A Concrete Analysis of Wagner's k-List Algorithm over $\mathbb{Z}_p$

Antoine Joux[1], Hunter Kippen[2], and Julian Loss[1]

[1] CISPA Helmholtz Center for Information Security, Germany, joux@cispa.de, loss@cispa.de
[2] University of Maryland, College Park, MD, USA, hkippen@umd.edu

Abstract. Since its introduction by Wagner (CRYPTO '02), the k-list algorithm has found significant utility in cryptanalysis. One important application thereof is in computing forgeries on several interactive signature schemes that implicitly rely on the hardness of the ROS problem formulated by Schnorr (ICICS '01). The current best attack strategy for these schemes relies the conjectured runtime of the k-list algorithm over $\mathbb{Z}_p$. The tightest known analysis of Wagner's algorithm over $\mathbb{Z}_p$ is due to Shallue (ANTS '08). However, it hides large polynomial factors and leaves a gap with respect to desirable concrete parameters for the attack. In this work, we develop a degraded version of the k-list algorithm which provably enforces the heuristic invariants in Wagner's original. In the process, we devise and analyze a new list merge procedure that we dub the interval merge. We give a thorough analysis of the runtime and success probability of our degraded algorithm, and show that it beats the projected runtime of the analysis by Shallue for parameters relevant to the generalized ROS attack of Benhamouda et al. (EUROCRYPT '21). For a 256-bit prime p, and $k = 8$, our degraded k-list algorithm runs in time $\approx 2^{70.4}$, while Shallue's analysis states that the Wagner's original algorithm runs in time $\approx 2^{98.3}$.

1 Introduction

In 2002, Wagner introduced a generalization of the venerable *birthday problem* [30]. This generalized birthday problem tasks solvers with finding $x_i \in L_i$, $i \in \{1, \ldots, k\}$ such that $x_1 + \cdots + x_k = 0 \pmod p$, where p is a prime and each L_i is a list of random elements in $\mathbb{Z}_p$. To solve this problem, Wagner introduced a subexponential time algorithm dubbed the k-tree (or k-list) algorithm. This algorithm (and its variants) see use in cryptanalyzing myriad schemes and protocols.

Of particular importance, Wagner's k-list algorithm can be applied to attack the security of a variety of interactive signature schemes such as threshold-, multi-, and blind signature schemes [6, 12, 27, 30].

Currently, the runtime of these attacks depends on the conjecture that Wagner's algorithm [30] is correct, and outputs a solution in time $O(k \cdot$

$p^{\frac{1}{1+\ell}}$) where $k = 2^\ell$. Presently, the tightest analysis of the k-list algorithm over $\mathbb{Z}_p$ is due to Shallue [28]. Shallue was able to prove that Wagner's algorithm yields a solution with overwhelming probability in time $O(k \cdot p^{\frac{1}{\ell}})$. While Shallue's analysis asymptotically matches the conjecture, it hides large polynomial terms and leaves a substantial gap with respect to concrete parameters. For example, the aforementioned attacks of Drijvers et al. [12] and Benhamouda et al. [6] consider attacks on schemes over 256 and 512 bit groups which require anywhere from $k = 4$ to $k = 64$ lists. For these choices of parameters, the $+1$ term in the exponent can easily determine the difference between whether or not the attacks are feasible. In this work, we revisit Wagner's algorithm and provide the first meaningful analysis of its running time for practical values of k.

1.1 Contributions

We now give a more detailed overview of our problem statement and our contributions. For clarity, we present the original version of Wagner's k-list algorithm in Figure 1. In his beautifully simple algorithm, $k = 2^\ell$ initial lists $L_1^\ell, ..., L_k^\ell$ of m random elements in $\mathbb{Z}_p = I_\ell$ each are merged together in pairs. The merging of pairs creates $k/2$ new lists $L_1^{\ell-1}, ..., L_k^{\ell-1}$, where L_i consist of sums of the form $a + b$ where $a \in L_{2i-1}, b \in L_{2i}$ and $a + b$ lies in the interval $I_{\ell-1} := \left[\left|-\frac{p-1}{2}\right|, \left\lfloor\frac{p-1}{2}\right\rfloor\right]$ of half the size of I_ℓ. This step is now recursively repeated over ℓ many levels until a list L_1^0 is produced. The algorithm is deemed successful if $0 \in L_1^0$.

Existing Analyses and Their Limitations. Wagner's original analysis of the k-list algorithm provides a heuristic that the algorithm should succeed with constant probability in time $O(k \cdot p^{\frac{1}{1+\ell}})$ when elements in the initial lists are sampled from $\mathbb{Z}_p$. The analysis relies on the heuristic invariant that the elements in the lists are sampled uniformly and independently from the increasingly shrinking domains at each level.

However, it is simple to verify that the heuristic does not hold. At each level, the output lists take *all* pairs of elements that sum to fit inside the smaller domain (mod p). As such, it is possible for multiple output list elements to share an input element, which violates independence—lack of uniformity follows from the sum itself, as the sum of two uniform random variables is no longer uniform.

This presents a problem when attempting to analyze the algorithm without using the heuristic assumptions. It is extremely difficult to calculate the explicit probability distribution of output list elements due

k-list Algorithm

Input: $k = 2^\ell$ lists L_i of m random elements in $\mathbb{Z}_p$.
Output: $x_1, ..., x_k$ s.t. $x_i \in L_i$ and $\sum_i x_i = 0 \pmod p$.

For all $j \in [k]$, set $L_j^\ell := L_j$. For all $j \in [\ell]$, set $I_j := \left[\left\lfloor -\frac{p-1}{2^{\lfloor \log m \rfloor \cdot (\ell - j)+1}} \right\rfloor, \left\lfloor \frac{p-1}{2^{\lfloor \log m \rfloor \cdot (\ell - j)+1}} \right\rfloor\right]$.

- For $i = \ell$ down to 1 do:
  - For $j \in [2^{i-1}]$ do: $L_j^{i-1} = \{a + b \pmod p : a \in L_{2j-1}^i, b \in L_{2j}^i, a + b \in I_{i-1} \pmod p\}$
- If $0 \in L_1^0$, determine elements $x_1, ..., x_k$ s.t. $\sum_i x_i = 0$ and $x_i \in L_i$. Return $x_1, ..., x_k$ if such elements exist. Return $\perp$ otherwise.

Fig. 1: The original version of Wagner's k-list algorithm over $\mathbb{Z}_p$ [30].

to the lack of independence (and no guarantees on the exact nature of the dependence between list elements). Shallue worked around some of these constraints by utilizing the theory of Martingales to bound the non-uniformity and dependence of the elements in each merged list [28]. As discussed above, this results in loose bounds on the runtime that do not apply to practical values of k. Thus, his work leaves open the question of giving concrete bounds on the running time of Wagner's algorithm as well as the attacks that build on it.

Our Approach: Analyzing a Degraded Algorithm. In this work, we take a completely different approach to analyzing the running time of Wagner's algorithm. Instead of analyzing his algorithm directly, we instead consider a degraded version of the k-list algorithm. Our algorithm can be viewed as an all-around worse version of Wagner's original algorithm which provably achieves the heuristic invariant of the original algorithm and matches its running time and success probability when k is not too large. It does so through a modified list merge procedure that we dub the Interval Merge. Like its name suggests, the interval merge procedure first divides the input domain into suitably sized intervals and selects a single element per interval from both input lists (if they exist). Unique sums are then created from these selected elements. Finally, rejection sampling is employed to enforce the uniformity of elements in the merged list. Naturally, this modification does not come for free. We incur (at most) a constant factor loss per level of the tree. We present our main theorem on the runtime of the degraded k-list algorithm.

Theorem 1. *Given initials lists of size*

$$\left\lceil \alpha^{\frac{-(\ell-1)}{\ell+1}} \cdot c^{\frac{-(\ell-1)(\ell+2)}{2(\ell+1)}} \cdot p^{\frac{1}{\ell+1}} \right\rceil,$$

the degraded k-list algorithm in Figure 3 runs in time

$$\Theta\left(k \cdot \alpha^{\frac{-(\ell-1)}{\ell+1}} \cdot c^{\frac{-(\ell-1)(\ell+2)}{2(\ell+1)}} \cdot p^{\frac{1}{\ell+1}} \right),$$

and finds a solution with non-negligible probability, where α is the interval scaling factor input to IntervalMerge *and $k = 2^\ell$.*

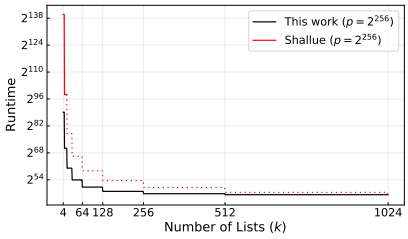
Note that we indeed manage to preserve the $+1$ in the denominator of the exponent(s). This allows our degraded variant of the algorithm to outperform Shallue's analysis [28] in small parameter regimes relevant to a wide variety of attacks on signature schemes.

For example, for a 256-bit prime p, and $k = 8$, our degraded k-list algorithm runs in time $\approx 2^{70.4}$, while Shallue's analysis states that the Wagner's original algorithm runs in time $\approx 2^{98.3}$. Furthermore, note that Shallue's analysis requires a technical assumption[3] that limits the range of parameter regimes in which it is valid. For a 256-bit p, Shallue's analysis holds only for k up to 8. However, in Figure 2, for ease of comparison, we extended the graphs showing the complexity beyond the limit of Shallue's analysis. For these graphs we invoke Theorem 1 with $c = (1/6)$, and $\alpha = 0.79591$ to calculate the required sizes of the initial lists, and plot the runtimes on the y-axis. We justify these choices of parameters with our analysis in Section 3. In combination with Shallue's analysis, our new algorithm substantiates the original performance claims from Wagner's paper for a wide range of parameters for both theory and practice.

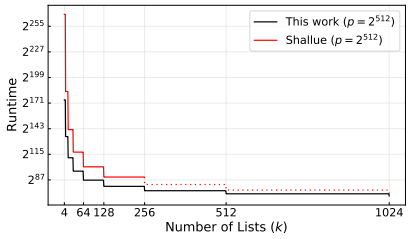
1.2 Related Work

Other analyses of the k-list algorithm. Wagner's k-list algorithm [30] has been re-analyzed multiple times since its original publication. In effect, there are two distinct versions of the algorithm, depending on the domain of the input list elements. The first considers lists of elements from an arbitrary finite field $\mathbb{Z}_p$ (which we discuss in this work). The second instead considers initial lists consisting of binary vectors (or equivalently from $\mathbb{Z}_{2^n}$), where the summation operator is replaced by coordinate-wise

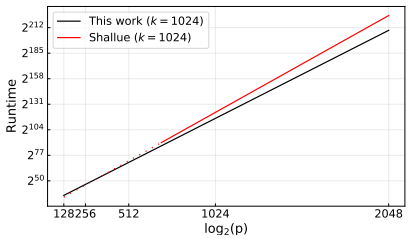
[3] For the range of parameters we are considering, this assumption can be written as $\log p > 70 \log k$.



(a) Comparison for 256-bit p



(b) Comparison for 512-bit p



(c) Comparison for $k = 1024$

Fig. 2: Runtime comparison between the degraded k-list algorithm (Figure 3), and the analysis of Wagner's algorithm [30] by Shallue [28] varying k for $p = \{2^{256}, 2^{512}\}$ (a)(b), and varying p for $k = 1024$ (c). The dotted segments indicates the extension of Shallue's analysis beyond the range where his technical assumption holds.

XOR. In this version, the goal of the k-list algorithm then becomes finding $x_i \in L_i$, $i \in \{0,1\}^n$ such that $x_1 \oplus \cdots \oplus x_k = \vec{0}$. The sums depicted in Figure 1 are replaced by finding list elements that match on an increasing number of low-order bits. Wagner's original analysis of this variant of the k-list algorithm states that it expects to find a single solution in time $O(k \cdot 2^{\frac{n}{1+\ell}})$ for $k = 2^\ell$.

For this variant of the algorithm, the analysis by Minder and Sinclair [24] is particularly illuminating. Minder and Sinclair were able to prove that Wagner's algorithm *does* indeed meet its conjectured runtime in the binary vector case and outputs a solution with high probability. Minder and Sinclair's analysis of the failure probability of Wagner's algorithm relies on the fact the bits of random binary vectors are independent of each other. This allows them to prove the uniformity of the elements in all merged lists. With this, Minder and Sinclair are then able to bound the covariance between any two candidate solutions to the algorithm. Thus,

they use Chebyshev's inequality to show the number of solutions is tightly concentrated around the expectation.

Note that the techniques used in Minder and Sinclair's analysis does not apply to arbitrary finite fields. In particular, the idea that the bits of input elements are independent of each other. One could certainly represent elements in $\mathbb{Z}_p$ by their bit-decomposition, but an XOR of two field elements in this representation will result in carries, breaking uniformity. We remark that Minder and Sinclair do provide a variant of their analysis for larger finite fields of prime powers $\mathbb{Z}_{p^n}$ for p prime, but this incurs a penalty of $\sqrt{p}$ on the runtime.

For the variant of the k-list algorithm acting on arbitrary finite fields, Lyubashevsky provided an analysis suited to solve the integer subset-sum problem [21]. Similarly to our proposed algorithm, Lyubashevsky only uses a subset of all valid summations during list merging. For this construction, the runtime is approximately $O(k \cdot p^{\frac{2}{1+\ell}})$. Shallue's analysis [28] can be viewed as an improvement to the efficiency of Lyubashevsky's, and thus also incurs a penalty in terms of the runtime ($O(k \cdot p^{\frac{1}{\ell}})$).

Applications of the k-list algorithm. Originally noted by Schnorr in 2001, multiple discrete logarithm-based blind signatures (including Schnorr [27] and Okamoto-Schnorr [26]) implicitly rely on an additional hardness assumption known as the ROS problem [14, 27]. An efficient solver for the ROS problem was noted to immediately produce a one more forgery attack on these signature schemes. Wagner [30] showed that the k-list algorithm for inputs over $\mathbb{Z}_p$, where p is the group modulus of the signature scheme, can be used to solve the ROS problem. In order to craft the proper inputs to the ROS solver, the attacker must use multiple *parallel* signing sessions.

More recently, Benhamouda et al. [6] showed that the ROS problem is solvable in polynomial time given that the dimension of the problem—the number of concurrent signing sessions—is larger than $\log p$. In addition to their polynomial time attack, Benhamouda et al. introduced a generalized variant that offered smooth trade-offs between the number of open sessions and attack difficulty. They did so by first applying Wagner's algorithm (but terminating early) to constrain the size of inputs to the ROS problem, and then applying their original polynomial time attack. As a consequence, the runtime of Benhamouda et al.'s generalized attack relies on the conjectured runtime of Wagner's algorithm over $\mathbb{Z}_p$. In addition, Benhamouda et al. were able to show that their attack techniques are able to be applied to threshold- and multi-signature schemes such as

those found in [15, 19, 22, 29] by refining techniques presented by Drijvers et al. [12].

The k-list algorithm has other cryptanalytic use cases. Many of these applications make use of the binary, i.e. XOR version, of the algorithm. Some direct applications help to break secret-key ciphers, to give a few examples [10, 18, 20]. The hash function proposed [2] was shown to be vulnerable to a k-list approach in [11], its successor FSB [1] was a SHA-3 candidate and despite additional precautions to withstand such attacks, some vulnerabilities remained as shown in [8].

Extensions of the k-list algorithm. Generalizations of Wagner's algorithm have been considered in many different directions. The case where the number of lists is not a power-of-two is considered in [25]. Its generalization to quantum computers is considered in [16]. Another extension led to the design of the representation technique giving improved algorithms for the subset-sum problem and the binary decoding problem [3–5,7,9,13,17].

2 Preliminaries

For any nonnegative interval $I = [a, b]$, $b \geq a \geq 0$, let $-I$ denote the complementary interval $-I := [-b, -a]$. Throughout the paper, we will need to make reference to the occupancy of complementary pairs of intervals.

Definition 1. *Given a complementary pair of intervals $I, -I$ and two lists L_1, L_2, we say the pair of intervals is* fully occupied *if there exist elements $x_1 \in L_1$ and $x_2 \in L_2$, such that $x_1 \in I$ and $x_2 \in -I$.*

Definition 2. *Given a complementary pair of intervals $I, -I$ and two lists L_1, L_2, we say the pair of intervals is* half occupied *if there exists an element $x_1 \in L_1$ and for all $x_2 \in L_2$, $x_1 \in I$ and $x_2 \notin -I$ or there exists an element $x_2 \in L_2$ and for all $x_1 \in L_1$, $x_2 \in -I$ and $x_1 \notin I$.*

Definition 3. *The discrete Triangle distribution $\mathcal{T}_{z-1}$ is defined by the following probability mass function:*

$$\mathcal{T}_{z-1}(k) = \begin{cases} \frac{k+z}{z^2} & -(z-1) \leq k \leq -1 \\ \frac{1}{z} & k = 0 \\ \frac{z-k}{z^2} & 1 \leq k \leq z-1 \end{cases}$$

We make use of a well known application of Azuma's inequality, named McDiarmid's inequality.

Theorem 2 (McDiarmid's Inequality [23]). *A function* $f : \mathcal{X}_1 \times \cdots \times \mathcal{X}_n \mapsto \mathbb{R}$ *satisfies the* bounded differences property *if there are constants* $c_1, c_2, \ldots, c_n$ *such that for all* $i = 1, 2, \ldots, n$ *and every* $x_1 \in \mathcal{X}_1, x_2 \in \mathcal{X}_2, \ldots x_n \in \mathcal{X}_n$*, we have*

$$\sup_{x_i' \in \mathcal{X}_i} \left| \begin{array}{r} f(x_1, \ldots, x_{i-1}, x_i, x_{i+1}, \ldots, x_n) - \\ f(x_1, \ldots, x_{i-1}, x_i', x_{i+1}, \ldots x_n) \end{array} \right| \leq c_i.$$

For such an f *satisfying the bounded differences property, consider independent random variables* $X_1, X_2, \ldots, X_n$ *where* $X_i \in \mathcal{X}_i$ *for all* i*. Then, for any* $\epsilon \geq 0$*,*

$$\mathbf{P}\left[f(X_1, X_2, \ldots, X_n) - \mathbf{E}\left[f(X_1, X_2, \ldots, X_n)\right] \leq -\epsilon\right] \leq \exp\left(-\frac{2\epsilon^2}{\sum_{i=1}^n c_i^2}\right)$$

3 Analysis of the Degraded k-List Algorithm

In this section, we present an analysis of a degraded version of the k-list algorithm.

Degraded k-List Algorithm

Input: $k = 2^\ell$ lists L_i of m uniform and independent elements in $\mathbb{Z}_p$, represented as integers in $\left[-\lfloor \frac{p}{2} \rfloor, \lfloor \frac{p}{2} \rfloor\right]$, and interval scaling factor $\alpha > 0$.
Output: $x_1, ..., x_k$ s.t. $x_i \in L_i$ and $\sum_i x_i = 0$.

- For $i = \ell$ down to 2 do:
  - For $j \in [2^{i-1}]$ do:
    - $L_j^{i-1} = \mathsf{IntervalMerge}(L_{2j-1}^i, L_{2j}^i, \alpha)$
- $L_1^0 = \{a + b : a \in L_1^1, b \in L_2^1, a + b = 0\}$
- If $L_1^0 \neq \emptyset$, determine elements $x_1, ..., x_k$ s.t. $\sum_i x_i = 0$ and $x_i \in L_i$. Return $x_1, ..., x_k$ if such elements exist. Return $\perp$ otherwise.

Fig. 3: A degraded version of Wagner's k-list algorithm that discards elements to maintain the uniformity and independence of lists at each level.

The performance of our algorithm is degraded (compared to Wagner's original algorithm) by discarding list elements at every level of the tree (except the last). By carefully choosing which elements to discard, the degraded algorithm maintains the uniformity and independence of list elements at every level. We introduce two procedures, IntervalMerge and

RejectionSample, that replace the traditional list-merge operation up to the last level of the tree (where the traditional list-merge is sufficient to find the last collision). IntervalMerge preserves the independence of output list elements by only combining one set of elements per matching "interval" in the input lists. The list elements created by IntervalMerge are sums of two uniform random variables, and as such are no longer uniform. Thus, we employ rejection sampling to further sculpt the output list into uniform and independent elements before inserting them into the output list(s) at each level.

Achieving a comparable success probability to the heuristic version requires the lists at the second to last level retain enough elements to guarantee a collision at the output level with a noticeable probability. As both IntervalMerge and RejectionSample discard list elements, the degraded algorithm will naturally require larger initial lists. Observe that the interval merge can be implemented in time linear in the size of the input lists. As we are only selecting (at most) a single pair of elements per interval, only a linear scan through each list is required. As such, we actually *improve* over Wagner's original algorithm in this regard. The lists need not be sorted nor stored in a hash table. Thus, the runtime of our degraded k-list algorithm is $O(k \cdot m_{in}^{(\ell)})$, where $m_{in}^{(\ell)}$ is the size of the initial input lists. For clarity, we restate our main runtime theorem here:

Theorem 1 *Given initials lists of size*

$$\left\lceil \alpha^{\frac{-(\ell-1)}{\ell+1}} \cdot c^{\frac{-(\ell-1)(\ell+2)}{2(\ell+1)}} \cdot p^{\frac{1}{\ell+1}} \right\rceil,$$

the degraded k-list algorithm in Figure 3 runs in time

$$\Theta\left(k \cdot \alpha^{\frac{-(\ell-1)}{\ell+1}} \cdot c^{\frac{-(\ell-1)(\ell+2)}{2(\ell+1)}} \cdot p^{\frac{1}{\ell+1}}\right),$$

and finds a solution with non-negligible probability, where α is the interval scaling factor input to IntervalMerge *and $k = 2^\ell$.*

Proof. For the degraded k-list algorithm described in Figure 3 to obtain a solution, lists at the second-to-last level L_1^1 and L_2^1 must contain enough elements such that a collision between the two lists is found. Thus, we want to analyze the sizes of these lists. The main tool for that is the fact that given input lists of size m_{in}, the IntervalMerge procedure in Figure 4 returns a list of size $c \cdot m_{in}$ for some constant $0 < c \leq 1$ with high probability.

Let us denote by $\mathcal{L}_{in}^{(i)}$, the domain of the elements in list $L_j^i \ \forall \ j$. Let us also denote by $m_{in}^{(i)}$, the number of elements in list $L_j^i \ \forall \ j$.

Then, for a collision to occur with constant probability, birthday bounds require

$$(m_{in}^{(1)})^2 = \Theta(|\mathcal{L}_{in}^{(1)}|). \tag{1}$$

Observe that $\mathcal{L}_{in}^{(i)}$ is the output range of the Interval Merge procedure in Figure 4 as executed on the input lists at level $i+1$. Thus, we have that

$$\mathcal{L}_{in}^{(i)} = \left[- \left\lfloor \left\lfloor \frac{|\mathcal{L}_{in}^{(i+1)}|}{\alpha \cdot m_{in}^{(i+1)}} \right\rfloor /2 \right\rfloor , \left\lfloor \left\lfloor \frac{|\mathcal{L}_{in}^{(i+1)}|}{\alpha \cdot m_{in}^{(i+1)}} \right\rfloor /2 \right\rfloor \right]$$

as the input interval $[-a, a]$ is substituted for $\mathcal{L}_{in}^{(i+1)}$. Then,

$$\begin{aligned} |\mathcal{L}_{in}^{(i)}| &= 2 \cdot \left\lfloor \left\lfloor \frac{|\mathcal{L}_{in}^{(i+1)}|}{\alpha \cdot m_{in}^{(i+1)}} \right\rfloor /2 \right\rfloor \\ &= \Theta \left(\frac{|\mathcal{L}_{in}^{(i+1)}|}{\alpha \cdot m_{in}^{(i+1)}} \right) \end{aligned} \tag{2}$$

In addition, as per our initial assumption, $m_{in}^{(i)} = c \cdot m_{in}^{(i+1)}$. Thus $m_{in}^{(i)} = c^{\ell-i} m_{in}^{(\ell)}$. Combining with (2), we now obtain a recurrence relation on the size of the input intervals

$$\begin{aligned} |\mathcal{L}_{in}^{(i)}| &= \Theta \left(\frac{|\mathcal{L}_{in}^{(i+1)}|}{\alpha c^{\ell-i-1} m_{in}^{(\ell)}} \right) \\ |\mathcal{L}_{in}^{(i)}| &= \Theta \left(\frac{|\mathcal{L}_{in}^{(\ell)}|}{\prod_{j=i}^{\ell-1} \alpha c^{\ell-j-1} m_{in}^{(\ell)}} \right) \end{aligned} \tag{3}$$

Substituting (3) for $|\mathcal{L}_{in}^{(1)}|$ in (1), we find that

$$\begin{aligned} (m_{in}^{(1)})^2 &= \Theta(|\mathcal{L}_{in}^{(1)}|) \\ (m_{in}^{(1)})^2 &= \Theta \left(\frac{|\mathcal{L}_{in}^{(\ell)}|}{\prod_{j=1}^{\ell-1} \alpha c^{\ell-j-1} m_{in}^{(\ell)}} \right) \\ (c^{\ell-1} m_{in}^{(\ell)})^2 &= \Theta \left(\frac{|\mathcal{L}_{in}^{(\ell)}|}{\prod_{j=1}^{\ell-1} \alpha c^{\ell-j-1} m_{in}^{(\ell)}} \right) \\ (c^{\ell-1} m_{in}^{(\ell)})^2 &= \Theta \left(\frac{p}{\alpha^{\ell-1} c^{(\ell-1)(\ell-2)/2} (m_{in}^{(\ell)})^{\ell-1}} \right) \end{aligned} \tag{4}$$

as $|\mathcal{L}_{in}^{(\ell)}| = p$. Solving (4) for $m_{in}^{(\ell)}$ completes the proof, as the interval merge procedure is called at most $2k$ times.

$$m_{in}^{(\ell)} = \Theta\left(\alpha^{\frac{-(\ell-1)}{\ell+1}} \cdot c^{\frac{-(\ell-1)(\ell+2)}{2(\ell+1)}} \cdot p^{\frac{1}{\ell+1}}\right)$$

□

Interval Merge

Input: Lists L_1, L_2 of m_{in} uniform and independent elements in $[-a, a]$, and interval scaling factor $\alpha > 0$.
Output: List L_{12} of uniform and independent elements in $[-\lfloor\lfloor\frac{2a}{\alpha \cdot m_{in}}\rceil/2\rfloor, \lfloor\lfloor\frac{2a}{\alpha \cdot m_{in}}\rceil/2\rfloor]$.

- Let $\mathcal{L} = \left\lfloor\frac{2a}{\alpha \cdot m_{in}}\right\rceil$.
- Let $\mathcal{L}_{\frac{1}{2}} = \lfloor\mathcal{L}/2\rfloor$.
- Let $N_{int} = \left\lfloor\frac{2a}{\mathcal{L}}\right\rfloor$.
- Sample N_{int} random numbers $\in [0, 1]$, $r_1, \ldots, r_{N_{int}}$.
- Let f be the linear function s.t. $f(0) = -a$ and $f(N_{int}) = a + 1$.
- For $j = 1$ to N_{int} do:
  - Let $I_j = [-a, a] \cap [f(j-1), f(j)] \cap \mathbb{Z}$.
  - Let x_1^* be the first element of $L_1 \in I_j$ otherwise $\perp$.
  - Let x_2^* be the first element of $L_2 \in -I_j$ otherwise $\perp$.
  - If $x_1^* \neq \perp$ AND $x_2^* \neq \perp$ then:
    - If RejectionSample$(\mathcal{T}_{|I_j|-1}, (x_1^* + x_2^*), \mathcal{L}_{\frac{1}{2}}\ ; r_j) =$ Accept then:
      - Add $(x_1^* + x_2^*)$ to L_{12}.
- Return L_{12}.

Fig. 4: The interval merge procedure that takes the place of the traditional list merge operation in the degraded k-list algorithm in Figure 3

3.1 Analysis of IntervalMerge

The main source of degradation of initial list sizes is due to the IntervalMerge procedure. The procedure is primarily designed to maintain the independence and uniformity of elements in the merged list. To ensure the degraded k-list algorithm terminates in the same number of iterations as Wagner's original, care must be taken to enforce similar constraints on the range of elements in the merged list. The independence of output elements is ensured by only allowing for sums of unique elements from each input list. This is achieved by first partitioning the input domain

Rejection Sampling

Input: The probability mass function of a discrete triangle distribution $\mathcal{T}_{z-1}$ (Definition 3), candidate output element x_{12} distributed according to $\mathcal{T}_{z-1}$, a target interval bound $a \leq z-1$, and randomness $r \in [0;1]$.
Output: Accept OR Reject.

- If $x_{12} \notin [-a, a]$ Return Reject.
- Let $p = \frac{\mathcal{T}_{z-1}(a)}{\mathcal{T}_{z-1}(x_{12})}$.
- If $r < p$ Return Accept, Otherwise Return Reject

Fig. 5: The rejection sampling procedure used during the execution of an interval merge. The procedure is designed to accept with a uniform probability.

into non-overlapping intervals of similar size. If (at most) one element is used (and not reused) in a sum per interval, then the uniqueness of each summand is guaranteed.

To bound the range of the output, we form the sums from complementary intervals symmetric about the origin. Then, to further shrink the range and ensure that elements in the output list are distributed uniformly, we apply rejection sampling to each sum. As such, the size of the merged list is precisely the number of *fully occupied* (see definition 1) complementary intervals multiplied by the proportion of sums that were accepted. The challenge is to have enough intervals such that we sufficiently shrink the range of output elements (as in the original k-list algorithm), while guaranteeing (with high probability) that enough complementary intervals are fully occupied.

Constructing the Intervals. Defining the interval boundaries is not straightforward. The input list elements are over a subset of the integers. We cannot guarantee that the optimal number of intervals perfectly partitions the input domain. Additionally, the algorithm invariant requires output elements to remain identically distributed. If intervals are allowed to vary in size, we must enforce the identical distributions of output elements through our rejection sampling procedure.

More formally, for input elements drawn from $[-a, a]$, IntervalMerge splits the input domain of each list into sub-intervals of size $\mathcal{L}$ and $\mathcal{L}+1$, where

$$\mathcal{L} = \lfloor (2a)/(\alpha \cdot m_{in}) \rceil. \tag{5}$$

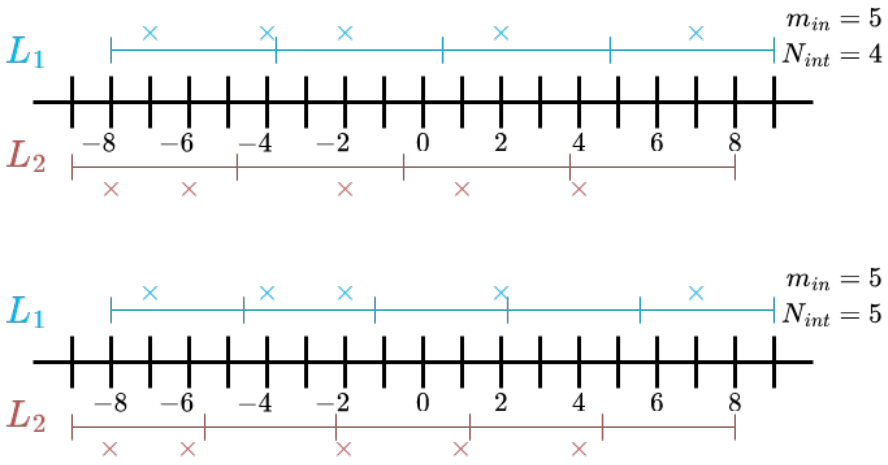


Fig. 6: Example intervals created during interval merge (Figure 4) for $m_{in} = 5$ input elements per list, and $N_{int} = \{4, 5\}$. Input elements are sampled from an input domain of $[-8, 8]$.

Here m_{in} is the number of elements in each input list. Note also that α controls the (approximate) number of intervals we create, since we define the number of intervals as $N_{int} = \lfloor (2a)/\mathcal{L} \rfloor$.

The interval boundaries are set by first defining the linear function $f()$, where $f(0) = -a$ and $f(N_{int}) = a+1$. Then, each interval $I_j,\ j \in [N_{int}]$ is constructed by taking the set of integers in $[f(j-1), f(j)]$. By inspection, we see that these sets are intervals of size $\mathcal{L}$ or $\mathcal{L}+1$. See Figure 6 for an example.

Output Range and Distribution. The output range and distribution of elements at every level of the degraded k-list algorithm is defined by the rejection sampling procedure in Figure 5. We first note the following.

Remark 1. All elements $x_{12} \in L_{12}$ inserted by the IntervalMerge procedure in Figure 4 are independent.

This is fairly straightforward. The independence of output elements is attained by construction, as each output element is formed from unique summands. Each defined subinterval is non-overlapping, and (at most) one element is used per subinterval per list.

The range and (exact) distribution is more involved. Elements in the output list(s) at every level of the degraded k-list algorithm solely consist

of elements accepted by the rejection sampling procedure in Figure 5. In this procedure, all elements outside the specified symmetric interval are automatically rejected.

For elements inside the specified interval, we flatten the probability distribution and make sure that the final distribution of outcomes is uniform. This is shown by computing the probability that the rejection sampling procedure outputs Accept:

$$\begin{aligned}
&\mathbf{P}\left[\mathsf{RejectionSample}(\mathcal{T}_{z-1}, x_{12}, a; r) = \mathsf{Accept}\right] \\
&= \sum_{i=-(z-1)}^{z-1} \mathbf{P}\left[\mathsf{Accept} \mid x_{12} = i\right] \cdot \mathbf{P}\left[x_{12} = i\right] \\
&= \sum_{i=-a}^{a} \mathbf{P}\left[r \leq \frac{\mathcal{T}_{z-1}(a)}{\mathcal{T}_{z-1}(x_{12})}\right] \cdot \mathcal{T}_{z-1}(x_{12}) \qquad (6)
\end{aligned}$$

where RejectionSample is the rejection sampling procedure specified in Figure 5, and (6) is due to the fact that the procedure always returns Reject when $x_{12} \notin [-a, a]$. Finishing the calculation yields

$$\begin{aligned}
&= \sum_{i=-a}^{a} \frac{\mathcal{T}_{z-1}(a)}{\mathcal{T}_{z-1}(x_{12})} \cdot \mathcal{T}_{z-1}(x_{12}) \qquad (7) \\
&= (2a+1) \cdot \mathcal{T}_{z-1}(a)
\end{aligned}$$

Observe in (7) that for all $i \in [-a, a]$, the probability of accepting given $x_{12} = i$ is $\mathcal{T}_{z-1}(a)$. Therefore, all elements accepted by the rejection sampling procedure are assigned the same probability mass. As such, we can state the following:

Remark 2. All elements x_{12} accepted by the RejectionSample procedure in Figure 5 and inserted into L_{12} by the IntervalMerge procedure in Figure 4 are identically distributed. Each element is drawn from the uniform distribution on $[-\mathcal{L}_{\frac{1}{2}}, \mathcal{L}_{\frac{1}{2}}]$ where $\mathcal{L}_{\frac{1}{2}} = \lfloor \mathcal{L}/2 \rfloor$, and $\mathcal{L}$ is as defined in (5). We are using the RejectionSample procedure with two different input distributions, $\mathcal{T}_{\mathcal{L}-1}$ or $\mathcal{T}_{\mathcal{L}}$. Yet, in both cases, we have the exact same output distribution.

Here, we are simply replacing the input a in Figure 5 with the interval boundary ($\mathcal{L}_{\frac{1}{2}}$) supplied during the Interval Merge procedure in Figure 4.

Furthermore, the sums supplied to the rejection sampling procedure are the sums of two independent uniform elements (a random variable that

is uniform over an interval is also uniform over any subinterval). Since the elements come from intervals of size either $\mathcal{L}$ or $\mathcal{L}+1$ which are symmetric of each other around 0, the sums belong to either $[-(\mathcal{L}-1);\mathcal{L}-1]$ or to $[-\mathcal{L};\mathcal{L}]$. The distribution is thus a discrete triangle distribution $\mathcal{T}_{z-1}$ (Definition 3). Here, the parameter z is set to the size of the complementary intervals that the summands x_1^* and x_2^* inhabit. This can be either $\mathcal{L}$ or $\mathcal{L}+1$ as shown in Figure 6.

When the size of the intervals is $\mathcal{L}$, the probability of the rejection sampling procedure returning Accept is

$$\begin{aligned}&(2\mathcal{L}_{\frac{1}{2}}+1)\cdot\mathcal{T}_{\mathcal{L}-1}(\mathcal{L}_{\frac{1}{2}})\\&=(2\lfloor\mathcal{L}/2\rfloor+1)\cdot\frac{\mathcal{L}-\lfloor\mathcal{L}/2\rfloor}{\mathcal{L}^2}.\end{aligned}\tag{8}$$

Similarly, when the size of the intervals is $\mathcal{L}+1$, we obtain

$$\begin{aligned}&(2\mathcal{L}_{\frac{1}{2}}+1)\cdot\mathcal{T}_{\mathcal{L}}(\mathcal{L}_{\frac{1}{2}})\\&=(2\lfloor\mathcal{L}/2\rfloor+1)\cdot\frac{(\mathcal{L}+1)-\lfloor\mathcal{L}/2\rfloor}{(\mathcal{L}+1)^2}.\end{aligned}\tag{9}$$

To simplify (8) and (9) we have two cases. One for when $\mathcal{L}$ is even, and another for when $\mathcal{L}$ is odd. We show only the even case here for brevity. When $\mathcal{L}$ is even, (8) becomes

$$\begin{aligned}&(2(\mathcal{L}/2)+1)\cdot\frac{(\mathcal{L})-(\mathcal{L}/2)}{(\mathcal{L})^2}\\&=\frac{\mathcal{L}+1}{2\mathcal{L}}\geq\frac{1}{2},\ \forall\ \mathcal{L}\geq 1.\end{aligned}\tag{10}$$

Similarly, (9) becomes

$$\begin{aligned}&(2(\mathcal{L}/2)+1)\cdot\frac{(\mathcal{L}+1)-(\mathcal{L}/2)}{(\mathcal{L}+1)^2}\\&=\frac{\mathcal{L}/2+1}{\mathcal{L}+1}\geq\frac{1}{2},\ \forall\ \mathcal{L}\geq 1.\end{aligned}\tag{11}$$

The odd case proceeds in the exact same manner, but replace $\lfloor\mathcal{L}/2\rfloor$ with $(\mathcal{L}-1)/2$ instead of $\mathcal{L}/2$. In both cases, the total probability mass for accepted sums is $\geq 1/2$.

3.2 Output List Size

In IntervalMerge, for each subinterval, an element is added to L_{12} if and only if that subinterval and its complement are *fully occupied* and the

sum of the first elements (in list order) passes the rejection sampling procedure. Therefore, we can view $\mathbf{E}[|L_{12}|]$ as the expected number of fully occupied pairs of complementary intervals (see Definition 1) multiplied by the expected loss in probability mass incurred by rejection sampling. To give a lower bound for $|L_{12}|$, we make use of McDiarmid's inequality (Theorem 2).

Applying the inequality requires defining a function that satisfies the bounded differences property. To this end, we define the function f as a $(2m_{in} + N_{int})$-input function that takes the values of the elements in the lists L_1 and L_2 and the randomness $r_j \ \forall \ j \in [N_{int}]$ supplied to the rejection sampling procedure in Figure 5. The return value for f is then set to the number of elements in the list L_{12} output by a single execution of the interval merge procedure in Figure 4 for a given interval scaling factor α.

Note that we must be careful when using the randomness to make sure that adding or removing one doubly occupied interval does not create an unwanted cascade of changes in the other decisions made in subsequent invocations of the interval merge procedure. This can be done by assigning each interval (at every level) its own randomness at the very beginning of the algorithm independently of whether this interval contributes or not. With this precaution in place, any change in the input list only has a localized effect and can change the size of any list that depends on this element by at most 1.

In addition, McDiarmid's inequality is a consequence of constructing a Doob Martingale that tracks the conditional expectation of f as its inputs are sampled. The sums passed to RejectionSample are determined by list order. As such, we ensure that for all $i \in [m_{in}]$, the $(2i-1)$-st input is the i-th element of L_1 and the input $2i$-th input is the i-th element of L_2.

The first step to applying McDiarmid's inequality is to calculate the expected value for f.

Lemma 1. *Let $\mathcal{X}^* := [-a, a] \subset \mathbb{Z}$, and $\mathcal{U}^* := [0, 1] \subseteq \mathbb{R}$ and let L_1 and L_2 be list of m_{in} uniform elements from $\mathcal{X}^*$. Further, let $N_{int}^{(\mathcal{L})}$ be the number of sub-intervals of size $\mathcal{L}$, let $N_{int}^{(\mathcal{L}+1)}$ be the number of sub-intervals of size $\mathcal{L}+1$ and let $N_{int} = N_{int}^{(\mathcal{L})} + N_{int}^{(\mathcal{L}+1)}$. Then, the expectation of the function $f : (\mathcal{X}^*)^{2m_{in}} \times (\mathcal{U}^*)^{N_{int}} \mapsto \mathbb{N}$, where the input $x_{2i-1} \in \mathcal{X}^*$ is the i-th element of L_1 and the input $x_{2i} \in \mathcal{X}^*$ is the i-th element of L_2 for all $i \in [m_{in}]$, and the return value is the number of elements in the list L_{12} output by the interval merge procedure in Figure 4–given the*

interval scaling factor α–is

$$\mathbf{E}\left[f(X_1,\ldots,X_{2m},r_1,\ldots,r_{N_{int}})\right] \geq \frac{1}{2}\cdot\left(N_{int}^{(\mathcal{L})}\cdot\left(1-\left(1-\frac{\mathcal{L}}{2a+1}\right)^{m_{in}}\right)^2 + N_{int}^{(\mathcal{L}+1)}\cdot\left(1-\left(1-\frac{\mathcal{L}+1}{2a+1}\right)^{m_{in}}\right)^2\right).$$

Proof. The the expected number of output list elements ($\mathbf{E}[f]$) for a single execution of the Interval merge procedure can be found by counting the (expected) number of times the rejection sampling procedure in Figure 5 returns Accept. However, the expected number of Accepts cannot be computed directly without first knowing the number of sums $x_1^* + x_2^*$ that were submitted to the rejection sampling procedure. As such, we make use of the law of iterated expectation

$$\begin{aligned}\mathbf{E}\left[f(X_1,\ldots,X_{2m},r_1,\ldots,r_{N_{int}})\right] &= \mathbf{E}\left[\#\ \mathsf{RejectionSample\ Accepts}\right]\\ &= \mathbf{E}\left[\mathbf{E}\left[\#\ \mathsf{RejectionSample\ Accepts}\mid N_{sum}\right]\right].\end{aligned} \tag{12}$$

where N_{sum} is the number of sums $x_1^* + x_2^*$ submitted to the rejection sampling procedure.

The expected number of Accepts given the number of sums created during the interval merge procedure is covered by our analysis of the rejection sampling procedure in the previous section. By construction, our rejection sampling procedure outputs a *uniform* distribution over a target interval. Each outcome that results in an Accept is assigned a constant probability mass (See (7) and Remark 2).

We state the following claim:

Claim. If the number of sums submitted to the RejectionSample procedure in Figure 5 by the IntervalMerge procedure in Figure 4 is N_{sum}, where each sum is distributed according to the triangle distributions $\mathcal{T}_{\mathcal{L}-1}$ or $\mathcal{T}_{\mathcal{L}}$, and the target output interval is $\mathcal{L}_{\frac{1}{2}}$, then

$$\mathbf{E}\left[\#\ \mathsf{RejectionSample\ Accepts}\mid N_{sum}\right] \geq (1/2)\cdot N_{sum},$$

where $\mathcal{L}$ is defined in (5), and $\mathcal{L}_{\frac{1}{2}} = \lfloor \mathcal{L}/2 \rfloor$.

Proof (Claim). The claim follows from (10) and (11), which show that the rejection sampling procedure outputs Accept with probability $\geq 1/2$, and the linearity of expectation.

Therefore, we can rewrite (12) as

$$\mathbf{E}\left[f(X_1, \ldots, X_{2m}, r_1, \ldots, r_{N_{int}})\right] \geq \mathbf{E}\left[(1/2) \cdot N_{sum}\right]. \tag{13}$$

So we now have simplified the expected value calculation of $f()$ to be at least half the (expected) number of sums $x_1^* + x_2^*$ created.

A sum is only created when the pair of intervals at a particular iteration is fully occupied (Definition 1). Determining the number of fully occupied intervals after sampling all input list elements is an instance of the balls-into-bins problem.

Let $\mathcal{A}_j$ be the event that intervals $I_j, -I_j$ are fully occupied after all $2m_{in}$ input list elements are sampled. Then, we can define the indicators $\mathcal{I}_j$, where $\mathcal{I}_j = 1$ when event $\mathcal{A}_j$ occurs. Then, the expected number of created sums can be expressed by applying the linearity of expectation to the sum of all $\mathcal{I}_j$,

$$\begin{aligned}\mathbf{E}\left[N_{sum}\right] &= \mathbf{E}\left[\sum_{j=0}^{N_{int}} \mathcal{I}_j\right] = \sum_{j=0}^{N_{int}} \mathbf{E}\left[\mathcal{I}_j\right] = \sum_{j=0}^{N_{int}} \mathbf{P}\left[\mathcal{A}_j\right] \\ &= N_{int}^{(\mathcal{L})} \cdot \mathbf{P}\left[\mathcal{A}^{(\mathcal{L})}\right] + N_{int}^{(\mathcal{L}+1)} \cdot \mathbf{P}\left[\mathcal{A}^{(\mathcal{L}+1)}\right] \end{aligned} \tag{14}$$

were $\mathcal{A}^{(\mathcal{L})}$ (resp. $\mathcal{A}^{(\mathcal{L}+1)}$) is shorthand for the event A_j for any given interval I_j of size $\mathcal{L}$ (resp. $\mathcal{L}+1$). Here (14) results from each interval (and interval pair) of the same size having equal probability of occupation.

The probability of $\mathcal{A}_j$ occurring is the probability that after sampling $2m_{in}$ elements (m_{in} for each list), interval I_j is occupied by (at least) one element from L_1 and $-I_j$ is occupied by (at least) one element from L_2. As these two sub-events are independent, we can calculate $\mathbf{P}\left[\mathcal{A}_j\right]$ as

$$\begin{aligned}\mathbf{P}\left[\mathcal{A}_j\right] &= \mathbf{P}\left[I_j \text{ is occupied}\right] \cdot \mathbf{P}\left[-I_j \text{ is occupied}\right] \\ &= \left(1 - \mathbf{P}\left[I_j \text{ is unoccupied after sampling } m_{in} \text{ elements}\right]\right)^2 \\ &= \left(1 - \left(1 - \frac{|I_j|}{2a+1}\right)^{m_{in}}\right)^2 \end{aligned} \tag{15}$$

where $1 - |I_j|/(2a+1)$ is the probability of a sampled element landing outside a particular interval. Combining (13) and (14) with (15) completes

the proof.

$$\begin{aligned}\mathbf{E}\left[f(X_1,\ldots,r_{N_{int}})\right] &\geq \mathbf{E}\left[(1/2)\cdot N_{sum}\right] \\ &= \frac{1}{2}\cdot\left(N_{int}^{(\mathcal{L})}\cdot\mathbf{P}\left[\mathcal{A}^{(\mathcal{L})}\right]+N_{int}^{(\mathcal{L}+1)}\cdot\mathbf{P}\left[\mathcal{A}^{(\mathcal{L}+1)}\right]\right) \\ &= \frac{1}{2}\cdot\left(N_{int}^{(\mathcal{L})}\cdot\left(1-\left(1-\frac{\mathcal{L}}{2a+1}\right)^{m_{in}}\right)^2\right. \\ &\qquad \left.+N_{int}^{(\mathcal{L}+1)}\cdot\left(1-\left(1-\frac{\mathcal{L}+1}{2a+1}\right)^{m_{in}}\right)^2\right)\end{aligned}$$

□

Now, we can state the lower bound on the output list size of IntervalMerge using McDiarmid's inequality.

Applying McDiarmid's Inequality.

Lemma 2. *Let $L_1, L_2, N_{int}^{(\mathcal{L})}, N_{int}^{(\mathcal{L}+1)}, N_{int}$ be as in Lemma 1. For any $\epsilon > 0$, the size of the output list L_{12} of the* IntervalMerge *procedure in figure 4 is at least*

$$\begin{aligned}&\frac{1}{2}\cdot\left(N_{int}^{(\mathcal{L})}\cdot\left(1-\left(1-\frac{\mathcal{L}}{2a+1}\right)^{m_{in}}\right)^2\right. \\ &\left.+N_{int}^{(\mathcal{L}+1)}\cdot\left(1-\left(1-\frac{\mathcal{L}+1}{2a+1}\right)^{m_{in}}\right)^2\right)-\epsilon\end{aligned}$$

with probability at least $1-\exp\left(-2\epsilon^2/(8m_{in}+N_{int})\right)$.

Proof. The proof follows directly from applying McDiarmid's inequality (Theorem 2) to the expected size of the output list from IntervalMerge (Lemma 1). To do so, we first make the following claim:

Claim. Let $\mathcal{X}^* := [-a, a] \subset \mathbb{Z}$, and $\mathcal{U}^* := [0, 1] \subseteq \mathbb{R}$. Then, the function $f : (\mathcal{X}^*)^{2m_{in}} \times (\mathcal{U}^*)^{N_{int}} \mapsto \mathbb{N}$, where the input $x_{2i-1} \in \mathcal{X}^*$ is the i-th element of L_1 and the input $x_{2i} \in \mathcal{X}^*$ is the i-th element of L_2 for all $i \in [m_{in}]$, and the return value is the number of elements in the list L_{12} output by the interval merge procedure in Figure 4–given the interval scaling factor α–satisfies the bounded differences property in Theorem 2. Furthermore, this property is satisfied with $c_i = 2$ for all $i \in [2m_{in}]$, and $c_i = 1$ for all $i \in [2m_{in} + 1, 2m_{in} + N_{int}]$. □

Proof (Claim). This can be seen as follows. Altering the value of a single list element changes which particular interval it occupies. Depending on the index of the element, it may change the value of the sum submitted to the rejection sampling procedure. With this in mind, it is simple to see how changing the value of a single list element can at most change the output of f by ± 2. If the element is contained in a half-occupied (Definition 2) pair of intervals, then moving the element can make at most one new pair of fully occupied intervals. This would raise the value of f by 1, should the newly created sum pass the rejection sampling procedure.

On the other hand, if the element is contained in a fully occupied pair of intervals, then moving the element can in fact remove two elements from the output list. Moving the element could leave its old interval pair half-occupied, and if the interval the element is moved to is already fully occupied, a different sum for that interval could be submitted to the rejection sampling procedure. This occurs when the element in question appears earlier in its list (thus selected for summation). If the new sum does not pass the rejection sampling (as the randomness remains unchanged), then this would lower the value of f by 2.

For the remaining N_{int} inputs to the function f, the value of f can change by at most 1 as these inputs govern the randomness for the rejection sampling procedure. As such, modifying these inputs merely alters the decision of the rejection sampling, which adds or subtracts an element from the output list. □

Additionally, note the assumption that the input list elements are sampled uniformly and independently. If we assign each list element to one of the function inputs, then each input is an independent random variable. Therefore, we satisfy all preconditions to applying the inequality. Here we apply the inequality in its single-sided variant, as a lower bound on the size of merged lists suffices for our analysis. Thus,

$$\begin{aligned}
\mathbf{P}[f(X_1, \ldots, X_{2m+N_{int}}) - \mathbf{E}\left[f(X_1, \ldots, X_{2m+N_{int}})\right] \leq -\epsilon] \\
&\leq \exp\left(-\frac{2\epsilon^2}{\sum_{i=1}^{2m_{in}} 2^2 + \sum_{i=1}^{N_{int}} 1^2}\right) \\
&\leq \exp\left(-\frac{2\epsilon^2}{8m_{in} + N_{int}}\right)
\end{aligned}$$

Now we can apply Lemma 1 to the expected value of f, and substitute the length of the output list L_{12} for the exact value of f.

$$\begin{aligned}\mathbf{P}\Bigg[|L_{12}| - \frac{1}{2}\cdot\Bigg(N_{int}^{(\mathcal{L})}\cdot\left(1-\left(1-\frac{\mathcal{L}}{2a+1}\right)^{m_{in}}\right)^2 \\ + N_{int}^{(\mathcal{L}+1)}\cdot\left(1-\left(1-\frac{\mathcal{L}+1}{2a+1}\right)^{m_{in}}\right)^2\Bigg) \leq -\epsilon\Bigg] \\ \leq \exp\left(-\frac{2\epsilon^2}{8m_{in}+N_{int}}\right)\end{aligned}$$

A bit of rearranging completes the proof.

$$\begin{aligned}\mathbf{P}\Bigg[|L_{12}| > \frac{1}{2}\cdot\Bigg(N_{int}^{(\mathcal{L})}\cdot\left(1-\left(1-\frac{\mathcal{L}}{2a+1}\right)^{m_{in}}\right)^2 \\ + N_{int}^{(\mathcal{L}+1)}\cdot\left(1-\left(1-\frac{\mathcal{L}+1}{2a+1}\right)^{m_{in}}\right)^2\Bigg) - \epsilon\Bigg] \\ > 1-\exp\left(-\frac{2\epsilon^2}{8m_{in}+N_{int}}\right) \qquad (16)\end{aligned}$$

□

3.3 Determining Constants

In order to make a proper comparison to the analysis by Shallue [28], we need to determine the constants involved. While the analysis in the previous section provides a complete characterization of the number of output elements at every level, the effect of the interval merge algorithm on the number of total initial list elements (and therefore the runtime) is unclear. For starters, the number of elements in each input list m_{in} changes on a per level basis. As such, the size of the subintervals $\mathcal{L}$ (Defined in (5)) necessarily changes as well.

We start by simplifying the result of Lemma 1 at the expense of some tightness.

Corollary 1. *Let notations be as in Lemma 1. Then*

$$\mathbf{E}\left[f(X_1,\ldots,X_{2m},r_1,\ldots,r_{N_{int}})\right] \geq \frac{1}{2}\left(N_{int}\cdot\left(1-\left(1-\frac{\mathcal{L}}{2a+1}\right)^{m_{in}}\right)^2\right).$$

This simplification is derived using the fact that

$$1 - \left(1 - \frac{\mathcal{L}}{2a+1}\right) < 1 - \left(1 - \frac{\mathcal{L}+1}{2a+1}\right).$$

Using the definition of $\mathcal{L}$ in Equation (5), observe that

$$\begin{aligned} 1 - \left(1 - \frac{\mathcal{L}}{2a+1}\right) &= 1 - \left(1 - \frac{\lfloor (2a)/(\alpha \cdot m_{in}) \rceil}{2a+1}\right) \\ &\approx 1 - \left(1 - \frac{1}{\alpha \cdot m_{in}}\right) \end{aligned}$$

for large enough input element domain $[-a, a]$. Similarly, the number of created intervals N_{int} can be approximated as

$$\begin{aligned} N_{int} &= \left\lfloor \frac{2a}{\mathcal{L}} \right\rfloor \\ &= \left\lfloor \frac{2a}{\lfloor (2a)/(\alpha \cdot m_{in}) \rceil} \right\rfloor \\ &\approx \alpha \cdot m_{in} \end{aligned} \tag{17}$$

Note that as a continues to grow, the approximation becomes arbitrarily accurate. Thus, we can restate the above corollary as follows.

Corollary 2. *Let notations be as in Lemma 1. Then*

$$\mathbf{E}\left[f(X_1 \ldots, r_{N_{int}})\right] \gtrapprox \frac{1}{2}\left(\alpha \cdot m_{in}\left(1 - \left(1 - \frac{1}{\alpha \cdot m_{in}}\right)^{m_{in}}\right)^2\right).$$

Asymptotically, the relative error of this approximation can be made arbitrarily close to 0.

We have now stated the expected value of f as a function of the quantity $\alpha \cdot m_{in}$. Ideally, we want to state the expected value of f as a fraction of m_{in}. This bounds the amount of elements lost to the interval merge procedure at every level.

Observe that

$$\lim_{m_{in} \to \infty} \frac{1}{2}\left(\alpha\left(1 - \left(1 - \frac{1}{\alpha \cdot m_{in}}\right)^{m_{in}}\right)^2\right) = \frac{1}{2}\left(\alpha\left(1 - e^{-1/\alpha}\right)^2\right) \tag{18}$$

Thus, as m_{in} grows large, we can optimize the fraction of remaining elements. Using numerical optimization, we found that (18) is maximized when $\alpha \approx 0.79591$, for a maximum value of ≈ 0.20363. With this in mind, we can then restate the corollary for a final time.

Corollary 3. *Let notations be as in Lemma 1 and fix* $\alpha = 0.79591$*. Then*

$$\mathbf{E}\left[f(X_1 \ldots, r_{N_{int}})\right] \gtrapprox 0.20363 \cdot m_{in}$$

Asymptotically, the relative error of this approximation can be made arbitrarily close to 0*.*

Note also that 'm_{in} large enough' is rather small, as convergence is quite quick (when $m_{in} = 10$, the absolute error is 0.014 for $\alpha = 0.8$). If we apply the above corollary to Lemma 2, we obtain a concrete lower bound on the number of output elements of each execution of the interval merge procedure in Figure 4.

Lemma 3. *Given two input lists* L_1*,* L_2 *of size* m_{in} *large enough, with elements sampled uniformly and independently from* $[-a, a] \subset \mathbb{Z}$ *large enough, for any* $\epsilon > 0$*, the size of the output list* L_{12} *of the* IntervalMerge *procedure in figure 4 is at least*

$$0.20363 \cdot m_{in} - \epsilon$$

for interval scaling parameter $\alpha = 0.79591$ *with probability at least* $1 - \exp\left(-2\epsilon^2/(8.79591 \cdot m_{in})\right)$*.*

By choosing ϵ to be a small fraction of m_{in}, we can bound the probability that the interval merge procedure returns a list of size $c \cdot m_{in}$ for some constant c. For concrete runtime parameters, we set c to $(1/6) = 0.1\bar{6}$. Thus,

Corollary 4. *Let notations be as in Lemma 3. The size of the output list* L_{12} *of the* IntervalMerge *procedure in figure 4 is at least*

$$(1/6) \cdot m_{in}$$

for interval scaling parameter $\alpha = 0.79591$ *with probability at least* $1 - \exp\left(-0.00031 \cdot m_{in}\right)$*.*

Now, all that remains is to combine Corollary 4 with Theorem 1 to obtain concrete parameters. Theorem 1 assumes that each input list is of the same size at every level. If we obtain larger lists than desired from the IntervalMerge procedure, then we can simply discard the excess—larger internal list sizes will result in a higher probability of finding a collision at the last level. With this in mind, we can find a lower bound on the probability that our degraded k-list algorithm meets the claimed runtime.

Corollary 5. *Let notations and input list sizes be as in Theorem 1 and let $c = 1/6, \alpha = 0.79591$. Then the success probability of the degraded k-List algorithm in Figure 3 is lower bounded as*

$$\prod_{i=1}^{\ell-1}\left(1-e^{-0.00031\cdot(1/6)^{\ell-i-1}\cdot m_{in}^{(\ell)}}\right)^{2^i}.$$

It can be verified that for all parameter settings used for Figure 2, this probability is 1 within computer precision. This is because for all relevant parameters, the term $(1/6)^{\ell}\cdot m_{in}^{(\ell)}$ will be much larger than 2^{ℓ}.

Proof. To begin, we need to calculate the joint probability that *all* lists have the requisite number of elements. Let B_j^i be the event that list L_j^i in the degraded k-list algorithm in Figure 3 has at least $c^{\ell-i}m_{in}^{(\ell)}$ elements where $m_{in}^{(\ell)}$ is the number of elements in each input list. Now, we can write the joint probability as

$$\begin{aligned}\mathbf{P}\left[\bigcap_{i=1}^{\ell-1}\bigcap_{j=1}^{2^i}B_j^i\right] &= \mathbf{P}\left[\bigcap_{h=1}^{2}B_h^1 \,\middle|\, \bigcap_{i=2}^{\ell-1}\bigcap_{j=1}^{2^i}B_j^i\right]\cdot\mathbf{P}\left[\bigcap_{i=2}^{\ell-1}\bigcap_{j=1}^{2^i}B_j^i\right]\\ &= \left(\mathbf{P}\left[B_1^1 \,\middle|\, \bigcap_{j=1}^{2}B_j^2\right]\right)^2\cdot\mathbf{P}\left[\bigcap_{i=2}^{\ell-1}\bigcap_{j=1}^{2^i}B_j^i\right] \qquad (19)\end{aligned}$$

where (19) is due to the fact that the sizes of lists at level i are solely dependent on the sizes of the lists at level $(i+1)$. In addition, the sizes of each list at level i are independent and identically distributed when conditioned on the event that lists at level $(i+1)$ are all the same size. Note that i goes to $\ell-1$, as the lists at level ℓ are the initial input lists. Their sizes are set to $m_{in}^{(\ell)}$ by definition.

We can continue decomposing the joint probability in (19), making use of the exhibited Markov property. Completing the decomposition yields

$$\prod_{i=1}^{\ell-1}\left(\mathbf{P}\left[B_1^i \,\middle|\, \bigcap_{j=1}^{2}B_j^{i+1}\right]\right)^{2^i} \qquad (20)$$

To obtain concrete parameters, we want to apply Corollary 4 to (20). Note that Corollary 4 is calculating the probability that a list at level i has at least $(1/6)\cdot m_{in}^{(i+1)}$ elements given that the input lists at level $(i+1)$ have *exactly* $m_{in}^{(i+1)}$ elements. In this regard, Corollary 4 provides

a lower bound on the probability. Therefore, we can state the following for $c = (1/6)$

$$\mathbf{P}\left[\bigcap_{i=1}^{\ell-1}\bigcap_{j=1}^{2^i} B_j^i\right] \geq \prod_{i=1}^{\ell-1}\left(1 - e^{-0.00031\cdot m_{in}^{(i+1)}}\right)^{2^i}$$
$$= \prod_{i=1}^{\ell-1}\left(1 - e^{-0.00031\cdot(1/6)^{\ell-i-1}\cdot m_{in}^{(\ell)}}\right)^{2^i} \tag{21}$$

where the list size at level $(i+1)$, $m_{in}^{(i+1)}$ is substituted for m_{in} in the application of Corollary 4. Thus, for any fixed number of lists $k = 2^\ell$, the joint probability that the size of the lists at each level of the degraded k-list algorithm depicted in Figure 3 shrink by at most $(1/6)$ is overwhelming in the number of inputs to the initial lists at level ℓ. □

Acknowledgements

This work is funded by the European Union, ERC-2023-STG-101116713. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union. Neither the European Union nor the granting authority can be held responsible for them. Hunter Kippen is supported in part by the Clark Doctoral Fellowship from the Clark School of Engineering, University of Maryland, College Park.

References

1. Augot, D., Finiasz, M., Manuel, P.G.S., Sendrie, N.: Sha-3 proposal: Fsb (2009), https://www.rocq.inria.fr/secret/CBCrypto/fsbdoc.pdf
2. Augot, D., Finiasz, M., Sendrier, N.: A fast provably secure cryptographic hash function. Cryptology ePrint Archive, Report 2003/230 (2003), https://eprint.iacr.org/2003/230
3. Austrin, P., Kaski, P., Koivisto, M., Nederlof, J.: Subset sum in the absence of concentration. In: Mayr, E.W., Ollinger, N. (eds.) 32nd International Symposium on Theoretical Aspects of Computer Science, STACS 2015, March 4-7, 2015, Garching, Germany. LIPIcs, vol. 30, pp. 48–61. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2015). https://doi.org/10.4230/LIPICS.STACS.2015.48, https://doi.org/10.4230/LIPIcs.STACS.2015.48
4. Becker, A., Coron, J.S., Joux, A.: Improved generic algorithms for hard knapsacks. In: Paterson, K.G. (ed.) Advances in Cryptology – EUROCRYPT 2011. Lecture Notes in Computer Science, vol. 6632, pp. 364–385. Springer, Heidelberg, Germany, Tallinn, Estonia (May 15–19, 2011). https://doi.org/10.1007/978-3-642-20465-4_21
5. Becker, A., Joux, A., May, A., Meurer, A.: Decoding random binary linear codes in $2^{n/20}$: How $1 + 1 = 0$ improves information set decoding. In: Pointcheval, D., Johansson, T. (eds.) Advances in Cryptology – EUROCRYPT 2012. Lecture Notes in Computer Science, vol. 7237, pp. 520–536. Springer, Heidelberg, Germany, Cambridge, UK (Apr 15–19, 2012). https://doi.org/10.1007/978-3-642-29011-4_31
6. Benhamouda, F., Lepoint, T., Loss, J., Orrù, M., Raykova, M.: On the (in)security of ROS. Journal of Cryptology **35**(4), 25 (Oct 2022). https://doi.org/10.1007/s00145-022-09436-0
7. Bernstein, D.J., Jeffery, S., Lange, T., Meurer, A.: Quantum algorithms for the subset-sum problem. In: Gaborit, P. (ed.) Post-Quantum Cryptography - 5th International Workshop, PQCrypto 2013. pp. 16–33. Springer, Heidelberg, Germany, Limoges, France (Jun 4–7, 2013). https://doi.org/10.1007/978-3-642-38616-9_2
8. Bernstein, D.J., Lange, T., Niederhagen, R., Peters, C., Schwabe, P.: FSBday. In: Roy, B.K., Sendrier, N. (eds.) Progress in Cryptology - INDOCRYPT 2009: 10th International Conference in Cryptology in India. Lecture Notes in Computer Science, vol. 5922, pp. 18–38. Springer, Heidelberg, Germany, New Delhi, India (Dec 13–16, 2009)
9. Bonnetain, X., Bricout, R., Schrottenloher, A., Shen, Y.: Improved classical and quantum algorithms for subset-sum. In: Moriai, S., Wang, H. (eds.) Advances in Cryptology – ASIACRYPT 2020, Part II. Lecture Notes in Computer Science, vol. 12492, pp. 633–666. Springer, Heidelberg, Germany, Daejeon, South Korea (Dec 7–11, 2020). https://doi.org/10.1007/978-3-030-64834-3_22
10. Boura, C., Canteaut, A.: Zero-sum distinguishers for iterated permutations and application to Keccak-f and Hamsi-256. In: Biryukov, A., Gong, G., Stinson, D.R. (eds.) SAC 2010: 17th Annual International Workshop on Selected Areas in Cryptography. Lecture Notes in Computer Science, vol. 6544, pp. 1–17. Springer, Heidelberg, Germany, Waterloo, Ontario, Canada (Aug 12–13, 2011). https://doi.org/10.1007/978-3-642-19574-7_1
11. Coron, J.S., Joux, A.: Cryptanalysis of a provably secure cryptographic hash function. Cryptology ePrint Archive, Report 2004/013 (2004), https://eprint.iacr.org/2004/013

12. Drijvers, M., Edalatnejad, K., Ford, B., Kiltz, E., Loss, J., Neven, G., Stepanovs, I.: On the security of two-round multi-signatures. In: 2019 IEEE Symposium on Security and Privacy. pp. 1084–1101. IEEE Computer Society Press, San Francisco, CA, USA (May 19–23, 2019). https://doi.org/10.1109/SP.2019.00050
13. Esser, A., Zweydinger, F.: New time-memory trade-offs for subset sum: Improving ISD in theory and practice. In: Hazay, C., Stam, M. (eds.) Advances in Cryptology – EUROCRYPT 2023, Part V. Lecture Notes in Computer Science, vol. 14008, pp. 360–390. Springer, Heidelberg, Germany, Lyon, France (Apr 23–27, 2023). https://doi.org/10.1007/978-3-031-30589-4_13
14. Fuchsbauer, G., Plouviez, A., Seurin, Y.: Blind schnorr signatures and signed ElGamal encryption in the algebraic group model. In: Canteaut, A., Ishai, Y. (eds.) Advances in Cryptology – EUROCRYPT 2020, Part II. Lecture Notes in Computer Science, vol. 12106, pp. 63–95. Springer, Heidelberg, Germany, Zagreb, Croatia (May 10–14, 2020). https://doi.org/10.1007/978-3-030-45724-2_3
15. Gennaro, R., Jarecki, S., Krawczyk, H., Rabin, T.: Secure distributed key generation for discrete-log based cryptosystems. Journal of Cryptology **20**(1), 51–83 (Jan 2007). https://doi.org/10.1007/s00145-006-0347-3
16. Grassi, L., Naya-Plasencia, M., Schrottenloher, A.: Quantum algorithms for the k-xor problem. In: Peyrin, T., Galbraith, S. (eds.) Advances in Cryptology – ASIACRYPT 2018, Part I. Lecture Notes in Computer Science, vol. 11272, pp. 527–559. Springer, Heidelberg, Germany, Brisbane, Queensland, Australia (Dec 2–6, 2018). https://doi.org/10.1007/978-3-030-03326-2_18
17. Howgrave-Graham, N., Joux, A.: New generic algorithms for hard knapsacks. In: Gilbert, H. (ed.) Advances in Cryptology – EUROCRYPT 2010. Lecture Notes in Computer Science, vol. 6110, pp. 235–256. Springer, Heidelberg, Germany, French Riviera (May 30 – Jun 3, 2010). https://doi.org/10.1007/978-3-642-13190-5_12
18. Joux, A.: Cryptanalysis of the EMD mode of operation. In: Biham, E. (ed.) Advances in Cryptology – EUROCRYPT 2003. Lecture Notes in Computer Science, vol. 2656, pp. 1–16. Springer, Heidelberg, Germany, Warsaw, Poland (May 4–8, 2003). https://doi.org/10.1007/3-540-39200-9_1
19. Komlo, C., Goldberg, I.: Frost: Flexible round-optimized schnorr threshold signatures **Version from "January 7, 2020"** (2020), https://crysp.uwaterloo.ca/software/frost/frost-extabs.pdf
20. Levieil, É., Fouque, P.A.: An improved LPN algorithm. In: Prisco, R.D., Yung, M. (eds.) SCN 06: 5th International Conference on Security in Communication Networks. Lecture Notes in Computer Science, vol. 4116, pp. 348–359. Springer, Heidelberg, Germany, Maiori, Italy (Sep 6–8, 2006). https://doi.org/10.1007/11832072_24
21. Lyubashevsky, V.: The parity problem in the presence of noise, decoding random linear codes, and the subset sum problem. In: International Workshop and International Workshop on Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (2005), https://api.semanticscholar.org/CorpusID:7748280
22. Maxwell, G., Poelstra, A., Seurin, Y., Wuille, P.: Simple schnorr multi-signatures with applications to bitcoin. Cryptology ePrint Archive, Report 2018/068 (2018), https://eprint.iacr.org/2018/068
23. McDiarmid, C.: On the method of bounded differences, p. 148–188. London Mathematical Society Lecture Note Series, Cambridge University Press (1989). https://doi.org/10.1017/CBO9781107359949.008

24. Minder, L., Sinclair, A.: The extended k-tree algorithm. Journal of Cryptology **25**(2), 349–382 (Apr 2012). https://doi.org/10.1007/s00145-011-9097-y
25. Nikolic, I., Sasaki, Y.: Refinements of the k-tree algorithm for the generalized birthday problem. In: Iwata, T., Cheon, J.H. (eds.) Advances in Cryptology – ASIACRYPT 2015, Part II. Lecture Notes in Computer Science, vol. 9453, pp. 683–703. Springer, Heidelberg, Germany, Auckland, New Zealand (Nov 30 – Dec 3, 2015). https://doi.org/10.1007/978-3-662-48800-3_28
26. Pointcheval, D., Stern, J.: Security arguments for digital signatures and blind signatures. Journal of Cryptology **13**(3), 361–396 (Jun 2000). https://doi.org/10.1007/s001450010003
27. Schnorr, C.P.: Security of blind discrete log signatures against interactive attacks. In: Qing, S., Okamoto, T., Zhou, J. (eds.) ICICS 01: 3rd International Conference on Information and Communication Security. Lecture Notes in Computer Science, vol. 2229, pp. 1–12. Springer, Heidelberg, Germany, Xian, China (Nov 13–16, 2001)
28. Shallue, A.: An improved multi-set algorithm for the dense subset sum problem. In: Algorithmic Number Theory: 8th International Symposium, ANTS-VIII Banff, Canada, May 17-22, 2008 Proceedings 8. pp. 416–429. Springer (2008)
29. Syta, E., Tamas, I., Visher, D., Wolinsky, D.I., Jovanovic, P., Gasser, L., Gailly, N., Khoffi, I., Ford, B.: Keeping authorities "honest or bust" with decentralized witness cosigning. In: 2016 IEEE Symposium on Security and Privacy. pp. 526–545. IEEE Computer Society Press, San Jose, CA, USA (May 22–26, 2016). https://doi.org/10.1109/SP.2016.38
30. Wagner, D.: A generalized birthday problem. In: Yung, M. (ed.) Advances in Cryptology – CRYPTO 2002. Lecture Notes in Computer Science, vol. 2442, pp. 288–303. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 18–22, 2002). https://doi.org/10.1007/3-540-45708-9_19