# Algorithms for the Sparse Random 3XOR Problem

Charles Bouillaguet, Claire Delaplace

# Algorithms for the Sparse Random 3XOR Problem


**Charles Bouillaguet**
University of Lille, France
charles.bouillaguet@univ-lille.fr

**Claire Delaplace**
Ruhr University Bochum, Germany
claire.delaplace@rub.de



## Abstract

We present two new algorithms for a variant of the 3XOR problem with lists consisting of $n$-bit vectors whose coefficients are drawn randomly according to a Bernoulli distribution of parameter $p < 1/2$. We show that in this particular context the problem can be solved much more efficiently than in the general setting. We first present a simple adaptation of the folklore quadratic algorithm that discards heavy vectors in a preprocessing step. This leads to a linear algorithm with overwhelming success probability for $p < 1/11$, and is sub-quadratic for all $p < 1/2$. We also describe a variant of this method which succeeds deterministically, which is also linear for $p < 1/47$ and always sub-quadratic.

We finally propose a randomized algorithm with a sub-quadratic time complexity when the lists consists of vector of fixed Hamming weight, and discuss possible further improvements.



**2012 ACM Subject Classification** Theory of computation → Computational complexity and cryptography; Theory of computation

**Keywords and phrases** Algorithms, 3-xor problem, random sparse 3-xor


## 1 Introduction

Given three lists $A$, $B$ and $C$ of $n$-bit vectors, the 3XOR problem consists in deciding the existence of (or even finding) a triplet $(\mathbf{x}, \mathbf{y}, \mathbf{z}) \in A \times B \times C$ such that $\mathbf{x} \oplus \mathbf{y} \oplus \mathbf{z}$ is equal to a given target, often assumed to be zero (here the $\oplus$ symbol represent the exclusive-OR or XOR).

This problem can be seen as a variant of the celebrated 3SUM problem, where this time the input list items are seen as integers and we must have $x + y + z = 0$. Many geometric problems can be reduced to 3SUM in sub-quadratic time, and those problem are said to be 3SUM hard [6]. Although the 3XOR problem has enjoyed less interest in the complexity theory field, there exists a few such reductions. For instance, it is a fact that any $\mathcal{O}\left(N^{2-\epsilon}\right)$ algorithm for the 3XOR problem with input lists of size $N$ would imply faster-than-expected algorithms for listing triangles in a graph [14, 7]. Another result due to [4] show that an algorithm solving the 3XOR problem in time $\Omega\left(n^{2-o(1)}\right)$ also reduces the time complexity of the offline SetDisjointness and SetIntersection problems.

The 3XOR problem also has some cryptographic applications, in which the input lists consist of uniformly random vectors (the cryptographic community makes this assumption "by default"). In particular, we can mention Nandi's attack [11] against the COPA mode of authenticated encryption, or the more recent attack against the two-round single-key Even-Mansour cipher by Leurent and Sibleyras [9]. May and Both have been considering a variant of the 3XOR problem, the *approximate 3-list birthday problem*: given three lists of $N$ uniformly random elements of $\{0,1\}^n$ the goal consist in finding triplets $(\mathbf{x}, \mathbf{y}, \mathbf{z})$ in the list such that the hamming weight of $\mathbf{x} \oplus \mathbf{y} \oplus \mathbf{z}$ is small [2].

The simplest possible algorithm to solve the 3XOR problem is the quadratic algorithm, which consists in taking all $\mathbf{x} \oplus \mathbf{y} \in A \times B$ and checking whether they belong to $C$. Using an optimal static dictionary [5] to hold $C$, this results in a time complexity of $\mathcal{O}\left(|A| \cdot |B| + |C|\right)$.

In the particular case where $|A| = |B| = |C| = N$ this algorithm runs in time $\mathcal{O}\left(N^2\right)$. In the following, this simple algorithm will be referred to as QuadraticAlgorithm.

When the input lists are made of random vectors, the decisional variant of the problem may be trivial: if the input lists are too long (resp. too short), then the existence (resp. absence) of a "3XOR triplet" ($\mathbf{x} \oplus \mathbf{y} \oplus \mathbf{z} = 0$) in the input may be asserted with high probability without even observing the input. In this setting, a computational variant of the problem, namely actually finding *a single* 3XOR triplet given the input lists makes more sense.

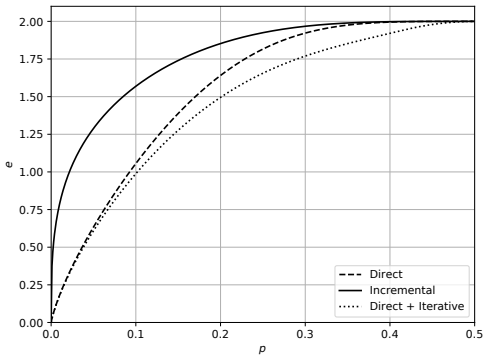
We believe that with random vectors, the hardest case occurs when the size of the input lists $N$ is chosen such that they contain one (and only one) solution with high probability. In any case, if the input lists where longer, they could always be truncated to this size. In the case where the vectors are drawn uniformly at random in $\{0,1\}^n$, this means that $N \approx 2^{n/3}$. In this particular case, the quadratic algorithm is mostly the only option to recover the solution. Some improvements of this method exist [3, 4], however these improvements allows only to gain a polynomial factor in $n$. It is not clear today whether it is possible to find an algorithm for this problem with complexity below $N^{2-o(1)}$.

**Contributions.** In this paper, we focus on a variant of the 3XOR problem where the elements of the lists are random and *sparse*. More precisely, each input bit is drawn independently at random according to a Bernoulli distribution of parameter $0 < p < 1/2$ — the "dense" random case corresponds to $p = 1/2$. The sparse variant of the problem is quite different from its dense counterpart.

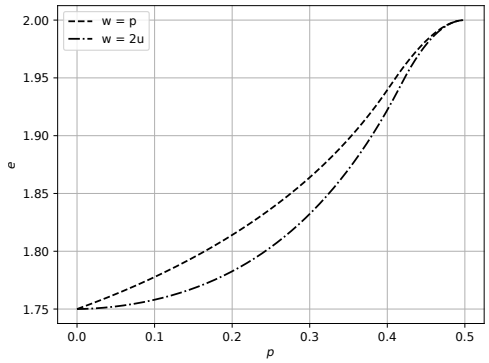
We first give the probability that the input actually contains a 3XOR triplet for given $N$ and $p$. To the best of our knowledge, this result was not readily available from the existing litterature, not even in the simple case where $p = 1/2$. We then describe three algorithms to solve the random sparse 3XOR problem. The simplest possible one (Section 3) works by discarding useless input vectors (whose hamming weight is above a well-chosen threshold), then searches a solution using the quadratic algorithm. This first algorithm returns the solution with overwhelming probability. We also propose an incremental version of this algorithm (Section 4) which deterministically returns a solution if there is one in the input. These algorithms have a time complexity of $\mathcal{O}\left(N + N^e\right)$, for some parameter $e < 2$ when $p < 1/2$. The evolution of this parameter $e$ in function of $p$ is shown in Figure 1 (left diagram). In particular, both algorithms are shown to be *linear* when $p$ is small enough — this stands in strong contrast with the dense case.

In the rest of the paper we propose alternatives to the quadratic algorithms to deal with instances $(A_i, B_j, C_k)$ where the elements of the lists have fixed Hamming weight, respectively $i, j, k$. In this context, useless heavy vectors have already been discarded and the sparsity of the input vectors has to be exploited differently. We use techniques inspired from decoding algorithms.

This is an interesting case as all instances of the sparse 3XOR problem can be converted to several independent sub-instance of this type. In the first of these algorithms (Section 5) we select randomly a subset $J$ of the indices and "guess" that a 3XOR triplet has only zeroes on columns in $J$. From here, we consider the sublists $A'$, $B'$ and $C'$ of $A_i, B_j$, and $C_k$, consisting only of vectors whose coefficients indexed by $j \in J$ are zeroes. We solve this smaller instance with the quadratic algorithm. If no solution is found, we try again with a different $J$. For a well chosen size of $J$, this algorithm is at least as fast as the quadratic algorithm. In the particular case where $i = j = k$, we show that the complexity of the algorithm is between $N^{7/4}$ (when $p$ is close to zero) and $N^2$ (when $p$ is close to $1/2$), where




**(a)** The DIRECT algorithm of section 3 and the INCREMENTAL algorithm of section 4 run in time $\mathcal{O}\left(N + N^e\right)$ where $e$ is shown here.




**(b)** The ITERATIVE algorithm of section 5 runs in time $\tilde{\mathcal{O}}\left(N^e\right)$ where $e$ is shown here.

**Figure 1** Exponents in the time complexity in function of $p$ when the input lists contain one 3XOR triplet with high probability.

$N$ is the size of the input lists.

Finally, we discuss possible improvement of this method in section 6, which basically consists in re-iterating the filtering steps a constant number of times instead of solving the sub-instance directly with the quadratic algorithm. This borrows the main technique of the "nearest neighbors" algorithm of May and Ozerov [10] (which is used in a decoding algorihtm). Given a parameter $t$, we split the indices in $t$ slices. We select randomly a subset $J_1$ of the indices belonging to the first slice and guess that the solution is zero over the columns in $J_1$. We then build the sublists $A_i^{(1)}, B_j^{(1)}, C_k^{(1)}$ of the vectors whose coefficients indexed by $\ell \in J_1$ are zero. After that we select a random subsets $J_2$ of the indices belonging to the second slice and re-iterate until we obtain the lists $A_i^{(t)}, B_j^{(t)}, C_k^{(t)}$, which we process with the quadratic algorithm. The trick is that, if one of our guess $J_\ell$ was wrong, we do not have to restart the whole process, but only starting from $J_\ell$. Although we did not fully investigate the time complexity of this algorithm, we believe that this method should be more efficient in practice than the previous one.

**Motivation** The algorithms described in this paper have no concrete application that we know of, and we don't really care. However they can be used to obtain the —non-trivial and "interesting"— result shown in Appendix C.

# 2 Preliminaries

## 2.1 Notations, Definition and Useful Properties

Let $\mathbf{x} = x_0 x_1 \dots x_{n-1}$ be an $n$-bit string (we use "bit string" and "vector" as synonyms). We denote by $\mathrm{wt}(\mathbf{x})$ its Hamming weight. We denote by $\mathtt{\sim x}$ the negation of $\mathbf{x}$ (XORing 1 to each bit) and by $\mathbf{x}$ `&` $\mathbf{y}$ the bit-wise AND of $\mathbf{x}$ and $\mathbf{y}$. We denote by $\mathbf{x}_{\setminus j}$ the bit-string $x_0 \dots x_{j-1} x_{j+1} \dots x_{n-1}$; more generally, if $J$ is a subset of $\{0, \dots, n-1\}$, we denote by $\mathbf{x}_{\setminus J}$ the sub-string of $\mathbf{x}$, where all $x_j$ for $j \in J$ have been discarded. Let $A$ be a list ; we denote by $|A|$ the number of elements in $A$. Let $A[i]$ be the $i$-th element of $A$. We denote by $A_i$ the sublist of $A$ such that $A_i = \{\mathbf{x} \in A | \mathrm{wt}(\mathbf{x}) = i\}$.

▶ **Definition 1** (3XOR triplet). *Let* $\mathbf{x}, \mathbf{y}$ *and* $\mathbf{z}$ *be three* $n$*-bit strings. We say that* $(\mathbf{x}, \mathbf{y}, \mathbf{z})$ *is a* 3XOR triplet *if* $\mathbf{x} \oplus \mathbf{y} \oplus \mathbf{z} = 0$.

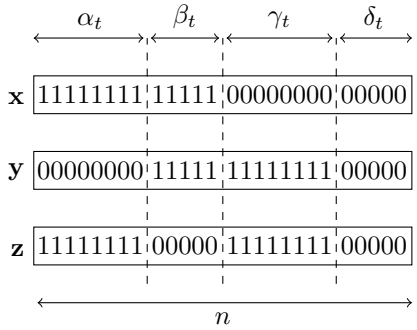



**Figure 2** Shape of a sparse random 3XOR triplet $t = (\mathbf{x}, \mathbf{y}, \mathbf{z})$, up to column permutation.

For any triplet $t = (\mathbf{x}, \mathbf{y}, \mathbf{z})$ of $n$-bit strings and any index $0 \leq j < n$, we denote by $\mathbf{t}_j$ the 3-bit string $x_j y_j z_j$ and we say that it is the *type* of column $j$ in $t$. In a 3XOR triplet, the possible column types are $\{000, 011, 101, 110\}$. Therefore, up to columns permutations, the shape of a 3XOR triplet can be described by Figure 2. Given again a triplet $t = (\mathbf{x}, \mathbf{y}, \mathbf{z})$ of $n$-bit strings, we consider the following functions $\alpha_t = wt(\mathbf{x}\ \texttt{\&}\ \mathbf{z}), \beta_t = wt(\mathbf{x}\ \texttt{\&}\ \mathbf{y}), \gamma_t = wt(\mathbf{y}\ \texttt{\&}\ \mathbf{z})$ and $\delta_t = wt(\texttt{\~}\mathbf{x}\ \texttt{\&}\ \texttt{\~}\mathbf{y}\ \texttt{\&}\ \texttt{\~}\mathbf{z})$. In any case, if $(\mathbf{x}, \mathbf{y}, \mathbf{z})$ is a 3XOR triplet we have $\alpha_t + \beta_t + \gamma_t + \delta_t = n$.

▶ **Definition 2** (3XOR problem with distribution)**.** *Let $\mathcal{D}$ be a probability distribution over $\{0,1\}^n$. Let $A$, $B$ and $C$ be three lists of elements drawn independently at random according to $\mathcal{D}$. A* solution *to the instance of the* 3XOR problem *given by $(A, B, C)$ is a 3XOR triplet $(\mathbf{x}, \mathbf{y}, \mathbf{z}) \in A \times B \times C$. A* random 3XOR triplet *is a triplet $(\mathbf{x}, \mathbf{y}, \mathbf{z})$ chosen according to $\mathcal{D}$* conditioned *to $\mathbf{x} \oplus \mathbf{y} \oplus \mathbf{z} = 0$.*

Because of randomness of the input, the question of the existence of a 3XOR triplet in $(A, B, C)$ may be easy to decide with low probability of error without even observing $A, B$ and $C$, depending on $n, \mathcal{D}$ and the size of the lists. Therefore, our main focus is on the search problem (actually producing a solution, not merely deciding its existence).

**Bounds for Binomial Distributions.** Let $\mathcal{B}(n, p)$ denote the binomial distribution with parameters $n, p$. We denote by log the logarithm in basis 2 and by $H$ the binary entropy function, meaning that $H(x) = -x\log(x) - (1-x)\log(1-x)$, for all $0 < x < 1$. The following standard bounds for the binomial coefficient can be derived from Stirling's formula:

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

We make heavy use of tail bounds for binomial distributions, notably the Chernoff bound (2) and the tighter classical inequality (3), a proof of which can be found in [1] amongst others.

$$\Pr(X \leq k) \leq \exp\left(-\frac{1}{2p}\frac{(np-k)^2}{n}\right), \quad \text{if } \frac{k}{n} < p \tag{2}$$

$$\Pr(X \leq an) \leq \exp -nD(a,p) \quad \text{if } a < p, \tag{3}$$

$$\Pr(X \geq an) \leq \exp -nD(a,p) \quad \text{if } a > p,$$

where $D(a,p) = a\ln\frac{a}{p} + (1-a)\ln\frac{1-a}{1-p}$ is the Kullback-Leibler divergence between an $a$-coin and a $p$-coin.

**Computational Model.** We consider a transdichotomous word Random Access Machine (word-RAM) model. In this model, we have access to a machine in which each "memory cell" contains a $n$-bit word. We assume that the usual arithmetic and bit-wise operations on $n$-bit words, as well as the comparison of two $n$-bit integers and memory access with $n$-bit addresses can be done in constant time. In other terms, we assume that the machine is large enough to accommodate the instance of the problem at hand.

## 2.2 Properties of Random Sparse 3XOR Triplets

Let $0 < p < 1/2$ be fixed. We denote by $Ber_p$ the Bernoulli distribution of parameter $p$ (if $x \xleftarrow{\$} Ber_p$, then $\Pr(x = 1) = p$). Let $\mathcal{D}$ the distribution over $\{0,1\}^n$, where each bit is drawn independently from $Ber_p$. This paper focus on the 3XOR problem with input distribution $\mathcal{D}$.

**Existence of a 3XOR triplet.** Let $a, b$ and $c$ be random bits drawn according to $Ber_p$; the probability that they XOR to zero is $(1-p)\left(1-2p+4p^2\right)$. It follows that if $\mathbf{x}, \mathbf{y}$ and $\mathbf{z}$ are random bit strings drawn according to $\mathcal{D}$, then $\Pr(\mathbf{x} \oplus \mathbf{y} \oplus \mathbf{z} = 0) = (1-p)^n(1-2p+4p^2)^n$. Given three lists $A, B$ and $C$ of random bit strings drawn according to $\mathcal{D}$, each of size $N$, the number of 3XOR triplets in $(A, B, C)$ is a random variable, denoted by $Y$ below, that follows a binomial distribution. We have the following result.

▶ **Theorem 3.** *Let* $q_0 = (1-p)\left(1-2p+4p^2\right)$, $q_1 = (1-p)\left(1-4p+8p^2-4p^3\right)$ *and* $q_2 = (1-p)\left(1-3p+4p^2\right)$. *Then:*

***i)*** $\mathrm{E}\,Y = N^3 {q_0}^n$

***ii)*** $\Pr(Y = 0) \leq 3N^{-1}\left({q_1}/{q_0}^2\right)^n + 3N^{-2}\left({q_2}/{q_0}^2\right)^n + N^{-3}(1/q_0)^n$.

***iii)*** $\Pr(Y = 0) \leq \dfrac{3}{(\mathrm{E}\,Y)^{1/3}} + \dfrac{3}{(\mathrm{E}\,Y)^{2/3}} + \dfrac{1}{\mathrm{E}\,Y}$.

The proof of this theorem can be found in Appendix A.1. The inequality *ii)* is tighter than *iii)*, but *iii)* is often more practical. A first difference between the sparse case and the dense case is that in (exponentially) smaller input lists are sufficient to ensure the existence of a 3XOR triplet with high probability.

For instance, with $p = 1/16$ and $n = 512$, there is *one expected* 3XOR triplet in the input with $N = 2^{44.41}$ (compare this to $N = 2^{170.7}$ with $p = 1/2$). Point *ii)* of theorem 3 states that if we want this solution to be there with probability 99%, we need $N \geq 2^{46.70}$ — the input lists must be 6.5 times larger.

**Expected Density of a 3XOR Triplet.** Let us now consider a sparse random 3XOR triplet $t = (\mathbf{x}, \mathbf{y}, \mathbf{z})$; for a given column $j$, we find that

$$u := \Pr(\mathbf{t}_j = 110) = \Pr(\mathbf{t}_j = 101) = \Pr(\mathbf{t}_j = 011) = p^2/(1-2p+4p^2),$$
$$v := \Pr(\mathbf{t}_j = 000) = (1-p)^2/(1-2p+4p^2).$$

Note that $3u + v = 1$. This means that $(\alpha_t, \beta_t, \gamma_t, \delta_t)$ is a vector of random variables that follows a multinomial distribution of parameters $n$ and $(u, u, u, v)$. Therefore, $\alpha_t, \beta_t$ and $\gamma_t$ individually follow the binomial distribution $\mathcal{B}(n, u)$ and it follows that the expected "density" of $\mathbf{x}$, $\mathbf{y}$ and $\mathbf{z}$ is $2u$. This is always smaller than $p$ when $0 < p < 1/2$. In other terms: *random triplets drawn from $\mathcal{D}$ have density $p$, but random* 3XOR *triplets drawn from $\mathcal{D}$ have smaller density.* The algorithms described in this paper take advantage of this fact.

## 2.3 Methodology

We decided to focus on the case where $N = poly(n) \cdot {q_0}^{-n/3}$. In that case we can expect to have a constant number of 3XOR triplets in $A \times B \times C$ with high probability thanks to theorem 3. This decision can be justified in two ways.

- First, if the input lists are too long, then we may simply look only at the first $poly(n) \cdot {q_0}^{-n/3}$ entries. Theorem 3 then tells us that we can still expect to find a solution with high probability.
- Second, if the input lists contain "too many" elements, there will be trivial and uninteresting solutions. Indeed, if $N \geq [1/(1-p)]^n$, we can expect the string $000\ldots0$ to be present in all three lists. It follows that "return $(0,0,0)$" would be a constant-time algorithm for the sparse random 3XOR problem with a high success probability.

The algorithms we present below would also work for $N < q_0^{-n/3}$, but since the lists consists of random elements, it is unlikely that a solution exists in this case. We can however imagine the following scenario, when a solution following the distribution $\mathcal{D}$ is created and injected inside of smaller lists. In this case, our algorithms will find it with the claimed probability.

Consistently with this assumption on the size of the input, we assume that there is a 3XOR triplet $t^* = (\mathbf{x}^*, \mathbf{y}^*, \mathbf{z}^*)$ in the input, and we study the performance of our algorithms in returning this triplet $t^*$.

# 3 A Very Simple, "Direct", Algorithm

In this section, we present the simplest possible algorithm that solves the sparse 3XOR problem with interesting theoretical guarantees when $p$ is small. Its most striking feature is that it succeeds with overwhelming probability in linear time when $p$ is small enough.

It follows from our preliminary observations that random sparse 3XOR triplets are *very* sparse. This can be exploited in the following simple way: choose a *threshold weight* $w$ ; remove from the input all vectors of hamming weight greater than or equal to $w$ and run the quadratic algorithm on the filtered lists.

Choosing a small value of $w$ leads to smaller "filtered" instances and thus to a smaller running time for the actual computation using the quadratic algorithm. However, if $w$ is too small, then a potential solution present in the input might be discarded by the filtering step. To avoid this, we choose the value of $w$ in the range $]2nu, np[$ — above the expected density of random 3XOR triplets so that we do not discard them, and below the density of the input lists in order to actually discard input vectors that are too heavy.

Any value of $w$ in this "admissible range" ensures an exponentially small failure probability. Indeed, the solution present in the original input is discarded by the filtering step if and only if the weight of either $\mathbf{x}^*, \mathbf{y}^*$ or $\mathbf{z}^*$ is greater than or equal to $w$. Define $\epsilon$ with $2un(1+\epsilon) = w$ ; it follows that $\epsilon$ is strictly positive. We know that the expected weight of $\mathbf{x}^*, \mathbf{y}^*$ and $\mathbf{z}^*$ is $2un$, therefore the Chernoff bound (2) shows that either has weight greater than $w$ with probability less than $\exp(-nu\epsilon^2)$. A union bound then ensures that the solution is discarded with probability less than $3\exp(-nu\epsilon^2)$, so the algorithm succeeds with overwhelming probability as long as the original input contains at least one solution.

We now discuss the complexity of the algorithm. Filtering the input lists takes time $\mathcal{O}(N)$. Let $X \sim \mathcal{B}(n,p)$ be a (binomial) random variable modeling the weight of an input vector of density $p$. Such a vector is kept by the filtering step if its weight is less than $w$, and this happens with probability $s := \Pr(X < w)$. Let $A', B', C'$ denote the filtered lists; their

sizes are stochastically independent random variables following binomial distributions of parameters $N, s$ and their expected size is $Ns$. The expected running time of the quadratic algorithm on the filtered instance is therefore $\mathrm{E}\left(|A'||B'| + |C'|\right) = N^2s^2 + Ns$.

How to choose $w$? We could target the middle of the admissible range $]2nu; np[$, however a much better choice is $w = np^3 + (1-p^2)2nu = 2nu(1+\epsilon)$ with $\epsilon = \frac{p(1-2p)^2}{2}$. The idea is that the filtering threshold weight drifts closer to $2nu$ when $p$ gets closer to zero. This leads to the DIRECT algorithm shown below.

**Algorithm 1** A "direct" algorithm for the sparse 3XOR problem.

**function** FILTER(L, w)
    **return** $\{\mathbf{x} \in L \mid \mathrm{wt}(\mathbf{x}) < w\}$

**function** DIRECT($A, B, C$)
    Set $w \leftarrow n\left(p^3 + (1-p^2)2u\right)$.
    Set $A' \leftarrow$ FILTER$(A, w)$, $B' \leftarrow$ FILTER$(B, w)$ and $C' \leftarrow$ FILTER$(C, w)$.
    **return** QUADRATICALGORITHM$(A', B', C')$.

Using the binomial tail bound (3), we see that the DIRECT algorithm has an expected running-time of:

$$\mathrm{E}\,T = N + N^2\exp(-\lambda n) + o(N) \qquad \text{with} \qquad \lambda = 2D\left(\frac{(2-3p+2p^2)(2p+1)p^2}{1-2p+4p^2}, p\right)$$

Let $e = \log_N(\mathrm{E}\,T - N)$, so that the algorithm runs in time $\mathcal{O}\left(N + N^e\right)$. A quick calculation shows that:

$$e = 2 + 6\frac{D\left(\frac{(2-3p+2p^2)(2p+1)p^2}{1-2p+4p^2}, p\right)}{\ln(1-p)(1-2p+4p^2)}.$$

The graph of $e$ is shown in Fig. 1a. When $p$ goes to zero, the exponent $e$ reaches a limit of zero. When $p$ goes to $\frac{1}{2}$, then $e = 2$. In between, $e$ is increasing. Using the bisection algorithm, we find that $e \leq 1$ when $p \leq 0.0925$. It follows that the DIRECT algorithm is linear when $p \leq \frac{1}{11}$. Looking at Fig. 1a, we conjecture that $e \leq 2p(1 - 2\ln p)$. Establishing this is left for future work.

It is worth noting that the unwieldy expression of $e$ would be greatly simplified from using the simpler Chernoff bound (2) instead of (3). However, the resulting upper-bound on the complexity of the algorithm is much looser for small $p$: it results in $\lim e = 1$ when $p$ goes to zero.

## 4 An Incremental Version

The DIRECT algorithm uses an *a priori* threshold on the density of solution to reduce the size of the instance. This can be improved by starting with a low threshold weight and progressively increasing it while no solution is found. The resulting INCREMENTAL algorithm deterministically reveals the 3XOR triplet present in the input, and it also does so in linear time for small values of $p$.

The main idea is the following: when $w \geq np$, then the filtering step does not reduce significantly the size of the lists. Thus, all iterations after the $np$-th cost essentially $\Omega\left(N^2\right)$. However, the $w$-th iteration (with filtering weight $w$) is done if and only if any of $\mathbf{x}^*, \mathbf{y}^*$

**Algorithm 2** An improved, "incremental" algorithm for the sparse 3XOR problem.

```
function Incremental(A, B, C)
    for w = 0, 1, 2, ..., n do
        Set A' ← Filter(A, w), B' ← Filter(B, w) and C' ← Filter(C, w).
        S ← QuadraticAlgorithm(A', B', C').
        if S ≠ ⊥ then
            return S
```

and $\mathbf{z}^*$ has Hamming weight greater than or equal to $w$. Their expected weight is $2un$, and therefore the probability that the most expensive iterations take place is exponentially small.

▶ **Theorem 4.** *The* Incremental *algorithm returns a solution if it exists in time* $\mathcal{O}\left(N + N^e\right)$, *where*

$$e = 2 + 3\frac{2D\left(\frac{1}{1+\sqrt[3]{\left(1-\frac{1}{p}\right)^2\left(1-\frac{1}{2u}\right)}}, p\right) + D\left(\frac{1}{1+\sqrt[3]{\left(1-\frac{1}{p}\right)^2\left(1-\frac{1}{2u}\right)}}, \frac{2p^2}{1-2p+4p^2}\right)}{\ln(1-p)(1-2p+4p^2)}.$$

The proof of this Theorem is given in Appendix A.2. The graph of $e$ is also shown in Fig. 1a. We find again that $e$ is increasing, $\lim_{p\to 0} e = 0$, $\lim_{p\to\frac{1}{2}} e = 2$ and $e \leq 1$ when $p \leq 0.02155$. Thus, this unfailing algorithm is linear when $p \leq \frac{1}{47}$. We conjecture that $e \leq 3\sqrt[3]{2p} - \frac{4}{5}2p - \frac{1}{5}(2p)^2$ but we leave for future work to prove it. It is worthwhile noting that the complexity of this algorithm is significantly higher than that of the Direct algorithm of section 3. The difference comes from the difference in success probability: this one is unfailing.

In practice, in order to avoid repeating the same work several time, we propose to dispatch the entries of the lists according to their Hamming weight (e.g. the list $A_i$ is the sub-list of $A$ containing only elements of Hamming weight equal to $i$), and treat independently all instances $(A_i, B_j, C_k)$ such that the values $i, j, k$ make it possible to find a solution. More details about how we should proceed is given in Appendix B.

## 5 A Better Algorithm for the Base Case

The previous algorithms (Direct and Incremental) all exploit the sparsity of the 3XOR triplets by filtering the input lists in order to reduce their size, and they use the quadratic algorithm as the "last resort" solution. In this section, we give a better algorithm for the base case where the lists have already been reduced to low-density vectors. The same filtering ideas are therefore not directly applicable, and we instead use a different technique, inspired by the Information Set Decoding algorithm of Lee-Brickell [8].

We consider the setting where we are given three lists $A_i, B_j$ and $C_k$ composed of vectors of hamming weight exactly $i$, $j$ and $k$ respectively, of possibly different sizes. Up to renaming, we assume without loss of generality that $|A_i| \leq |B_j| \leq |C_k|$. Our goal is find a 3XOR triplet with high probability if there is one in the input lists, or return $\perp$ if there is none.

The restriction to input vectors of fixed hamming weight is not really stringent ; indeed, if the input lists were made of vectors of arbitrary hamming weight, then the following strategy could be implemented: partition each input list in $n$ parts according to the Hamming weight ; solve the original problem by searching the $\binom{n+3}{3}$ sub-instances $A_{\mu+\nu} \times B_{\nu+\lambda} \times C_{\mu+\lambda}$ subject to $\mu + \nu + \lambda \leq n$. The point is that any potential 3XOR triplet $t^*$ present in the input is contained in the sub-instance where $\mu, \nu$ and $\lambda$ denote the number of columns of type 110, 101 and 011 of $t^*$, respectively.

The main algorithmic idea is the following: if there is a 3XOR triplet $t^*$ in the input lists, then it necessarily has $\delta = n - (i+j+k)/2$ columns of "type 000". We randomly guess a subset $J$ of these columns and keep only the vectors from the input lists that are zero on all columns of $J$. This produces a smaller sub-instance and we solve it using the quadratic algorithm. If no solution is found, we try again.

As opposed to the two previous algorithms, here the generated sub-instances are not sparser but instead denser than the original input. Correctly choosing the number $s$ of columns that are clamped to zero is critical for performance. This leads to the following ITERATIVE algorithm.

**Algorithm 3** A sub-quadratic algorithm for the base case.

**function** CLAMP(L, J)
  // Return the sublist made of vectors which are 0 on the columns in $J$.
  **return** $\{\mathbf{x}_{\setminus J} \mid (\mathbf{x} \in L) \wedge (\forall j \in J, x_j = 0)\}$

**function** ITERATIVE($A_i, B_j, C_k$)
  Let $\delta \leftarrow n - (i+j+k)/2$.
  Set $s$ to the integer $0 \leq s \leq \delta$ that minimizes $\frac{\binom{n-i}{s}\binom{n-j}{s}}{\binom{n}{s}\binom{\delta}{s}}|A_i| \cdot |B_j| + \frac{\binom{n}{s}}{\binom{\delta}{s}}|C_k|$.
  Let $I \leftarrow H\left(s/n\right) - (\delta/n)H\left(s/\delta\right)$
  **for** $t = 0, 1, \ldots, n2^{nI}$ **do**
    $J \leftarrow$ uniformly random subset of $\{1, 2, \ldots, n\}$ of size $s$.
    $A' \leftarrow$ CLAMP($A_i, J$), $B' \leftarrow$ CLAMP($B_i, J$), $C' \leftarrow$ CLAMP($C_i, J$).
    $S \leftarrow$ QUADRATICALGORITHM($A', B', C'$).
    **if** $S \neq \perp$ **then return** $S$
  **return** $\perp$

To discuss the properties of the algorithm, we assume as usual that there is a 3XOR triplet $t^* = (\mathbf{x}^*, \mathbf{y}^*, \mathbf{z}^*)$ in the input. The following lemma establishes the success probability and justifies the choice of $s$.

▶ **Lemma 5.** *The* ITERATIVE *algorithm fails to returns $t^*$ with probability $e^{-n}$ and its expected running time is less than* $\frac{\binom{n-i}{s}\binom{n-j}{s}}{\binom{n}{s}\binom{\ell}{s}}|A_i| \cdot |B_j| + \frac{\binom{n}{s}}{\binom{\ell}{s}}|C_k|$.

**Proof.** When the "golden triplet" $t^*$ belongs to $A' \times B' \times C'$, then the loop stops and the algorithm succeeds. Let $r$ denote the probability (over the random choice of $J$) that a given triplet in the input is discarded by the "clamping" step; we have $r = \binom{\delta}{s}/\binom{n}{s}$. The expected number of iterations therefore follows a geometric distribution of parameter $r$, and its expectation is therefore $1/r$. Thanks to the bounds on binomial coefficients (1), we obtain:

$$\text{E}\left[\#\text{ iterations}\right] = \frac{1}{r} \leq \frac{2^{nH(s/n)}}{\sqrt{2\pi s(1-s/n)}} \Big/ \frac{2^{\delta H(s/\delta)}}{\sqrt{8s(1-s/\delta)}} \leq cst \times 2^{nI}$$

The probability that $n/r$ iterations occur without success is $(1-r)^{n/r}$ and by concavity of $\ln 1 - x$ we find that this is less than $e^{-n}$. Therefore, the algorithm fails only with exponentially small probability after having done $n$ times the expected number of iterations.

The total time spent in CLAMP is dominated by $|C_k|/r$. After clamping, the expected size of the sub-instances is $|A'| = |A_i|\binom{n-i}{s}/\binom{n}{s}$. The same goes for $B_j$ and $C_k$. Therefore, the expected cost of solving a single sub-instance using the quadratic algorithm is:

$$\frac{\binom{n-i}{s}\binom{n-j}{s}}{\binom{n}{s}\binom{n}{s}}|A_i||B_j| + \frac{\binom{n-k}{s}}{\binom{n}{s}}|C_k|.$$

The lemma follows, because the component in $|C_k|$ of the cost of solving subproblems is dominated by the cost of filtering. ◀

It is not straightforward to state anything meaningful about the complexity of the algorithm in general. However, when the input contains a 3XOR triplet, then the ITERATIVE algorithm always returns it faster than the quadratic algorithm. Indeed, assume that the input contains a 3XOR triplet ; had we chosen $s = 0$ then no clamping would take place and the original input lists would have been fed to the quadratic algorithm. The whole procedure would then stop and succeed during the first iteration, with a running time equal to that of the quadratic algorithm (up to negligible terms). The choice of $s$ guarantees a *better* expected running time. This value can be found by exhaustive search over $n$ elements. It follows that the algorithm is at most quadratic.

**Equidistributed Inputs.** To progress in the analysis, we therefore restrict ourselves to a simpler setting: following again our methodology discussed in section 2.3, we start from three input lists $A, B$ and $C$ of size $poly(n){q_0}^{-n/3}$ — therefore containing one 3XOR triplet with high probability. We choose a constant $w$ and we build the sublists $A_{wn}, B_{wn}$ and $C_{wn}$ by keeping only the vectors of weight $wn$ from $A, B$ and $C$. In this setting, the expected size of the input lists is $N_w = \mathrm{E}\,|A_{wn}| = \mathrm{E}\,|B_{wn}| = \mathrm{E}\,|C_{wn}| = N\binom{n}{wn}p^{wn}(1-p)^{n(1-w)}$. The running time of the ITERATIVE algorithm on input $(A_{wn}, B_{wn}, C_{wn})$ is then a function of $n$ and $p$ only.

If the input contains a 3XOR triplet, then it has $\delta = n\left(1 - \frac{3}{2}w\right)$ columns of "type 000". The value of the $s$ parameter can be provided manually and/or decided automatically. Define:

$$L = -\frac{1}{3}\log q_0 + H(w) + w\log p + (1-w)\log(1-p)$$

$$I = H\left(\frac{s}{n}\right) - \frac{\delta}{n}H\left(\frac{s}{\delta}\right)$$

$$R = H\left(\frac{s}{n}\right) - (1-w)H\left(\frac{s}{n(1-w)}\right)$$

With these notations, the size of the input lists is $N_w = poly(n)2^{nL}$, the number of iterations of the loop is $n2^{nI}$ and the expected size of the clamped sub-lists is $2^{n(L-R)}$. It follows that the total time spent clamping the lists is $2^{n(I+L)}$ and the total time spend in the quadratic algorithm is $2^{n(I+2(L-R))}$. Therefore, in this setting, the ITERATIVE algorithm runs in time $\tilde{\mathcal{O}}\left({N_w}^e\right)$ where the exponent is $e = \frac{I}{L} + \max\left(1, 2 - 2\frac{R}{L}\right)$.

Given a choice of $w$ (and a value of $s$), this allows the exponent to be computed as function of $p$ — as shown in Fig. 1b for $w = p$ and $w = 2u$. The figure strongly suggests that the exponent $e$ reaches a limit of $\frac{7}{4}$ when $p$ goes to zero. This would make the ITERATIVE algorithm asymptotically better than the quadratic algorithm. It is indeed provably the case, at least for some values of $w$. We consider two interesting cases : $w = p$ and $w = 2u$. With $w = p$, the "filtering" targets the average density of the input vectors, yielding only a polynomial reduction in size. with $w = 2u$, the filtering targets the expected density of 3XOR triplets.

▶ **Theorem 6.** *For $w \in \{2u, p\}$, the exponent $e$ reaches a limit of $\frac{7}{4}$ when $p$ goes to zero.*

The proof amounts to choosing $s/\delta = 1 - e^{-1/2}$ when $w = p$ and $s/\delta = 1 - \sqrt[4]{2u}$ when $w = 2u$, then compute the limits. More details can be found in Appendix A.3.

**Application to the Direct Algorithm.** This algorithm for the base case can be used to improve the DIRECT algorithm of section 3. After discarding the vectors of weight greater than $w$ from the input, the DIRECT algorithm feeds the filtered lists to the quadratic algorithm. The proposed modification consists in partitioning the filtered lists by hamming weight, and solving the $\mathcal{O}\left(n^3\right)$ sub-instances using the ITERATIVE algorithm.

The complexity of this procedure is dominated by the running time of the ITERATIVE algorithm on the heaviest balanced sub-instance $(A_w, B_w, C_w)$, with $w/n = p^3 + 2u(1-p^2)$. Given a value of $p$, we can compute $w$ ; from there, we can compute the exponent $e$ of the iterative algorithm (by choosing the optimal value of $s/\delta$ numerically); we then know that the sub-instance can be dealt with in time $(Ns)^e$, where $s$ is the quantity defined in section 3 ; this in turn allows us to compute the exponent of the combination of the DIRECT algorithm with the ITERATIVE algorithm, numerically, for a given value of $p$. The result is again shown in Fig. 1a.

Proving an upper-bound on the exponent of the combination is left as an interesting open problem.

## 6 Possible Improvement and Discussion

We discuss a possible improvement of the previous algorithm. After the clamping step, the sub-instances may not be fully dense. Therefore it would make sense to use the ITERATIVE algorithm recursively instead of using the quadratic algorithm. More precisely, in order to make this idea work, we propose to rely on an technique due to May and Ozerov to improve Information Set Decoding [10].

**Algorithm 4** Sparse Random 3XOR with Improved Clamping

**function** PERMUTED($A_i, B_j, C_k$)
  // Returns a 3XOR triplet in $A_i \times B_j \times C_k$ w.h.p. or $\perp$ if none exists
  **for** $\kappa = 1, \ldots, Poly(n)$ **do** ▷ $\deg Poly(n) > (t-1)/2$
    $\delta \leftarrow n - (i+j+k)/2$
    $q \leftarrow$ random permutation of $\{0, \ldots, n-1\}$
    $A_i \leftarrow q(A_i)$, $B_j \leftarrow q(B_j)$, $C_k \leftarrow q(C_k)$ ▷ Permute the columns of the lists
    Set $s$ to the value that minimize the runtime
    $S \leftarrow$ RECURSIVE($A_i, B_j, C_k, s, 1$)
    **return** $S$
  **return** $\perp$

**function** RECURSIVE($A_i, B_j, C_k, s, \ell$)
  **if** $\ell = t+1$ **then**
    **return** QUADRATICALGORITHM($A_i, B_j, C_k$)
  **else**
    $I \leftarrow H(s/n) - (\delta/n)H(s/\delta)$
    **for** $0 \leq \mu < n2^{nh_\ell I}$ **do**
      $J \leftarrow$ random subset of $h_\ell s$ columns in $[(h_1 + \cdots + h_{\ell-1})n : (h_1 + \cdots + h_\ell)n]$
      $A' \leftarrow$ CLAMP($A_i, J$), $B' \leftarrow$ CLAMP($B_j, J$), $C' \leftarrow$ CLAMP($C_k, J$)
      $S \leftarrow$ RECURSIVE($A', B', C', \ell+1$)
      **if** $S \neq \perp$ **then**
        **return** $S$
  **return** $\perp$

Once again, we are in the setting where the lists contains entries of fixed Hamming weight, respectively $i, j$ and $k$. Let us fix a constant $t$ and positive reals $h_1, \ldots, h_t$ such that $h_1 + \cdots + h_t = 1$. We know that if there is a 3XOR triplet $t^*$ in the input, then there are at $\delta = n - (i + j + k)/2$ columns of "type 000" in $t^*$.

Here is the main idea of the method. We choose a total number $s$ of columns to clamp, as in the ITERATIVE algorithm. However, we do the clamping in $t$ stages:

1. Choose a random subset $J_1$ of $h_1 s$ columns in the interval $[0 : h_1 n[$, and assume that the type of all columns in $J_1$ is 000. Compute the sub-lists $A_i^{(1)}, B_j^{(1)}, C_k^{(1)}$ by clamping the original input lists over $J_1$.
2. Choose a random subet $J_2$ of $h_2 s$ columns in the interval $[h_1 n : (h_1 + h_2)n[$. Compute sublists $A_i^{(2)}, B_j^{(2)}, C_k^{(2)}$ by clamping the output of the previous stage over $J_2$
3. Continue for $t$ steps until we came up with small sublists $A_i^{(t)}, B_j^{(t)}, C_k^{(t)}$ and clamping has been done on $s$ columns.
4. Solve the resulting instance $(A_i^{(t)}, B_j^{(t)}, C_k^{(t)})$ using the quadratic algorithm.

In the ITERATIVE algorithm, if the choice of $J$ is wrong, then all the clamping that has been done must be discarded. Here, only a part of it will be done in vain, but the choices that have been made *before* the bad one can still stand. We assume that the columns of type 000 are uniformly distributed in $[0, n-1]$. If not, we can randomize the instance by randomly permuting the columns.

▶ **Lemma 7.** *The* PERMUTED *Algorithm finds a solution present in the input with overwhelming probability.*

The proof is given in Appendix A.4. Due to the number of parameters that have to be taken into account, the complexity of this algorithm is not easy to analyze, and we leave it as an open problem. We claim however, that, in order to reach the best time complexity, we need to fix the values of the parameters $h_1, \ldots, h_t$ such that the time spent clamping the lists is mostly the same in each level.

## References


1 R. Arratia and L. Gordon. Tutorial on large deviations for the binomial distribution. *Bulletin of Mathematical Biology*, 51(1):125 – 131, 1989. URL: `http://www.sciencedirect.com/science/article/pii/S0092824089800527`, `doi:https://doi.org/10.1016/S0092-8240(89)80052-7`.

2 Leif Both and Alexander May. The approximate k-list problem. *IACR Transactions on Symmetric Cryptology*, 2017(1):380–397, 2017.

3 Charles Bouillaguet, Claire Delaplace, and Pierre-Alain Fouque. Revisiting and improving algorithms for the 3xor problem. *IACR Transactions on Symmetric Cryptology*, 2018(1):254–276, 2018.

4 Martin Dietzfelbinger, Philipp Schlag, and Stefan Walzer. A subquadratic algorithm for 3xor. *arXiv preprint arXiv:1804.11086*, 2018.

5 Michael L. Fredman, János Komlós, and Endre Szemerédi. Storing a sparse table with o(1) worst case access time. *J. ACM*, 31(3):538–544, June 1984. URL: `http://doi.acm.org/10.1145/828.1884`, `doi:10.1145/828.1884`.

6 Anka Gajentaan and Mark Overmars. On a class of $\mathcal{O}(n^2)$ problems in computational geometry. *Computational geometry*, 5(3):165–185, 1995.

7 Zahra Jafargholi and Emanuele Viola. 3sum, 3xor, triangles. *CoRR*, abs/1305.3827, 2013. URL: `http://arxiv.org/abs/1305.3827`, `arXiv:1305.3827`.

8 Pil Joong Lee and Ernest Brickell. An observation on the security of McEliece's public-key cryptosystem. In *Workshop on the Theory and Application of of Cryptographic Techniques*, pages 275–280. Springer, 1988.

9 Gaëtan Leurent and Ferdinand Sibleyras. Low-memory attacks against two-round even-mansour using the 3-xor problem. In Alexandra Boldyreva and Daniele Micciancio, editors, *Advances in Cryptology - CRYPTO 2019 - 39th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2019, Proceedings, Part II*, volume 11693 of *Lecture Notes in Computer Science*, pages 210–235. Springer, 2019. URL: `https://doi.org/10.1007/978-3-030-26951-7_8`, `doi:10.1007/978-3-030-26951-7\_8`.

10 Alexander May and Ilya Ozerov. On computing nearest neighbors with applications to decoding of binary linear codes. In *EUROCRYPT*, pages 203–228, 2015.

11 Mridul Nandi. Revisiting Security Claims of XLS and COPA. *IACR Cryptology ePrint Archive*, 2015:444, 2015.

12 S.M. Ross. *Probability Models for Computer Science*. Elsevier Science, 2002. URL: `https://books.google.fr/books?id=fG3iEZ8f3CcC`.

13 Wayne E. Smith. Various optimizers for single-stage production. *Naval Research Logistics Quarterly*, 3(1-2):59–66, 1956. URL: `https://onlinelibrary.wiley.com/doi/abs/10.1002/nav.3800030106`, `arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1002/nav.3800030106`, `doi:10.1002/nav.3800030106`.

14 Emanuele Viola. Reducing 3xor to listing triangles, an exposition. Technical report, Northeastern University, College of Computer and Information Science, May 2012. Available at `http://www.ccs.neu.edu/home/viola/papers/xxx.pdf`.


# A Missing Proofs

## A.1 Proof of Theorem: Existence of a Solution

Recall that $Y$ is the random variable that counts the number of 3XOR triplets in $A \times B \times C$ (taken over the random choice of $A, B$ and $C$). To establish the theorem, we want to estimate $\Pr[Y > 0]$. The expected value of $Y$ is bounded away from zero, so a concentration inequality would bring the result. We cannot use a Chernoff-type bound because the $N^3$ triplets in $A \times B \times C$ are identically distributed, but not independent of each other.

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

**Proof of theorem 3.** The expected value of $Y$ is easy to determine. Because the elements of the lists are identically distributed, $\Pr(A[i] \oplus B[j] \oplus C[k] = 0)$ is independent of $i, j$ and $k$, and we get:

$$\mathrm{E}(Y) = \mathrm{E}\left(\sum X(i,j,k)\right) = \sum \mathrm{E}(X(i,j,k)) = \sum \Pr(A[i] \oplus B[j] \oplus C[k] = 0) = \mathrm{E}(Y) = N^3 {q_0}^n.$$

Because $Y$ is the sum of binary random variables, we are entitled to use Ross's conditional expectation inequality [12]:

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

As argued above, the value of the term under the sum is independent of $i, j$ and $k$, so this boils down to: $\Pr(Y > 0) \geq E(Y)/E(Y \mid X(0,0,0) = 1)$. It remains to compute the number

of solutions knowing that there is at least one.

$$E(Y \mid X(0,0,0)=1) = \sum \Pr\left(A[i] \oplus B[j] \oplus C[k] = 0 \mid A[0] \oplus B[0] \oplus C[0] = 0\right)$$
$$= {q_0}^{-n} \sum \Pr\left(A[i] \oplus B[j] \oplus C[k] = 0 \ \wedge \ A[0] \oplus B[0] \oplus C[0] = 0\right)$$

To study this joint distribution, we simply distinguish 8 cases by considering separately the situation where $i=0$, $j=0$ and $k=0$ (resp $\neq 0$ for each index). We introduce the shorthand $p_{ijk} = \Pr\left(A[i] \oplus B[j] \oplus C[k] = 0 \ \wedge \ A[0] \oplus B[0] \oplus C[0] = 0\right)$ and assume that $i,j,k > 0$. Then the two joined events are in fact independent and $p_{ijk} = {q_0}^{2n}$. But when at least one indice is zero, this is no longer the case ; the extreme situation is $p_{000} = {q_0}^n$. By symmetry between the three input lists, we find that $p_{ij0} = p_{i0k} = p_{0jk}$ and $p_{i00} = p_{0j0} = p_{00k}$.

Let us compute $p_{0jk}$. What happens here depends mostly on the hamming weight of $A[0]$. Indeed $B[j]$ and $C[k]$ have to sum to one where $A[0]$ is one, and to sum to zero where $A[0]$ is zero. Two bits drawn according to $Ber_p$ sum to one with probability $2p(1-p)$, and they sum to zero with probability $p^2+(1-p)^2$. Therefore we get (using the definition of the binomial distribution and the binomial theorem):

$$p_{0jk} = \sum_{w=0}^{n} \Pr\left(wt(A[0])=w\right) \Pr\left(B[j] \oplus C[k] = A[0] \wedge B[0] \oplus C[0] = A[0] \mid wt(A[0])=w\right)$$
$$= \sum_{w=0}^{n} \binom{n}{w} p^w (1-p)^{n-w} \left( \left[2p(1-p)\right]^w \left[p^2+(1-p)^2\right]^{n-w} \right)^2$$
$$= \sum_{w=0}^{n} \binom{n}{w} \left[4p^3(1-p)^2\right]^w \left[(1-p)\left(1-2p+2p^2\right)^2\right]^{n-w}$$
$$= \left(4p^3(1-p)^2 + (1-p)\left(1-2p+2p^2\right)^2\right)^n$$
$$= {q_1}^n$$

Next, we determine $p_{00k}$. Here, $C[k]$ has to be equal to $A[0] \oplus B[0]$. This then mostly depends on the hamming weight of $A[0] \oplus B[0]$, and the situation is somewhat symmetrical:

$$p_{00k} = \sum_{w=0}^{n} \Pr\left(wt(A[0] \oplus B[0])=w\right) \Pr\left(C[k] = A[0] \oplus B[0] \wedge C[0] = A[0] \oplus B[0] \mid wt(A[0] \oplus B[0])=w\right)$$
$$= \sum_{w=0}^{n} \binom{n}{w} \left[2p(1-p)\right]^w \left[p^2+(1-p)^2\right]^{n-w} \left(p^w(1-p)^{n-w}\right)^2$$
$$= \sum_{w=0}^{n} \binom{n}{w} \left[2p^3(1-p)\right]^w \left[(1-p)^2(1-2p+2p^2)\right]^{n-w}$$
$$= \left(2p^3(1-p) + (1-p)^2(1-2p+2p^2)\right)^n$$
$$= {q_2}^n$$

We can now write:

$$\mathrm{E}\left(Y \mid X(0,0,0)=1\right) = {q_0}^{-n}\left[(N-1)^3 {q_0}^{2n} + 3(N-1)^2 {q_1}^n + 3(N-1){q_2}^n + {q_0}^n\right]$$
$$= N^3 {q_0}^n + 3N^2 \left(q_1/q_0\right)^n + 3N\left(q_2/q_0\right)^n + 1 - \Delta \qquad \text{with:}$$
$$\Delta = (3N^2 - 3N + 1){q_0}^n + 3(2N-1)\left(q_1/q_0\right)^n + 3\left(q_2/q_0\right)^n$$

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

finally have (using the convexity of $1/(1+x)$ in the last step):

$$\begin{aligned}\Pr\left(Y>0\right) &\geq \mathrm{E}\left(Y\right)/\mathrm{E}\left(Y \mid X(0,0,0)=1\right)\\ &\geq \frac{N^3{q_0}^n}{N^3{q_0}^n + 3N^2\left(q_1/q_0\right)^n + 3N\left(q_2/q_0\right)^n + 1 - \Delta}\\ &\geq \frac{1}{1 + 3N^{-1}\left(q_1/{q_0}^2\right)^n + 3N^{-2}\left(q_2/{q_0}^2\right)^n + 1/\mathrm{E}\left(Y\right)}\\ \Pr\left(Y=0\right) &\leq 3N^{-1}\left(q_1/{q_0}^2\right)^n + 3N^{-2}\left(q_2/{q_0}^2\right)^n + 1/\mathrm{E}\left(Y\right)\end{aligned}$$

This establishes the second point of the theorem.

It it worthwhile noting that the second moment inequality $\Pr(Y>0) \geq (\mathrm{E}\left(X\right))^2/(\mathrm{E}\left(X^2\right))$ gives exactly the same bound; computing the variance $\sigma^2 = (\mathrm{E}\left(X^2\right)) - (\mathrm{E}\left(X\right))^2$ and using Chebyshev's inequality also yields the result of the theorem.

Next, using $\mathrm{E}\left(Y\right) = N^3{q_0}^n$, we may rewrite our last inequality in terms of the expectation:

$$\Pr\left(Y=0\right) \leq 3\frac{\left({q_1}^3/{q_0}^5\right)^{n/3}}{\left(\mathrm{E}\left(Y\right)\right)^{-1/3}} + 3\frac{\left({q_2}^3/{q_0}^4\right)^{n/3}}{\left(\mathrm{E}\left(Y\right)\right)^{-2/3}} + \frac{1}{\mathrm{E}\left(Y\right)}.$$

We claim that ${q_1}^3/{q_0}^5$ and ${q_2}^3/{q_0}^4$ are both smaller than one over $(0, 1/2)$. This follows from the facts that: *i*) they are both equal to 1 when $p=0$ and *ii*) they decreasing functions of $p$ over this interval. The latter claim an be established by actually computing their derivatives and checking that they are negative (all factors are easily seen to take only positive values):

$$\begin{aligned}\frac{\partial}{\partial p}\left(\frac{{q_1}^3}{{q_0}^5}\right) &= -\frac{6p(4p^3-8p^2+4p-1)^2(2p^2-6p+3)(2p-1)^2}{(4p^2-2p+1)^6(1-p)^3},\\ \frac{\partial}{\partial p}\left(\frac{{q_2}^3}{{q_0}^4}\right) &= -\frac{6p(4p^2-3p+1)^2(4p^2-6p+3)(1-2p)}{(4p^2-2p+1)^5(p-1)^2}.\end{aligned}$$

This establishes the third point of the theorem. ◀

## A.2 Proof of the Complexity of the Incremental Algorithm

**Proof of Theorem 4.** Let $T_i$ denote the running time of the $i$-th iteration and $S$ the number of iterations done when the algorithm stops. The total running time is then given by $T = \sum_{i=0}^{n}[S \geq i]T_i$. The two random variables $[S \geq i]$ and $T_i$ are not independent, however we claim that $\mathrm{E}\left([S \geq i]T_i\right) \leq \left(\mathrm{E}\left[S \geq i\right]\right)\left(T_i\right)$ — *i.e.* they are negatively correlated. The point is that is that the fact that input lists contain a "large weight" triplet $t^*$ can only reduce the expected size of filtered lists by at most one, and therefore reduce the expected running time of processing them. It follows that $\mathrm{E}\,T \leq \sum_{i=0}^{n}\Pr(S \geq i)(\mathrm{E}\,T_i)$.

Next, let us consider two random variables following binomial distributions $X_{2u} \sim \mathcal{B}(n, 2u)$ and $X_p \sim \mathcal{B}(n, p)$. From the previous section, we know that $\mathrm{E}\,T_i = N^2s^2 + Ns$, where $s = Pr(X_p \leq i)$. In addition, following the same reasoning as in section 3, we have $\Pr(S \geq i) \leq 3\Pr(X_{2u} \geq i)$. This gives:

$$\mathrm{E}\,T \leq 3N^2\sum_{i=0}^{n}\Pr(X_p \leq i)^2 Pr(X_{2u} \geq i)$$

Set $u_i = \Pr(X_p \leq i)^2 Pr(X_{2u} \geq i)$. Our goal is to upper-bound the sum of the $u_i$'s. To this end, we split the sum in three parts:

$$\sum_{i=0}^{n} u_i \leq \sum_{i=0}^{2nu} u_i + \sum_{i=2nu}^{np} u_i + \sum_{i=np}^{n} u_i$$

First, we note that $\Pr(X_p \leq i)$ is increasing when $i \leq 2nu$, because $2u < p$. Therefore we have $u_i \leq \Pr(X_p \leq 2nu)^2$, and because $Pr(X_{2u} \geq i)$ is greater than $\frac{1}{2}$ when $i \leq 2nu$, we find that $u_i \leq 2u_{2nu}$ when $0 \leq i \leq 2nu$. A symmetric argument shows that $2u_{np} \geq u_i$ for all $np \leq i \leq n$. Let $M$ denote the largest $u_i$ for $2nu \leq i \leq np$ ; then the sum of all the $u_i$'s is less than $2nM$. We use (3) to get an upper-bound on $M$. Set $f(x) = 2D(x,p) + D(x,2u)$, so that $u_i \leq e^{-nf\left(\frac{i}{n}\right)}$. We next seek the maximum of $f$, and for this we compute its derivative:

$$f'(x) = 2\log\frac{x}{p} - 2\log\frac{x-1}{p-1} + \log\frac{x}{2u} - \log\frac{x-1}{2u-1}$$

Solving $f'(x_0) = 0$ reveals only one possible real solution, that satisfies:

$$1 - \frac{1}{x_0} = \sqrt[3]{\left(1-\frac{1}{p}\right)^2\left(1-\frac{1}{2u}\right)}$$

It appears that $1 - 1/x_0$ is the geometric mean of $1-1/p, 1-1/p$ and $1-1/(2u)$; therefore we find that $2u \leq x_0 \leq p$ (in other terms, the largest $u_i$ actually has an index in the range $[2nu; np]$). It follows that the total expected cost of the algorithm is upper-bounded by:

$$\mathrm{E}\,T \leq N + 6nN^2 e^{-n[2D(x_0,p)+D(x_0,2u)]}$$

Setting again $e = \log_N(\mathrm{E}\,T - N)$, the INCREMENTAL algorithm runs in time $\mathcal{O}\left(N + N^e\right)$, with:

$$e = 2 + 3\frac{2D\left(\frac{1}{1+\sqrt[3]{\left(1-\frac{1}{p}\right)^2\left(1-\frac{1}{2u}\right)}}, p\right) + D\left(\frac{1}{1+\sqrt[3]{\left(1-\frac{1}{p}\right)^2\left(1-\frac{1}{2u}\right)}}, \frac{2p^2}{1-2p+4p^2}\right)}{\ln(1-p)(1-2p+4p^2)}.$$

◀

## A.3 Proofs for the Analysis of the Iterative Algorithm

**Proof of theorem 6.** First of all, let us start by considering the case where $w = np$ — we "filter" the input list by keeping the most common weight. We set the number of clamped columns to $s/\delta = \left(1 - e^{-\frac{1}{2}}\right)$. This gives

$$L = -\frac{1}{3}\log_2 q_0$$

$$I = H\left(\left(1-e^{-\frac{1}{2}}\right)\left(1-\frac{3}{2}p\right)\right) - \left(1-\frac{3}{2}p\right)H\left(1-e^{-\frac{1}{2}}\right)$$

$$R = H\left(\left(1-e^{-\frac{1}{2}}\right)\left(1-\frac{3}{2}p\right)\right) - (1-p)H\left(\left(1-e^{-\frac{1}{2}}\right)\left(1-\frac{3}{2}p\right)\frac{1}{1-p}\right)$$

This allows the exponent $e$ to be computed, at least numerically. It is not well-defined at $p = 0$, but the limit can be computed: we find that $\lim_{p\to 0}\frac{I}{L} = \frac{3}{4}$ while $\lim_{p\to 0}\frac{R}{L} = \frac{1}{2}$ (these limits can be computed automatically by the Maple computer algebra system ; unfortunately the open-source SageMath system fails). This means that the exponent reaches a limit of $\frac{7}{4}$ when $p$ goes to zero.

Let us next consider another interesting case, namely $w = 2u$ — we target the expected density of 3XOR triplets. This time, we set the number of clamped columns so that

$s/\delta = 1 - \sqrt[4]{2u}$. This gives:

$$L = -\frac{1}{3}\log q_0 + H(2u) + 2u\log p + (1-2u)\log(1-p)$$

$$I = H\left(\left(1-\sqrt[4]{2u}\right)(1-3u)\right) - (1-3u)\,H\left(1-\sqrt[4]{2u}\right)$$

$$R = H\left(\left(1-\sqrt[4]{2u}\right)(1-3u)\right) - (1-2u)H\left(\left(1-\sqrt[4]{2u}\right)\frac{1-3u}{1-2u}\right)$$

We again find that $\lim_{p\to 0}\frac{I}{L} = \frac{3}{4}$ while $\lim_{p\to 0}\frac{R}{L} = \frac{1}{2}$. Therefore, the exponent also reaches a limit of $\frac{7}{4}$ when $p$ goes to zero. ◀

## A.4 Proof regarding the Permuted Algorithm

**Proof of Lemma 7.** We denote by $t^* = (\mathbf{x}^*, \mathbf{y}^*, \mathbf{z}^*)$ the solution we aim to recover.

Let $A_i^{(\ell)}, B_j^{(\ell)}, C_k^{(\ell)}$ denote the lists that are taken as input by the RECURSIVE algorithm alongside with the index $\ell$. Let $\mathbf{x}_\ell^*$ denote the vector $\mathbf{x}^*_{\setminus I_\ell}$, where $I_\ell = J_1 \cup \ldots J_{\ell-1}$, for $\ell > 1$ and $I_1 = \emptyset$. We define $\mathbf{y}_\ell^*$ and $\mathbf{z}_\ell^*$ accordingly.

Let us denote by $\pi_\ell$ the probability that the triplet $(\mathbf{x}_{\ell+1}^*, \mathbf{y}_{\ell+1}^*, \mathbf{z}_{\ell+1}^*)$ is in $A_i^{(\ell+1)} \times B_j^{(\ell+1)} \times C_k^{(\ell+1)}$, knowing that $(\mathbf{x}_\ell^*, \mathbf{y}_\ell^*, \mathbf{z}_\ell^*)$ is in $A_i \times B_j \times C_k$. Assuming that there is a fraction $h_\ell$ of the $\delta$ the columns of "type 000" in the interval of size $h_\ell n$ inside which we choose $J_\ell$, we have (using the bounds on binomial coefficients (1)):

$$\pi_\ell \geq \binom{h_\ell\delta}{h_\ell s} / \binom{h_\ell n}{h_\ell s} \geq cst \cdot 2^{h_\ell(\delta H(s/\delta) - nH(s/n))} = cst \cdot 2^{-nh_\ell I}.$$

At each step $\ell$, if the solution is not found, we restart the procedure up to $\approx n/\pi_\ell$ times. Assuming that $(\mathbf{x}_\ell^*, \mathbf{y}_\ell^*, \mathbf{z}_\ell^*)$ is in $A_i^{(\ell)}, B_j^{(\ell)}, C_k^{(\ell)}$, the probability that we do not find the solution after this many iterations is

$$\Pr\left[\text{fail at step } \ell\right] \approx (1-\pi_\ell)^{\frac{n}{\pi_\ell}} \leq e^{-n},$$

In particular, this is true for $i = 1$. This means that the algorithm will return the solution with overwhelming probability, as long as the 000 columns are uniformly distributed (i.e. there are $h_l\delta$ columns of type 000 in each slice of size $h_\ell n$).

It remains to show that there exist a permutation $Q$ of the columns of the lists, such that the columns of type 000 in the solution are uniformly distributed. We claim that such a permutation can be found in roughly $n^{(t-1)/2}$ iterations of the PERMUTED Algorithm.

Let $Q$ be a random permutation of the columns of the lists. Let $\delta^*$ be the exact number of columns of type 000 in the solution. We say that $Q$ is "good enough" if after applying $Q$ to $A$, $B$ and $C$, the columns of type 000 are uniformly distributed. In other words, in each slice of $h_\ell n$ columns we would like to have about $s_\ell\delta^*$ columns of type 000. The probability that $Q$ satisfies this condition is given by

$$\begin{aligned}
\Pr\left[Q \text{ good enough}\right] &= \binom{h_1 n}{h_1\delta^*} \cdots \binom{h_t n}{h_t\delta^*} / \binom{n}{\delta^*} \\
&\geq cst \cdot \frac{\sqrt{n}2^{h_1 nH(\delta^*/n)} \cdots 2^{h_t nH(\delta^*/n)}}{\sqrt{h_1 n \ldots h_t n}\, 2^{nH(\delta^*/n)}} \\
&\geq cst \cdot \frac{\sqrt{n}2^{nH(\delta^*/n)}}{\sqrt{(h_1\ldots h_t)n^t}\, 2^{nH(\delta^*/n)}} \\
&\geq cst \cdot n^{(1-t)/2}.
\end{aligned}$$

It follows that the expected number of iteration of the PERMUTED Algorithm is $\mathcal{O}\left(n^{(t-1)/2}\right)$. Making at least $Poly(n)$ iterations of the PERMUTED Algorithm where the degree of the polynomial is greater than $(t-1)/2$ should ensure that a good enough permutation is found. ◀

## B "Practical" Considerations for the Incremental Algorithm

The iterative algorithm outlined in Section 4 may terminate earlier than its direct counterpart of section 3, but it may also do more work (this happens in particular when the chosen filtering threshold weight is "just right" and all previous iterations have been wasted). However, it can be modified to do less work in all circumstances.

Let $A_i$ denote the sub-list formed by all vectors of $A$ of hamming weight exactly $i$ ; $B_i$ and $C_i$ are defined accordingly. This partitions each input list in $n$ parts. The original problem could then be solved by searching the $n^3$ sub-instances $A_i \times B_j \times C_k$ for all $i, j, k$. In fact, some of these sub-instance cannot contain a 3XOR triplet. This is obvious with $i = 1, j = 1$ and $k = 5$ for instance: $A$ and $B$ are too sparse to cancel the heavier $C$. A 3XOR triplet $t$ necessarily belongs to $A_{\mu+\nu} \times B_{\nu+\lambda} \times C_{\mu+\lambda}$, where $\mu, \nu$ and $\lambda$ denote the number of columns of type 110, 101 and 011 of $t$, respectively. This is subject to the constraint that $\mu + \nu + \lambda \leq n$, and there are $\binom{n+3}{3}$ such possibilities.

All these "admissible" sub-instances are not equally likely to contain a solution, and they require a variable amount of time to search. Finding the order in which to process them to minimize the expected running time is a classical scheduling problem, namely that of minimizing the weighted sum of completion times on a single machine ($1||\sum w_i C_i$ in the usual nomenclature). It can be solved optimally in polynomial time using Smith's ratio rule [13]: process the sub-instances by decreasing order of cost-efficiency (probability of success divided by time required). It turns out that the cost-efficiency of searching $A_{\mu+\nu} \times B_{\nu+\lambda} \times C_{\mu+\lambda}$ is very well correlated to $\mu + \nu + \lambda$, which counts the number of non-000 columns of the solution. This yields the following "practical" algorithm.

**Algorithm 5** A refined, more "practical" iterative algorithm.

```
1: function PRACTICAL(A, B, C, w)
2:     Partition A (resp. B and C) by hamming weight into A_0, ..., A_n (resp. B_i, C_i)
3:     for m = 0, ..., n do
4:         for each (μ, ν, λ) such that μ + ν + λ = m do
5:             if μ + ν < w, ν + λ < w and μ + λ < w then
6:                 Search A_{μ+ν} × B_{ν+λ} × C_{μ+λ} using the quadratic algorithm.
7:                 If a solution has been found, report it and stop.
8:     return ⊥
```

With the "if" statement of line 4, the PRACTICAL algorithm succeeds in reporting a solution $(\mathbf{x}^*, \mathbf{y}^*, \mathbf{z}^*)$ if all the three components have hamming weight less than $w$ ; therefore it succeeds at the same conditions than the DIRECT algorithm of section 3 with the same threshold $w$, but faster. Without the "if" statement (or with $w = +\infty$), it always succeeds, but faster than the "iterative" algorithm outlined in section 4. In all cases, the speedup is at most polynomial.

## C Computation of a sparse 3XOR

```
#!/usr/bin/env python3
from hashlib import sha512
```

```
# Des. Codes Cryptogr. 33(2)
a = "Mark Goresky and Andrew Klapper: Periodicity and Correlation Properties of d-FCSR Sequences. (2004)"

# TCC (2) 2015
b = "Ran Canetti, Yael Tauman Kalai and Omer Paneth: On Obfuscation with Random Oracles. (2015)"

# Cryptologia 10(1)
c = "David Kahn: Secrets of the Codebreakers. (1986)"

# CT-RSA 2007
d = "Mario Lamberger, Norbert Pramstaller, Christian Rechberger and Vincent Rijmen: Second Preimages for SMASH. (2007)"

# EUROCRYPT 2013
e = "Patrick Derbez, Pierre-Alain Fouque and Jérémy Jean: Improved Key Recovery Attacks on Reduced-Round AES in the Single-Key Setting. (2013)"

# Financial Cryptography 2003
f = "Javier Herranz and Germán Sáez: Verifiable Secret Sharing for General Access Structures, with Application to Fully Distributed Proxy Signatures. (2003)"

# C2SI 2019
g = "Yongge Wang and Qutaibah M. Malluhi: Reusable Garbled Turing Machines Without FHE. (2019)"

# CRYPTO 2000
h = "Masayuki Abe and Tatsuaki Okamoto: Provably Secure Partially Blind Signatures. (2000)"

# Cryptologia 38(2)
i = "Chris Christensen: The National Cash Register Company Additive Recovery Machine. (2014)"

#  ICISC 2003
j = "Jonathan Katz: Binary Tree Encryption: Constructions and Applications. (2003)"

# CRYPTO 1994
k = "Olivier Delos and Jean-Jacques Quisquater: An Identity-Based Signature Scheme with Bounded Life-Span. (1994)"

# J. Mathematical Cryptology 9(2)
l = "Shlomi Dolev, Juan A. Garay, Niv Gilboa, Vladimir Kolesnikov and Yelena Yuditsky: Towards efficient private distributed computation on unbounded input streams. (2015)"


def H(s):
    """Apply SHA-512 to a string and converts the hash to an integer"""
    return int.from_bytes(sha512(s.encode('utf8')).digest(), byteorder='big')

# An unexpected relationship through SHA-512... with ``Secrets of the Codebreakers'' ?!?
assert (H(a) & H(b) & H(c) & H(d)) ^ (H(e) & H(f) & H(g) & H(h)) ^ (H(i) & H(j) & H(k) & H(l)) == 0
```