# Cryptanalysis of a Generalized Subset-Sum Pseudorandom Generator

**Charles Bouillaguet**
Sorbonne Université, CNRS, LIP6, F-75005 Paris, France

**Florette Martinez**
Sorbonne Université, CNRS, LIP6, F-75005 Paris, France

**Damien Vergnaud**
Sorbonne Université, CNRS, LIP6, F-75005 Paris, France

**Abstract**

We present attacks on a generalized subset-sum pseudorandom generator, which was proposed by von zur Gathen and Shparlinski in 2004. Our attacks rely on a sub-quadratic algorithm for solving a vectorial variant of the 3SUM problem, which is of independent interest. The attacks presented have complexities well below the brute-force attack, making the generators vulnerable. We provide a thorough analysis of the attacks and their complexities and demonstrate their practicality through implementations and experiments.

**2012 ACM Subject Classification** Security and privacy → Cryptography

**Keywords and phrases** Cryptography, pseudo-random generator, subset-sum problem, 3SUM problem, cryptanalysis

**Digital Object Identifier** 10.4230/LIPIcs.MFCS.2023.23

**Supplementary Material** *Software*: `https://github.com/floretteM/Knapsack`
archived at `swh:1:dir:dd38a4a2c04ee30599a006d1882aa1975d4fc778`

**Funding** This work was supported in part by the French Agence Nationale de la Recherche under project "GORILLA" (ANR-20-CE39-0002). Preliminary computational experiments were carried out using the Grid'5000 testbed, supported by a scientific interest group hosted by Inria and including CNRS, RENATER and several Universities as well as other organizations (see `https://www.grid5000.fr`).

**Acknowledgements** The authors are grateful to the anonymous reviewers for their insightful comments and valuable suggestions.

## 1 Introduction

A pseudo-random number generator is a deterministic algorithm that runs in polynomial time using a short random seed as its input, and produces a long sequence that is indistinguishable from a truly random sequence in polynomial time. The versatile applications of pseudo-random numbers have been extensively explored in the literature, particularly in cryptography where they are employed for tasks such as key generation, encryption, and digital signatures.

In 1985, Rueppel and Massey introduced the *knapsack generator* (or *subset sum generator*) [16] whose security ultimately relies on the NP-hard *modular subset sum problem*: given integers $\omega_0, \dots, \omega_{n-1}$, $t$ and $q$, find a subset of the $\omega_i$'s that sum to $t$ modulo $q$, i.e. to find bits $x_0, \dots, x_{n-1} \in \{0, 1\}$ such that

$$\sum_{i=0}^{n-1} x_i \omega_i = t \bmod q.$$

In the *knapsack generator*, the modulus $q$ is usually taken as a power of 2, $q = 2^n$, the *weights* $\omega_0, \dots, \omega_{n-1}$ are kept secret and given $n$ secret *control bits* $u_0, \dots, u_{n-1}$, one extends them using a linear feedback generator (a fast but non-cryptographically secure pseudo-random number generator) to obtain a flow of pseudo-random bits $(u_i)_{0 \leq i \leq N+n-2}$. For $i \in \{0, \dots, N-1\}$, one then computes

$$v_i = \sum_{j=0}^{n-1} u_{i+j}\omega_j \bmod 2^n$$

and outputs $y_i$ which are the $\rho = n - \ell$ leading bits of $v_i$ where $\ell$ is a given parameter.

In 2011, Knellwolf and Meier [12] presented a cryptanalysis of this generator. They used a *guess-and-determine* strategy coupled with lattice-based techniques to recover most of the key in relevant instances of the generator. In order to run said attack, they needed to guess all the $n$ initial control bits. Hence their attack had a time complexity $\Omega(2^n)$. In 2009, Von zur Gathen and Shparlinski [20] presented the *fast knapsack generator* that had a far smaller key and was sensibly faster but had not undergone a serious cryptanalysis until recently [14]. We consider another variant of the subset sum pseudorandom generator, suggested by von zur Gathen and Shparlinski in 2004 [19].

The family of generators proposed by von zur Gathen and Shparlinski can be described in an abstract way using two integer parameters $\lambda$ and $n$ and three independent components:

- a control-sequence generator $\mathsf{CSG} : \{0,1\}^\lambda \times \mathbb{N} \to \{0,1\}^n$;
- an Abelian cyclic group $(\mathbb{G}, +)$ of order $q$ where the group law is denoted additively;
- a deterministic and public conversion function $\Psi : \mathbb{G} \to \{0,1\}^\rho$ where $\rho$ denotes the output length of the pseudo-random generator.

The seed of this generalized subset-sum generator consists in a bit-string $\mathsf{seed}_0 \in \{0,1\}^\lambda$ and $n$ group elements $P_1, \dots, P_n \in \mathbb{G}$. The bit size of the seed is thus equal to $\lambda + n \cdot \lceil \log_2(q) \rceil$. At each iteration $i \in \mathbb{N}$, the control-sequence generator generates an $n$-bit string $\mathbf{v_i} = (v_i^1, \dots, v_i^n) = \mathsf{CSG}(\mathsf{seed}_0, i)$, computes the group element $Q_i$ defined by

$$Q_i = [v_i^1]P_1 + \cdots + [v_i^n]P_n \in \mathbb{G}$$

and outputs $s_i = \Psi(Q_i) \in \{0,1\}^\rho$.

In the Rueppel-Massey classical subset sum generator, the group $\mathbb{G}$ is thus the group of modular residue $\mathbb{G} = \mathbb{Z}_q$, the control-sequence generator is defined by a linear feedback shift register and the conversion function is a truncation. In [19], von zur Gathen and Shparlinski proposed to use for $\mathbb{G}$ the group of rational points of an elliptic curve defined over a (prime) finite field, a linear feedback shift register as the control-sequence generator and again a truncation for the conversion function (more precisely, truncation of the x-coordinate of the elliptic curve point $Q_i$). They proposed to use $\lambda = n$ and an elliptic curve defined over a finite field $\mathbb{Z}_p$ where $p$ is a $n$-bit prime number. By the Hasse-Weil theorem [9], the number of group elements $q$ is around $2^n$ and the total seed size is $\simeq n + n \cdot n = n \cdot (n+1)$. They suggested that $\Psi$ should discard $\ell = \log_2(n)$ low-order bits of the $x$-coordinate of the point before using it as pseudo-random output.

Von zur Gathen and Shparlinski claimed that: “the only available attack on this generator is the brute force search over all parameters defining this generator” and thus using $n$ as small as 12 should provide a 128-bit security level. The statistical properties of the sequences generated by this pseudo-random generator were analyzed in [4, 1, 8] but its cryptographic security has not been studied up to the present article. We present a simple attack against this generator and a lattice-based attack on another variant derived from this abstraction. In the instantiation suggested by von zur Gathen and Shparlinski, our attack has complexity $\mathcal{O}\left(2^{1.78n}\right)$ well below the $\mathcal{O}\left(2^{n(n+1)}\right)$ brute-force attack. Our attacks rely on a sub-quadratic algorithm for solving a vectorial variant of the 3SUM problem, which is of independent interest. We provide a thorough analysis of the attacks and their complexities, and demonstrate their practicality through implementations and experiments.

## 2 High-level description of the attack

We consider the case where the control sequence generated by the CSG is known by the adversary. If this is not the case, they can simply try all possible values for $\mathsf{seed}_0 \in \{0,1\}^\lambda$ which increases the complexity of the attack by a multiplicative factor $2^\lambda$.

We assume that the control sequence generator outputs uniform and independent $n$-bit strings $\mathbf{v_i} = \mathsf{CSG}(\mathsf{seed}_0, i)$ for each $i \in \mathbb{N}$. Note that in the concrete schemes that we attack, this is obviously false; we nevertheless carry our analysis under this assumption and our experimental results will show that it actually holds in practice.

Let us suppose that an adversary finds three indices $i, j, k$ such that $\mathbf{v_i} + \mathbf{v_j} = \mathbf{v_k}$ as vectors of integers, i.e. where the addition is performed coordinate-wise over $\mathbb{Z}$. In this case, the adversary knows that the relation $Q_i + Q_j = Q_k$ holds in the group $\mathbb{G}$. They are not given the actual values of the points $Q_i, Q_j$ and $Q_k$ but only the values $\Psi(Q_i)$, $\Psi(Q_j)$ and $\Psi(Q_k)$. Assume that the number of preimages through $\Psi$ is limited and that the adversary can efficiently compute them; they can simply check if $\Psi(X+Y) = \Psi(Q_k)$ for all $(X, Y)$ such that $\Psi(X) = \Psi(Q_i)$ and $\Psi(Y) = \Psi(Q_j)$. If there exists only one such pair $(X, Y)$ then the adversary can safely assume that $Q_i = X$, $Q_j = Y$ (and $Q_k = X + Y$).

The number of pairs $(X, Y)$'s such that

$$\Psi(X+Y) = \Psi(Q_k) \tag{1}$$

is difficult to estimate and depends heavily on the group $\mathbb{G}$ and the conversion function $\Psi$. In [17], Shoup studied the computational complexity of the discrete logarithm in Abelian groups in the context of algorithms which do not exploit any special properties of the encodings of group elements. Shoup introduced the *generic group model* where each group element is encoded as a unique and arbitrary binary string (picked uniformly at random and independent of the actual group structure). As a consequence, it is not possible for an algorithm in this model to exploit any special properties of the encodings and group elements can only be operated on using an oracle that provides access to the group operations. If we make a similar assumption on the group $\mathbb{G}$ and the conversion function $\Psi$ is a truncation of $\ell$ bits out of the $(\log_2 q)$-bit encodings of group elements, then we can expect that the number of preimages is close to $2^\ell$ and the number of pairs $(X, Y)$ different from $(Q_i, Q_j)$ for which (1) holds is expected to be

$$2^\ell \cdot 2^\ell / 2^{\log_2(q)-\ell} \simeq 2^{3\ell}/q. \tag{2}$$

In particular if $\rho > 2 \cdot \log_2(q)/3$, one expects the number of candidates for $(Q_i, Q_j, Q_k)$ to be constant in a "generic" group. It is worth mentioning that this assumption does not hold in the classical knapsack generator that uses the group $\mathbb{G} = \mathbb{Z}_m$ since in this case, the number of candidates for a single equation will be about $2^{2\ell}$.

Each relation $\mathbf{v_i} + \mathbf{v_j} = \mathbf{v_k}$ gives two relations in the group $\mathbb{G}$:

$$Q_i = v_i^1 P_1 + \cdots + v_i^n P_n \qquad \text{and} \qquad Q_j = v_j^1 P_1 + \cdots + v_j^n P_n$$

If the adversary collect sufficiently many linearly independent such relations, they would be able to retrieve all the weights used in the generalized knapsack generator.

**Table 1** Tabulating all solutions of $x + y = z$ for $x, y, z \in \{0, 1\}$.

| $x$ | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
|---|---|---|---|---|---|---|---|---|
| $y$ | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| $z$ | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 |
| $x+y$ | 0 | 0 | 1 | 1 | 1 | 1 | 2 | 2 |

In the following, we describe and analyse an algorithm to find "good triplets" of indices $(i, j, k)$ such that $\mathbf{v_i} + \mathbf{v_j} = \mathbf{v_k}$ and show how to use it to attack the elliptic knapsack generator when $\rho = n - \log_2(n)$ (as suggested by von zur Gathen and Shparlinski).

## 3 Finding "Good Triplets"

Assume that three lists $A, B$, and $C$, each of size $N$, are made of uniformly random $n$-bit strings. Let $Y$ be the random variable that counts the number of triplets $(x, y, z) \in A \times B \times C$ such that $x + y = z$ when $x, y$ and $z$ are seen over $\mathbb{Z}^n$ and not modulo 2. When this relation holds, we call $(x, y, z)$ a "good triplet". Our goals in this section are twofold: 1) lower-bound the probability that $A, B$ and $C$ contain a good triplet and 2) design an algorithm to find good triplets efficiently.

As a warm-up, examining the simplest case ($n = 1$) is interesting (cf. Table 1). Looking at this table, we see that $\Pr(x + y = z) = 3/8$. We next prove the following

▶ **Theorem 1.** $\mathrm{E}(Y) = N^3 \left(\frac{3}{8}\right)^n$, *and* $\Pr(Y = 0) \leq \frac{1}{N^3}\left(\frac{8}{3}\right)^n + \frac{3}{N}\left(\frac{10}{9}\right)^n + \frac{3}{N^2}\left(\frac{4}{3}\right)^n$.

The proof is given in Appendix A. It boils down to estimating the variance of $Y$ and using the second-moment inequality. With $N = \alpha(8/3)^{n/3}$, Theorem 1 yields:

$$\Pr(Y = 0) \leq \frac{1}{\alpha^3} + \frac{3}{\alpha}(0.801...)^n + \frac{3}{\alpha^2}(0.69...)^n.$$

Therefore, setting $\alpha = 10$ is sufficient to ensure that a good triplet exists with probability 99.9%. In addition, it follows from the theorem that

$$\Pr(Y = 0) \leq \frac{1}{\mathrm{E}(Y)} + \frac{3}{(\mathrm{E}(Y))^{2/3}} + \frac{3}{(\mathrm{E}(Y))^{1/3}}. \tag{3}$$

### 3.1 A Simple Sub-Quadratic Algorithm to Find Good Triplets

Finding a "good triplet" (such that $x + y = z$) can be done using a naive quadratic algorithm: for all pairs $(x, y)$ in $A \times B$, check if $x + y \in C$; if so, return $(x, y, x + y)$ ; after this loop, return $\perp$ (to handle the case where the algorithm fails). This could potentially be sped up a little by exploiting the fact that $x$ and $y$ are necessarily disjoint.

In this section, we present a simple algorithm to find a good triplet more efficiently. We work under the assumption that the input lists have size $N := \alpha(8/3)^{n/3}$ for some constant $\alpha \geq 4$. Under this condition, (3) ensures that there is a good triplet with probability at least $\frac{3}{64}$. This assumption will be relaxed in the next section.

Looking again at Table 1, we see that $\Pr(x = 1 \mid x + y = z) = 1/3$ while $\Pr(z = 1 \mid x + y = z) = 2/3$. In other terms, even though $x, y, z$ are sampled uniformly at random, if we restrict our attention to good triplets, then $x$ and $y$ are biased towards zero (sparse) while $z$ is biased towards 1 (dense).

This observation suggests an algorithm to find good triplets efficiently: remove from $A, B$ (resp. $C$) input vectors of Hamming weight different from $n/3$ (resp. $2n/3$), then run the naive quadratic algorithm on what remains.

▶ **Theorem 2.** *This algorithm terminates in $\mathcal{O}\left(N^e\right)$ with $e = 2\ln(9/4)/\ln(8/3) \approx 1.654$ and succeeds with probability $\Omega\left(\frac{1}{n}\right)$.*

**Proof.** Let $H$ denote the binary entropy function, meaning that $H(x) = -x\log_2(x) - (1-x)\log_2(1-x)$, for all $0 < x < 1$. The following standard bounds for the binomial coefficient can be derived from Stirling's formula:

$$\frac{2^{nH(x)}}{\sqrt{8nx(1-x)}} \leq \binom{n}{xn} \leq \frac{2^{nH(x)}}{\sqrt{2\pi nx(1-x)}}, \qquad (0 < x < 1/2) \tag{4}$$

It follows from the discussion juste before the statement of the theorem that there are $3^n$ good triplets on $n$ bits (out of $8^n$ triplets in total). The number of good triplets that satisfy the weight condition imposed by the algorithm is

$$N = \binom{n}{n/3, n/3, n/3} = \binom{n}{2n/3}\binom{2n/3}{n/3} \geq \frac{2^{nH(2/3)}}{\sqrt{n4/3}}\frac{2^{2n/3}}{2\sqrt{n/3}} = \frac{3\sqrt{3}}{8n}3^n.$$

If the input list contain a good triplet, then the algorithm described above returns it with probability greater than $0.65/n$. The claimed time complexity is in fact a consequence of the *next* theorem (Theorem 5), and we will therefore not prove it here. ◀

## 3.2 Sub-Quadratic Algorithm with Overwhelming Success Probability

We generalize the algorithm of the previous section by relaxing the weight condition. This yields Algorithm 1. It takes an additional argument $w$ controlling the maximum allowed weight. In the sequel, all the stated complexities must be understood "up to a constant factor". Let $\epsilon$ denote a constant in the open interval $\left(0; \frac{1}{6}\right)$. Let $X \sim \mathcal{B}(n,p)$ be a binomial random variable. We will use the classical inequality (5) given below, a proof of which can be found in [2] amongst others. Here, $D(a,p)$ is the Kullback-Leibler divergence between an $a$-coin and a $p$-coin:

$$\begin{aligned} &\Pr(X \leq an) \leq \exp(-nD(a,p)) && \text{if } a < p. \\ &\Pr(X \geq an) \leq \exp(-nD(a,p)) && \text{if } a > p, \\ &\quad D(a,p) = a\ln\frac{a}{p} + (1-a)\ln\frac{1-a}{1-p}. \end{aligned} \tag{5}$$

We denote by $\mathrm{wt}(x)$ the Hamming weight of a bit string $x$.

**Algorithm 1** Find good triplets.

```
1: function FindTriplet($A, B, C, w$)
2:     $A' \leftarrow \{x \in A \mid \mathrm{wt}(x) \leq w\}$
3:     $B' \leftarrow \{y \in B \mid \mathrm{wt}(y) \leq w\}$
4:     for all $x, y \in A' \times B'$ do
5:         if $x + y \in C$ then
6:             return $(x, y, z)$
7:     return $\perp$
```

▶ **Lemma 3.** *With $w = n\left(\frac{1}{3} + \epsilon\right)$, if the input contains a good triplet, then Algorithm 1 returns $\perp$ with probability less than $2\exp(-2n\epsilon^2)$.*

**Proof.** Assume that the input lists contain a good triplet $(x^*, y^*, z^*)$. It will be discarded if and only if the weight of either $x^*, y^*$ is greater than $w$. We know that the weight of $x^*$ and $y^*$ follows a binomial distribution of parameters $(n, 1/3)$, therefore (5) shows that either has weight greater than $n(1/3 + \epsilon)$ with probability less than $\exp(-nD(1/3 + \epsilon, 1/3))$.

The ("well-known") fact that $D(p + \epsilon, p) \geq 2\epsilon^2$ combined with union bound (for $x^*$ and $y^*$) then yields the announced result. ◀

▶ **Lemma 4.** *Let $T$ denote the running time of Algorithm 1 with $w = n\left(\frac{1}{3} + \epsilon\right)$. Then $\mathrm{E}\left[T\right] \leq N + N^2 \exp\left[-2nD\left(\frac{1}{3} + \epsilon, \frac{1}{2}\right)\right]$.*

**Proof.** Filtering the input lists and keeping only low-weight vectors can be done in linear time. Given the complexity of the naive quadratic algorithm, the total time complexity is simply $T = N + |A'| \cdot |B'|$.

Let $X \sim \mathcal{B}(n, 1/2)$ be a binomial random variable modeling the weight of a random $n$-bit vector. Such a vector belongs to $A'$ or $B'$ if its weight is less than or equal to $w$, and this happens with probability $s := \Pr(X \leq w)$. The binomial tail bound (5) yields $s \leq \exp\left[-nD\left(\frac{1}{3} + \epsilon, \frac{1}{2}\right)\right]$.

The sizes of $A'$ and $B'$ are stochastically independent random variables following a binomial distribution of parameters $(N, s)$ with expectation $Ns$. The expected running time of the quadratic algorithm on $A'$ and $B'$ is therefore $\mathrm{E}\left(|A'| \times |B'|\right) = \mathrm{E}\,|A'| \times \mathrm{E}\,|B'| = N^2 s^2$. Combining this with the upper bound on $s$ gives the announced result. ◀

▶ **Theorem 5.** *Write $e = 2 \cdot \dfrac{\ln(9/4)}{\ln(8/3)} \approx 1.654$ For all $d > e$ there is an algorithm that runs in time $\mathcal{O}\left(N^d\right)$, where $N$ denotes the size of the input list and fails to reveal a good triplet present in the input with negligible probability (in $n$).*

**Proof.** Let $e < d < 2$ be a complexity exponent greater than the bound $e$ given in the statement of the theorem. There always exist $\epsilon > 0$ such that

$$d = 2 - 6\frac{D\left(\frac{1}{3} + \epsilon, \frac{1}{2}\right)}{\frac{1}{3}\ln\frac{8}{3} + \epsilon \ln 2}.$$

Indeed, setting $\epsilon = 0$ in this expression yields the lower-bound exponent $e$ of the theorem, and the expression of $d$ is increasing as a function of $\epsilon$; it reaches $d = 2$ for $\epsilon = 1/6$.

Let $N_0 := (8/3)^{n/3}$, so that input lists of size $N_0$ contain a single good triplet in average. We distinguish two cases depending of the size of the input lists.

Suppose that $N \leq 2^{\epsilon n} N_0$, where $N$ denotes the size of the input lists. In this case run Algorithm 1 with $w = n\left(\frac{1}{3} + \epsilon\right)$. Lemma 3 guarantees the exponentially small failure probability while lemma 4 tells us that the expected running time $T$ is less than $N + N^2 \exp[-2nD\left(\frac{1}{3} + \epsilon, \frac{1}{2}\right)]$.

A quick calculation shows that the algorithm then runs in time $\mathcal{O}\left(N^d\right)$ – the value of $d$ has been chosen for this purpose. The theorem is proved in this case.

If $N > 2^{n\epsilon} N_0$, then slice the input lists in chunks of size $4N_0$ and run Algorithm 1 with $w = n/3$ on each successive chunk until a solution is found. Each chunk contains a good triplet with probability at least $\frac{3}{64}$ thanks to (3). The algorithm reveals this triplet, if it exists, with probability $\Omega\left(\frac{1}{n}\right)$, because it always works if the algorithm of the previous section works.

There are $2^{\epsilon n}/4$ chunks (i.e., exponentially many). Because the chunks are disjoint parts of the input lists, success in a chunk is independent from the others. Therefore the probability that this process fails to reveal a good triplet is negligible. The running time of this procedure is $\mathcal{O}\left(NN_0^{e-1}\right)$. Because $N_0 \le N$, this is less than $\mathcal{O}\left(N^e\right)$. ◀

▶ Remark 6. $A_b, B_b$ and $C_b$ the subsets of strings of $A$, $B$ and $C$ whose first bit is equal to $b$ (for $b \in \{0,1\}$), then a good triplet necessarily belongs to one of the three sets $A_0 \times B_0 \times C_0$, $A_0 \times B_1 \times C_1$ or $A_1 \times B_0 \times C_1$ (and the search for a good triplet in $A \times B \times C$ thus reduces to the search in those three sets). The expected cardinality of $A_0, A_1, B_0, B_1, C_0$ and $C_1$ is $N/2$ and applying this idea recursively, one obtains (assuming the division is always done in a "balanced" manner) a time complexity $T(N)$ which heuristically satisfies the recursion $T(N) = 3T(N/2) + O(N)$ (and thus $T(N) = O(N^{\log_2(3)}) = O(N^{1.59})$). This improves a bit the complexity of our algorithm, but this "divide-and-conquer" approach is probably more complex to implement. It would be interesting to see if one can combine this approach with our filtering technique.

## 4 Practical Key-recovery Attack on the von zur Gathen-Shparlinski Elliptic Subset Sum Generator

In this section, we consider the instantiation of the knapsack generator suggested by von zur Gathen and Shparlinski in [19]. In particular, the group $\mathbb{G}$ is an elliptic curve $E$ defined over a (prime) finite field $\mathbb{F}_p$ (where $p \ge 5$ is an $n$-bit prime number). It is a rational curve given by the following Weierstrass equation

$$E : y^2 = x^3 + ax + b$$

for some $a, b \in \mathbb{F}_p$ with $4a^3 + 27b^2 \ne 0$. It is well known that the set $E(\mathbb{F}_p)$ of $\mathbb{F}_p$-rational points (including the special point $O$ at infinity) forms an Abelian group with an appropriate composition rule (denoted additively) where $O$ is the neutral element – for more details on elliptic curves, we refer to [5, 21]. Von zur Gathen and Shparlinski suggested to use a conversion function $\Psi : E \to \{0,1\}^\rho$ that simply truncates $\ell = \log_2(n)$ least significant bits of the x-coordinate of a point (with $\rho = n - \ell$). An $n$-bit linear feedback shift register is used as the control-sequence generator (as in the Rueppel-Massy classical knapsack generator) and the overall seed length is thus $n(n+1)$ bits.

### 4.1 Attack on the Elliptic Subset Sum Generator

The adversary first "guesses" $\mathsf{seed}_0$. In other terms, all subsequent steps have to be repeated $2^n$ times, one for each possible value of $\mathsf{seed}_0$.

Following the analysis from Section 3, one needs to construct three sets $A$, $B$, $C$ of independent $n$-bit strings size $N = \alpha(8/3)^{n/3} \le \alpha 2^{0.472n}$ in order to find a good triplet $(i,j,k)$ such that $\mathbf{v_i} + \mathbf{v_j} = \mathbf{v_k}$ time $\mathcal{O}\left(N^{1.654\ldots}\right) = \mathcal{O}\left(2^{0.78n}\right)$ with probability at least $1 - 7/\alpha$. We need to have $n/2$ such good triplets in order to find the $n$ points $P_1, \ldots, P_n$ used as weights in this elliptic knapsack generator, and we can hope to obtain them with constant positive probability from an output sequence made of $\Omega\left(n^2(8/3)^{n/3}\right)$ output values in $\{0,1\}^\rho$. In our implementation, we do not distinguish the sets $A$, $B$, and $C$ and simply run the algorithm from the previous section with $A = B = C$ the sets of all vectors $\mathbf{v_i}$ corresponding to all known outputs $s_i \in \{0,1\}^\rho$.

Note that, as in the classical knapsack generator, the control sequence is not made of independent $n$-bit strings since if one denotes $(u_n)_{n\ge 0}$ the sequence output by the linear feedback shift register, we have

$$\mathbf{v_i} = (v_i^1, \ldots, v_i^n) = (u_i, u_{i+1}, \ldots, u_{i+n-1}) \in \{0,1\}^n$$

for $i \in \mathbb{N}$. The analysis given in Section 3 does not apply to such sequences but we make the heuristic assumption that these $n$-bit tuples are "sufficiently" random and that our algorithm will succeed with a similar probability (this heuristic is validated by our experiments).

We then follow the general idea given above but for each good triplet $(i, j, k)$ such that $\mathbf{v_i} + \mathbf{v_j} = \mathbf{v_k}$, if the adversary finds two points $X$ and $Y$ on the elliptic curve such that $\Psi(X) = s_i$, $\Psi(Y) = s_j$ and $\Psi(X + Y) = s_k$, then this gives rise to two possible relations:

1. $X = Q_i$, $Y = Q_j$ (and $X + Y = Q_i + Q_j = Q_k$), but also
2. $X = -Q_i$, $Y = -Q_j$ (and $X + Y = -(Q_i + Q_j) = -Q_k$).

This is due to the fact that on an elliptic curve, a point and its negative have representations with much in common since they share the same the x-coordinate (and the y-coordinates are opposites). This "non-genericness" of elliptic curves is well-known and has important consequences in cryptography[1]. However, with a truncation of $\log_2(n)$ bits of the x-coordinate of the points, we expect the number of points triple compatible with $(s_i, s_j, s_k)$ to be equal to only 2.

Note that for the first good triplet computed in the attack, the sign is not a problem since the generator parametrized with the $n$ points $P_1, \dots, P_n$ outputs the same sequence as the one parametrized with the $n$ points $-P_1, \dots, -P_n$. The adversary can then pick up arbitrarily $(Q_i, Q_j) = (X, Y)$ or $(Q_i, Q_j) = (-X, -Y)$. However, for the subsequent relations obtained from other good triplets, the sign choice may be incompatible with the first one and this will result in a system with no solutions. In order to be able to solve the system, we need to have $n$ linear relations among the points $P_1, \dots, P_n$ and each good triplet gives us two such relations (the third one is by construction is a linear combination of the two others and is useless in solving the linear system). Assuming that $n$ is even, one needs to make $n/2 - 1$ choices for the sign of each relation (after the first one), and the adversary can simply "guess" all such signs. This multiplies the running time of the algorithm by a factor $2^{n/2-1}$.

Once the $n/2$ good triplets have been found, the algorithm can inverse the system and obtain the $n$ points $P_1, \dots, P_n$. From these values, the points $P_1, \dots, P_n$. The overall complexity of the attack is thus

$$\mathcal{O}\left(2^n(2^{0.78n} + 2^{2\log(n)} + 2^{n/2-1}) + \mathsf{poly}(n)\right) = \mathcal{O}\left(2^{1.78n}\right)$$

binary operations.

## 4.2 Experimental Results

We implement our attack using sagemath v.9.5 on a laptop. Our codes are available at `https://github.com/floretteM/Knapsack`. In our implementation, we did not need the exhaustive search on the signs mentioned above since we instead looked for good triplets that involve a point already found (and this sets up the sign with certainty). This makes the probability of finding such good triplets more complex to analyse but this trick works well in practice.

We first consider the elliptic curve defined by the equation $y^2 = x^3 + 5x + 5$ over $\mathbb{F}_p$ where $p = 2^{16} - 15$. This curve contains $q = 65111$ points. We present the attack when the control sequence $(\mathbf{v_i})$ is known and we consider $n = 16$ as suggested by von zur Gathen and Shparlinski. The key size in this setting is equal to 256 bits. We present in Table 2 the number $m$ of outputs needed and the time necessary to recover the secret weights with probability at least 50% when $\ell$ bits are missing.

[1] For instance, the ECDSA signature scheme is malleable in the sense that if the pair of integers $(r, s)$ is

**Table 2** Key-recovery with exhaustive search and $q$ a 16-bit integer.

| $\ell$ | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| $m$ | 1000 | 1000 | 1000 | 1000 | 1000 | 1885 |
| time | $6.9s$ | $5.3s$ | $5.6s$ | $5.02s$ | $5.7s$ | $26.7s$ |

When 7 bits are truncated we cannot recover the weights even with 3000 outputs. But we earlier observed that the algorithm would not work well if $\ell > \log_2(q)/3$, see (2). With the proposed choice of $\ell = \log_2 n = 5$, our results are coherent with the heuristic.

To test the limits of our attack, we also implement it for elliptic curves with larger group orders (*i.e.* for parameters larger than those suggested by von zur Gathen and Shparlinski). This gives an algorithm with overall complexity $\mathcal{O}\left(2^n(2^{0.78n} + 2^{n/2}) \cdot \mathsf{poly}(n, \log(p))\right)$.

We consider the elliptic curve defined by the equation $y^2 = x^3 + x + 14$ over $\mathbb{F}_p$ where $p = 2^{40} + 15$ but still $n = 16$. With this choice we can focus on recovering the points of the elliptic curves from the outputs without being to bothered by finding the good triplets. This curve contains $q = 1099510687747$ points. We present in Table 3 the number $m$ of outputs needed and the time necessary to recover the secret weights with probability at least 50% when $\ell$ bits are missing.

**Table 3** Key-recovery with exhaustive search and $q$ a 40-bit integer.

| $\ell$ | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|
| $m$ | 1885 | 1885 | 1885 | 1885 | 1885 | 1885 | 1885 | 1885 | 1750 |
| time | $2.1s$ | $2.1s$ | $2.08s$ | $2.5s$ | $2.6s$ | $2.1s$ | $3.5s$ | $8.3s$ | $26.7s$ |

## 5 Practical Key-recovery Attack on the Subset Product Generator

Following the generalization of the knapsack generator to elliptic curves proposed by von zur Gathen and Shparlinski, it is natural to consider other variants using Abelian groups of interest in cryptography. The most natural choice is to use (a subgroup of) the multiplicative group of a finite field $\mathbb{Z}_p$ for some prime number $p$. This group is certainly not generic since there exist sub-exponential time discrete logarithm algorithms in these groups, but it seems that representation of group elements by the unique member of its class in $\{0, \ldots, p-1\}$ is sufficiently "generic" that using truncation of their bit-representation as a conversion function would permit an adversary to mount a lattice-based attack on this generator even if a quarter of the bits of each group elements is discarded when computing the output of the generator.

More precisely, in this section, we consider a multiplicative variant of the subset sum generator where:

- the control-sequence generator is a linear feedback shift register with a $\lambda$-bit seed;
- the Abelian cyclic group $(\mathbb{G}, \cdot)$ is the multiplicative group of a (prime) finite field $\mathbb{Z}_p$ (note that it is denoted multiplicatively);
- the public conversion function $\Psi : \mathbb{G} \to \{0,1\}^\rho$ where $\rho = \lfloor \alpha \cdot \log_2(p) \rfloor$ is simply the truncation of $\lceil (1-\alpha)\log_2(p) \rceil$ bits of the unique member of its group element class in $\{0, \ldots, p-1\}$. The notation $\div$ denotes the euclidean division.

a valid signature of a given message then so is $(r, -s)$ [18].

We call this generator the *subset product generator*.

## 5.1 Description of the Attack

In this setting, the seed consists in a bit-string $\mathsf{seed}_0 \in \{0,1\}^\lambda$ and $n$ group elements $g_1, \dots, g_n \in \mathbb{Z}_p^*$. The bit size of the seed is thus equal to $\lambda + n \cdot \lceil \log_2(p) \rceil$. At each iteration $i \in \mathbb{N}$, the control-sequence generator generates an $n$-bit string $\mathbf{v_i} = (v_i^1, \dots, v_i^n) = \mathsf{CSG}(\mathsf{seed}_0, i)$, computes the group element $h_i$ defined by

$$h_i = g_1^{v_i^1} \cdots g_n^{v_i^n} \in \mathbb{Z}_p^*$$

and outputs $s_i = \Psi(h_i) = h_i \operatorname{div} 2^\ell \in \{0,1\}^k$ where $p$ is a $(k+\ell)$ -bit long prime number (with $k = \lfloor \alpha \cdot \log_2(p) \rfloor$).

A straightforward adaptation of the attack of the Section 4 gives an attack with complexity $O(2^\lambda \cdot (2^{0.78n} + p^{2(1-\alpha)})$ for $\alpha \geq 2/3$. Note that the complexity does not involve the $O(2^{n/2})$ term that came from the indecision on the signs in the elliptic curve variant of the knapsack generator. We remark that one can improve the complexity of the attack by replacing the brute-force search on the missing bits with the use of Coppersmith technique to retrieve them.

**Coppersmith's method.** Coppersmith's method [7, 6] is a technique to find small integer zeroes of univariate or bivariate polynomials modulo a given integer. It has been generalized for finding small roots of (modular) multivariate polynomial equations with integer coefficients by several authors and notably used to attack algebraic pseudo-random generators (see [11, 10, 3, 14] and references therein). These techniques work by constructing a Euclidean lattice associated with the system of equations, and then finding short vectors in this lattice using lattice reduction algorithms. In its most basic variant, given a polynomial $f(X_1, \dots, X_k)$ defined modulo an integer $p$, one can find a "small" root $(x_1, \dots, x_k) \in \mathbb{Z}_p^k$ under the condition that $|x_i| \leq B_i$ for some bounds $(B_1, \dots, B_k)$. The method succeeds (heuristically) in polynomial time when (up to small constant factors),

$$\prod_{i \in M} B_1^{i_1} \dots B_k^{i_k} \leq p$$

when $f$ can be written as a sum of monomials of the form

$$f(X_1, \dots, X_k) = \sum_{i \in M} a_i X_1^{i_1} \dots X_k^{i_k}$$

for some $a_i \in \mathbb{Z}_p^*$. For this simple variant, the lattice is constructed using only the polynomial $f$ but there exist variants – with better upper-bounds on the root $(x_1, \dots, x_k)$ – using lattices of higher dimensions with shifts or powers of the polynomial $f$ (see [11] for details).

**Description of the attack.** For a vector $\mathbf{v_i}$ output by the control sequence generator, we have

$$h_i = g_1^{v_i^1} \cdots g_n^{v_i^n} \in \mathbb{Z}_p^*$$

with $h_i = (2^\ell s_i + x_i)$ where $x_i \in \{0, \dots, 2^\ell - 1\}$ is some value unknown to the adversary. Given a good triplet $(i, j, k)$ with $\mathbf{v_i} + \mathbf{v_j} = \mathbf{v_k}$, we have $h_i \cdot h_j = h_k \bmod p$ and thus:

$$(2^\ell s_i + x_i) \cdot (2^\ell s_j + x_j) = (2^\ell s_k + x_k) \bmod p.$$

The unknowns $(x_i, x_j, x_k)$ are thus "small" roots of an equation of the form

$$Ax_i + Bx_j + x_i x_j - x_k + C = 0 \bmod p$$

where $A = 2^\ell s_i$, $B = 2^\ell s_j$ and $C = (2^\ell s_i \cdot 2^\ell s_j - 2^\ell s_k) \bmod p$ are values known by the adversary. One can thus apply Coppersmith's technique to this polynomial and the basic technique (without using shifts or powers of the polynomial) will succeed if $|x_i|, |x_j|, |x_k| \le p^{1/5}$. A simple trick allows us to improve readily this bound by setting $y = x_i x_j - x_k$ such that $|y| \le 2^{2\ell}$ and solving the equation

$$g(x_i, x_j, y) = Ax_i + Bx_j + y + C = 0 \bmod p$$

in $(x_i, x_j, y)$ is sufficient to recover $(x_i, x_j, x_k)$. Using the basic Coppersmith's technique (again without using shifts or powers of this polynomial), this attack will succeed (heuristically) in polynomial-time if $|x_i|, |x_j|, |x_k| \le p^{1/4}$. For $\alpha \ge 3/4$, we thus obtain an attack with the overall complexity

$$\mathcal{O}\left(2^\lambda \cdot 2^{0.78n} + n \cdot \mathsf{poly}(\log_2(p))\right) = \mathcal{O}\left(2^\lambda \cdot 2^{0.78n}\right).$$

▶ Remark 7. Note that we can improve the bound on the size of the "small" root by using shifts and powers of the polynomial $g(x_i, x_j, y)$. For instance, if one considers the family of fours polynomials

$$\{g, x_i \cdot g, x_j \cdot g, g^2\}$$

that vanishes in $(x_i, x_j, y)$ modulo $p$ with total multiplicity $(1 + 1 + 1 + 2) = 5$ and involve the following set of monomials:

$$\{x_i, x_j, y, x_i^2, x_i x_j, x_i y, x_j^2, x_j y, y^2\}$$

with a sum of degrees equal to $(1 + 1 + 2 + 2 + 2 + 3 + 2 + 3 + 4) = 20$, we obtain that the Coppersmith's method succeeds (heuristically) if $|x_i|, |x_j|, |x_k| \le p^{5/20} = p^{1/4}$ (see [11]). This gives the same bound as above. However, if we reintroduce the variable $x_k$ and replace the monomial $x_i x_j$ by $y + x_k$, the total degree of the set of monomials decreases to 19 and this decreases the bound to $p^{5/19}$. It is possible to decrease a bit further the exponent of $p$ in this bound, at the cost of using a lattice of higher dimension in Coppersmith's technique using the technique of unravelled linearization from [10] (see also [3]).

## 5.2 Experimental Results

**Exhaustive search on the truncated bits.** We consider first the finite field $\mathbb{F}_p$ with $p = 2q+1$ and $q = 99839$. We choose weights in the cyclic multiplicative group $\mathbb{G}$ of order $q$ made by the non-quadratic residues of $K$ minus zero. We present the attack when the control sequence $(\mathbf{v_i})$ is known and we consider $n = 16$ for which the key size is equal to 256 bits. We present in Table 4 the number $m$ of outputs needed and the time necessary to recover the secret weights with probability at least 50% when $\ell$ bits are missing.

When 7 bits are truncated we cannot recover the weights even with 1885 outputs.

Now we consider the finite field $\mathbb{F}_p$ with $p = 2q + 1$ and

$$q = 72536599031050480402372360602698911648481683373808860129469667649180998227293$$

a 256-bit number, but still $n = 16$. With this choice we can focus on recovering the points from the outputs without being bothered by finding the good triplets.

**Table 4** Key-recovery with exhaustive search and $q$ a 16-bit integer.

| $\ell$ | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| $m$ | 1000 | 1000 | 1000 | 1000 | 1000 | 1885 |
| time | $0.51s$ | $0.45s$ | $0.44s$ | $0.47s$ | $0.58s$ | $2.1s$ |

**Table 5** Key-recovery with exhaustive search and $q$ a 256-bit integer.

| $\ell$ | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|
| $m$ | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 |
| time | $0.46s$ | $0.50s$ | $0.48s$ | $0.43s$ | $0.55s$ | $0.70s$ | $0.87s$ | $1.9s$ | $6.6s$ |

**Coppersmith method.** We consider the attack on the second group with $p = 2q + 1$ and $q$ a 256-bit number. First, we implement the attack with the single polynomial $g = Ax_i + Bx_j + y + C$. As the Coppersmith method is a bit more unpredictable, we present in Table 6 the number $m$ of outputs needed and the time necessary to recover the weights with probability at least 50% when $\ell$ bits are missing.

**Table 6** Key-recovery with Coppersmith method and $q$ a 256-bit integer.

| $\ell$ | 2 | 4 | 8 | 16 | 32 | 62 | 63 |
|---|---|---|---|---|---|---|---|
| $m$ | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 |
| time | $0.71s$ | $0.67s$ | $0.68s$ | $0.61s$ | $0.63s$ | $0.51s$ | $0.55s$ |

If we follow the heuristic in Coppersmith's method we should be able to retrieve the weights up to $\ell = 64$ and $\ell = 64$ is the first instance where the attack stops working. If we try to consider the family of polynomials $\{g, x_i g, x_j g, yg, g^2\}$ instead the improvement on the upper-bound from $p^{1/4}$ to $p^{5/19}$ would not be significant for 256-bit integers.

## References

1 Omran Ahmadi and Igor E. Shparlinski. Exponential sums over points of elliptic curves. *J. Number Theory*, 140:299–313, 2014.

2 R. Arratia and L. Gordon. Tutorial on large deviations for the binomial distribution. *Bulletin of Mathematical Biology*, 51(1):125–131, 1989.

3 Aurélie Bauer, Damien Vergnaud, and Jean-Christophe Zapalowicz. Inferring sequences produced by nonlinear pseudorandom number generators using Coppersmith's methods. In Marc Fischlin, Johannes Buchmann, and Mark Manulis, editors, *PKC 2012: 15th International Conference on Theory and Practice of Public Key Cryptography*, volume 7293 of *Lecture Notes in Computer Science*, pages 609–626, Darmstadt, Germany, May 21–23 2012. Springer, Heidelberg, Germany. `doi:10.1007/978-3-642-30057-8_36`.

4 Simon R. Blackburn, Alina Ostafe, and Igor E. Shparlinski. On the distribution of the subset sum pseudorandom number generator on elliptic curves. *Unif. Distrib. Theory*, 6(1):127–142, 2011.

5 Ian F. Blake, Gadiel Seroussi, and Nigel P. Smart. *Elliptic curves in cryptography*, volume 265 of *Lond. Math. Soc. Lect. Note Ser.* Cambridge: Cambridge University Press, 1999.

6 Don Coppersmith. Finding a small root of a bivariate integer equation; factoring with high bits known. In Ueli M. Maurer, editor, *Advances in Cryptology – EUROCRYPT'96*, volume 1070 of *Lecture Notes in Computer Science*, pages 178–189, Saragossa, Spain, May 12–16 1996. Springer, Heidelberg, Germany. `doi:10.1007/3-540-68339-9_16`.

7 Don Coppersmith. Finding a small root of a univariate modular equation. In Ueli M. Maurer, editor, *Advances in Cryptology – EUROCRYPT'96*, volume 1070 of *Lecture Notes in Computer Science*, pages 155–165, Saragossa, Spain, May 12–16 1996. Springer, Heidelberg, Germany. doi:10.1007/3-540-68339-9_14.

8 Edwin D. El-Mahassni. On the distribution of the elliptic subset sum generator of pseudorandom numbers. *Integers*, 8(1):article a31, 7, 2008.

9 Helmut Hasse. Zur Theorie der abstrakten elliptischen Funktionenkörper. III: Die Struktur des Meromorphismenringes. Die Riemannsche Vermutung. *J. Reine Angew. Math.*, 175:193–208, 1936.

10 Mathias Herrmann and Alexander May. Attacking power generators using unravelled linearization: When do we output too much? In Mitsuru Matsui, editor, *Advances in Cryptology – ASIACRYPT 2009*, volume 5912 of *Lecture Notes in Computer Science*, pages 487–504, Tokyo, Japan, December 6–10 2009. Springer, Heidelberg, Germany. doi:10.1007/978-3-642-10366-7_29.

11 Ellen Jochemsz and Alexander May. A strategy for finding roots of multivariate polynomials with new applications in attacking RSA variants. In Xuejia Lai and Kefei Chen, editors, *Advances in Cryptology – ASIACRYPT 2006*, volume 4284 of *Lecture Notes in Computer Science*, pages 267–282, Shanghai, China, December 3–7 2006. Springer, Heidelberg, Germany. doi:10.1007/11935230_18.

12 Simon Knellwolf and Willi Meier. Cryptanalysis of the knapsack generator. In Antoine Joux, editor, *Fast Software Encryption – FSE 2011*, volume 6733 of *Lecture Notes in Computer Science*, pages 188–198, Lyngby, Denmark, February 13–16 2011. Springer, Heidelberg, Germany. doi:10.1007/978-3-642-21702-9_11.

13 Donald Ervin Knuth. *The art of computer programming, Volume 4B: Combinatorial Algorithms, Part 2.* Addison-Wesley, 2022.

14 Florette Martinez. Attacks on pseudo random number generators hiding a linear structure. In Steven D. Galbraith, editor, *Topics in Cryptology – CT-RSA 2022*, volume 13161 of *Lecture Notes in Computer Science*, pages 145–168, Virtual Event, March 1–2 2022. Springer, Heidelberg, Germany. doi:10.1007/978-3-030-95312-6_7.

15 S.M. Ross. *Probability Models for Computer Science.* Elsevier Science, 2002.

16 Rainer A. Rueppel and James L. Massey. Knapsack as a nonlinear fonction. In *IEEE International Symposium on Information Theory.* IEEE Press, NY, 1985.

17 Victor Shoup. Lower bounds for discrete logarithms and related problems. In Walter Fumy, editor, *Advances in Cryptology – EUROCRYPT'97*, volume 1233 of *Lecture Notes in Computer Science*, pages 256–266, Konstanz, Germany, May 11–15 1997. Springer, Heidelberg, Germany. doi:10.1007/3-540-69053-0_18.

18 Jacques Stern, David Pointcheval, John Malone-Lee, and Nigel P. Smart. Flaws in applying proof methodologies to signature schemes. In Moti Yung, editor, *Advances in Cryptology – CRYPTO 2002*, volume 2442 of *Lecture Notes in Computer Science*, pages 93–110, Santa Barbara, CA, USA, August 18–22 2002. Springer, Heidelberg, Germany. doi:10.1007/3-540-45708-9_7.

19 Joachim von zur Gathen and Igor Shparlinski. Predicting subset sum pseudorandom generators. In Helena Handschuh and Anwar Hasan, editors, *SAC 2004: 11th Annual International Workshop on Selected Areas in Cryptography*, volume 3357 of *Lecture Notes in Computer Science*, pages 241–251, Waterloo, Ontario, Canada, August 9–10 2004. Springer, Heidelberg, Germany. doi:10.1007/978-3-540-30564-4_17.

20 Joachim von zur Gathen and Igor E. Shparlinski. Subset sum pseudorandom numbers: fast generation and distribution. *J. Math. Cryptol.*, 3(2):149–163, 2009. doi:10.1515/JMC.2009.007.

21 Lawrence C. Washington. *Elliptic curves. Number theory and cryptography.* Boca Raton, FL: Chapman and Hall/CRC, 2nd ed. edition, 2008.

## A Proof of Theorem 1

We now proceed to prove theorem 1.

**Proof.** Let $x, y, z, u, v$ denote five independent random bits, and set:

$$\begin{aligned}
\rho &= \Pr(x + y = z)\\
\sigma &= \Pr(u + v = z \mid x + y = z)\\
\tau &= \Pr(u + y = v \mid x + y = z)
\end{aligned}$$

We already know that $\rho = 3/8$. Building a simple table as above shows that $\sigma = \tau = 5/12$ (see Table 7).

**Table 7** Tabulating $u + v$, $x + y$, $u + y$ for $x, y, z, u, v \in \{0, 1\}$.

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $u$ | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| $v$ | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| $x$ | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| $y$ | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| $z$ | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 |
| $u+v$ | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 |
| $x+y$ | 0 | 0 | 1 | 1 | 1 | 1 | 2 | 2 | 0 | 0 | 1 | 1 | 1 | 1 | 2 | 2 | 0 | 0 | 1 | 1 | 1 | 1 | 2 | 2 | 0 | 0 | 1 | 1 | 1 | 1 | 2 | 2 |
| $u+y$ | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 2 | 2 | 1 | 1 | 2 | 2 | 1 | 1 | 2 | 2 | 1 | 1 | 2 | 2 |

Let $X(i, j, k)$ denote the binary random variable that takes the value 1 if and only if $A[i] + B[j] = C[k]$, so that $Y = \sum X(i, j, k)$. Unless mentioned otherwise, all sums are taken over $0 \leq i, j, k < N$ ; we omit the indices to alleviate notations.

The expected value of $Y$ is easy to determine. Because the elements of the lists are identically distributed, $\Pr(A[i] + B[j] = C[k])$ is independent of $i, j$ and $k$ and its value is $\rho^n$. We get:

$$\mathrm{E}(Y) = \mathrm{E}\left(\sum X(i,j,k)\right) = \sum \mathrm{E}(X(i,j,k)) \quad = \quad \sum \Pr(A[i] + B[j] = C[k]) = N^3 \left(\frac{3}{8}\right)^n .$$

Because $Y$ is the sum of binary random variables, we are entitled to use the "conditional expectation inequality" [15] (see also [13, MPR]):

$$\Pr(Y > 0) \geq \sum_{i=1}^{n} \frac{\mathrm{E}(Y_j)}{\mathrm{E}(Y \mid Y_i = 1)}. \tag{6}$$

Which, in our case, gives:

$$\Pr(Y > 0) \geq \sum \frac{\mathrm{E}(X(i,j,k))}{\mathrm{E}(Y \mid X(i,j,k) = 1)}.$$

As argued above, the value of the term under the sum is independent of $i, j$ and $k$, so this boils down to: $\Pr(Y > 0) \geq \left(\frac{3}{8}\right)^n / E(Y \mid X(0,0,0) = 1)$. It remains to compute the expected number of good triplets under the assumption that there is at least one. This yields:

$$E(Y \mid X(0,0,0) = 1) = \sum \Pr(A[i] + B[j] = C[k] \mid A[0] + B[0] = C[0])$$

We split this sum into 8 parts by considering separately the situation where $i = 0$, $j = 0$ and $k = 0$ (resp. $\neq 0$ for each summation index). We introduce the shorthand $p_{ijk} = \Pr\left(A[i] + B[j] = C[k] \mid A[0] + B[0] = C[0]\right)$ and we assume that $i, j, k > 0$. Because $A[i]$ is sampled independently from $A[0]$ (resp. $B$, $C$), the two events inside the conditional probability are in fact independent and therefore $p_{ijk} = \left(\frac{3}{8}\right)^n$. But when at least one index is zero, this is no longer the case. The extreme situation is $p_{000} = 1$.

When there is a single non-zero summation index, the situation is rather simple. If $x + y = z$, then $x + U = z$ if and only if $U = y$, and this happens with probability $2^{-n}$ because $U$ is uniformly random. This shows that $p_{i00} = p_{0j0} = p_{00k} = 2^{-n}$.

It remains to deal with the case of two non-zero summation indices. In fact, $p_{ij0}$ is simply $\sigma^n$, while both $p_{i0k}$ and $p_{0jk}$ are equal to $\tau^n$ (by the symmetry between the role of the first two lists).

It follows that

$$\begin{aligned}
&\mathrm{E}\left(Y \mid X(0,0,0) = 1\right) \\
&= (N-1)^3 \left(\frac{3}{8}\right)^n + 3(N-1)^2 \left(\frac{5}{12}\right)^n + 3(N-1) \cdot 2^{-n} + 1 \\
&= N^3 \left(\frac{3}{8}\right)^n + 3N^2 \left(\frac{5}{12}\right)^n + 3N2^{-n} + 1 - \Delta
\end{aligned}$$

$$\text{with} \qquad \Delta = \left(3N^2 - 3N + 1\right)\left(\frac{3}{8}\right)^n + 3(2N-1)\left(\frac{5}{12}\right)^n + 3 \cdot 2^{-n}.$$

The "error term" $\Delta$ is always positive for $N \geq 1$. Going back to the beginning, we have:

$$\begin{aligned}
\Pr\left(Y > 0\right) &\geq \frac{N^3(3/8)^n}{N^3(3/8)^n + 3N^2(5/12) + 3N(1/2)^n + 1 - \Delta} \\
&\geq \frac{1}{1 + 3N^{-1}(10/9)^n + 3N^{-2}(4/3)^n + N^{-3}(8/3)^n}
\end{aligned}$$

Using the convexity of $x \longmapsto 1/(1+x)$, we obtain

$$\Pr(Y = 0) \leq 3N^{-1}(10/9)^n + 3N^{-2}(4/3)^n + N^{-3}(8/3)^n. \qquad \blacktriangleleft$$