Invertible Bloom Lookup Tables with Less Memory and Randomness

Nils Fleischhacker
Ruhr University Bochum, Germany

Kasper Green Larsen
Aarhus University, Denmark

Maciej Obremski
National University of Singapore, Singapore

Mark Simkin
Ethereum Foundation, Aarhus, Denmark

Abstract

In this work we study Invertible Bloom Lookup Tables (IBLTs) with small failure probabilities. IBLTs are highly versatile data structures that have found applications in set reconciliation protocols, error-correcting codes, and even the design of advanced cryptographic primitives. For storing n elements and ensuring correctness with probability at least $1-\delta$, existing IBLT constructions require $\Omega(n(\frac{\log(1/\delta)}{\log(n)}+1))$ space and they crucially rely on fully random hash functions.

We present new constructions of IBLTs that are simultaneously more space efficient and require less randomness. For storing n elements with a failure probability of at most δ, our data structure only requires $\mathcal{O}(n+\log(1/\delta)\log\log(1/\delta))$ space and $\mathcal{O}(\log(\log(n)/\delta))$-wise independent hash functions.

As a key technical ingredient we show that hashing n keys with any k-wise independent hash function $h : U \to [Cn]$ for some sufficiently large constant C guarantees with probability $1-2^{-\Omega(k)}$ that at least $n/2$ keys will have a unique hash value. Proving this is non-trivial as k approaches n. We believe that the techniques used to prove this statement may be of independent interest.

We apply our new IBLTs to the encrypted compression problem, recently studied by Fleischhacker, Larsen, Simkin (Eurocrypt 2023). We extend their approach to work for a more general class of encryption schemes and using our new IBLT we achieve an asymptotically better compression rate.

2012 ACM Subject Classification Theory of computation → Bloom filters and hashing

Keywords and phrases Invertible Bloom Lookup Tables

Digital Object Identifier 10.4230/LIPIcs.ESA.2024.54

Related Version *Full Version*: `https://eprint.iacr.org/2023/918` [8]

Funding *Nils Fleischhacker*: Funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy - EXC 2092 CASA - 390781972.
Kasper Green Larsen: Supported by a DFF Sapere Aude Research Leader grant No 9064-00068B.
Maciej Obremski: Supported by the MOE tier 2 grant T2EP20223-0024, Breaking the Box – On Security of Cryptographic Devices.

1 Introduction

The Invertible Bloom Lookup Table (IBLT) is a very elegant data structure by Goodrich and Mitzenmacher [11]. It functions much like a dictionary data structure, supporting insertions, deletions and the retrieval of key-value pairs. What is special about the IBLT, is that upon initialization, one decides on a threshold n. Now, regardless of how many key-value pairs are present in the IBLT, the space usage will always remain proportional to n. Of course this comes at a cost, namely that the retrieval operations will temporarily stop functioning, when the number of pairs stored in the IBLT exceeds n. When the number of stored pairs falls below n again, the IBLT will resume supporting retrieval queries.

The above functionality is extremely useful in many applications. Consider for instance the set reconciliation problem [15, 6]. Here two parties Alice and Bob hold sets S_A and S_B of key-value pairs. Think of these sets as two replicas of a database storing key-value pairs. In applications where insertions and deletions into the database must be supported quickly, we may allow the two sets S_A and S_B to be slightly inconsistent, such that a client performing an operation on the database will not have to wait for synchronization among the two replicas. Instead, Alice and Bob will every now and then synchronize their two sets S_A and S_B. For this purpose, Alice maintains an IBLT for her set S_A, which she may send to Bob. Upon receiving the IBLT, Bob then deletes every element from his set S_B from Alice's IBLT. If $|(S_A \setminus S_B) \cup (S_B \setminus S_A)|$ is less than the threshold n, Bob can retrieve the key-value pairs in $S_A \setminus S_B$. Since the space usage of IBLTs is only proportional to the threshold n, this allows for the communication between Alice and Bob to be proportional to $|(S_A \setminus S_B) \cup (S_B \setminus S_A)|$ and not $|S_A|$ or $|S_B|$. This may result in significant savings, when the sets S_A and S_B are large, but very similar. IBLTs have also seen uses in numerous other applications, ranging from distributed systems applications [18, 16] over fast error-correcting codes [17] to cryptography [1, 9, 10].

The surprising functionality of IBLTs is supported via hashing. In more detail, the original IBLT construction by Goodrich and Mitzenmacher consists of an array A of m cells along with a hash function h mapping keys to k distinct entries in A for a tuneable parameter k. Each cell of A has three fields, a *count*, a *keySum* and a *valueSum*. When inserting a key-value pair (x, y), we compute the k positions $h(x) = (i_1, \ldots, i_k)$, increment the *count* field in $A[i_j]$, add x to the *keySum* of $A[i_j]$ and add y to the *valueSum* of $A[i_j]$ for each $j = 1, \ldots, k$. A deletion of a key-value pair is simply supported by reversing these operations, i.e. decrementing *count* and subtracting x from *keySum* and y from *valueSum*. To support the retrieval of the value associated with a query key x, we again compute $h(x) = (i_1, \ldots, i_k)$ and examine the entries $A[i_j]$. If we find such an entry where the *count* field is one, then we know that only one key-value pair hashed there. We can thus compare the *keySum* to x, and if they are equal, we can return the *valueSum*. If the *keySum* is different from x, or we find a cell with a *count* of zero, we may return that x is not in the IBLT. Finally, if all k *count* fields are at least two, we return "Don't know". If the number of cells m is $2nk$, then the chance that a key-value pair hashes to at least one unique entry (no collisions) is around $1 - 2^{-\Omega(k)}$ whenever the number of key-value pairs stored in the IBLT does not exceed the threshold n.

Peeling

The simple functionality above supports Insertions, Deletions and Get operations, where a Get operation retrieves the value associated with a query key x. Using space $O(nk)$, the Get operation succeeds with probability $1 - 2^{-\Omega(k)}$. However, in several applications, such as set reconciliation, one is more interested in outputting the list of all key-value pairs present in the IBLT. For this purpose, a ListEntries operation is also supported. To list all key-value pairs in the IBLT, we repeatedly look for a cell in A with a *count* of one. When we find such a cell $A[i]$, we output $(x, y) =(A[i].\mathit{keySum}, A[i].\mathit{valueSum})$ and then delete (x, y) from the IBLT. This process of *peeling* the key-value pairs reduces the *count* of other fields and thus increases the chance that we can continue peeling key-value pairs. Concretely, the ListEntries operation can be shown to succeed with probability $1 - \Omega(n^{-k+2})$ when the number of key-value pairs present in the IBLT does not exceed the threshold n. The peeling success probability thus far exceeds that of the simple Get operation when hashing to at least $k = 3$ entries.

Supporting False Deletions

The attentive reader may have observed that the simple version of the IBLT described above critically assumes that no deletions are performed on key-value pairs that are not already present in the IBLT. In the set reconciliation example, this is insufficient as there may be key-value pairs in S_B that are not in S_A, which will cause false deletions. A simple extension to the IBLT ensures that it also functions if the total number of present key-value pairs plus the number of false deletions does not exceed the threshold n. For set reconciliation, this is equivalent to $|S_A \setminus S_B| + |S_B \setminus S_A| \leq n$. To support such false deletions, we add a *hashSum* field to every cell and include another hash function g mapping keys to a sufficiently large output domain $[R]$. When inserting key-value pairs, $g(x)$ is added to the *hashSum* field of $A[i_j]$ and subtracted during deletions. To retrieve the value associated with a key x, we proceed as before, but whenever the *count* is either -1 or 1, we also perform a check that the *hashSum* is equal to g applied to the *keySum*. If not, we treat the cell as if the count was at least 2. For ListEntries, a peeling operation also includes such checks and furthermore, when a *count* is -1, we may instead insert $(x, y) =$(-*keySum*,-*valueSum*) if g applied to -*keySum* equals -*hashSum*. A second source of error is when the same key has been inserted with multiple different values. We ignore this issue here, and remark that the ListEntries in the original IBLT also fails in recovering keys with multiple associated values.

Memory Usage and Randomness

In this paper, we focus on the more interesting ListEntries operation and ignore the Get operation. Requiring that ListEntries succeeds with probability $1 - \delta$, the classic IBLT uses space $O(n(\lg(1/\delta)/\lg n + 1))$, since we must set $k = O(1 + \lg_n(1/\delta))$ to make $n^{-k+1} \leq \delta$, and the space usage is $m = O(nk)$ cells. Notice here, and throughout the paper, that space is measured in number of *cells* of the IBLT. In terms of bit complexity, the *count* field needs $O(\lg n)$ bits, the *keySum* and *valueSum* fields need $O(\lg |U| + \lg n)$ bits when keys and values come from a universe U. Finally, in both previous IBLTs and our new construction, the *hashSum* field needs $O(\lg(1/\delta) + \lg n)$ bits. Thus each cell of the table costs $O(\lg(|U|n/\delta))$ bits.

The analysis of the classic IBLT critically assumes that the hash function h is truly fully random. This is of course unrealistic in practice. But where many typical data structures can make due with $O(\lg(1/\delta))$ or $O(\lg n)$-wise independent hash functions, this is not known to be the case for the IBLT. Concretely, the standard analysis of the peeling process of the IBLT requires a union bound over exponentially many events (for every set of $2 \leq j \leq n$ keys S, for every set T of $jk/2$ entries of A, we have a failure event saying that $h(x) \in T$ for all $x \in S$). With exponentially many events in the union bound, each of them must occur with probability at most $\exp(-\Omega(n))$ for the union bound to be useful. This requires a seed length of $\Omega(n)$ bits for a hash function and thus cannot be implemented with k-wise independence for k significantly less than n. It could be the case that a more refined analysis could show that less randomness suffices, but this has not yet been demonstrated.

We remark that it is possible to show that tabulation hashing [4, 19] may be used to support peeling, but this also requires a random seed of length proportional to n, since it requires a character size of at least $(1 + \Omega(1))n$, and the space usage is at least the number of characters. Finally, we mention that it may also be possible to use the splitting trick of Dietzfelbinger and Rink [5], but as far as we are aware of, it would be not more efficient than tabulation hashing in this context.

1.1 Our Contributions

Our main contribution is a new version of the IBLT that is both more space efficient and that can be implemented with much less randomness. We call our new data structure a Stacked IBLT and show the following:

▶ **Theorem 1.** *Let δ be less than a sufficiently small constant. Given a threshold n, the Stacked IBLT supports Insertions, Deletions and ListEntries operations, where ListEntries succeeds with probability $1-\delta$ when the number of key-value pairs is no more than n. Furthermore, it uses space $O(n + \lg(1/\delta)\,\mathrm{lglg}(1/\delta))$ cells and requires only $O(\lg(\lg(n)/\delta))$-wise independent hashing.*

Comparing this to the classic IBLT, our construction outperforms it for any $\delta = n^{-\omega(1)}$ and more importantly, it can be implemented with a small random seed. Our Stacked IBLT also supports false deletions like the classic IBLT and ListEntries succeeds with the claimed probability if the number of key-value pairs plus the number of false deletions does not exceed n.

We note that such small failure probabilities are important in cryptographic applications, like the ones that rely on encrypted compression [3, 14]. A data-dependent failure of a data structure leaks information about its contents, even if one can not see the contents of the data structure itself. In cryptographic applications, where security should commonly break with at most a negligible probability, using a (encrypted) data structure, which fails with an inverse polynomial probability is insufficient. An adversary could deduce information about encrypted data by just observing, whether a cryptographic protocol successfully terminates or not.

The overall idea in the Stacked IBLT is to construct arrays $A_1, \ldots, A_{\lg n}$ where A_i has $Cn/2^i$ entries. Each of the arrays has its own hash function h_i mapping keys to a single entry in A_i. To support the ListEntries operation, we start by peeling all elements in A_1 that hash uniquely. We then proceed to A_2 and so forth. The critical property we require is that each time we peel, we successfully peel at least half of all remaining key-value pairs. In this way, the number of entries in the next A_i to peel from, is always a constant factor larger than the number of remaining key-value pairs. When we reach $A_{\lg n}$, we finally peel the last key-value pair. In this way, all we need from the hash functions h_i, is that at least half the key-value pairs hash uniquely with probability $1 - \delta/\lg n$. We prove that this is the case if the h_i's are just $O(\lg(\lg(n)/\delta))$-wise independent:

▶ **Theorem 2.** *Let $x_1, \ldots, x_n \in U$ be a set of n distinct keys from a universe U and let $h : U \to [Cn]$ be a hash function drawn from a $2k$-wise independent family of hash functions. If $C \geq 8e$, then with probability at least $1 - 4 \cdot (8e/C)^{\min\{k,n/C\}}$ it holds that there are no more than $n/2$ indices i such that there exists a $j \neq i$ with $h(x_i) = h(x_j)$.*

In addition to allowing implementations with limited independence, the geometrically decreasing sizes of the arrays A_i also result in the improved space usage compared to classic IBLTs.

While Theorem 2 might at first sight appear to follow from standard approaches for analyzing hash functions with limited independence, there are in fact several difficult obstacles that we need to overcome to prove it. In particular, as k approaches n, the obvious approaches fail miserably. Furthermore, our Stacked IBLTs critically needs Theorem 2 to hold for k all the way up to n. We believe the ideas we use to overcome this barrier are interesting in their own right and may prove useful in future work. We thus discuss these ideas and the barriers we overcome in Section 1.2.

Let us also comment on the constant $8e$. It is not as small as one could hope, but it is small enough that we have chosen to state it explicitly rather than hide it in O-notation. Presumably our analysis could be tightened further to reduce it by a constant factor, but we have focused on a clean exposition of the proof.

Finally, let us also comment that when the number of remaining key-value pairs drop below $\lg(1/\delta)$, Theorem 2 is insufficient to guarantee a success probability of $1 - \delta/\lg n$ due to the $\min\{k, n/c\}$ in the exponent. For this reason, we change strategy and replace some of the arrays A_i by matrices with multiple rows. We leave the details to later sections and mention here that this is what causes the $\mathcal{O}(\lg(1/\delta)\operatorname{lglg}(1/\delta))$ term in the space usage of the Stacked IBLT.

In terms of computational efficiency our construction is slightly worse than that of Goodrich and Mitzenmacher. Retrieving all key-value pairs from their IBLT has a computational cost of $\mathcal{O}(n \cdot (1 + \lg(1/\delta)/\lg(n))$, while our construction requires $\mathcal{O}(n \cdot \lg(n/\delta))$. In our opinion, however, this is a small price to pay for achieving smaller IBLTs that require less randomness.

Encrypted Compression

We apply our new data structure to the encrypted compression problem, studied by Fleischhacker, Larsen, and Simkin [10]. Here one is given an array of ciphertexts of a homomorphic encryption scheme, where at most t are encryptions of non-zero values. The goal of an encrypted compression scheme is to compress this vector as much as possible, without knowing what is inside the ciphertexts, i.e. without knowing which entries in the vector are encryptions of zero and which are not. Apart from being theoretically interesting, this problem also naturally appears as part of larger cryptographic protocols [3, 14]. We show that following the approach of Fleischhacker, Larsen, and Simkin one can use our stacked IBLT data structure to obtain better encrypted compression schemes. Additionally, we show how their approach can be generalized to work for arbitrary homomorphic encryption schemes. Note that their work, required the encryption schemes to have plaintext spaces that grow at least linearly with the desired upper bound on the error rate of their data structure. Due to the page limit, we defer the formal description of this application, along with the full construction and proof to the full version of this paper [8].

Rateless IBLTs

In a work subsequent to ours, Yang, Gilad, Alizadeh [20] consider the setting of rateless IBLTs. Here an encoder has a fixed set of source symbols and would like to encode them into an infinite sequence of coded symbols. Without going into detail, these coded symbols should have several high-level properties: The computation of the coded symbols should not depend on a fixed a-priori threshold of how many source symbols will be in the data structure. The sequence of generated coded symbols should be linear in the sense that two sequences of coded symbols can be subtracted to obtain a sequence of coded symbols that represents the set difference of the corresponding sets. For any number of source symbols, one should be able to decode them back from a sufficiently long prefix of the sequence of coded symbols.

As noted by Yang, Gilad, Alizadeh, the IBLT of Goodrich and Mitzenmacher [11] does not satisfy these properties as the size of the data structure needs to be fixed at the start and there is no clear way of viewing it as a infinite sequence of coded symbols. We will not prove this formally in our work, but note that our stacked IBLTs naturally have these properties, as they can be constructed starting from the smallest array and repeatedly building the larger arrays on top of it, viewing the array cells as coded symbols.

Some More Related Works

A variant of IBLTs that may appear similar to ours are irregular IBLTs, as originally already suggested by Goodrich and Mitzenmacher [11] and also studied by Lázaro and Balázs Matuz [13], where different set elements are encoded using a different amount of hash functions. We note that our construction is regular, since it is oblivious to the specific value of any one set element and all elements get treated equally. We believe this to be helpful for applications, like encrypted compression, where the set elements are not visible to the encoder generating the data structure.

In a recent work, that appeared subsequent to ours, by Belazzougui, Kucherov, Walzer [2], the authors consider IBLTs with very small failure probabilities as we do here. The idea behind their construction is to augment the original IBLT of Goodrich and Mitzenmacher with a smaller backup stash data structure. When decoding of the main IBLT fails, their peeling resorts to recovering the missing elements from the stash. In comparison, our stacked IBLTs can conceptually be seen as an iterative version of this idea, as we have a sequence of smaller and smaller "stashes", moving on to peeling the smaller ones, when peeling the bigger ones fails repeatedly. Furthermore, their work considers fully random hash functions, whereas our work gets away with using hash functions with limited independence. Their construction results in a sketch that is asymptotically comparable in size and has a better expected, but worse worst-case decoding time.

1.2 Technical Contributions

When analysing events involving hash functions of limited independence, one typically considers higher moments of a sum of random variables that each depends only on a constant number of hash values. For our Theorem 2, the natural random variables to consider would be the random variables $X_{i,j}$ taking the value 1 if $h(x_i) = h(x_j)$. Clearly there are no more than $n/2$ indices i such that there exists $j \neq i$ with $h(x_i) = h(x_j)$ if $\sum_{i \neq j} X_{i,j} \leq n/2$. To upper bound $\Pr[\sum_{i \neq j} X_{i,j} > n/2]$, we raise both sides of the inequality to the k'th power and use that $\Pr[\sum_{i \neq j} X_{i,j} > n/2] = \Pr[(\sum_{i \neq j} X_{i,j})^k > (n/2)^k]$. Using Markov's inequality, this probability is at most $\mathbb{E}[(\sum_{i \neq j} X_{i,j})^k]/(n/2)^k$. Expanding the k'th power of the sum into a sum of monomials and using linearity of expectation, we have $\mathbb{E}[(\sum_{i \neq j} X_{i,j})^k] = \sum_{T \in \{(i,j):i \neq j\}^k} \mathbb{E}[\prod_{(i,j) \in T} X_{i,j}]$. Since each product depends on at most $2k$ hash values, and h is $2k$-wise independent, we can analyse each monomial as if h was truly random.

For the purpose of proving our theorem, this approach actually suffices to establish the theorem for $k < \sqrt{n}$. However, for our application in IBLTs we need the theorem to hold for k up to $\Omega(n)$. The problem is that as k approaches n, using that $\sum_{i \neq j} X_{i,j}$ is small as a proxy for having many elements hash to a unique position is lossy. In essence, this is because ℓ elements hashing to the same value contributes around ℓ^2 to $\sum_{i \neq j} X_{i,j}$ whereas it actually only corresponds to ℓ elements not hashing to a unique value. For this reason, $\mathbb{E}[(\sum_{i \neq j} X_{i,j})^k]$ is simply too large to give a meaningful bound from Markov's inequality when $k = \Omega(\sqrt{n})$. In fact, it is not only the higher-moments method that is doomed, but any approach based on arguing that $\Pr[\sum_{i \neq j} X_{i,j} > n/2]$ is small will fail. Consider for instance the case where k is $\Theta(n)$. Our Theorem 2 shows that the probability that less than $n/2$ keys hash uniquely is $\exp(-\Omega(n))$. If we consider $\sum_{i \neq j} X_{i,j}$ and even assume that h is truly random, then the probability that the first $n/\lg n$ keys all hash to the first $n/\lg^3 n$ entries is $(C \lg^3 n)^{-n/\lg n} \geq \exp(-O(n \lg\lg n/\lg n))$ for constant $C > 0$. But when this happens, we have $\sum_{i \neq j} X_{i,j} \geq (n/\lg^3 n) 2 \binom{\lg^2 n}{2} \approx n \lg n$. That is, $\Pr[\sum_{i \neq j} X_{i,j} > n/2] \geq \exp(-O(n \lg\lg n/\lg n))$.

In light of this, it is not a priori clear which random variables are sensible to analyse, keeping in mind that they should depend on only few hash values (for the sake of limited independence) and yet accurately capture the event that at least $n/2$ elements hash to a unique value. We present two alternative proofs circumventing this barrier.

In the first, and completely self-contained proof, we carefully define random variables $Y_{i,j}$ that actually depend *on all* hash values. We then consider the k'th moment of a sum involving these $Y_{i,j}$'s and argue that most monomials are 0 due to the special definition of the $Y_{i,j}$'s. Now that there are only very few non-zero monomials left, we upper bound our $Y_{i,j}$'s by the $X_{i,j}$'s above, bringing us back into a setup with monomials depending on at most $2k$ hash values. Compared to going directly from the $X_{i,j}$'s, what we win is that there are much fewer monomials left in the sum. The initial pruning of monomials using the more involved $Y_{i,j}$'s is a key technical innovation that we have not seen before and believe may be an inspiration in future work analysing random variables of limited independence.

In the second proof, we invoke a previous theorem on k-wise independence fooling combinatorial rectangles [7, 12]. This proof is shorter than the first, but relies on the heavy lifting done in previous works and does not yield the explicit small constant in our theorem.

2 Preliminaries

Let X, Y be sets, we denote by $|X|$ the size of X and by $X \triangle Y$ the symmetric set difference of X and Y, i.e., $X \triangle Y = (X \cup Y) \setminus (X \cap Y) = (X \setminus Y) \cup (Y \setminus X)$. We write $x \leftarrow X$ to denote the process of sampling a uniformly random element $x \in X$. Let $\boldsymbol{v} \in X^n$ be a vector. We write v_i to denote its i-th component. Let $\boldsymbol{M} \in X^{n \times m}$ be a matrix. We write $\boldsymbol{M}[i,j]$ to denote the cell in the i-th row and j-th column. We write $[n]$ to denote the set $\{1, \ldots, n\}$. We write lg without a specified base to denote the logarithm to base two.

3 Hashing Uniquely with Limited Independence

In this section, we prove our main technical result, Theorem 2, which we restate here for convenience.

▶ **Theorem 2** (restated). *Let $x_1, \ldots, x_n \in U$ be a set of n distinct keys from a universe U and let $h : U \to [Cn]$ be a hash function drawn from a $2k$-wise independent family of hash functions. If $C \geq 8e$, then with probability at least $1 - 4 \cdot (8e/C)^{\min\{k, n/C\}}$ it holds that there are no more than $n/2$ indices i such that there exists a $j \neq i$ with $h(x_i) = h(x_j)$.*

As discussed in Section 1.2, the straight forward approach of analysing moments of a sum $\sum_{i<j} X_{i,j}$ with $X_{i,j}$ being an indicator for $h(x_i) = h(x_j)$, does not give the desired result. In essence, this is because a collision of ℓ elements contributes roughly ℓ^2 to the sum.

In this section, we present two alternative proofs circumventing this barrier. We start by giving the self-contained proof that introduces an elegant new trick to analysing k-wise independent random variables. We then give a proof invoking results on k-wise independence fooling combinatorial rectangles. The remark that the second proof does not yield the explicit constants in Theorem 2.

3.1 Proof via Moments

Our first step in the proof of Theorem 2 is thus to make a far less obvious definition of random variables.

Proof. Define random variables $Y_{i,j}$ with $i \neq j$ taking the value 1 if $h(x_i) = h(x_j)$ and furthermore, for all a with $\min\{i,j\} < a < \max\{i,j\}$ we have $h(x_i) \neq h(x_a)$. Otherwise, $Y_{i,j}$ takes the value 0. Observe that if elements $x_{i_1}, \dots, x_{i_\ell}$ are all those that hash to a concrete value v, and $i_1 < i_2 < \cdots < i_\ell$, then $Y_{i_1,i_2} = Y_{i_2,i_1} = Y_{i_2,i_3} = \cdots = Y_{i_\ell,i_{\ell-1}} = 1$ and all other $Y_{i,j}$'s with i or j in $\{i_1, \dots, i_\ell\}$ are zero. The random variable $Y_{i,j}$ is thus 1 if x_i and x_j hash to the same v, and furthermore, i and j are consecutive in the sorted order of all elements hashing to v. Critically, a collision of ℓ elements contribute only $2\ell - 2$ to $\sum_{i \neq j} Y_{i,j}$. On the negative side, these random variables $Y_{i,j}$ clearly depend on more than two hash values unlike the $X_{i,j}$'s.

Letting $S = \{x_1, \dots, x_n\}$, observe that if there more than $n/2$ keys $x \in S$ such that there is a $y \in S \setminus \{x\}$ with $h(x) = h(y)$, then $\sum_{i \neq j} Y_{i,j} > n/2$. Let $r = \min\{k, n/C\}$. Using Markov's, we get

$$\Pr\left[\sum_{i \neq j} Y_{i,j} > n/2\right] = \Pr\left[\left(\sum_{i \neq j} Y_{i,j}\right)^r > (n/2)^r\right] < \frac{\mathbb{E}\left[\left(\sum_{i \neq j} Y_{i,j}\right)^r\right]}{(n/2)^r}. \tag{1}$$

We thus focus on bounding $\mathbb{E}[(\sum_{i \neq j} Y_{i,j})^r]$. Expand it into its monomials

$$\mathbb{E}\left[\left(\sum_{i \neq j} Y_{i,j}\right)^r\right] = \sum_{(i_1,j_1),\dots,(i_r,j_r)} \mathbb{E}\left[\prod_{h=1}^{r} Y_{i_h,j_h}\right].$$

Here the sum ranges over all lists of r pairs (i_h, j_h) with $i_h \neq j_h$. Notice that the product is 1 if and only if all the indicators involved are 1. For a monomial $\prod_{h=1}^{r} Y_{i_h,j_h}$, think of the pairs (i_h, j_h) as edges of a graph with the elements $x_1, \dots, x_n$ as nodes. The critical observation is that if any node in this graph has at least three distinct neighbors, then $\prod_{h=1}^{r} Y_{i_h,j_h} = 0$. To see this, assume the node x_i has at least three distinct neighbors. If x_i has two neighbors x_{j_1}, x_{j_2} with $j_1 < j_2 < i$, then we cannot have both $Y_{j_1,i} = Y_{i,j_1} = 1$ and $Y_{j_2,i} = Y_{i,j_2} = 1$. This is because, by definition, $Y_{j_1,i}$ can only be 1 if there are no elements x_a with $h(x_a) = h(x_{j_1})$ and $j_1 < a < i$. But $a = j_2$ is an example of such an element when we also require $Y_{j_2,i} = Y_{i,j_2} = 1$. A similar argument applies to the case that x_i has two neighbors x_{j_1}, x_{j_2} with $i < j_1 < j_2$. Notice that this also implies that the monomial is 0 if the corresponding graph has a cycle since the node of largest index on the cycle has an edge to two distinct neighbors of lower index. In combination, the monomial can only be non-zero if the corresponding edges form connected components corresponding to paths (possibly with duplicate edges).

Let $\mathcal{G}^r$ denote the set of all ordered lists $\mathcal{L}$ of r pairs $\mathcal{L} := (i_1, j_1), \dots, (i_r, j_r)$ (with $i_h \neq j_h$ for all h) such that every connected component in the corresponding graph $G(\mathcal{L})$ forms a path. Then

$$\mathbb{E}\left[\left(\sum_{i \neq j} Y_{i,j}\right)^r\right] = \sum_{\mathcal{L} \in \mathcal{G}^r} \mathbb{E}\left[\prod_{(i,j) \in \mathcal{L}} Y_{i,j}\right].$$

Now consider a monomial $\prod_{(i,j) \in \mathcal{L}} Y_{i,j}$ for an $\mathcal{L} \in \mathcal{G}^r$. Define $X_{i,j}$ as the random variable taking the value 1 if $h(x_i) = h(x_j)$ and 0 otherwise. Here we use that $Y_{i,j} \leq X_{i,j}$ and thus $\prod_{(i,j) \in \mathcal{L}} Y_{i,j} \leq \prod_{(i,j) \in \mathcal{L}} X_{i,j}$. Therefore

$$\mathbb{E}\left[\left(\sum_{i \neq j} Y_{i,j}\right)^r\right] \leq \sum_{\mathcal{L} \in \mathcal{G}^r} \mathbb{E}\left[\prod_{(i,j) \in \mathcal{L}} X_{i,j}\right].$$

What we have achieved is to upper bound $\mathbb{E}[(\sum_{i<j} Y_{i,j})^r]$ by the contribution from monomials corresponding to graphs consisting of paths. Furthermore, for these monomials, we have replaced the $Y_{i,j}$ variables by the simpler $X_{i,j}$ variables that each only depend on two hash values. This allows us to handle the limited independence of h.

Next, we bound $\mathbb{E}[\prod_{(i,j)\in\mathcal{L}} X_{i,j}]$ for an $\mathcal{L} \in \mathcal{G}^r$. With the graph interpretation $G(\mathcal{L})$ of $\mathcal{L}$ in mind, we observe that the product is 1 if and only if, for every connected component in $G(\mathcal{L})$, all nodes in the component hash to the same value. Furthermore, the monomial depends on at most $2r \le 2k$ hash values and thus the random variables behave as if h was truly random. For a connected component with q_i nodes, the probability all nodes hash to the same is precisely $(Cn)^{-(q_i-1)}$. If the total number of nodes in $G(\mathcal{L})$ having at least one neighbor is q and the total number of connected components in $G(\mathcal{L})$ formed by these nodes and their edges is c, then

$$\mathbb{E}\left[\prod_{(i,j)\in\mathcal{L}} X_{i,j}\right] = (Cn)^{-q+c}.$$

For every $q \le 2r$ and every $c \le q/2$, let $\mathcal{G}^r_{q,c} \subseteq \mathcal{G}^r$ be the subset of lists $\mathcal{L}$ for which the corresponding graph $G(\mathcal{L})$ has c non-singleton connected components and those connected components together have q nodes. Then

$$\mathbb{E}\left[\left(\sum_{i\neq j} Y_{i,j}\right)^r\right] \le \sum_{q=2}^{2r}\sum_{c=1}^{q/2}\sum_{\mathcal{L}\in\mathcal{G}^r_{q,c}} \mathbb{E}\left[\prod_{(i,j)\in\mathcal{L}} X_{i,j}\right] = \sum_{q=2}^{2r}\sum_{c=1}^{q/2} |\mathcal{G}^r_{q,c}|(Cn)^{-q+c}.$$

We thus need to bound $|\mathcal{G}^r_{q,c}|$. Here we show the following

▶ **Lemma 3.** *For all $q \le 2r$, $c \le q/2$ it holds that*

$$|\mathcal{G}^r_{q,c}| \le \left(\frac{4er}{q}\right)^{q-c} 2^r q^r n^q q^{-c}.$$

Before we prove the lemma, let us use to finish our proof of Theorem 2. Continuing our calculations above using Lemma 3, we have that

$$\begin{aligned} |\mathcal{G}^r_{q,c}|(Cn)^{-q+c} &\le \left(\frac{4er}{qCn}\right)^{q-c} 2^r q^r n^q q^{-c} \\ &= \left(\frac{4er}{qC}\right)^{q} \left(\frac{4er}{Cn}\right)^{-c} 2^r q^r. \end{aligned}$$

Since we set $r = \min\{k, n/C\}$ and require $C \ge 8e$, we have $(4er/(Cn)) \le 1/4$ and thus exploiting that the sum over c is a geometric series we get

$$\sum_{c=1}^{q/2} |\mathcal{G}^r_{q,c}|(Cn)^{-q+c} \le 2\left(\frac{4er}{qC}\right)^{q}\left(\frac{4er}{Cn}\right)^{-q/2} 2^r q^r = 2\left(\frac{4ern}{Cq^2}\right)^{q/2} 2^r q^r$$

Using again that $n/C \ge r$ and $r \ge q/2$, we have $4ern/(Cq^2) \ge 4er^2/q^2 \ge e$ and thus we may again use a geometric series to conclude

$$\mathbb{E}\left[\left(\sum_{i\neq j} Y_{i,j}\right)^r\right] \le \sum_{q=2}^{2r}\sum_{c=1}^{q/2} |\mathcal{G}^r_{q,c}|(Cn)^{-q+c} \le 4\left(\frac{4ern}{C(2r)^2}\right)^r (4r)^r = 4\left(\frac{4en}{C}\right)^r.$$

Plugging this back into the bound (1) we got from Markov's inequality, we finally conclude

$$\Pr\left[\sum_{i\neq j} Y_{i,j} > n/2\right] \leq 4 \cdot \left(\frac{8e}{C}\right)^r.$$

Recalling that $r = \min\{k, n/C\}$ completes the proof. ◀

Counting Graphs (Proof of Lemma 3)

To bound $|\mathcal{G}^r_{q,c}|$, we first recall that every $\mathcal{L} \in \mathcal{G}^r_{q,c}$ corresponds to a graph consisting of c non-singleton connected components, each forming a path of q_i nodes with $q = \sum_i q_i$. The set of (undirected) edges in $G(\mathcal{L})$ thus has cardinality $q - c \leq r$. We now argue that any $\mathcal{L} \in \mathcal{G}^r_{q,c}$ can be uniquely described by an element in

$$\mathcal{U} := \binom{r}{q-c} \times (\{0,1\} \times [q-c])^{r-(q-c)} \times \binom{2(q-c)}{q} \times [n]^q \times [q]^{2(q-c)-q}.$$

Here $\binom{r}{q-c}$ is the set of all $(q-c)$-sized subsets of a universe of cardinality r. Notice that this indirectly specifies a surjective function from $\mathcal{U}$ to $\mathcal{G}^r_{q,c}$ and thus

$$|\mathcal{G}^r_{q,c}| \leq \binom{r}{q-c}(2(q-c))^{r-(q-c)}\binom{2(q-c)}{q} n^q q^{q-2c}.$$

To describe an $\mathcal{L} \in \mathcal{G}^r_{q,c}$ with an element from $\mathcal{U}$, use an element in $\binom{r}{q-c}$ to specify the first occurence of each edge in $\mathcal{L}$ (where an edge (i,j) is first if neither (i,j) or (j,i) occurs earlier in $\mathcal{L}$). For each of the $r-(q-c)$ remaining edges in order, use an element in $\{0,1\} \times [q-c]$ to specify it as a copy of one of the $q-c$ first edges, where $\{0,1\}$ indicates whether to reverse the order of the end points. Next observe that the $q-c$ first edges have $2(q-c)$ end points of which precisely q are unique. Specify the first occurence of each unique node on these edges using an element in $\binom{2(q-c)}{q}$. Next use an element in $[n]$ for each such node in order to specify it among the nodes $x_1, \ldots, x_n$. Finally, for the remaining $2(q-c)-q$ end points, specify them as an index into the q first occurrences of unique nodes. This information uniquely describes $\mathcal{L}$.

Using that $\binom{2(q-c)}{q} \leq 2^{2(q-c)}$ and the general inequality $\binom{r}{q-c} \leq (er/(q-c))^{q-c}$, we conclude

$$\begin{aligned}
|\mathcal{G}^r_{q,c}| &\leq \left(\frac{er}{q-c}\right)^{q-c} (2(q-c))^{r-(q-c)} 2^{2(q-c)} n^q q^{q-2c} \\
&\leq \left(\frac{2er}{q}\right)^{q-c} (2q)^{r-(q-c)} 2^{2(q-c)} n^q q^{q-2c} \\
&= \left(\frac{4er}{q}\right)^{q-c} 2^r q^r n^q q^{-c}.
\end{aligned}$$ ◀

Let us finish by commenting on our choice of bounding $\sum_{i\neq j} Y_{i,j}$ rather than $\sum_{i<j} Y_{i,j}$. This choice was made for simplicity, but one may wonder whether focusing on the latter might result in tighter constants. This does not seem to be the case, as then the assumption that there are more than $n/2$ keys $x \in S$ such that there is a $y \in S \setminus \{x\}$ with $h(x) = h(y)$, does not imply $\sum_{i<j} Y_{i,j} > n/2$ (we use $\sum_{i\neq j} Y_{i,j} > n/2$), but only $\sum_{i<j} Y_{i,j} > n/4$. We would thus lose a constant factor in Markov's.

4 Proof via k-Wise Independence Fools Combinatorial Rectangles

We now give a second proof based on k-wise independence fooling combinatorial rectangles. This proof was communicated to us by an anonymous reviewer.

We first introduce the notion of a combinatorial rectangle. A combinatorial rectangle is a function $f : [m]^n \to \{0,1\}$ which is specified by n coordinate functions $f_i : [m] \to \{0,1\}$ as $f(x_1,\ldots,x_n) = \prod_{i\in m} f_i(x_i)$. We now use the following result, typically attributed to [7], although we cannot directly find this statement in the version available online. A clean introduction to combinatorial rectangles and bounded independence can, for instance, be found in [12].

▶ **Theorem 4.** *Let $X_1,\ldots,X_n$ be k-wise independent random variables with uniform marginal distributions over $[m]$. Let f be a combinatorial rectangle. Then there is a constant $a > 0$ such that*

$$\left|\mathbb{E}_{X_1,\ldots,X_n}\left[f(X_1,\ldots,X_n)\right] - \mathbb{E}_{x\in[m]^n}\left[f(x)\right]\right| \le e^{-ak},$$

where $\mathbb{E}_{x\in[m]^n}$ denotes a uniform random $x \in [m]^n$.

With this tool in place, we now prove Theorem 2.

Proof. Recall that we are hashing into Cn bins. Let $x_1,\ldots,x_n \in U$ denote the n keys and let $h : U \to [Cn]$ denote a hash function drawn randomly from a $2k$-wise independent family of hash functions. Let X_i be the random variable taking the value $h(x_i)$.

Let $J \subseteq [Cn]$ be the indices of a subset of the bins, with $|J| = t$ for a parameter t to be determined. Define random variables Z_j taking the value 1 if no element hashes to the value j and 0 otherwise. The probability that all bins indexed by J are empty is $\mathbb{E}[\prod_{j\in J} Z_j]$. If we now define functions $f_i : [Cn] \to \{0,1\}$ taking the value 1 on $h(x_i) \notin J$ and the value 0 for $h(x_i) \in J$, we have that $\prod_{j\in J} Z_j = \prod_{i=1}^n f_i(X_i)$, i.e. $\prod_{j\in J} Z_j$ is in effect a combinatorial rectangle. By Theorem 4, we have

$$\mathbb{E}\left[\prod_{j\in J} Z_j\right] \le \mathbb{E}_{x\in[Cn]^n}\left[f(x)\right] + e^{-ak}.$$

But $\mathbb{E}_{x\in[Cn]^n}[f(x)] = (1 - t/Cn)^n \le e^{-t/C}$. We now require $t < aCk$ and conclude $\mathbb{E}[\prod_{j\in J} Z_j] \le 2e^{-t/C}$.

Next, observe that if there are less than $n/2$ elements that hash to a unique value, then the number of occupied bins is at most $3n/4$. Vice versa, the number of unoccupied bins is at least $Cn - 3n/4$. If we also have $t < Cn - 3n/4$, then we may bound the expected number of t-sized subsets of bins that are empty. That is, if we let $Y_J = \prod_{j\in J} Z_j$, then we have just shown

$$\mathbb{E}\left[\sum_{J\in\binom{Cn}{t}} Y_J\right] \le \binom{Cn}{t} 2e^{-t/C}.$$

On the other hand, we may also lower bound the expectation by

$$\mathbb{E}\left[\sum_{J\in\binom{Cn}{n}} Y_J\right] \ge \Pr\left[\sum_j Z_j > Cn - 3n/4\right]\binom{Cn - 3n/4}{t}.$$

$\mathsf{Init}(\boldsymbol{h})$	$\mathsf{ListEntries}((T_0,\dots,T_{\lceil \lg n\rceil -1}),\boldsymbol{h})$
for $0 \le i < \lg(n) - \lg(\tau)$	$S' := \emptyset$
$\quad T_i := \mathsf{BasicInit}(1, \lceil Cn2^{-i}\rceil, \boldsymbol{h}_i)$	**for** $0 \le i < \lceil \lg n \rceil$
for $0 \le i < \lg(\tau)$	$\quad T_i := \mathsf{BasicDelete}(T_i, S')$
$\quad i' := \lfloor \lg(n) - \lg(\tau) \rfloor + i$	$\quad S' := S' \cup \mathsf{BasicListEntries}(T_i)$
$\quad T_{i'} := \mathsf{BasicInit}(2^i, \lceil C\tau 2^{-i}\rceil, \boldsymbol{h}_{i'})$	**return** S'
return $(T_0,\dots,T_{\lceil \lg n\rceil -1})$	
$\mathsf{Insert}((T_0,\dots,T_{\lceil \lg n\rceil -1}), S, \boldsymbol{h})$	$\mathsf{Delete}((T_0,\dots,T_{\lceil \lg n\rceil -1}), \tilde{S}, \boldsymbol{h})$
for $0 \le i < \lceil \lg n \rceil$	**for** $0 \le i < \lceil \lg n \rceil$
$\quad T_i := \mathsf{BasicInsert}(S, \boldsymbol{h}_i)$	$\quad T_i := \mathsf{BasicDelete}(\tilde{S}, \boldsymbol{h}_i)$
return $(T_0,\dots,T_{\lceil \lg n\rceil -1})$	**return** $(T_0,\dots,T_{\lceil \lg n\rceil -1})$

Figure 1 Our stacked IBLT construction using basic IBLTs as specified in Figure 2 as a building block. We have that $\tau = C_0 \lg(1/\delta)$ for a sufficiently large constant $C_0 > 0$.

Combining the two yields

$$\begin{aligned}
\Pr\left[\sum_j Z_j > Cn - 3n/4\right] &\le 2e^{-t/C} \cdot \frac{\binom{Cn}{t}}{\binom{Cn-3n/4}{t}} \\
&\le 2 \cdot \left(\frac{e^{-1/C}(Cn-t)}{Cn - 3n/4 - t}\right)^t \\
&= 2 \cdot \left(e^{-1/C}\left(1 + \frac{3}{4(C - 3/4 - t/n)}\right)\right)^t \\
&\le 2 \cdot \left(e^{-1/C + (3/4)\cdot 1/(C-3/4-t/n)}\right)^t
\end{aligned}$$

If we require $t < n$ and C at least a sufficiently large constant, then this is $\exp(-\Omega(t/C))$. Setting $t = \min\{aCk, n\}$ completes the proof. ◀

5 Smaller IBLTs with Limited Independence

In this section, we present a new construction of IBLTs, which we call stacked IBLTs, that is both asymptotically smaller and requires less randomness (in Subsection 5.3 we also argue that the analysis of the original IBLT cannot be strengthened to give bounds comparable to our stacked IBLT).

5.1 Stacked IBLTs

In this section we introduce our new Stacked IBLTs that are more space efficient and allow for a lower randomness complexity. Essentially the construction consists of $\lg n$ stacked smaller IBLTs. These IBLTs will be decoded in order and each is sized, such that we will be able to prove that it allows decoding at least half the remaining entries. This means that after decoding all $\lg n$ IBLTs, at most a single element is left to decode which can then be trivially decoded.

BasicInit$(\rho, \gamma, \boldsymbol{h})$	BasicListEntries$((\boldsymbol{K}, \boldsymbol{V}, \boldsymbol{C}), \boldsymbol{h})$

$\boldsymbol{K} := 0^{\rho\times\gamma}$
$\boldsymbol{V} := 0^{\rho\times\gamma}$
$\boldsymbol{C} := 0^{\rho\times\gamma}$
return $(\boldsymbol{K}, \boldsymbol{V}, \boldsymbol{C})$
	$S' := \emptyset$
	for $(i, j) \in [\rho] \times [\gamma]$
	  if $\boldsymbol{C}[i, j] = 1$
	    $(k, v) := (\boldsymbol{K}[i, j], \boldsymbol{V}[i, j])$
	    $S' := S' \cup \{(k, v)\}$
	return S'

BasicInsert$((\boldsymbol{K}, \boldsymbol{V}, \boldsymbol{C}), S, \boldsymbol{h})$	BasicDelete$((\boldsymbol{K}, \boldsymbol{V}, \boldsymbol{C}), \tilde{S}, \boldsymbol{h})$

foreach $(k, v) \in S$
  foreach $i \in [\rho]$
    $j := h_i(k)$
    $\boldsymbol{K}[i, j] := \boldsymbol{K}[i, j] + k$
    $\boldsymbol{V}[i, j] := \boldsymbol{V}[i, j] + v$
    $\boldsymbol{C}[i, j] := \boldsymbol{C}[i, j] + 1$
return $(\boldsymbol{K}, \boldsymbol{V}, \boldsymbol{C})$
	foreach $(k, v) \in \tilde{S}$
	  foreach $i \in [\rho]$
	    $j := h_i(k)$
	    $\boldsymbol{K}[i, j] := \boldsymbol{K}[i, j] - k$
	    $\boldsymbol{V}[i, j] := \boldsymbol{V}[i, j] - v$
	    $\boldsymbol{C}[i, j] := \boldsymbol{C}[i, j] - 1$
	return $(\boldsymbol{K}, \boldsymbol{V}, \boldsymbol{C}, \boldsymbol{h})$

Figure 2 A simplified version of a basic IBLT for key space $\mathcal{K}$ and universe $\mathcal{U}$. Both $\langle\mathcal{K}, +\rangle$ and $\langle\mathcal{U}, +\rangle$ need to form groups. The basic IBLT requires a number of rows ρ, a number of columns γ and a vector of hash functions $\boldsymbol{h} \in \{h : \mathcal{K} \to [\gamma]\}^\rho$ to initialize.

Let n be the threshold for an IBLT and $\delta > 0$ a desired failure probability. We can think of our Stacked IBLT as consisting of multiple rows, with a k-wise independent hash function associated with each row for $k = \Theta(\lg(\lg(n)/\delta))$. An element is hashed into one position in each row and stored there, like in the classic IBLT. The key novelty of our solution is that the number of entries per row varies. Moreover, while a classic IBLT focuses on peeling all elements, our analysis is based on peeling a constant fraction of the elements from each row.

More formally, let $\tau = C_0 \lg(1/\delta)$ for a sufficiently large constant $C_0 > 0$ and assume first that $n \geq \tau$. For $i = 0, \ldots, \lg(n/\tau)$, our IBLT has one row R_i with $Cn2^{-i}$ entries. Here $C > 0$ is a sufficiently large constant, where $C = 8e$ is provably sufficient. Finally, for $i = 0, \ldots, \lg(\tau)$, it has a group G_i consisting of 2^i rows all with $C\tau 2^{-i}$ entries. In case $n < C_0 \lg(1/\delta)$, our structure has a group G_i of 2^i rows for every $i = \lg(\tau/n), \ldots, \lg(\tau)$. In the group G_i, every row has $C\tau 2^{-i}$ entries. The IBLT uses $\sum_{i=0}^{\lg(n/\tau)} Cn2^{-i} + \sum_{i=0}^{\lg(\tau)} C\tau = O(n + \lg(1/\delta)\,\mathrm{lglg}(1/\delta))$ space. In the formal description of our Stacked IBLT construction, shown in Figure 1, we do not explicitly distinguish between the rows R_i and groups G_i, but rather view them as smaller IBLTs that we call $T_1, \ldots, T_{\lg n}$. For the analysis, however, distinguishing the smaller IBLTs with one row and those with multiple rows is helpful.

▶ **Theorem 1** (restated). *Given a threshold n, the Stacked IBLT supports* Insert, Delete, *and* ListEntries *operations, where* ListEntries *succeeds with probability $1 - \delta$ if the number of key-value pairs is no more than n. Furthermore, it uses space $O(n + \lg(1/\delta)\,\mathrm{lglg}(1/\delta))$ and requires only $O(\lg(\lg(n)/\delta))$-wise independent hashing.*

▶ Remark 5. We note that a k-wise independent hash function from a universe U to a universe of size γ requires $O(k \lg(U))$ bits. Since we require $O(\lg(n/\delta)$ such functions, we observe that the total number of random bits we need is $O(\lg(n/\delta)(\lg(1/\delta) + \lg\lg(n))\lg(U))$ bits. Regarding running times, the Insert and Delete operations both require $O(\lg(n/\delta))$

evaluations of a $O(\lg(\lg(n)/\delta))$-wise independent hash function, plus insertions in the table entries. The running time is dominated by the evaluations of the hash functions, for a total time of $O(k\lg(n/\delta))$ per element.

Proof of Theorem 1. To analyse the probability that peeling succeeds, we focus on the case of $n \geq \tau$. The other case is just a special case.

To argue that peeling succeeds with high probability, we consider a very restrictive form of peeling and argue that even this process succeeds. Concretely, for $i = 0, \dots, \lg(n/\tau)$, consider peeling all elements that land alone in R_i (after having peeled elements landing alone in R_j with $j < i$). Then, for $i = 0, \dots, \lg(\tau)$ in turn, select the row of G_i where most elements hash alone and peel those elements. To prove that this process succeeds in peeling all elements with probability at least $1 - \delta$, we define the events E_i occuring if there are more than $n2^{-(i+1)}$ elements left after peeling from $R_0, \dots, R_i$. Similarly, define F_i as the event that more than $\tau 2^{-(i+1)}$ elements remain after peeling from $R_0, \dots, R_{\lg(\tau)}, G_0, \dots, G_i$. We observe that if $F_{\lg(\tau)}$ does not occur, then there are no more than $1/2$ elements left, i.e. peeling succeeded.

The key step in our proof is to argue that the following two inequalities hold:

$$\Pr[E_i \mid \cap_{j=0}^{i-1}\overline{E_j}] \leq \frac{\delta}{4(\lg(n/\tau) - i + 1)^2}. \tag{2}$$

and

$$\Pr[F_i \mid \cap_{j=0}^{\lg(n/\tau)}\overline{E_j} \cap_{j=0}^{i-1} \overline{F_j}] \leq \delta^2/2. \tag{3}$$

Observe that these two are sufficient as

$$\begin{aligned}
\Pr[\overline{F_{\lg(\tau)}}] &\geq \Pr[\cap_{j=0}^{\lg(n/\tau)}\overline{E_j} \cap_{j=0}^{\lg(\tau)} \overline{F_j}] \\
&= \prod_{i=0}^{\lg(n/\tau)} (1 - \Pr[E_i \mid \cap_{j=0}^{i-1}\overline{E_j}]) \prod_{i=0}^{\lg(\tau)} (1 - \Pr[F_i \mid \cap_{j=0}^{\lg(n/\tau)}\overline{E_j} \cap_{j=0}^{i-1} \overline{F_j}]) \\
&\geq \prod_{i=0}^{\lg(n/\tau)} \left(1 - \frac{\delta}{4(\lg(n/\tau) - i + 1)^2}\right) \left(1 - \delta^2/2\right)^{\lg(\tau)+1} \\
&\geq 1 - \sum_{i=0}^{\lg(n/\tau)} \frac{\delta}{4(i+1)^2} - \frac{(\lg(\tau)+1)\delta^2}{2} \\
&\geq 1 - \frac{\delta\pi^2}{24} - \frac{\delta}{2} \\
&\geq 1 - \delta.
\end{aligned}$$

We start by showing (2). Observe that conditioned on $\cap_{j=0}^{i-1}\overline{E_j}$, we know that no more than $n2^{-i}$ elements remain after peeling from $R_0, \dots, R_{i-1}$. We may condition on an arbitrary such set as the hash functions across the rows are independent. So let S be a set of at most $n2^{-i}$ elements. The probability that there are more than $n2^{-(i+1)}$ elements that do no hash alone in R_i is clearly maximized when $|S|$ is $n2^{-i}$. Theorem 2 gives us that this probability is at most $4(8e/C)^{\min\{k/2, n2^{-i}/C\}}$. For $C \geq 16e$, this is at most $4 \cdot 2^{-\min\{k/2, n2^{-i}/C\}}$. Since $k = \Theta(\lg(\lg(n)/\delta))$, we have $2^{-k/2} < \delta/(4\lg_2^2 n) \leq \delta/(4(\lg(n/\tau)-i+1)^2)$ for a big enough constant in the Θ-notation. We also have $n2^{-i}/C = \tau 2^{\lg(n/\tau)-i}/C$. For big enough constant C_0 (in the definition of τ), this is at least $2\lg(1/\delta)(\lg(n/\tau)-i+1)+2 \geq \lg(1/\delta)+2\lg(\lg(n/\tau)-i+1))+2$ (and this is by a large margin) and we conclude $2^{-n2^{-i}/C} \leq (\delta/4)/(\lg(n/\tau) - i + 1))^2$.

To show (3), note again that conditioned on $\cap_{j=0}^{\lg(n/\tau)}\overline{E_j} \cap_{j=0}^{i-1} \overline{F_j}$, there are at most $\tau 2^{-i}$ elements left after peeling from $R_0, \dots, R_{\lg(n/\tau)}, G_0, \dots, G_{i-1}$. Again, condition on an arbitrary set S of remaining elements. The probability of F_i is clearly maximized if

$|S| = \tau 2^{-i}$. We split the proof in two cases. First, assume $\tau 2^{-i} \geq 4C$. Since each of the 2^i rows of G_i have $C\tau 2^{-i}$ entries, and the rows have independent hash functions, it follows by Theorem 2 and $C \geq 16e$, that

$$\Pr[F_i \mid \cap_{j=0}^{\lg(n/\tau)} \overline{E_j} \cap_{j=0}^{i-1} \overline{F_j}] \leq \left(4 \cdot 2^{-\min\{k, \tau 2^{-i}/C\}}\right)^{2^i} \leq \left(2^{-\min\{k/2, \tau 2^{-i}/(2C)\}}\right)^{2^i}.$$

Here the last inequality assumes $k = \Theta(\lg(\lg(n)/\delta))$ is at least a sufficiently large constant. We also use $\tau 2^{-i}/C - 2 \geq \tau 2^{-i}/C - \tau 2^{-i}/(2C)$. We clearly have $2^{-k/2} \leq \delta^2/2$ for a big enough constant in the Θ-notation. We also have $(2^{-\tau 2^{-i}/(2C)})^{2^i} = 2^{-\tau/(2C)}$. This is again smaller than $\delta^2/2$ for big enough constant C_0 in the definition of $\tau = C_0 \lg(1/\delta)$. Finally, for the case where $|S| = \tau 2^{-i} < 4C$, we note that one row of G_i has $C|S|$ entries and thus the expected number of elements that collide with another is no more than $|S|^2/(C|S|) = |S|/C$. By Markov's inequality, the probability that more than $|S|/2$ collide is no more than $2/C < 1/2$. By independence of the rows, the chance that peeling fails is at most 2^{-2^i}. Since $\tau 2^{-i} < 4C$, we have $2^i \geq \tau/(4C) = C_0 \lg(1/\delta)/(4C)$. For C_0 a big enough constant, this implies $2^{-2^i} < \delta^2/2$. ◀

5.2 Supporting Subtraction

Most applications of IBLTs require that decoding is possible after computing the *difference* between two different IBLTs. That is, given two IBLTs A, B, encoding sets S_A and S_B respectively, decoding $A - B$ should result in $S_A \triangle S_B$ as long as $|S_A \triangle S_B| \leq n$, even if the sets encoded in IBLTs A and B are much larger than n individually. The IBLT $A - B$ is obtained by subtracting the two data structures cell by cell.

Our stacked IBLTs can be made to support such an operation in a manner similar to the original IBLT construction. We modify the basic IBLT from Section 5.1 to have an additional hash sum matrix $\boldsymbol{H}$ where the values $g(k)$ for keys k for some appropriate hash function g are added up. During peeling both cells with a count of one or minus one can be peeled, whenever the hash of the key sum cell matches the hash stored in the hash sum cell. The stacked IBLTs supporting subtraction are explained in detail the full version of this paper [8].

5.3 Lower Bound on the Size of IBLTs

The original IBLT analysis by Goodrich and Mitzenmacher [11] shows that using truly random hash functions and space $\mathcal{O}(nk)$ one can achieve a failure probability of $\mathcal{O}(n^{-k+2})$. Stated in terms of δ and n, the space usage of their solution is thus $\Omega(n \lg_n(1/\delta))$. One may wonder, whether their analysis is tight or whether one could prove that IBLTs actually only require $o(nk)$ space for a similar failure probability.

It turns out their space bound is essentially tight and can not be improved by much. Assume we have an IBLT of size m storing keys $k_1, \dots, k_n$. Furthermore assume $h_1, \dots, h_k$ are perfectly random hash functions, which map each key to exactly k distinct locations. For an IBLT to be decodable, we must be able to find a cell with a count of one at each step of the peeling process. If $kn \geq cm \lg m$ for some sufficiently large constant c, then each cell will have at least $c \lg m$ elements in expectation and thus by Chernoff bound with high probability all cells have a count strictly larger than one. Thus it must hold that $kn < cm \lg m$. Consider two distinct keys that are inserted into the IBLT. The probability that both keys are hashed into exactly the same cells is

$$\binom{m}{k}^{-1} \geq \left(\frac{em}{k}\right)^{-k} \geq \left(\frac{en}{c \lg m}\right)^{-cm \lg m/n} \geq n^{-cm \lg m/n}.$$

If we want the IBLT to be correct with probability at least $1-\delta$, then it has to holds that

$$n^{-cm\lg m/n} \leq \delta$$

and thus

$$\frac{cm\lg m\lg n}{n} > \lg(1/\delta) \iff m\lg m > \frac{n\lg(1/\delta)}{c\lg n}.$$

For this to hold, it must also hold that

$$m\lg(n\lg(1/\delta)) > \frac{n\lg(1/\delta)}{c\lg n} \iff m > \frac{n\lg(1/\delta)}{c\lg(n)\lg(n\lg(1/\delta))}$$

and thus it must be true that

$$m > \frac{n\lg(1/\delta)}{c\lg^2(n\lg(1/\delta))} \geq \frac{n\lg_n(1/\delta)}{c\lg^2(n\lg(1/\delta))}$$

for any choice of $n \geq 2$.

References

1 Giuseppe Ateniese, Michael T. Goodrich, Vassilios Lekakis, Charalampos Papamanthou, Evripidis Paraskevas, and Roberto Tamassia. Accountable storage. In Dieter Gollmann, Atsuko Miyaji, and Hiroaki Kikuchi, editors, *ACNS 17: 15th International Conference on Applied Cryptography and Network Security*, volume 10355 of *Lecture Notes in Computer Science*, pages 623–644, Kanazawa, Japan, July 10–12 2017. Springer, Heidelberg, Germany. doi:10.1007/978-3-319-61204-1_31.

2 Djamal Belazzougui, Gregory Kucherov, and Stefan Walzer. Better space-time-robustness trade-offs for set reconciliation. In Karl Bringmann, Martin Grohe, Gabriele Puppis, and Ola Svenson, editors, *ICALP 2024: 51th International Colloquium on Automata, Languages and Programming*, LIPIcs, Tallinn, Estonia, July 8–12 2024. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

3 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.

4 Søren Dahlgaard, Mathias Bæk Tejs Knudsen, Eva Rotenberg, and Mikkel Thorup. Hashing for statistics over K-partitions. In Venkatesan Guruswami, editor, *56th Annual Symposium on Foundations of Computer Science*, pages 1292–1310, Berkeley, CA, USA, October 17–20 2015. IEEE Computer Society Press. doi:10.1109/FOCS.2015.83.

5 Martin Dietzfelbinger and Michael Rink. Applications of a splitting trick. In Susanne Albers, Alberto Marchetti-Spaccamela, Yossi Matias, Sotiris E. Nikoletseas, and Wolfgang Thomas, editors, *ICALP 2009: 36th International Colloquium on Automata, Languages and Programming, Part I*, volume 5555 of *Lecture Notes in Computer Science*, pages 354–365, Rhodes, Greece, July 5–12 2009. Springer, Heidelberg, Germany. doi:10.1007/978-3-642-02927-1_30.

6 David Eppstein, Michael T. Goodrich, Frank Uyeda, and George Varghese. What's the difference? efficient set reconciliation without prior context. *ACM SIGCOMM Computer Communication Review*, 41(4):218–229, August 2011. doi:10.1145/2043164.2018462.

7 Guy Even, Oded Goldreich, Michael Luby, Noam Nisan, and Boban Veličković. Efficient approximation of product distributions. *Random Structures & Algorithms*, 13(1):1–16, August 1998. doi:10.1002/(SICI)1098-2418(199808)13:1<1::AID-RSA1>3.0.CO;2-W.

8 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. URL: `https://eprint.iacr.org/2023/918`.

9 Nils Fleischhacker, Kasper Green Larsen, and Mark Simkin. Property-preserving hash functions for hamming distance from standard assumptions. In Orr Dunkelman and Stefan Dziembowski, editors, *Advances in Cryptology – EUROCRYPT 2022, Part II*, volume 13276 of *Lecture Notes in Computer Science*, pages 764–781, Trondheim, Norway, May 30 – June 3 2022. Springer, Heidelberg, Germany. `doi:10.1007/978-3-031-07085-3_26`.

10 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`.

11 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`.

12 Parikshit Gopalan and Amir Yehudayoff. Concentration for limited independence via inequalities for the elementary symmetric polynomials. *Theory of Computing*, 16(17):1–29, December 2020. `doi:10.4086/toc.2020.v016a017`.

13 Francisco Lázaro and Balázs Matuz. Irregular invertible bloom look-up tables. In *11th International Symposium on Topics in Coding, ISTC 2021*, pages 1–5, Montreal, Quebec, Canada, August 30 – September 3 2021. IEEE Computer Society Press.

14 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`.

15 Yaron Minsky, Ari Trachtenberg, and Richard Zippel. Set reconciliation with nearly optimal communication complexity. *IEEE Transactions on Information Theory*, 49(9):2213–2218, September 2003. `doi:10.1109/TIT.2003.815784`.

16 Michael Mitzenmacher and Rasmus Pagh. Simple multi-party set reconciliation. *Distributed Computing*, 31:441–453, October 2017. `doi:10.1007/s00446-017-0316-0`.

17 Michael Mitzenmacher and George Varghese. Biff (bloom filter) codes: Fast error correction for large data sets. In Giuseppe Caire, Michelle Effros, Hans-Andrea Loeliger, and Alexander Vardy, editors, *2012 IEEE International Symposium on Information Theory*, pages 483–487. IEEE Computer Society Press, July 1–6 2012. `doi:10.1109/ISIT.2012.6284714`.

18 A. Pinar Ozisik, Gavin Andresen, George Bissias, Amir Houmansadr, and Brian Levine. Graphene: A new protocol for block propagation using set reconciliation. In Joaquin Garcia-Alfaro, Guillermo Navarro-Arribas, Hannes Hartenstein, and Jordi Herrera-Joancomartí, editors, *Data Privacy Management, Cryptocurrencies and Blockchain Technology, ESORICS 2017 International Workshops, DPM 2017 and CBT 2017*, volume 10436 of *Lecture Notes in Computer Science*, pages 420–428, Oslo, Norway, September 14–15 2017. Springer, Heidelberg, Germany. `doi:10.1007/978-3-319-67816-0_24`.

19 Mikkel Thorup. Fast and powerful hashing using tabulation. *Communications of the Association for Computing Machinery*, 60(7):94–101, July 2017. `doi:10.1145/3068772`.

20 Lei Yang, Yossi Gilad, and Mohammad Alizadeh. Practical rateless set reconciliation. In Aruna Seneviratne, Darryl Veitch, Minlan Yu, and Vyas Sekar, editors, *ACM SIGCOMM 2024: Proceedings of the ACM SIGCOMM 2024 Conference*, Sydney, Australia, August 4–8 2024. ACM Press.