CR10: Cryptanalysis Autumn 2018

Lecture 2 — September 18th, 2018

Lecturer G. Hanrot, E. Kirshanova *Scribe: TA Duy Hoang*

The k-list problem. An algorithm for dense modular subset sum.

1 The 2-list Problem (classic birthday problem)

1.1 Problem

One of the best known combinatorial tool in cryptology is the birthday problem. In this section we will introduce the 2-list (classic birthday) problem and some of its applications.

Definition 1 (The 2-list Problem)**.** *Given two lists L_1, L_2 of elements drawn uniformly and independently at random from $\{0,1\}^n$, find $x_1 \in L_1$ and $x_2 \in L_2$ such that $x_1 \oplus x_2 = 0$ (the $\oplus$ symbol represents the bitwise exclusive-or operation).*

Lemma 2. *A solution (x_1, x_2) exists with good probability if $|L_1|, |L_2| > 2^{n/2}$. Furthermore, (x_1, x_2) can be found in $\widetilde{\mathcal{O}}\left(2^{n/2}\right)$ time.*

Proof. Consider the following experiment:

1. Generate L_1 uniform binary vectors from. $\{0,1\}^n$.
2. Generate L_2 random independent binary vectors from $\{0,1\}^n$.

We have

$$\begin{aligned}
\Pr\left[\exists x_2 \in L_2 : x_2 = x_1 \text{ for some } x_1 \in L_1\right] &= 1 - \Pr\left[\text{ all } x_2 \text{ differ from } L_1\right] \\
&= 1 - \prod_{i=1}^{|L_2|} \Pr\left[L_2[i] \notin L_1\right] \\
&= 1 - \prod_{i=1}^{|L_2|} \frac{2^n - |L_1|}{2^n} \\
&= 1 - \left(1 - \frac{|L_1|}{2^n}\right)^{|L_2|} \\
&> 1 - e^{-\frac{|L_1| \times |L_2|}{2^n}} = \Theta(1) \text{ for } |L_1| \times |L_2| > 2^n.
\end{aligned}$$

A merge-join sorts the two lists L_1, L_2 and scans the two sorted lists, returning any matching pairs. The merge-join requires $\mathcal{O}(n \log n)$ time where $n = \max\left(|L_1|, |L_2|\right)$. □

Remarks

1. We can increase the success probability by choosing a larger L_i.
2. Fact 2 holds not only for $(\mathbb{F}_2, \oplus)$. but also for other groups.

1.2 Some applications of the 2-list problem

Collision search for a hash function

Definition 3. *For a hash function $h : \{0,1\}^* \to \{0,1\}^n$, a collision is a pair (x_1, x_2) $(x_1 \neq x_2)$ such that $h(x_1) = h(x_2)$.*

Assuming, h behaves like a random function (i.e images of h are distributed uniformly and independent at random in $\{0,1\}^n$). Set:

$$L_1 = \{(x_1, h(x_1), x_1[1] = 0)\}$$
$$L_2 = \{(x_2, h(x_2), x_2[1] = 1)\}$$

Fact 4. *If $|L_1| \approx |L_2| \approx 2^{n/2}$, then a collision for h exists with high probability and can be found in time $\widetilde{\mathcal{O}}\left(2^{n/2}\right)$.*

A discrete logarithm collision algorithm

Definition 5. *Let G be a group of order N and h be a generator of G (i.e ord$(h) = N$). In the discrete logarithm problem, we are given (h, b) such that $b = h^x$, and asked to find x.*

We can rewrite: $h^x = b$ as $h^y = bh^z$ for $x = y - z$. we construct:

$$L_1 = \{(y, h^y), y \overset{\$}{\leftarrow} [0, N-1]\}$$
$$L_2 = \{(z, bh^z), z \overset{\$}{\leftarrow} [0, N-1]\}$$

Run a collision search algorithm on L_1, L_2.

Fact 6. *If $|L_1| \approx |L_2| > \sqrt{N}$, then a solution x can be found in $\widetilde{\mathcal{O}}(\sqrt{N})$ time. Concretely, we expect to find $(y, z) \in L_1 \times L_2$ such that $y - z = x \mod N$.*

2 The k-list algorithm

In this section we will present the k-list problem and an algorithm to solve it. The problem is defined as follows [2].

Definition 7. *Given k lists $L_1, \ldots, L_k$ of elements drawn uniformly and independently at random from $\{0,1\}^n$, find $x_1 \in L_1, \ldots, x_k \in L_k$ such that $x_1 \oplus \cdots \oplus x_k = 0$.*

We allow the lists to be extended to any desired length, and so it may aid the intuition to think of each element of each list as being generated by a random (or pseudorandom) oracle R_i, so that the j^{th} element of L_i is $R_i(j)$. Similar to Fact 2, it is easy to see that a solution to the k-list problem exists with good probability so long as $|L_1| \times \cdots \times |L_k| > 2^n$.

2.1 The 4-list problem

Before presenting an algorithm for the k-list problem, first we consider the problem with $k = 4$. Let $low_\ell(x)$ denote the least significant ℓ bits of x, and let L_{ij} contain all pairs from $L_i \times L_j$ that agree on their ℓ least significant bits. We will use the following basic properties of the problem:

Observation 8. *We have $low_\ell(x_i \oplus x_j) = 0$ if and only if $low_\ell(x_i) = low_\ell(x_j)$.*

Observation 9. *Given lists L_i, L_j, we can generate all pairs (x_i, x_j) satisfying $x_i \in L_i, x_j \in L_j$, and $low_\ell(x_i \oplus x_j) = 0$, in time $\widetilde{\mathcal{O}}(\max_i\{|L_i|\})$.*

Observation 10. *If $x_1 \oplus x_2 = x_3 \oplus x_4$, then $x_1 \oplus x_2 \oplus x_3 \oplus x_4 = 0$.*

Observation 11. *If $low_\ell(x_1 \oplus x_2) = 0$ and $low_\ell(x_3 \oplus x_4) = 0$, then we have $low_\ell(x_1 \oplus x_2 \oplus x_3 \oplus x_4) = 0$, and this case $\Pr[x_1 \oplus x_2 \oplus x_3 \oplus x_4 = 0] = 2^\ell/2^n$ for uniformly random and independently chosen x_i's.*

These properties suggest an algorithm for the 4-list problem. First, we extend the list $L_1, \dots, L_4$ until they each contain about 2^ℓ elements, where ℓ will be determined below. Then, we generate a large list L_{12} of values $x_1 \oplus x_2$ such that $low_\ell(x_1 \oplus x_2) = 0$. Similarly, we also have a list L_{34} of values $x_3 \oplus x_4$ where $low_\ell(x_3 \oplus x_4) = 0$. Finally, we search for matches between L_{12} and L_{34}. By Observation 10, any such match will satisfy $x_1 \oplus x_2 \oplus x_3 \oplus x_4 = 0$ and hence will yield a solution to 4-list problem. See Figure 1 for an illustration of this algorithm.

The complexity of this algorithm may be analyzed as follows.

$$\mathbb{E}[|L_{12}|] = \mathbb{E}[|L_{34}|] = \sum_{(x_1,x_2) \in L_1 \times L_2} \Pr[low_\ell(x_1 \oplus x_2) = 0] = \frac{|L_1| \times |L_2|}{2^\ell} = 2^\ell.$$

Observation 11 ensures that any pair of elements from $L_{12} \times L_{34}$ yields a match with probability $2^\ell/2^n$. Therefore,

$$\begin{aligned}\mathbb{E}[|E_{1234}|] &= \sum_{(x_{12},x_{34}) \in L_{12} \times L_{34}} \Pr[x_1 \oplus x_2 \oplus x_3 \oplus x_4 = 0 | low_\ell(x_1 \oplus x_2) = low_\ell(x_3 \oplus x_4)] \\ &= \frac{\mathbb{E}[|L_{12}|] \times \mathbb{E}[|E_{34}|]}{2^{n-\ell}} \\ &= 2^{3\ell - n}.\end{aligned}$$

The latter equation is at least 1 when $\ell \geq n/3$. Consequently, if we set $\ell = n/3$, we expect to find a solution to the 4-list problem. Since the size of all the lists appearing in the process above are of size $2^{n/3}$, the algorithm can be implemented in $\widetilde{\mathcal{O}}(2^{n/3})$ time and space.

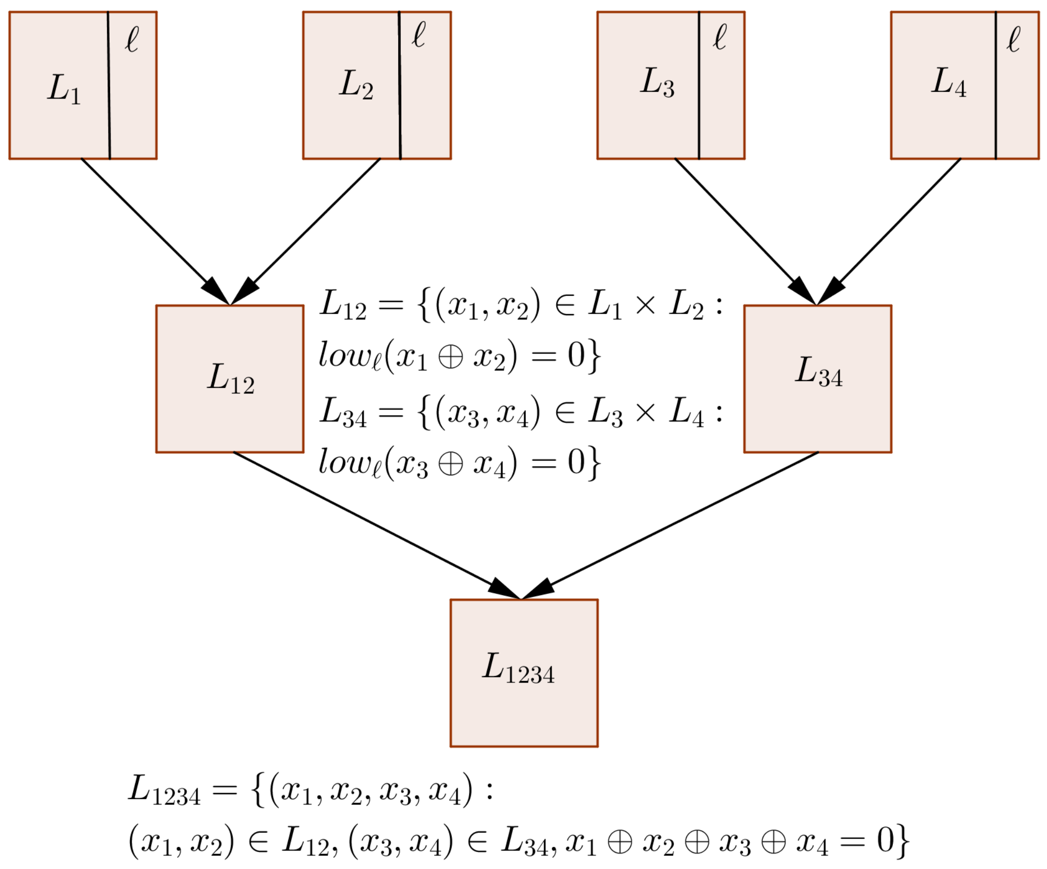
ℓ
L_1
ℓ
L_2
ℓ
L_3
ℓ
L_4
L_{12}
$L_{12} = \{(x_1, x_2) \in L_1 \times L_2 :$
$low_\ell(x_1 \oplus x_2) = 0\}$
$L_{34} = \{(x_3, x_4) \in L_3 \times L_4 :$
$low_\ell(x_3 \oplus x_4) = 0\}$
L_{34}
L_{1234}
$L_{1234} = \{(x_1, x_2, x_3, x_4) :$
$(x_1, x_2) \in L_{12}, (x_3, x_4) \in L_{34}, x_1 \oplus x_2 \oplus x_3 \oplus x_4 = 0\}$

Figure 1: A visual representation of the algorithm with $k = 4$.

2.2 Algorithm for k-list problem

The above algorithm finds only solutions with a special property, namely, $x_1 \oplus x_2$ and $x_3 \oplus x_4$ are zero in their *low* ℓ bits. However, this restriction was made merely for ease of presentation, and it can be eliminated. We can pick randomly ℓ bits of value α, and look for pairs $(x_1, x_2) \in L_1 \times (L_2 \oplus \alpha)$, $(x_3, x_4) \in L_3 \times (L_4 \oplus \alpha)$. Moreover, the value 0 in the problem's statement, i.e., the requirement $x_1 \oplus \cdots \oplus x_k = 0$, is not essential, and can be replaced by any other constant c without increasing complexity. We can replace L_k with $L'_k = L_k \oplus c$, then any solution $x_1 \oplus \cdots \oplus x'_k = 0$ will be a solution to $x_1 \oplus \cdots \oplus x_k$ and vice versa. Therefore, without loss of generality we assume that $c = 0$.

As a corollary, when $k > k'$ the complexity of the k-list problem can be no larger than the complexity of k'-list problem. We pick arbitrary values $x_{k'+1}, \ldots, x_k$ from $L_{k'+1}, \ldots, L_k$ and fix this choice. Then, we set $c := x_{k'+1} \oplus \cdots \oplus x_k$ and find a solution for the k'-list problem with $x_1 \oplus \cdots \oplus x_{k'} = c$. For instance, this shows that we can solve the k-list problem with complexity at most $\widetilde{\mathcal{O}}(2^{n/3})$ for all $k \geq 4$. So, can we do better?

More interestingly, we can use the idea of the 4-list algorithm to solve the k-list problem even faster than cube-root time for larger values of k. We extend the 4-list algorithm above as follows. Let k be a power of two (we consider arbitrary k case later). We replace the binary tree of depth 2 from Figure 1 with a binary tree of depth $\log k$. We set $\ell_h := hn/(1 + \log k)$ for low significant bits to create a new list at level h from two lists at level $h-1$. Namely, the k-list algorithm introduced by David Wagner [2] can be described as follows.

Algorithm 1 The k-list algorithm (Wagner's algorithm) for $k = 2^m$

```
Input: $L_1, \dots, L_{2^m} \subset \{0,1\}^n, |L_i| = 2^{\frac{n}{m+1}}$ for all $i = 1, \dots, 2^m$
Output: $(x_1, \dots, x_{2^m}) \in L_1 \times \dots \times L_{2^m}$ such that $x_1 \oplus \dots \oplus x_{2^m} = 0$ or fail
1: Let $\ell \leftarrow \frac{n}{m+1}$
2: for $i = 1$ to $m-1$ do                                              ▷ depth of the tree
3:     for $j = 1$ to $2^m$, $j = j \times 2^i$ do                      ▷ $j$-th level of the tree
4:         Compute $L_{j \dots j+2^i-1}$ by merging 2 lists $L_{j \dots j+2^{i-1}-1}$, $L_{j+2^{i-1} \dots j+2^i-1}$ on the $n\ell$ low significant bits.
5: Compute $L_{1 \dots 2^m}$ by merging 2 lists $L_{1 \dots 2^{m-1}}$ and $L_{2^{m-1}+1 \dots 2^m}$ on all $n$ bits
6: if $L_{1 \dots 2^m} \neq \emptyset$ then return any element of $L_{1 \dots 2^m}$
7: else return fail
```

Theorem 12. *The expected size of the output list $L_{1 \dots 2^m}$ in **Algorithm 1** is 1. The running time and the memory of **Algorithm 1** are*

$$T(kList) = M(kList) = \widetilde{\mathcal{O}}\left(k 2^{\frac{n}{\log k + 1}}\right).$$

In particular, for $k = 2^{\sqrt{n}}$, setting $\ell = 2^{\sqrt{n}}$, we obtain $T(kList) = M(kList) = \widetilde{\mathcal{O}}\left(2^{2\sqrt{n}}\right)$.

Proof. (sketch) The starting lists $L_1, \dots, L_k$ are of size $2^\ell = 2^{\frac{n}{\log k+1}}$. By induction, one can show that all intermediate lists created on step 4 are of expected size 2^ℓ. In step 5 we have,

$$\mathbb{E}[|L_{1 \dots 2^m}|] = \sum_{(x_1, \dots, x_{2^m}) \in L_{1 \dots 2^{m-1}} \times L_{2^{m-1}+1 \dots 2^m}} \Pr[x_1 \oplus \dots \oplus x_{2^m} = 0 | X = Y]$$

where $X = low_{(m-1)\ell}(x_1 \oplus \dots \oplus x_{2^{m-1}})$ and $Y = low_{(m-1)\ell}(x_{2^{m-1}+1} \oplus \dots \oplus x_{2^m})$. Therefore

$$\mathbb{E}[|L_{1 \dots 2^m}|] = \frac{2^{2\ell}}{2^{n-(m-1)\ell}} = 1, \quad \text{since } \ell = \frac{n}{m+1}.$$

The running time of merging any two lists on step 4 is $\widetilde{\mathcal{O}}\left(2^{\frac{n}{m+1}}\right) = \widetilde{\mathcal{O}}\left(2^{\frac{n}{\log k+1}}\right)$, we perform this merging $k \log k$ times. Therefore, the total running time is $\widetilde{\mathcal{O}}\left(k 2^{\frac{n}{\log k+1}}\right)$. The memory is bounded by the size of the lists we store and is $\widetilde{\mathcal{O}}\left(k 2^{\frac{n}{\log k+1}}\right)$.

When $k \gg 2$, we can evaluate the tree in postfix order, discarding lists when they are no longer needed. In this way, we will need storage for only about $\log k$ lists. If we take $k = 2^{\sqrt{n}-1}$, then the algorithm will run in $\widetilde{\mathcal{O}}\left(2^{2\sqrt{n}}\right)$ time and $\widetilde{\mathcal{O}}\left(2^{\sqrt{n}}\right)$ space using this optimization. □

Remarks

1. We can also obtain an algorithm for the general k-list problem when k is not a power of two. We take $k' = 2^{\lfloor \log k \rfloor}$ to be the largest power of two less than k, and we use the list-elimination trick above. Namely, if $k = 2^m + j$ for $1 \le j \le 2^m - 1$, we set $L_1, \dots, L_{k'}$ of

size $\frac{n}{\log k'+1}$ and run the algorithm for k'-list problem with $c = x_{2^m+1} \oplus \cdots \oplus x_{2^m+j}$ for any $(x_{2^m+1}, \ldots, x_{2^m+j}) \in L_{2^m+1} \times \cdots \times L_{2^m+j}$. However, the results are less satisfying: The algorithm for $k = 2^m + j$ obtained in this way runs no faster than the algorithm for $k = 2^m$.

2. We can also apply the algorithm to the group $(\mathbb{Z}/m\mathbb{Z}, +)$ where m is arbitrary. Let $[a, b] = \{x \in \mathbb{Z}/m\mathbb{Z} : a \leq x \leq b\}$ denote the interval of elements between a and b (wrapping modulo m), and define L_{12} to represent the solutions to $x_1 + x_2 \in [a, b]$ with $x_1 \in L_1$ and $x_2 \in L_2$. Then we may solve a 4-list problem over $\mathbb{Z}/m\mathbb{Z}$ by computing L_{1234} from 2 lists L_{12} and L_{34} where $[a, b] = \left[-m/2^{\ell+1}, m/2^{\ell+1} - 1\right]$ and $\ell = \frac{1}{3}\log m$. In general k, we will present an algorithm and its analysis proposed by Vadim Lyubashevsky [1] for k-list problem over $\mathbb{Z}/m\mathbb{Z}$ in the next section.

Open Problems

1. *Other values of k.* We have present an improved algorithm only for the case where k is a power of two. An open question is whether this restriction can be removed. A case of particular interest might be where $k = 3$: is there any group operation $+$ where solutions to $x_1 + x_2 + x_3 = 0$ can be found more efficiently than a naive square-root birthday search?

2. *Other combining operations.* We can ask for other operations $+$ where the k-list problem $x_1 + \cdots + x_k = c$ has efficient algorithms.

2.3 Modified Wagner's algorithm

In this section, we will present a modified Wagner's algorithm over $\mathbb{Z}_N$ and analyze its probability of success. These results introduced by Vadim Lyubashevsky in [1]. For the analysis of this algorithm, let us recall a well-known bound on tail distributions of sums of independent random variables.

Lemma 13. *(Chernoff Bound) Let $S_1, \ldots, S_m$ be a sequence of m independent Bernoulli random variables, such that* $\Pr[S_i = 1] = p$. *Let $S = S_1 + \cdots + S_m$. Then for $0 \leq \gamma \leq 1$. We have,*

$$\Pr[S < (p - \gamma)] \leq e^{-2m\gamma^2}.$$

A key subroutine in the modified Wagner's algorithm is ListMerge, which takes as input two lists L_1 and L_2 of integers in the interval $\left[-\frac{R}{2}, \frac{R}{2}\right)$ and outputs elements $b + c \in \left[-\frac{Rp}{2}, \frac{Rp}{2}\right)$ where $b \in L_1, c \in L_2$. Here $p < 1$ is a parameter set at the beginning.The algorithm takes k lists each containing $tN^{\frac{2}{\log k}}$ number uniformly and independently distributed in the range $[0, N)$, and returns k numbers (one from each list) whose sum is zero. The following lemma provides a relationship between how many numbers there are in L_1, L_2 lists and how many numbers there are in the new list depending on p. Intuitively, the larger the difference between R and Rp, the fewer numbers we will have in the new list.

Lemma 14. *Let L_1 and L_2 be lists of numbers in the range $\left[-\frac{R}{2}, \frac{R}{2}\right]$ and p and c be positive reals such that $e^{-\frac{c}{12}} < p < \frac{1}{8}$. Let L_3 be a list of numbers $a_1 + a_2$ such that $a_1 \in L_1, a_2 \in L_2$, and $a_1 + a_2 \in \left[-\frac{Rp}{2}, \frac{Rp}{2}\right)$. If L_1 and L_2 each contain at least $\frac{c}{p^2}$ independent, uniformly distributed numbers in $\left[-\frac{R}{2}, \frac{R}{2}\right)$, then with probability greater than $1 - e^{-\frac{c}{4}}$, L_3 contains at least $\frac{c}{4p^2}$ independent, uniformly distributed numbers in the range $\left[-\frac{Rp}{2}, \frac{Rp}{2}\right)$.*

Proof. We will construct the list L_3 that consist entirely of independent, uniformly distributed numbers in the range $\left[-\frac{Rp}{2}, \frac{Rp}{2}\right)$ and there is at least $\frac{c}{4^2}$ elements with high probability. First, we will show a lemmas about the distribution of the numbers from list L_1 and L_2 in the range $\left[-\frac{R}{2}, \frac{R}{2}\right)$ use Lemma 13.

Claim 15. *With probability greater than $1 - e^{-\frac{c}{2}}$, any interval of length Rp that is fully contained inside range $\left[-\frac{R}{2}, \frac{R}{2}\right)$, has at least $\frac{c}{2p}$ numbers from L_1 (similarly L_2).*

Proof. (Claim 15): L_1(similarly L_2) has $\frac{c}{p^2}$ independent uniformly distributed numbers in the range $\left[-\frac{R}{2}, \frac{R}{2}\right)$. For each one, the probability that it falls into a specified interval that has size Rp is p. Let $S_i = 1$ if i^{th} number falls into the interval and $S_i = 0$ in otherwise. By applying the Chernoff Bound from Lemma 13 with $\gamma = \frac{p}{2}, m = \frac{c}{p^2}$, we get $\Pr\left[S < \frac{c}{2p}\right] \leq e^{-\frac{c}{2}}$. ☐

Claim 16. *With probability greater than $1 - e^{-\frac{c}{3}}$, any interval of length $\frac{Rp}{2}$ that is fully contained inside range $\left[-\frac{R}{2}, \frac{R}{2}\right)$, has at least $\frac{c}{12p}$ numbers from L_1 (similarly L_2).*

Proof. The proof similar to with proof in Claim 15. ☐

Now we continue with the proof of Lemma 14. Consider the intervals

$$A_\ell = \left[-\frac{R}{2} + \frac{(1+3\ell)Rp}{2}, -\frac{R}{2} + \frac{(2+3\ell)Rp}{2}\right)$$

for integers ℓ where $0 \leq \ell \leq \frac{1}{2p}$. Notice that the intervals are disjoint, have size $\frac{Rp}{2}$, and there is a distance of at least Rp between every two intervals. From each interval A_ℓ, select an a_ℓ from L_1 that is in the interval. By Claim 16, such an a_ℓ exists in each interval with probability greater than $1 - e^{-\frac{c}{3}}$. Therefore

$$\Pr\left[\text{ Each of the } \frac{1}{2p} \text{ intervals contains an element from } L_1\right] \tag{1}$$

$$= 1 - \sum_{\ell=0}^{1/2p} \Pr\left[A_\ell \text{ does not contain any element from } L_1\right] \tag{2}$$

$$\geq 1 - \frac{1}{2p}e^{-\frac{c}{3}}. \tag{3}$$

For each A_ℓ pick $a_\ell \in A_\ell$ from L_1. We have the following facts:

1. By our choice of the intervals, the distance between any a_i and a_j is at least Rp.
2. Since $p < \frac{1}{8}$, $-\frac{R}{2} + \frac{Rp}{2} \leq a_i \leq \frac{R}{2} - \frac{Rp}{2}$ for all i.

For each ℓ, define B_ℓ to be the list of numbers in L_2 in the range $\left[-a_\ell - \frac{Rp}{2}, -a_\ell + \frac{Rp}{2}\right)$. Now we notice that since all the a_ℓ are at a distance of at least Rp from each other, the lists B_ℓ are disjoint. By Claim 15, with probability greater than $1 - e^{-\frac{c}{2}}$ each B_ℓ contains at least $\frac{c}{2p}$ numbers. So the probability that each of the B_ℓ contain more than $\frac{Rp}{2}$ elements is greater than $1 - \frac{1}{2p}e^{-\frac{c}{2}}$. Since adding $(a_\ell + B_\ell) \subset \left[-\frac{Rp}{2}, \frac{Rp}{2}\right)$, we create $\frac{1}{2p} \cdot \frac{c}{2p} = \frac{c}{4p^2}$. They are uniformly distributed in the range $\left[-\frac{Rp}{2}, \frac{Rp}{2}\right)$ because all the numbers in B_ℓ are uniformly distributed in the range $\left[-a_\ell - \frac{Rp}{2}, -a_\ell + \frac{Rp}{2}\right)$, and they are independent because all the B_ℓ are disjoint. By the union bound, the probability of having $\frac{c}{4p^2}$ elements is greater than

$$1 - \frac{1}{2p}e^{-\frac{c}{3}} - \frac{1}{2p}e^{-\frac{c}{2}} > 1 - \frac{1}{p}e^{-\frac{c}{3}} > 1 - e^{-\frac{c}{4}}$$

The last inequality follows from the assumption that $e^{-\frac{c}{12}} < p$. $\square$

The algorithm **MergeLists(L_1, L_2)** takes two lists each containing $\frac{c}{p^2}$ numbers and returns a list L_3 with the properties described in the statement of Lemma 14 can be described as in Algorithm 2.

Algorithm 2 MergeLists(L_1, L_2)

```
1: Sort $L_1$, $L_2$
2: for $\ell = 0$ to $\frac{1}{2p}$ do
3:     Pick an $a_\ell$ from $L_1$ that's in the interval $A_\ell$
4:     Define list $B_\ell$ to be elements from $L_2$ in the range $\left[-a_\ell - \frac{Rp}{2}, -a_\ell + \frac{Rp}{2}\right)$
5:     for each $b \in B_\ell$ do
6:         Add the element $a_\ell + b$ to the list $L_3$
```

The step that takes the longest time in the algorithm is the sorting of the lists. Thus, the MergeList algorithm requires $\widetilde{\mathcal{O}}\left(\frac{c}{p^2}\right)$ time.

We denote $I_i = \left[-\frac{N^{1-\frac{i}{\log k}}}{2}, \frac{N^{1-\frac{i}{\log k}}}{2}\right)$ for all $0 \leq i \leq \log k$, we have all the numbers in lists at level 0 are in the range I_0. The modified Wagner's algorithm can be described as follows.

Algorithm 3 Modified Wagner's algorithm over $\mathbb{Z}_N$

```
Input: $L_1, \dots, L_k \subset \mathbb{Z}_N, |L_i| = tN^{\frac{2}{\log k}}, t = poly(k)$ for all $i = 1, \dots, k$
Output: $(x_1, \dots, x_k) \in L_1 \times \cdots \times L_k$ such that $\sum_{i=1}^k x_i = 0 \mod N$
1: For all elements $x$ in $L_1, \dots, L_k$ if $(x \geq N/2)$ then $(x := x - N/2)$
2: for $i = 1$ to $\log k - 1$ do
3:     for $j = 1$ to $k$, $j = j \times 2^i$ do
4:         Compute $L_{j \dots j+2^i-1}$ by merging 2 lists $L_{j \dots j+2^{i-1}-1}$, $L_{j+2^{i-1} \dots j+2^i-1}$ ▷ use MergeLists algorithm
5: if $L_{1 \dots k} \neq \emptyset$ then return any element of $L_{1 \dots k}$
6: else return fail
```

Theorem 17. ***Algorithm 3*** *solves k-list problem over $\mathbb{Z}_N$ for $\left(8 < N^{\frac{1}{\log k}} < e^{\frac{t}{4k^2}}\right)$ with probability at least $1 - ke^{-\frac{t}{4k^2}}$ in time and memory is $\widetilde{\mathcal{O}}\left(ktN^{\frac{2}{\log k}}\right)$.*

Proof. The Algorithm 3 builds a tree of depth $\log k$ whose nodes are lists of numbers such that on the i^{th} level the list belong to I_i. In particular, the output list $L_{out} \subset I_{\log k} = \left[-\frac{1}{2}, \frac{1}{2}\right) \cap \mathbb{Z} = 0$. Having kept track of how each element is composed of elements from level 0, we have solved the problem (assuming the final list is nonempty) since we have found $x_1, \dots, x_k$ such that $x_1 + \cdots + x_k = 0 \mod N$. We will show that by induction that with probability at least $1 - ke^{-\frac{t}{4k^2}}$ for all $0 \le i \le \log k$ on level i, we have $\frac{k}{2^i}$ lists each containing $\frac{t}{4^i} N^{\frac{2}{\log k}}$ independently, uniformly distributed numbers in the range I_i

1. At $i = 0$, $L_1, \dots, L_k \subset I_0$ the distribution is guaranteed by the input
2. Assume that it is true for some level i where $0 \le i \le \log k$
3. We want to show that it is true for level $i + 1$

We pair up the $\frac{k}{2^i}$ list at level i and from each pair, create one list in the range I_{i+1}. So we have $\frac{k}{2^{i+1}}$ list with numbers in the range I_{i+1}. To prove that each list contains $\frac{t}{4^i} \cdot N^{\frac{2}{\log k}}$ independent, uniformly distributed elements in the range I_{i+1}. We use Lemma 14 with the following parameters:

$$R = N^{1 - \frac{i}{\log k}}, c = \frac{t}{4^i}, p = N^{-\frac{1}{\log k}}.$$

Thus, if we take two lists L_1 and L_2 at level i, by the inductive hypothesis they each have $\frac{c}{p^2}$ independent uniformly distributed elements in the range $\left[-\frac{R}{2}, \frac{R}{2}\right)$. By Lemma 14, we can create a new list L_3 with $\frac{c}{4p^2} = \frac{t}{4^{i+1}} \cdot N^{\frac{2}{\log k}}$ independent, uniformly distributed elements in the range

$$\left[-\frac{Rp}{2}, \frac{Rp}{2}\right) = \left[-\frac{N^{1-\frac{i+1}{\log k}}}{2}, \frac{N^{1-\frac{i+1}{\log k}}}{2}\right) = I_{i+1}$$

where each element is the sum of a number from L_1 and a numbers from L_2. The probability of success is at least $1 - e^{-\frac{c}{4}}$. Since $c = \frac{t}{4^i}$ and $i < \log k$, $c > \frac{t}{k^2}$ and so the probability of success for each list merge is at least $1 - e^{-\frac{t}{4k^2}}$. The total number of times that we combine lists during the whole algorithm is $k - 1$, so the probability that everything is successful is at least $1 - ke^{-\frac{t}{4k^2}}$.

Since we are combining lists of at most $tN^{\frac{2}{\log k}}$ elements and doing $k - 1$ list combines, so the total running time is $\widetilde{\mathcal{O}}\left(ktN^{\frac{2}{\log k}}\right)$. □

Corollary 18. *Given k independent lists each consisting of $tN^{\frac{2}{\log k}}$ independent uniformly distributed elements in the range $[0, N)$ and a target b, there exists an algorithm that with probability at least $1 - ke^{-\frac{t}{4k^2}}$ returns one element from each of the k lists such that the sum of k elements is $b \mod N$. The running time of this algorithm is $\widetilde{\mathcal{O}}\left(ktN^{\frac{2}{\log k}}\right)$.*

Proof. We subtract the target b from every element in the first list, then we apply algorithm 3. Since exactly one number from every list got used, it mean that $-b$ got used once. And since the sum is 0, we found an element from each list such that their sum is $b \mod N$. □

3 An algorithm for Dense Random Modular subset Sum (RMSS)

In this section, we present the algorithm introduced by [1] to solve Random Modular Subset Sum.

Definition 19. *(RMSS problem) Given a modulus N, and n numbers $a_1, \ldots, a_n$ that are independent, uniformly distributed in the range $[0, N)$, S is a target. Find a set $I \subset [n]$ such that $\sum_{i \in I} a_i = S \mod N$.*

Consider the following random variable X which defines a distribution on the range $[0, N)$:

$$X := \sum_{i=1}^{n} x_i a_i \mod N.$$

where $x_1, \ldots, x_n$ be n-bits, such that each x_i is drawn uniformly and independently from $\{0, 1\}$. We want to know when the distribution of random variable X is close to uniform distribution (U) in the range $[0, N)$, thus it is vital that we formalize what we mean by "close".

Definition 20. *Let X and Y be a random variables taking values in a probability space A. The statistical distance between X and Y, denoted $\Delta(X, Y)$, is*

$$\Delta(X, Y) = \frac{1}{2} \sum_{a \in A} |\Pr[X = a] - \Pr[Y = a]|$$

Proposition 21. *Let $X_1, \ldots, X_k$ and $Y_1, \ldots, Y_k$ be two lists of independent random variables. Then*

$$\Delta(X_1, \ldots, X_k), Y_1, \ldots, Y_k)) \leq \sum_{i=1}^{k} \Delta(X_i, Y_i)$$

The authors in [4] proved that if $N = 2^{cn}$ where $c < 1$, then for almost all choices of a_i, $\Delta(X, U)$ is small. This result is now known as the leftover hash lemma. Formally,

Proposition 22. *(Impagliazzo, Naor [4]) Let $N = 2^{cn}$ with $c < 1$. Then the probability over all choices for $(a_1, \ldots, a_n)$ that $\Delta(X, U) < 2^{-\frac{(1-c)n}{4}}$ is greater than $1 - 2^{-\frac{(1-c)n}{2}}$.*

Definition 23. *We call the list $(a_1, \ldots, a_n)$ is well-distributed if it is one of the good choices from Proposition 22, i.e, $\Delta(X, U) < 2^{-\frac{(1-c)n}{4}}$.*

The next lemma says that if we have a *well-distributed* list of a_i, then we can create another list by taking random subsets of the a_i, and the statistical distance of this new list from a list of uniformly distributed numbers will be small.

Lemma 24. *For a given n well-distributed numbers $a_1, \ldots, a_n$ modulo $N = 2^{cn}$, $c < 1$. Then*

$$\Delta((X_1, \ldots, X_m), (U_1, \ldots, U_m)) \leq m 2^{-\frac{(1-c)}{4}}$$

where the X_i are random variables distributed according X above and U_i are random variables distributed uniformly in the range $[0, N)$.

Proof. Apply Proposition 21. □

The idea for the algorithm in [1] will be to first take the n given numbers modulo $N = 2^{n^\epsilon}$ and construct $\frac{1}{2}n^{1-\epsilon}$ lists each containing $n^2 N^{\frac{2}{\log \frac{1}{2}n^{1-\epsilon}}}$ numbers that are almost independent and uniformly distributed in $[0, N)$. But we cannot do that directly because we don't have enough numbers. By Proposition 22 and Lemma 24, we can say that with high probability the lists are not too far from being uniformly distributed. Now that we have lists that are big enough, so we can apply Corollary 18 to find one number from each list such that the sum of the numbers is the target. And since every number in the list is some subset of the numbers in the group from which the list was generated, we have found a subset of the original n numbers which sums to the target S. The algorithm can be described as follows.

Algorithm 4 Algorithm for dense RMSS problem

```
Input: $a_1, \ldots, a_n, S$, $N = 2^{n^\epsilon}$ for $\epsilon < 1$
Output: $I \subset [n]$ such that $\sum_{i \in I} a_i = S \mod N$
1: Divide all the $a_i$'s into $\frac{1}{2}n^{1-\epsilon}$ groups each containing $2n^\epsilon$ numbers $(a_1, \ldots, a_{2n^\epsilon}), \ldots, (a_{n-2n^\epsilon+1}, \ldots, a_n)$
2: for group $i = 1$ to $\frac{1}{2}n^{1-\epsilon}$ do
3:     Construct $L_i = \left\{\sum_{j=0}^{2n^\epsilon} a_{i \cdot 2n^\epsilon + j - 1} x_j \text{ for } x \overset{iid}{\leftarrow} \{0,1\}^{2n^\epsilon}\right\}$ of size $|L_i| = n^2 N^{\frac{2}{\log \frac{1}{2}n^{1-\epsilon}}}$
4: Subtract $S$ from any list $L_i$
5: Run algorithm 3 on $L_1, \ldots, L_{\frac{1}{2}n^{1-\epsilon}}$ with parameter $k = \frac{1}{2}n^{1-\epsilon}, t = n^2$
```

Theorem 25. *Given n numbers $a_1, \ldots, a_n$, that independent and uniformly distributed in the range $[0, N)$ where $N = 2^{n^\epsilon}$ for some $\epsilon < 1$, and a target S. **Algorithm 4** will output a solution for RMSS problem with probability at least $1 - 2^{-\Omega(n^\epsilon)}$ and it's requires $\widetilde{\mathcal{O}}\left(2^{\frac{n^\epsilon}{\log n}}\right)$ time.*

Proof. If we assume for second that the lists $L_1, \ldots, L_{\frac{1}{2}n^{1-\epsilon}}$ in line (5) consist of independent, uniformly distributed numbers, then we are applying the algorithm from Corollary 18, so with probability

$$1 - ke^{-\frac{t}{4k^2}} = 1 - \frac{1}{2}n^{1-\epsilon}e^{-\frac{n^2}{4(\frac{1}{2}n^{1-\epsilon})}} = 1 - e^{-\Omega\left(n^{2\epsilon}\right)}$$

the algorithm will return one element from each list such that the sum of elements is $S \mod N$. And this gives us the solution to the subset sum instance. The running time of the algorithm is $\widetilde{\mathcal{O}}\left(\left(\frac{1}{2}n^{1-\epsilon}n^2 N^{\frac{2}{\log \frac{1}{2}n^{1-\epsilon}}}\right)^2\right) = \widetilde{\mathcal{O}}\left(2^{\frac{n^\epsilon}{\log n}}\right)$.

The problem is that each list is not generated uniformly and independently from the interval $[0, N)$. We will first observe that with high probability, the a_i's in every group are *well-distributed.* The Proposition 22 implies that with probability at least $1 - 2^{-\frac{n^\epsilon}{4}}$ the numbers in any one group are *well-distributed*, and since there are $\frac{1}{2}n^{1-\epsilon}$ groups, the probability that all the groups are *well-distributed* is at least

$$1 - \frac{1}{2}n^{1-\epsilon}2^{-\frac{n^\epsilon}{4}} = 1 - 2^{-\Omega(n^\epsilon)}$$

Suppose that the numbers in each group are *well-distributed*, we can apply Lemma 24, then the statistical distance between the $\frac{1}{2}n^{1-\epsilon}n^2 2^{\frac{2n^\epsilon}{\log \frac{1}{2}n^{1-\epsilon}}}$ numbers generated by Algorithm 4 and numbers generated from the uniform distribution is at most

$$\frac{1}{2}n^{1-\epsilon}n^2 2^{\frac{2n^\epsilon}{\log \frac{1}{2}n^{1-\epsilon}}} 2^{-\frac{2n^\epsilon}{8}} = 2^{-\Omega(n^\epsilon)}$$

for all $\epsilon < 1$. The probability that the algorithm from Corollary 18 succeeds on this input is at most $2^{-\Omega(n^\epsilon)}$ less than the probability that it succeeds on the uniformly generated input. Thus the probability of success is at least

$$1 - e^{-\Omega(2n^\epsilon)} - 2^{-\Omega(n^\epsilon)} = 1 - 2^{-\Omega(n^\epsilon)}$$

So to summarize, the probability that every group is *well-distributed* is $1 - 2^{-\Omega(n^\epsilon)}$ and the probability that the algorithm succeeds given that all the groups are *well-distributed* is $1 - 2^{-\Omega(n^\epsilon)}$. Thus the probability that Algorithm 4 succeeds is at least $1 - 1 - 2 \cdot 2^{-\Omega(n^\epsilon)} = 1 - 1 - 2^{-\Omega(n^\epsilon)}$. $\square$

References

[1] Vadim Lyubashevsky. On Random High Density Subset Sums. *Electronic Colloquium on Computational Complexity (ECCC)*, 2005.

[2] David Wagner. A Generalized Birthday Problem. *CRYPTO 2002, LNCS*, Springer-Verlag, pp. 288–303

[3] Shallue Andrew. An Improved Multi-set Algorithm for the Dense Subset Sum Problem. *Algorithmic Number Theory*, Springer Berlin Heidelberg (2008), pp. 416–429

[4] Russell Impagliazzo, Moni Naor. Efficient Cryptographic Schemes Provably as Secure as Subset Sum. *Journal of Cryptology Volume 9,Number 4*, 1996, pp. 199-216