathbb{N} \to [p]^*$ denote the function that maps an integer to its canonical p-ary representation and let $\mathsf{proj}_p : [p]^* \to \mathbb{N}$ be its inverse. Let $q = p^m$ be an arbitrary prime power and let $\eta = \lceil \lambda / \log q \rceil =$

$\lceil \lambda/(m \log p) \rceil$ We then define an injective function

$$\mathsf{binToField} : \mathbb{F}_2^\lambda \to \mathbb{F}_q^\eta \quad \mathsf{binToField}(\boldsymbol{b}) = \boldsymbol{d}$$

where

$$d_i := \sum_{j=0}^{m-1} c_{im+j} x^j$$

where

$$\boldsymbol{c} := \mathsf{decomp}_q(\mathsf{proj}_2(\boldsymbol{b})).$$

We further specify the inverse function as

$$\mathsf{fieldToBin} : \mathbb{F}_q^\eta \to \mathbb{F}_2^\lambda \cup \{\bot\}$$

$$\mathsf{fieldToBin}(\boldsymbol{d}) := \begin{cases} \bot & \text{if } \mathsf{proj}_p(\boldsymbol{c}) \geq 2^\lambda \\ \mathsf{decomp}_2(\mathsf{proj}_q(\boldsymbol{c})) & \text{otherwise} \end{cases}$$

where

$$d_i = \sum_{j=0}^{m-1} c_{im+j} x^j.$$

For a given α, any $\lambda = \Omega(\log \alpha)$ leads to the required wunderbar pseudorandom vector.

3 A Ciphertext Compression Scheme for Small Fields

In this section we present a construction of a ciphertext compression scheme, that in contrast to [FLS23] also works if the encryption scheme is defined over an arbitrarily small field, even for $\mathbb{F}_2$.

3.1 The Generalized Helpful Lemma

Fleischhacker, Larsen, and Simkin [FLS23] state the following helpful lemma.

Lemma 10 (Helpful Lemma [FLS23, Lemma 13]). *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*

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

We generalize this lemma to vectors of $\mathbb{F}_q$ elements.

Lemma 11 (Generalized Helpful Lemma). *Let $\eta \in \mathbb{N}^+$, $K \subseteq \mathbb{F}_q^\eta$, $(m_1, \ldots, m_n) \in \mathbb{F}_q^n$, 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*

$$\Pr\left[\boldsymbol{k} \leftarrow K^n : \exists j \in [n].\, \boldsymbol{k}_j = \left(\sum_{i \in I} m_i\right)^{-1} \cdot \sum_{i \in I} m_i \cdot \boldsymbol{k}_i\right] \leq \frac{n}{|K|}$$

Proof. Observe that $\boldsymbol{k}_j \in K \subseteq \mathbb{F}_q^\eta$ can be interpreted as polynomials of degree at most $\eta - 1$ with coefficients in $\mathbb{F}_q$. Similarly, $m_i \in \mathbb{F}_q$ is simply a constant polynomial over $\mathbb{F}_q$ and the vector-scalar multiplications are in fact correct polynomial multiplications resulting in polynomials of degree at most $\eta - 1$ with coefficients in $\mathbb{F}_q$. Therefore, the lemma can be reinterpreted as working over the extension field $\mathbb{F}_{q^\eta}$. It then follows directly as a special case of Lemma 10 for $\mathbb{F}_{q^\eta}$. □

As in [FLS23], the following corollary follows from the observation that due to the birthday bound the statistical distance between sampling from K^n and $\not\in(K^n))$ is at most $n^2/|K|$.

Corollary 12. *Let $\eta \in \mathbb{N}^+$, $K \subseteq \mathbb{F}_q^\eta$, $(m_1, \ldots, m_n) \in \mathbb{F}_q^n$, 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*

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

$\mathsf{Encode}_1(S, s_1)$

for $0 \le i < \log t - \log \tau$
  $F_i := \mathsf{BasicEnc}_1(1, \lceil Ct2^{-i} \rceil, S, (i, s_1))$
for $0 \le i < \log \tau$
  $i' := \lfloor \log t - \log \tau \rfloor$
  $F_{i'} := \mathsf{BasicEnc}_1(2^i, \lceil C\tau 2^{-i} \rceil, S, (i', s_1))$
return $(F_0, \ldots, F_{\lceil \log t \rceil - 1})$

$\mathsf{Decode}_1((F_0, \ldots, F_{\lceil \log t \rceil - 1}), s_1)$

$S' := \emptyset$
for $0 \le i < \lceil \log t \rceil$
  $F_i := \mathsf{BasicDel}_1(F_i, S', (i, s_1))$
  $S' := S' \cup \mathsf{BasicDec}_1(F_i, (i, s_1))$
return S'

$\mathsf{Encode}_2(S, s_1, s_2)$

for $0 \le i < \log t - \log \tau$
  $F_i := \mathsf{BasicEnc}_2(1, \lceil Ct2^{-i} \rceil, S, (i, s_1), s_2)$
for $0 \le i < \log \tau$
  $i' := \lfloor \log t - \log \tau \rfloor$
  $F_{i'} := \mathsf{BasicEnc}_2(2^i, \lceil C\tau 2^{-i} \rceil, S, (i', s_1), s_2)$
return $(F_0, \ldots, F_{\lceil \log t \rceil - 1})$

$\mathsf{Decode}_2((F_0, \ldots, F_{\lceil \log t \rceil - 1}), s_1, s_2)$

$S' := \emptyset$
for $0 \le i < \lceil \log n \rceil$
  $F_i := \mathsf{BasicDel}_2(F_i, S', (i, s_1), s_2)$
  $S' := S' \cup \mathsf{BasicDec}_2(F_i, (i, s_1), s_2)$
return S'

Fig. 3. Variants of the simplified stacked IBLT of [FLOS23]. These use modified basic IBLTs specified in Figure 4 respectively as a building blocks. As with the original stacked IBLT we have $\tau = C_0\kappa$ for a sufficiently large constant $C_0 > 0$ and $C = 8e$. Changes between successive modifications are marked in gray.

3.2 Construction

The construction presented here essentially takes the construction of Fleischhacker, Larsen, and Simkin [FLS23], applies the improved IBLT construction of [FLOS23] and instantiates the wunderbar pseudorandom vector using the construction for $K \subseteq \mathbb{F}_q^\eta$ described in Section 2.4. For completeness, since [FLOS23] does not include a formal proof that their improved IBLT can be used as a drop-in replacement in [FLS23], we give a full formal proof of the construction here.

Before we give the actual construction we first specify two variants of the stacked IBLT construction of [FLOS23] as specified in Figure 1 and Figure 2 and prove several lemmas about them. These two variants are specified in Figure 3 and Figure 4. We now state and prove several lemmas about these two variants. The first lemma states that the first variant of the construction of [FLOS23] as described in Figure 3 still works as expected. Essentially this means that the construction of [FLOS23] still works as expected even if the truly random functions it uses are replaced by pseudorandom ones.

Lemma 13. *Let* PRF *be a variable output domain pseudorandom function as defined in Definition 8. Then for any set* $S \subseteq \mathbb{F}_q \times \mathbb{F}_q$ *with* $|S| \le n$ *and such that for all* $(i, m), (i', m') \in S$, $i \ne i'$ *it holds that*

$$\Pr[\mathsf{Decode}_1(\mathsf{Encode}_1(\rho, \gamma, S, s_1), s_1) = S] \ge 1 - 2^{-\kappa} - \mathsf{negl}(\lambda)$$

where the probability is taken over the uniform choice of s_1.

Proof. The lemma follows from Corollary 7 and by a simple reduction to the pseudorandom of PRF. Let S be an arbitrary set. We established the claimed bound by constructing an adversary $\mathcal{A}$ against the pseudorandomness of PRF as follows. We then related the success probability of $\mathcal{A}$, to the probability of Encode_1 and Decode_1 working as intended. On input 1^λ and given access to an oracle o that contains either a truly random function of $\mathsf{PRF}(s_1, \cdot, \cdot)$, $\mathcal{A}$ computes

$$S' = \mathsf{Decode}(\mathsf{Encode}(\rho, \gamma, S, \boldsymbol{h}), \boldsymbol{h})$$

$\mathsf{BasicEnc}_1(\rho, \gamma, S, (r, s_1))$

$\boldsymbol{M} := 0^{\rho \times \gamma}$
$\boldsymbol{K} := 0^{\rho \times \gamma}$
$\boldsymbol{C} := 0^{\rho \times \gamma}$
foreach $(d, m) \in S$
  foreach $i \in [\rho]$
    $j := \mathsf{PRF}(s_1, \gamma, (r, i, d))$
    $\boldsymbol{M}[i, j] := \boldsymbol{M}[i, j] + m$
    $\boldsymbol{K}[i, j] := \boldsymbol{K}[i, j] + d$
    $\boldsymbol{C}[i, j] := \boldsymbol{C}[i, j] + 1$
return $(\boldsymbol{M}, \boldsymbol{K}, \boldsymbol{C})$

$\mathsf{BasicDel}_1((\boldsymbol{K}, \boldsymbol{M}, \boldsymbol{C}), \tilde{S}, (r, s_1))$

foreach $(d, m) \in \tilde{S}$
  foreach $i \in [\rho]$
    $j := \mathsf{PRF}(s_1, \gamma, (r, i, d))$
    $\boldsymbol{M}[i, j] := \boldsymbol{M}[i, j] - m$
    $\boldsymbol{K}[i, j] := \boldsymbol{K}[i, j] - d$
    $\boldsymbol{C}[i, j] := \boldsymbol{C}[i, j] - 1$
return $(\boldsymbol{M}, \boldsymbol{K}, \boldsymbol{M})$

$\mathsf{BasicDec}_1((\boldsymbol{K}, \boldsymbol{M}, \boldsymbol{C}), (r, s_1))$

$S' := \emptyset$
for $(i, j) \in [\rho] \times [\gamma]$
  if $\boldsymbol{C}[i, j] = 1$
    $S' := S' \cup \{(\boldsymbol{K}[i, j], \boldsymbol{M}[i, j])\}$
return S'

$\mathsf{BasicEnc}_2(\rho, \gamma, S, (r, s_1), s_2)$

$\boldsymbol{M} := 0^{\rho \times \gamma}$
$\boldsymbol{K} := (0^{\eta})^{\rho \times \gamma}$

foreach $(d, m) \in S$
  foreach $i \in [\rho]$
    $j := \mathsf{PRF}(s_1, \gamma, (r, i, d))$
    $\boldsymbol{M}[i, j] := \boldsymbol{M}[i, j] + m$
    $\boldsymbol{k} := \mathsf{Entry}(s_2, d)$
    $\boldsymbol{K}[i, j] := \boldsymbol{K}[i, j] + (m \cdot \boldsymbol{k})$

return $(\boldsymbol{M}, \boldsymbol{K})$

$\mathsf{BasicDel}_2((\boldsymbol{K}, \boldsymbol{M}), \tilde{S}, (r, s_1, s_2))$

foreach $(d, m) \in \tilde{S}$
  foreach $i \in [\rho]$
    $j := \mathsf{PRF}(s_1, \gamma, (r, i, d))$
    $\boldsymbol{k} := \mathsf{Entry}(s_2, d)$
    $\boldsymbol{M}[i, j] := \boldsymbol{M}[i, j] - m$
    $\boldsymbol{K}[i, j] := \boldsymbol{K}[i, j] - (m \cdot \boldsymbol{k})$

return $(\boldsymbol{M}, \boldsymbol{K})$

$\mathsf{BasicDec}_2((\boldsymbol{K}, \boldsymbol{M}), (r, s_1, s_2))$

$S' := \emptyset$
for $(i, j) \in [\rho] \times [\gamma]$
  if $\boldsymbol{M}[i, j] \neq 0$
    $d := \mathsf{Index}\big(s_2, (\boldsymbol{M}[i, j])^{-1} \cdot \boldsymbol{K}[i, j]\big)$
    if $d \in [n]$
      $S' := S' \cup \{(d, \boldsymbol{M}[i, j])\}$
return S'

Fig. 4. The left hand side shows the basic IBLT as specified in Figure 2 but using a PRF with variable codomain as replacement for the truly random functions. The difference between the original basic IBLT and this one are marked in gray. The right hand side shows a modified basic IBLT that works without a count matrix and allows insertions using only addition and multiplication by constants. The differences are again marked in gray. As long as all inserted messages are non-zero, the decoding of all three filters will be the same with overwhelming probability.

but replaces invocations of $h_{i,j}(\cdot)$ with queries of the form $o(\gamma_i, (i, j, \cdot))$. If $S' = S$, $\mathcal{A}$ outputs 0, otherwise it outputs 1. Note that if o contains a truly random function, this perfectly simulates

$$\mathsf{Decode}(\mathsf{Encode}(\rho, \gamma, S, \boldsymbol{h}), \boldsymbol{h}).$$

If on the other hand o contains $\mathsf{PRF}(s_1, \cdot, \cdot)$, this perfectly simulates

$$\mathsf{Decode}_1(\mathsf{Encode}_1(\rho, \gamma, S, s_2), s_2).$$

From the pseudorandomness of PRF it follows that

$$\left| \begin{array}{l} \Pr[\mathsf{Decode}(\mathsf{Encode}(\rho, \gamma, S, \boldsymbol{h}), \boldsymbol{h}) = S] \\ -\Pr[\mathsf{Decode}_1(\mathsf{Encode}_1(\rho, \gamma, S, s_2), s_2) = S] \end{array} \right| \leq \mathsf{negl}(\lambda).$$

Combined with Corollary 7 the lemma immediately follows. □

The second variant of the stacked IBLT construction described in Figure 3 essentially applies the same modification to stacked IBLTs that [FLS23] applied to regular IBLTs. That is, detecting "peelable" entries no longer uses a count matrix, but instead uses a wunderbar pseudorandom vector. The following lemma essentially states that, as long as the encoded set does not contain any zero entries, the two variants of stacked IBLTs will decode the same set with high probability if the wunderbar pseudorandom vector operates over a large enough universe.

Lemma 14. *Let* $(\mathsf{Entry}, \mathsf{Index})$ *be a wunderbar pseudorandom vector. Then, for any* $r, \rho, \gamma \in \mathbb{Z}$*, any PRF key* s_2*, and any set* $S \subseteq [n] \times (\mathbb{F}_q \setminus \{0\})$ *such that* $|S| \leq t$ *and for all distinct* $(i, m), (i', m') \in S$*,* $i \neq i'$ *it holds that*

$$\begin{aligned} &\Pr\left[\begin{array}{c} \mathsf{BasicDec}_1(\mathsf{BasicEnc}_1(\rho, \gamma, S, (r, s_1)), (r, s_1)) \\ = \mathsf{BasicDec}_2(\mathsf{BasicEnc}_2(\rho, \gamma, S, (r, s_1), s_2), (r, s_1), s_2) \end{array}\right] \\ &\geq 1 - \frac{\rho\gamma(n^2 + n)}{|K|} \end{aligned}$$

where the probability is taken over the uniform choice of s_1*.*

Proof. Let S_1, S_2 be the sets decoded by $\mathsf{BasicDec}_1$ and $\mathsf{BasicDec}_2$. We consider two types of errors: There could be an $(d, m) \in S_1 \setminus S_2$ or an $(d, m) \in S_2 \setminus S_1$.

In the first case, since $\mathsf{BasicDec}_1$ is decoding the element, it must the case that (d, m) is mapped into a cell on its own. However, this implies that the corresponding cell in the output of $\mathsf{BasicEnc}_2$ will contain m in the value matrix and $m \cdot \mathsf{Entry}(s_2, d)$ in the key matrix. Therefore, since $m \neq 0$ and by the correctness of the wunderbar pseudorandom vector, $\mathsf{BasicDec}_2$ will also decode the same element.

In the second case, it must hold that several entries $m_1, \ldots, m_a$ got mapped to the same position, but it so happens that

$$\mathsf{Index}\Big(s_2, \big(\sum_{i=1}^{a} m_i\big)^{-1} \cdot \sum_{i=1}^{a} m_i \cdot \mathsf{Entry}(s_2, d)\Big) \in [n]$$

by using the pseudorandomness of the wunderbar pseudorandom vector and applying Corollary 12 we can conclude that this will happen for any particular cell with probability at most $(n^2 + n)/|K|$. Since there are $\rho\gamma$ cells, the lemma follows by a union bound over the number of cells. □

The following lemma states that deletion works as expected in both variants of the basic IBLTs described in Figure 4 and used as building blocks in the variants of the stacked IBLT described in Figure 3. That is, if a set S is encoded and a subset $\tilde{S}$ is *deleted* from the encoding, the result is *identical* to a fresh encoding of $S \setminus \tilde{S}$ in both constructions.

Lemma 15. *For any $r, \gamma, \rho \in \mathbb{Z}$, any PRF key s_1, any wunderbar pseudorandom vector s_2, any set $S \subseteq [n] \times \mathbb{F}_q$ such that for all distinct $(i,m),(i',m') \in S$, $i \neq i'$, and any subset $\tilde{S} \subseteq S$ it holds that*

$$\begin{aligned}&\mathsf{BasicDel}_1(\mathsf{BasicEnc}_1(\rho,\gamma,S,(r,s_1)),\tilde{S},(r,s_1))\\ =\ &\mathsf{BasicEnc}_1(\rho,\gamma,S\setminus\tilde{S},(r,s_1))\end{aligned}$$

and

$$\begin{aligned}&\mathsf{BasicDel}_2(\mathsf{BasicEnc}_2(\rho,\gamma,S,(r,s_1),s_2),\tilde{S},r,s_1,s_2)\\ =\ &\mathsf{BasicEnc}_2(\rho,\gamma,S\setminus\tilde{S},(r,s_1),s_2)\end{aligned}$$

where the probability is taken over the choice of $\boldsymbol{h}$, s_1, and s_2.

Proof. The lemma follows easily by observing that deletion exactly subtracts the values that were added during encoding in both cases. □

The following lemma now states that also the second variant of the stacked IBLT described in Figure 3 works as intended, as long as the wunderbar pseudorandom vector operates over a large enough universe K and the set S does not contain any zero entries.

Lemma 16. *Let* PRF *be a variable output domain pseudorandom function as defined in Definition 8. Let* $(\mathsf{Entry},\mathsf{Index})$ *be a wunderbar pseudorandom vector. Then there exists a large enough constant $C' > 0$ such that for any set $S \subseteq \mathbb{F}_q \times \{\mathbb{F}_q\}$ with $|S| \leq t$ and such that for all distinct $(i,m),(i',m') \in S$, $i \neq i'$ it holds that*

$$\begin{aligned}&\Pr[\mathsf{Decode}_2(\mathsf{Encode}_2(\rho,\gamma,S,s_1,s_2),s_1,s_2)=S]\\ \geq\ &1-2^{-\kappa}-\frac{C'(n^2+n)(t+\kappa\log\kappa)}{|K|}-\mathsf{negl}(\lambda)\end{aligned}$$

where the probability is taken over the uniform choice of s_1 and s_2.

Proof. Let $S \subseteq \mathbb{F}_q \times \{\mathbb{F}_q\}$ with $|S| \leq t$ be arbitrary. Consider the two decoding procedures running in parallel. Clearly, for the end result to differ, one of the executions of $\mathsf{BasicDec}_{1/2}$ has to result in different outputs.

Let $0 \leq \tilde{\imath} < \lceil \log t \rceil$ be an index, such that for all executions of $\mathsf{BasicDec}_{1/2}$ with $i \leq \tilde{\imath}$ the outputs were identical. Let $\tilde{S}$ be the set S' *before* the ith execution of $\mathsf{BasicDec}_{1/2}$. Clearly $\tilde{S}'$ is the same in both cases. Since $\mathsf{BasicDec}_1$ decodes elements if and only if they happen to be alone in their cell, $\mathsf{BasicDec}_1$ never causes any false positives and it must always hold that $\tilde{S} \subseteq S$. It thus follows from Lemma 15, that the outputs of the $\tilde{\imath}$th executions of $\mathsf{BasicDec}_{1/2}$ are

$$\mathsf{BasicDec}_1(\mathsf{BasicEnc}_1(\rho_{\tilde{\imath}},\gamma_{\tilde{\imath}},S\setminus\tilde{S},(\tilde{\imath},s_1)),(\tilde{\imath},s_1))$$

and

$$\mathsf{BasicDec}_2(\mathsf{BasicEnc}_2(\rho_{\tilde{\imath}},\gamma_{\tilde{\imath}},S\setminus\tilde{S},(\tilde{\imath},s_1),s_2),(\tilde{\imath},s_1),s_2)$$

for some choice of $\rho_{\tilde{\imath}}$ and $\gamma_{\tilde{\imath}}$.

By Lemma 14 the probability that the output differs is then at most $\rho_{\tilde{\imath}}\gamma_{\tilde{\imath}}(n^2+n)/|K|$. With a simple union bound over all indices $0 \leq \tilde{\imath} < \log t$ and by observing that the entire datastructure overall has $O(t+\kappa\log\kappa)$ cells it then follows that there exists some large enough constant C' such that the output of Decode_2 differs from the output of Decode_1 with probability at most

$$\frac{n^2+n}{|K|}\cdot\sum_{0\leq i<\log t}\rho_i\gamma_i \leq \frac{C'(n^2+n)(t+\kappa\log\kappa)}{|K|}.$$

Since by Lemma 13 the output of Decode_1 is correct with probability $1-2^{-\kappa}-\mathsf{negl}(\lambda)$, the lemma follows by another union bound. □

$\widehat{\mathsf{BasicEnc}}(\rho, \gamma, \boldsymbol{c}, (r, s_1), s_2))$

$\boldsymbol{M} := \mathsf{Enc}0^{\rho \times \gamma}$

$\boldsymbol{K} := \mathsf{Enc}(0^{\delta})^{\rho \times \gamma}$

foreach $d \in [|\boldsymbol{c}|]$

  foreach $i \in [\rho]$

    $j := \mathsf{PRF}(s_1, \gamma, (r, i, d))$

    $\boldsymbol{M}[i, j] := \boldsymbol{M}[i, j] \boxplus c_d$

    $\boldsymbol{k} := \mathsf{Entry}(s_2, d)$

    $\boldsymbol{K}[i, j] := \boldsymbol{K}[i, j] \boxplus (c_d \boxdot \boldsymbol{k})$

Fig. 5. The $\widehat{\mathsf{BasicEnc}}$ procedure is a modified version of $\mathsf{BasicEnc}_2$ to allow filling the filter under additively homomorphic encryption. Changes are marked in gray.

$\mathsf{Compress}(\mathsf{pk}, \boldsymbol{c})$	$\mathsf{Decompress}(\mathsf{sk}, (\boldsymbol{F}, s_1, s_2))$
$s_1 \leftarrow \{0,1\}^{\lambda}$	$\boldsymbol{F}' := \mathsf{Dec}(\mathsf{sk}, \boldsymbol{F})$
$s_2 \leftarrow \mathsf{Sample}(1^{\lambda}, 1^n)$	**return** $\mathsf{Decode}_2(\boldsymbol{F}', s_1, s_2))$
for $0 \le i < \log t - \log \tau$	
$F_i := \widehat{\mathsf{BasicEnc}}(1, \lceil Cn2^{-i} \rceil, \boldsymbol{c}, (i, s_1), s_2)$	
for $0 \le i < \log \tau$	
$i' := \lfloor \log t - \log \tau \rfloor$	
$F_{i'} := \widehat{\mathsf{BasicEnc}}(2^i, \lceil C\tau 2^{-i} \rceil, \boldsymbol{c}, (i', s_1), s_2)$	
return $((F_0, \ldots, F_{\lceil \log t \rceil - 1}), s_1, s_2)$	

Fig. 6. A ciphertext compression scheme for arbitrary additively homomorphic encryption schemes for $\mathcal{F}^{\mathsf{ip}}_{t,\mathsf{pk}}$.

We now specify a final variant of the basic encoding procedure in Figure 5. Essentially the only important difference between $\widehat{\mathsf{BasicEnc}}$ and $\mathsf{BasicEnc}_2$ is that the former acts on an encrypted version of the encoded set (represented by a vector of ciphertexts). This now finally allows us to state the actual ciphertext compression scheme in Figure 6 and we state the correctness of the compression scheme in Theorem 17.

Theorem 17. *Let $\mathcal{E} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ be an additively homomorphic encryption scheme with plaintext space $\mathbb{F}_q$ and ciphertext length $\xi = \xi(\lambda)$. Let $(\mathsf{Sample}, \mathsf{Entry}, \mathsf{Index})$ be a wunderbar pseudorandom vector with index recovery for a universe $K \subseteq \mathbb{F}^{\eta}$ with $|K| \ge C'(n^2 + n)(t + \kappa \log \kappa) \cdot 2^{\kappa}$ for a large enough constant $C' > 0$ and let PRF be a pseudorandom function with variable codomain. Then $(\mathsf{Compress}, \mathsf{Decompress})$ as specified in Figure 6 is a $(1 - 2^{-(\kappa-1)} - \mathsf{negl}(\lambda))$-correct $(\lambda + (t + \kappa \log \kappa)\eta\xi)/(n\xi))$ compressing ciphertext compression scheme for $\mathcal{F}^{\mathsf{ip}}_t$.*

Before we prove this theorem we will state the following simple corollary that follows simply by instantiating the construction with the wunderbar pseudorandom vector from Section 2.4 and holds for all reasonable encryption schemes with ciphertext size $\xi = \Omega(\lambda)$.

Corollary 18. *Let $\mathcal{E} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ be an additively homomorphic encryption scheme with plaintext space $\mathbb{F}_q$ and ciphertext length $\xi = \Omega(\lambda)$. Let PRP be a pseudorandom permutation over $\{0,1\}^{\kappa + C'' + 2\log n + \log(t + \kappa \log \kappa)}$ for some large enough constant C'' and let PRF be a pseudorandom function with variable codomain. Then $(\mathsf{Compress}, \mathsf{Decompress})$ as specified in Figure 6 can be instantiated to be a $(1 - 2^{-(\kappa-1)} - \mathsf{negl}(\lambda))$-correct and $\tilde{O}(\frac{\kappa t + \kappa^2}{n \log q})$ compressing[7] ciphertext compression scheme for $\mathcal{F}^{\mathsf{ip}}_t$.*

[7] The soft-O notation $\tilde{O}(\cdot)$ ignores log factors in κ and n.

Proof (Theorem 17). First observe that $\mathsf{Compress}$ executes exactly Encode_2 on the set $S = \{(d, m) \in [n] \times \mathbb{F}_q \mid \mathsf{Dec}(\mathsf{Gen}^{-1}(\mathsf{pk}), c_d)\}$ but under homomorphic encryption. Each cell in the encoding is computed as the inner product of the ciphertext vector and some plaintext vector. It thus follows from the $\mathcal{Z}_{\mathsf{ip}}$ validity of $\boldsymbol{c}$, that after decryption step in $\mathsf{Decompress}$ we have $\boldsymbol{F}' = \mathsf{Encode}_2(S, s_1, s_2)$. However, since by design any $(d, 0) \in S$ does not influence the value of $\boldsymbol{F}'$ we have in fact that $\boldsymbol{F}' = \mathsf{Encode}_2(S', s_1, s_2)$ where $S' := \{(d, m) \in S \mid m \neq 0\} = \mathsf{sparse}()$. Since $\boldsymbol{c} \in \mathcal{F}^{\mathsf{ip}}_{t,\mathsf{pk}}$ and thus $|S'| \leq t$, we can apply Lemma 16 that

$$
\begin{aligned}
& \Pr[\mathsf{Decompress}(\mathsf{sk}, \mathsf{Compress}(\mathsf{pk}, \boldsymbol{c})) = \mathsf{sparse}(\mathsf{Dec}(\mathsf{sk}, \boldsymbol{c})] \\
= \; & \Pr[\mathsf{Decode}_2(\mathsf{Encode}_2(S', s_1, s_2), s_1, s_2) = S'] \\
\geq \; & 1 - 2^{-\kappa} - \frac{O(n^2(t + \kappa \log \kappa))}{|K|} - \mathsf{negl}(\lambda) \\
= \; & 1 - 2^{-\kappa} - \frac{O(n^2(t + \kappa \log \kappa))}{\Omega(n^2(t + \kappa \log \kappa) \cdot 2^{\kappa}} - \mathsf{negl}(\lambda) \\
\geq \; & 1 - 2^{-(\kappa - 1)} - \mathsf{negl}(\lambda)
\end{aligned}
$$

as claimed.

To see the compression factor, consider that the output of $\mathsf{Compress}$ consists of s_1 and s_2, both of which have length $O(\lambda)$ as well as the encrypted stacked IBLT without counters. The IBLT consists of pairs of value and key matrices. The value matrices combined have $O(t + \kappa \log \kappa)$ entries of 1 ciphertext each and the key matrices combined have $O(t + \kappa \log \kappa)$ entries of η ciphertexts each. Thus overall the output of $\mathsf{Compress}$ has a length of $O(\lambda + (t + \kappa \log \kappa)\eta\xi)$ bits leading to the claimed compression factor. □

References

ACLS18. Sebastian Angel, Hao Chen, Kim Laine, and Srinath T. V. Setty. PIR with compressed queries and amortized query processing. In *2018 IEEE Symposium on Security and Privacy*, pages 962–979, San Francisco, CA, USA, May 21–23, 2018. IEEE Computer Society Press. doi:10.1109/SP.2018.00062. 1

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

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, 1.3

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

CRT06. Emmanuel J. Candès, Justin Romberg, and Terence Tao. Robust uncertainty principles: Exact signal reconstruction from highly incomplete frequency information. *IEEE Transactions on Information Theory*, 52(2):489–509, February 2006. doi:10.1109/TIT.2005.862083. 1.3

Don06. David L. Donoho. Compressed sensing. *IEEE Transactions on Information Theory*, 52(4):1289–1306, April 2006. doi:10.1109/TIT.2006.871582. 1.3

FLOS23. Nils Fleischhacker, Kasper Green Larsen, Maciej Obremski, and Mark Simkin. Invertible bloom lookup tables with less memory and randomness. Cryptology ePrint Archive, Report 2023/918, 2023. https://eprint.iacr.org/2023/918. 1, 1.1, 1.2, 2.2, 2.2, 7, 1, 2, 3, 3.2, 3.2

FLS23. Nils Fleischhacker, Kasper Green Larsen, and Mark Simkin. How to compress encrypted data. In Carmit Hazay and Martijn Stam, editors, *Advances in Cryptology – EUROCRYPT 2023, Part I*, volume 14004 of *Lecture Notes in Computer Science*, pages 551–577, Lyon, France, April 23–27, 2023. Springer, Heidelberg, Germany. doi:10.1007/978-3-031-30545-0_19. 1, 1.1, 1.2, 4, 1.3, 2.1, 2.1, 2.1, 2.4, 2.4, 3, 3.1, 10, 3.1, 3.2, 3.2

GI10. Anna Gilbert and Piotr Indyk. Sparse recovery using sparse matrices. *Proceedings of the IEEE*, 98(6):937–947, June 2010. doi:10.1109/JPROC.2010.2045092. 1.3

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, 2.2

IKOS04. Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, and Amit Sahai. Batch codes and their applications. In László Babai, editor, *36th Annual ACM Symposium on Theory of Computing*, pages 262–271, Chicago, IL, USA, June 13–16, 2004. ACM Press. doi:10.1145/1007352.1007396. 1

JWR04. Mark Johnson, David Wagner, and Kannan Ramchandran. On compressing encrypted data without the encryption key. In Moni Naor, editor, *TCC 2004: 1st Theory of Cryptography Conference*, volume 2951 of *Lecture Notes in Computer Science*, pages 491–504, Cambridge, MA, USA, February 19–21, 2004. Springer, Heidelberg, Germany. doi:10.1007/978-3-540-24638-1_27. 1.3

KHJ+09. Demijan Klinc, Carmit Hazay, Ashish Jagmohan, Hugo Krawczyk, and Tal Rabin. On compression of data encrypted with block ciphers. In James A. Storer and Michael W. Marcellin, editors, *DCC 2009: 19th Data Compression Conference*, pages 213–222, Snowbird, UT, USA, March 16–18 2009. IEEE Computer Society Press. doi:10.1109/DCC.2009.71. 1.3

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, 1.3

MR23. Muhammad Haris Mughees and Ling Ren. Vectorized batch private information retrieval. In Thomas Ristenpart and Patrick Traynor, editors, *2023 IEEE Symposium on Security and Privacy*, pages 1812–1827, San Francisco, CA, USA, May 22–25 2023. IEEE Computer Society Press. doi:10.1109/SP46215.2023.00104. 1

SW73. Davoid Slepian and Jack Wolf. Noiseless coding of correlated information sources. *IEEE Transactions on Information Theory*, 19(4):471–480, July 1973. doi:10.1109/TIT.1973.1055037. 1.3

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. Association for Computing Machinery. doi:10.1145/2517488.2517497. 1