



**HAL**  
open science

# Faster Algorithms for the Sparse Random 3XOR Problem

Charles Bouillaguet, Claire Delaplace

► **To cite this version:**

Charles Bouillaguet, Claire Delaplace. Faster Algorithms for the Sparse Random 3XOR Problem. 2019. hal-02306917v1

**HAL Id: hal-02306917**

**<https://hal.science/hal-02306917v1>**

Preprint submitted on 7 Oct 2019 (v1), last revised 2 Oct 2021 (v4)

**HAL** is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

# 1 **Faster Algorithms for the Sparse Random 3XOR**

## 2 **Problem**

3 **Charles Bouillaguet** 

4 University of Lille, France

5 charles.bouillaguet@univ-lille.fr

6 **Claire Delaplace**

7 Ruhr University Bochum, Germany

8 claire.delaplace@rub.de

### 9 **Abstract**

---

10 We present two new algorithms for a variant of the 3XOR problem with lists consisting of  $N$   $n$ -bit  
11 vectors whose coefficients are drawn randomly according to a Bernoulli distribution of parameter  
12  $p < 1/2$ . We show that in this particular context the problem can be solved much more efficiently  
13 than in the general setting. In particular, we present two new algorithms. The first one has a  
14 time complexity which is both  $\mathcal{O}(N^{1+2.583p})$  and  $\mathcal{O}(N^{2-(1-2p)^{2.1}})$ . The second one has a time  
15 complexity which is almost linear in  $N$  for small values of  $p$   $p \leq 0.15$  and has a time complexity of  
16  $\tilde{\mathcal{O}}(N^{2-1.97(1-2p)^{2.37}})$  for  $p > 0.13$ . The analysis of these algorithms reveal a “phase change” for a  
17 certain threshold  $p$ .

18 **2012 ACM Subject Classification** Theory of computation → Computational complexity and cryp-  
19 tography; Theory of computation

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

21 **Funding** *Charles Bouillaguet*: author-specific funding acknowledgements

## 1 Introduction

Given three lists  $A$ ,  $B$  and  $C$  of  $n$ -bit vectors, the 3XOR problem consists in 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).

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 [5]. 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  $\mathcal{O}(N^{2-\epsilon})$  algorithm for the 3XOR problem with input lists of size  $N$  would imply faster-than-expected algorithms for listing triangles in a graph [11, 6]. Another result due to [3] show that an algorithm solving the 3XOR problem in time  $\Omega(n^{2-o(1)})$  also reduces the time complexity of the offline SETDISJOINTNESS and SETINTERSECTION.

The 3XOR problem also has some cryptographic applications, in which the input lists consists of uniformly random vectors (the cryptographic community makes this assumption “by default”). In particular, we can mention Nandi’s attack [10] 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 [8].

May and Both have been considering a variant of the 3XOR problem, the *approximate 3-list birthday problem* where giving three lists of 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 [1].

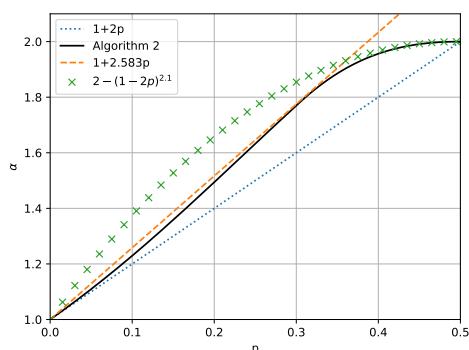
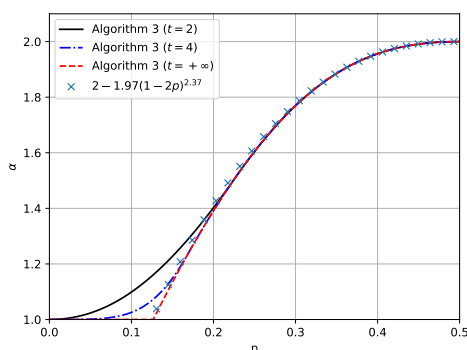
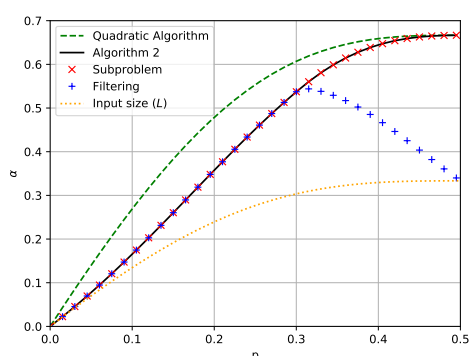
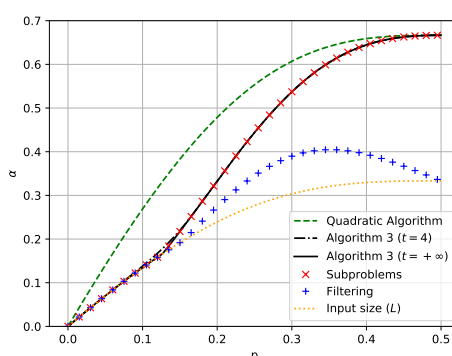
The simplest possible algorithm to solve the 3XOR problem is the quadratic algorithm, which consists in taking all xors  $\mathbf{x} \oplus \mathbf{y} \in A \times B$  and checking whether this belongs to the last list  $C$ . Using an optimal static dictionary [4] to hold  $C$ , this results in a time complexity of  $\mathcal{O}(|A||B| + |C|)$ . In the particular case where  $|A| = |B| = |C| = N$ , so that only one solution exists (with high probability) this algorithm runs in time  $\mathcal{O}(N^2)$ .

We focus on the case where  $N$  is such that there is one and only one solution with large probability. In the case where the vectors are drawn uniformly at random in  $\{0, 1\}^n$ , this means that  $N = 2^{n/3}$ . In this particular case, the quadratic algorithm is mostly the only option to recover the solution. Some improvements of this method exist [2, 3], however these improvements allows only to gain a polynomial factor in  $n$  compared to the quadratic algorithm. It is not clear today whether it is possible to find an algorithm for this problem with complexity below  $N^{2-o(1)}$ .

To some extent, the problem we consider in this paper is dual to the “approximate 3-list birthday problem” [1]: starting from (dense) random lists  $A, B, C$ , this asks for *approximate* 3XOR triplets  $(\mathbf{x}, \mathbf{y}, \mathbf{z})$  — triplets that approximately sum to zero, (*i.e.* such that the hamming weight of  $\mathbf{x} \oplus \mathbf{y} \oplus \mathbf{z}$  is small). Here, we start from random input lists with vectors of small hamming weight, and we want to find an exact match — an actual 3XOR triplet.

**Contributions.** In this paper, we focus on a variant of the 3XOR problem, where the elements of the lists are sparse and random. More precisely, each bit of each vector is drawn at random according to a Bernoulli distribution of parameter  $p < 1/2$  (the “dense” random case corresponds to  $p = 1/2$ ). A first consequence is that (exponentially) smaller input lists are sufficient to ensure the existence of a 3XOR triplet with high probability.

As a second consequence, we show that the 3XOR problem can be solved much faster than  $\mathcal{O}(N^2)$ . We take advantage of the fact that the proportion of indices  $j$  such that our

(a) Algorithm 2 (the complexity is  $N^\alpha$ ).(b) Algorithm 3 (the complexity is  $N^\alpha$ ).(c) Algorithm 2 (the complexity is  $2^{\alpha n}$ ).(d) Algorithm 3 (the complexity is  $2^{\alpha n}$ ).

■ **Figure 1** Complexities of our Algorithms.

68 solution  $(\mathbf{x}, \mathbf{y}, \mathbf{z})$  satisfies  $\mathbf{x}_j = \mathbf{y}_j = \mathbf{z}_j = 0$  is greater than  $1/4$  in expectation to come up  
 69 with two new algorithms described in sections 3 and 3.

70 In the first one, we select randomly a subset  $J$  of the indices, guess that the solution  
 71 satisfies  $\mathbf{x}_j = \mathbf{y}_j = \mathbf{z}_j = 0$  for all  $j \in J$ . From here, we consider the sublists  $A'$ ,  $B'$  and  $C'$   
 72 of  $A, B$ , and  $C$ , consisting only of vectors whose coefficients indexed by  $j \in J$  are zero. We  
 73 solve this smaller instance with the quadratic algorithm. If no solution is found, we restart.

74 In our second algorithm, we borrow the main technique of the “nearest neighbors”  
 75 algorithm of May and Ozerov [9] (which is an algorithm to decode linear codes). Given a  
 76 parameter  $t$ , we split the indices in  $t$  slices. We select randomly a subset  $J_1$  of the indices  
 77 belonging to the first slice and guess that the solution satisfies  $\mathbf{x}_j = \mathbf{y}_j = \mathbf{z}_j = 0$  for all  
 78  $j \in J_1$ . We then build the sublists  $A_1, B_1, C_1$  of the vectors whose coefficients indexed by  
 79  $j \in J_1$  are zero. After that we select a random subsets  $J_2$  of the indices belonging to the  
 80 second slice and build the sublists  $A_2, B_2, C_2$  of  $A_1, B_1, C_1$  whose coefficients indexed by  
 81  $j \in J_2$  are zero and so on an so forth, until we obtain the lists  $A_t, B_t, C_t$ , which we process  
 82 with the quadratic algorithm. The trick is that, if one of our guess  $J_k$  was wrong, we do not  
 83 have to restart the whole guess, but only starting from  $J_k$ .

84 These algorithms are polynomial in  $N$  (the size of the lists) and exponential in  $n$  (the  
 85 number of bits of the input vectors). In both case, we focus on the exponent in the complexity,  
 86 and we disregard all lower-order terms. The complexities of the two algorithms given in this  
 87 paper are shown in fig. 1. Taking  $p = 1/8$ , for instance, three lists of  $2^{0.165n}$  should contain

88 at least one solution. Finding it with the quadratic algorithm would take time  $2^{0.33n}$ . Our  
 89 first algorithm finds it in time  $2^{0.213n}$ , while our second algorithm finds it in time  $2^{0.17n}$ .

90 **Organisation** We recall the computational model as well as important properties regarding  
 91 the distribution of the zeroes and ones in the triplets in Section 2. Then, we present our  
 92 first new algorithm and study its complexity in Section 3. Finally we present the second  
 93 algorithm and its complexity in Section 4.

## 94 2 Preliminaries

### 95 2.1 Notations, Definition and Useful Properties

96 Let  $\mathbf{x}$  be an  $n$ -bit string. We have  $\mathbf{x} = x_0x_1 \dots x_{n-1}$ . Let  $A$  be a list we denote by  $|A|$  the  
 97 size of  $A$ , that is the number of elements in  $A$ . For any triplet  $(\mathbf{x}, \mathbf{y}, \mathbf{z}) \in \{0, 1\}^n$ , and any  
 98 index  $0 \leq j < n$ , we call *type* of the column  $j$  the 3-bit string  $x_jy_jz_j$ . For any  $\mathbf{x} = x_0 \dots x_{n-1}$ ,  
 99 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  
 100  $[0 : n]$ , we denote by  $\mathbf{x}_{\setminus J}$  the sub-string of  $\mathbf{x}$ , where all  $x_j$  for  $j \in J$  have been discarded.

101 Let us now define formally our version of the 3XOR problem.

102 ► **Definition 1** (3XOR triplet). *Let  $(\mathbf{x}, \mathbf{y}, \mathbf{z}) \in \{0, 1\}^n$ , we say that the triplet  $(\mathbf{x}, \mathbf{y}, \mathbf{z})$  is a*  
 103 *3XOR triplet if  $\mathbf{x} \oplus \mathbf{y} \oplus \mathbf{z} = 0$ .*

104 ► **Definition 2** (3XOR problem with distribution  $\mathcal{D}$ ). *Let  $\mathcal{D}$  be a distribution over  $\{0, 1\}^n$ . Let*  
 105  *$A, B$  and  $C$  be three lists of elements drawn independently in  $\{0, 1\}^n$ , according to  $\mathcal{D}$ . A*  
 106 *solution to the 3XOR problem is a 3XOR triplet  $(\mathbf{x}, \mathbf{y}, \mathbf{z}) \in A \times B \times C$ .*

107 We say that  $n$  is the *dimension* of the problem.

108 We denote by  $\log$  the logarithm in basis 2, by  $Ber_p$  the Bernoulli distribution of parameter  
 109  $p$ , and by  $H$  the binary entropy function, meaning that  $H(x) = -x \log(x) - (1-x) \log(1-x)$ ,  
 110 for all  $0 < x < 1$ . The following standard approximation for the binomial coefficient can be  
 111 derived from Stirling's formula:

$$112 \quad \frac{2^{nH(x)}}{\sqrt{8nx(1-x)}} \leq \binom{n}{xn} \leq \frac{2^{nH(x)}}{\sqrt{2\pi nx(1-x)}}, \quad (0 < x < 1/2) \quad (1)$$

113 We denote by the notation *cst* any constant. We also use the following result.

114 ► **Lemma 3** (Chernoff's lower bound). *Let  $X_1 \dots X_n$  be independent random variables taking*  
 115 *values in  $\{0, 1\}$ , and let  $X = \sum_{i=1}^n X_i$ , then for every  $0 < \epsilon < 1$ ,*

$$116 \quad \mathbb{P}[X \leq (1 - \epsilon)\mathbb{E}[X]] \leq e^{-\frac{\epsilon^2\mathbb{E}[X]}{2}}.$$

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

123 **The Quadratic Algorithm for 3XOR** For the sake of completeness, we recall the Quadratic  
 124 Algorithm for 3XOR in Algorithm 1.

**Algorithm 1** QuadraticAlgorithm

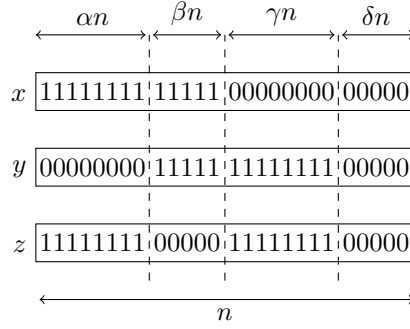
---

```

1: function QUADRATICALGORITHM(A,B,C)
2:   // Returns a 3XOR triplet  $(\mathbf{x}, \mathbf{y}, \mathbf{z}) \in A \times B \times C$  or  $\perp$  if none exist.
3:   Initialize a static dictionary  $\mathcal{C}$  with the content of  $C$ .
4:   for all  $\mathbf{x}, \mathbf{y} \in A \times B$  do
5:     if  $(\mathbf{x} \oplus \mathbf{y}) \in \mathcal{C}$  then return  $(\mathbf{x}, \mathbf{y}, \mathbf{v})$ 
6:   return  $\perp$ 

```

---



■ **Figure 2** Shape of a sparse random 3XOR triplet, up to column permutation.

## 2.2 Structural Properties of Sparse 3XOR Triplets

125

126 Let  $(\mathbf{x}, \mathbf{y}, \mathbf{z})$  be a 3XOR triplet. For each column  $j$ , the type  $\mathbf{t}_j$  of the  $j$ -th column belongs  
 127 to  $\{000, 011, 101, 110\}$ . Let  $\alpha, \beta, \gamma, \delta \in [0, 1]$  be four parameters satisfying  $\alpha + \beta + \gamma + \delta = 1$   
 128 and such that  $\alpha n$  columns are of type 101,  $\beta n$  columns are of type 110,  $\gamma n$  are of type 000.  
 129 Modulo columns permutations, the shape of a 3XOR triplet can be described by Figure 2.  
 130 We claim that if  $\mathbf{x}$ ,  $\mathbf{y}$  and  $\mathbf{z}$  are all drawn from the uniform distribution over  $\{0, 1\}^n$ , then  
 131  $\mathbb{E}[\alpha] = \mathbb{E}[\beta] = \mathbb{E}[\gamma] = \mathbb{E}[\delta] = \frac{1}{4}$ .

132

133 Let  $\mathcal{D}$  the distribution over  $\{0, 1\}^n$ , where each bit is drawn independently from  $Ber_p$ ,  
 134 with  $p < \frac{1}{2}$ . Let  $(\mathbf{x}, \mathbf{y}, \mathbf{z})$  be a triplet of  $n$ -bit vectors drawn from  $\mathcal{D}$  (i.e.  $\mathbb{P}[x_j = 1] = \mathbb{P}[y_j =$   
 135  $1] = \mathbb{P}[z_j = 1] = p$  for all  $j$ ). Then, for a given column  $j$  of type  $\mathbf{t}_j$ , we have

$$\begin{aligned}
 136 \quad \mathbb{P}[x_j \oplus y_j \oplus z_j = 0] &= \mathbb{P}[\mathbf{t}_j = 000] + \mathbb{P}[\mathbf{t}_j = 011] + \mathbb{P}[\mathbf{t}_j = 101] + \mathbb{P}[\mathbf{t}_j = 110] \\
 137 \quad &= (1-p)(1-2p+4p^2)
 \end{aligned}$$

139 It follows that  $\mathbb{P}[\mathbf{x} \oplus \mathbf{y} \oplus \mathbf{z} = 0] = (1-p)^n(1-2p+4p^2)^n$ . Now let us consider three lists  
 140  $A$ ,  $B$  and  $C$ , each consisting of  $N$   $n$ -bit vectors whose coordinates are drawn independently  
 141 from  $Ber_p$ . The expected number  $S$  of 3XOR triplets in  $A \times B \times C$  is

$$142 \quad \mathbb{E}[S] = \frac{N^3}{(1-p)^n(1-2p+4p^2)^n}.$$

143 As such, as soon as  $N \geq (1-p)^{-n/3}(1-2p+4p^2)^{-n/3}$ , there will be a 3XOR triplet in  
 144  $A \times B \times C$  with high probability. Let us therefore define  $L = -\frac{1}{3} \log(1-p)(1-2p+4p^2)$  and  
 145  $N = 2^{nL}$ . The algorithms we describe take as input three lists  $A$ ,  $B$  and  $C$ , each of  $N$  elements  
 146 drawn independently from  $\mathcal{D}$ . Their goal is to find a 3XOR triplet  $(x, y, z) \in A \times B \times C$ .

## 6 Faster Algorithms for the Sparse Random 3XOR Problem

147 Let us now consider a sparse random 3XOR triplet  $(\mathbf{x}, \mathbf{y}, \mathbf{z})$ . For a given column  $j$ , we  
 148 would like to estimate the probability that  $j$  is of a certain type. The only possibilities are  
 149 000, 011, 101, 110. Furthermore

$$150 \quad \mathbb{P}[\mathbf{t}_j = 110 \mid 3\text{-xor}] = \mathbb{P}[\mathbf{t}_j = 101 \mid 3\text{-xor}] = \mathbb{P}[011 \mid 3\text{-xor}] = p^2/(1 - 2p + 4p^2),$$

$$151 \quad \mathbb{P}[\mathbf{t}_j = 000 \mid 3\text{-xor}] = (1 - p)^2/(1 - 2p + 4p^2).$$

153 As  $0 < p < \frac{1}{2}$ , it follows that the most common column type in a sparse random 3XOR triplet  
 154 is 000. For the remaining of this paper, we denote by  $\delta$  the quantity  $(1 - p)^2/(1 - 2p + 4p^2)$ :  
 155 its the expected proportion of 000 columns in a random sparse 3XOR triplet.

156 We take advantage of this column repartition to propose several new algorithms to solve  
 157 the following problem.

### 3 A Simple Sparse 3-XOR Algorithm

159 For the sake of simplicity, let us assume that such a 3XOR triplet exists in the input lists  
 160 and denote it by  $(\mathbf{x}^*, \mathbf{y}^*, \mathbf{z}^*)$ . Let  $\delta^*$  denote the proportion of 000 columns in this triplet.  
 161 Because of the randomness of the input lists, we assume that  $(\mathbf{x}^*, \mathbf{y}^*, \mathbf{z}^*)$  is a random 3XOR  
 162 triplet. As such, we expect to find  $\mathbb{E}[\delta^*] = \delta$ .

163 The algorithms described in this section exploit the same underlying idea exposed in  
 164 section 2.2: because the most frequent ‘‘column type’’ in a 3XOR triplet is 000, we try to  
 165 reduce the size of the instance by guessing that the 3XOR triplet contained in the input is  
 166 000 on some columns.

167 Let  $\delta_0$  be a parameter chosen such that with overwhelming probability, at least  $\delta_0 n$  columns  
 168 are of type 000 in  $(\mathbf{x}^*, \mathbf{y}^*, \mathbf{z}^*)$ . Using the Chernoff bound, we may pick  $\delta_0 = (1 - n^{-1/3}) \delta$ .

---

**Algorithm 2** A Simple Sparse Random 3XOR Algorithm.

---

```

1: function FILTER(L, J)
2:   // Return the sublist made of vectors which are 0 on the columns in J.
3:   return  $\{\mathbf{x}_{\setminus J} \mid (\mathbf{x} \in L) \wedge (\forall j \in J, x_j = 0)\}$ 
4: function CASHEW(A, B, C)
5:   // Returns a 3XOR triplet w.h.p. if there is one in  $A \times B \times C$ .
6:   repeat
7:      $J \leftarrow$  uniformly random subset of  $\{1, 2, \dots, n\}$  of size  $\kappa \delta_0 n$ .
8:      $A' \leftarrow$  FILTER(A, J),  $B' \leftarrow$  FILTER(B, J),  $C' \leftarrow$  FILTER(C, J).
9:      $S \leftarrow$  QUADRATICALGORITHM( $A', B', C'$ ).
10:  until  $S \neq \perp$ 
11:  return S

```

---

169 Algorithm 2 is inspired by information set decoding techniques (notably by the Lee-  
 170 Brickell [7] algorithm). It takes a parameter  $0 \leq \kappa \leq 1$ , whose value is discussed below.

171 **► Theorem 4.** *The expected time complexity of CASHEW is*

- 172 *i)  $\Omega(N^{1+2p})$  and  $\mathcal{O}(N^{1+2.583p})$*   
 173 *ii)  $\mathcal{O}(N^{2-(1-2p)^{2.1}})$*

174 The rest of this section is devoted to proving theorem 4. Let us denote:

$$175 \quad I = H(\kappa \delta_0) - \delta^* H(\kappa \delta_0 / \delta^*)$$

$$176 \quad R = -\kappa \delta_0 \log_2(1 - p)$$

177

- 178 ► **Lemma 5.** *i)* The expected total time spent in FILTER is  $\mathcal{O}(2^{n(I+L)})$ .  
 179 *ii)* The expected total time spent in QUADRATICALGORITHM is  $\mathcal{O}(2^{n(I+2(L-R))})$ .  
 180 *iii)* The expected complexity of CASHEW is  $\mathcal{O}(2^{n(I+L)} + 2^{n(I+2(L-R))})$ .

181 **Proof.** We first observe that FILTER is linear in the size of its input. The probability that  
 182 the choice of  $J$  is compatible with  $(\mathbf{x}^*, \mathbf{y}^*, \mathbf{z}^*)$  is:

$$183 \quad \mathbb{P}[x_j^* = y_j^* = z_j^* = 0 \text{ for all } j \in J] = \binom{\delta^* n}{\kappa \delta_0 n} / \binom{n}{\kappa \delta_0 n}.$$

184 The expected number of iterations of CASHEW is the inverse of this probability, and thanks  
 185 to eq. (1), it is upper-bounded by:

$$186 \quad \mathbb{E}[\# \text{ iterations}] \leq \frac{2^{nH(\kappa \delta_0)}}{\sqrt{2\pi n \kappa \delta_0 (1 - \kappa \delta_0)}} / \frac{2^{n\delta^* H(\kappa \delta_0 / \delta^*)}}{\sqrt{8n \kappa \delta_0 / \delta^* (1 - \kappa \delta_0 / \delta^*)}}$$

$$187 \quad \leq cst \times 2^{n[H(\kappa \delta_0) - \delta^* H(\kappa \delta_0 / \delta^*)]}$$

188 This establishes point *i*).

189 We now estimate the size of the subproblems  $(A', B', C')$ . A random sparse vector of  
 190 density  $p$  is zero on the  $\kappa \delta_0 n$  columns chosen in  $J$  with probability  $q = (1 - p)^{\kappa \delta_0 n}$ . Because  
 191  $A, B$  and  $C$  are made of independent and uniformly random sparse vectors of density  $p$ , we  
 192 find that  $\mathbb{E}[|A'|] = q|A|$  (an identical result holds for  $B'$  and  $C'$ ). It follows that the expected  
 193 complexity of solving a single subproblem is  $|A'| \times |B'| = q^2 N^2$ . This establishes point *ii*).  
 194 Point *iii*) is a trivial consequence of *i*) and *ii*). ◀

196 By choosing the value of  $\kappa$ , we can adjust the expecting number of iterations and the  
 197 size of the subproblems. A quick examination reveals that  $I + L$  is an increasing function  
 198 of  $\kappa$ , while  $I + 2(L - R)$  first decreases to a minimum then increases again when  $\kappa$  varies  
 199 between 0 and 1. It turns out that there are two distinct situations, depending on  $p$ : the  
 200 best option is either to balance the cost of filtering and solving the subproblems (for small  $p$ )  
 201 or to minimize the total time spent dealing with the subproblems (for large  $p$ ).

202 ► **Lemma 6.** *There exist a constant  $p^* = 0.30534\dots$  such that the value of  $\kappa$  minimizing the*  
 203 *expected running time of CASHEW is:*

$$204 \quad \kappa = \begin{cases} -L/(2\delta \log 1 - p) & \text{when } p < p^* \\ 4 + 6/(p - 2) & \text{when } p > p^* \end{cases}$$

205 *The threshold  $p^*$  is the single value that equates the two alternatives.*

206 **Proof.** Let us consider the expected values of  $I$  and  $R$  over the random choice of a 3XOR  
 207 triplet (i.e. assuming that  $\delta^* = \delta_0 = \delta$ ):

$$208 \quad I' = H(\kappa \delta) - \delta H(\kappa)$$

$$209 \quad R' = -\kappa \delta \log_2(1 - p)$$

211 It follows from lemma 5 that the optimum expected value of  $\kappa$  is the one that minimizes  
 212  $\max\{I' + L, I' + 2(L - R')\}$  — this minimizes the exponent in the complexity.

213 Let  $\kappa_1$  denote the solution of  $I' + L' = I' + 2(L - R')$  (this is the value of  $\kappa$  that  
 214 balances the cost of filtering and solving the subproblems). Let  $\kappa_2$  denote the solution of  
 215  $\partial(I' - 2R')/\partial\kappa = 0$  (this is the value of  $\kappa$  that minimizes the total expected time required to  
 216 solve the subproblems).



217 When  $\kappa_1 < \kappa_2$ , then  $\kappa_1$  minimizes the expected running time of CASHEW. Indeed, taking  
 218  $\kappa$  below  $\kappa_1$  increases the complexity of solving the subproblems (which is decreasing below  
 219  $\kappa_2$ ); taking  $\kappa > \kappa_1$  increases the complexity of the filtering step (which is always increasing).

220 When  $\kappa_1 > \kappa_2$ , then  $\kappa_2$  is the optimal value: moving beyond  $\kappa_2$  increases the complexity  
 221 of both steps. Because the cost of the filtering step is increasing as a function of  $\kappa$ , it is  
 222 smaller than that of solving the subproblems while  $\kappa < \kappa_1$ . As such, solving the subproblems  
 223 dominate the complexity, and the cost of this step is minimum when  $\kappa = \kappa_2$ .

224 A somewhat tedious calculation shows that  $\kappa_1 = -L/(2\delta \log 1 - p)$  and  $\kappa_2 = 4 + 6/(p - 2)$ .  
 225 Another tedious calculation shows that  $\kappa_1 < \kappa_2 \iff p < p^*$ , where  $p^*$  is the solution  $\blacktriangleleft$

226 Theorem 4 can be established by taking the optimal value of  $\kappa$  given by lemma 6, and  
 227 verifying numerically that the resulting complexity indeed satisfies the announced bounds.  
 228 This is visible on Fig. 1.

## 229 **4 Improved Filtering à la May and Ozerov**

230 In this section we describe an improved algorithm, which is highly inspired by that of May  
 231 and Ozerov for Information Set Decoding [9].

232 Let  $(\mathbf{x}^*, \mathbf{y}^*, \mathbf{z}^*)$  be a 3XOR triplet in  $A \times B \times C$  and let  $\delta^*$  be the proportion of columns  
 233 of type 000 in this triplet. Let us fix a constant  $t$  and positive reals  $s_1, \dots, s_t$  such that  
 234  $s_1 + \dots + s_t = 1$ . We again consider the parameter  $\delta_0$  such that, with overwhelming probability,  
 235 at least  $\delta_0 n$  columns are of type 000 in  $(\mathbf{x}^*, \mathbf{y}^*, \mathbf{z}^*)$ .

236 As in section 3, we define  $I = H(\delta_0 \kappa) - \delta_0 H(\kappa)$  and  $R = -\log(1 - p)\kappa\delta_0$ .

237 We assume that the columns of type 000 are uniformly distributed in  $[0, n - 1]$ . If not,  
 238 we can randomize the instance by permutating the columns randomly. After a polynomial  
 239 number of such permutation, the columns of type 000 should be uniformly distributed. Here  
 240 is the main idea of the method.

- 241 1. Choose a random subset  $J_1$  of  $s_1 \delta_0 n$  columns among the first  $s_1 n$ , and assume that for  
 242 all  $j \in J_1$ , the type  $\mathbf{t}_j$  of the column  $j$  is  $\mathbf{t}_j = 000$ .
- 243 2. Compute the sub-lists  $A_1, B_1, C_1$  such that for all  $\mathbf{x} \in A_1$ , (resp.  $B_1, C_1$ )  $x_j = 0$ .
- 244 3. Compute sublists  $A_2, B_2, C_2$  recursively in a similar way, and so on and so forth until we  
 245 came up with small sublists  $A_t, B_t, C_t$ .
- 246 4. Solve the instance with  $A_t, B_t, C_t$  using the quadratic algorithm.

247 Now let  $1 \leq i_0 \leq t$  be the first index for which our choice of the set  $J_{i_0}$  is wrong (i.e.  
 248  $\exists j \in J_{i_0}$  such that  $\mathbf{t}_j \neq 000$  in the solution). We do not have to restart computing the lists  
 249  $A_{i_0-1}, B_{i_0-1}, C_{i_0-1}$ , but we only have to restart the computation starting from  $A_{i_0}, B_{i_0}, C_{i_0}$ .

250 More formally, this yields Algorithm 3. The remainder of this section is devoted to  
 251 establishing its complexity.

252 **► Theorem 7.** *For any constant  $t$ , there exists  $s_1, \dots, s_t$  and a polynomial  $P$  such that*  
 253 *PISTACHIO finds a solution in expected time*

$$254 \quad P(n) \left[ 2^{n \left( L + I \frac{1 - (R/I)}{1 - (R/I)^t} \right)} + 2^{n(I + 2(L - R))} \right].$$

255 **► Lemma 8.** *Let  $(\mathbf{x}^*, \mathbf{y}^*, \mathbf{z}^*)$  be a 3XOR triplet in  $A \times B \times C$ . We assume that this is the*  
 256 *only 3XOR triplet in  $A \times B \times C$ . Let  $\delta^*$  denote the proportion of columns of type 000 in*  
 257  *$(\mathbf{x}^*, \mathbf{y}^*, \mathbf{z}^*)$ . Assume that these columns are uniformly distributed among all the columns of*  
 258  *$(\mathbf{x}^*, \mathbf{y}^*, \mathbf{z}^*)$ ; then  $\text{PEANUT}(A, B, C, 1)$  returns  $(\mathbf{x}^*, \mathbf{y}^*, \mathbf{z}^*)$  with overwhelming probability.*

**Algorithm 3** Sparse Random 3XOR with Improved Filtering

---

```

1: function PISTACHIO(A, B, C)
2:   // Returns a 3XOR triplet in  $A \times B \times C$  w.h.p. or  $\perp$  if none exists
3:   do
4:      $q \leftarrow$  random permutation of  $\{0, \dots, n-1\}$ 
5:      $A \leftarrow q(A), B \leftarrow q(B), C \leftarrow q(C)$   $\triangleright$  Permute randomly the columns of the lists
6:      $S \leftarrow$  PEANUT(A, B, C, 1)
7:   while  $S = \perp$ 
8:   return S

9: function PEANUT(A, B, C, i)
10:  if  $i = t + 1$  then
11:    return QUADRATICALGORITHM(A, B, C)
12:  else
13:    for  $0 \leq \ell < n2^{ns_i I}$  do
14:       $J \leftarrow$  random subset of  $\kappa\delta_0 s_i n$  columns in  $[(s_1 + \dots + s_{i-1})n : (s_1 + \dots + s_i)n]$ 
15:       $A' \leftarrow$  FILTER(A, J),  $B' \leftarrow$  FILTER(B, J),  $C' \leftarrow$  FILTER(C, J)
16:       $S \leftarrow$  PEANUT(A', B', C', i + 1)
17:    if  $S \neq \perp$  then
18:      return S
19:  return  $\perp$ 

```

---

259 **Proof.** Let  $A_i, B_i, C_i$  denote the lists that are taken as input by PEANUT alongside with the  
260 index  $i$ . Let  $\mathbf{x}_i^*$  denote the vector  $\mathbf{x}_{I_i}^*$ , where  $I_i = J_1 \cup \dots \cup J_{i-1}$ , for  $i > 1$  and  $I_1 = \emptyset$ . We  
261 define  $\mathbf{y}_i^*$  and  $\mathbf{z}_i^*$  accordingly.

262 Let us denote by  $\pi_i$  the probability that the triplet  $(\mathbf{x}_{i+1}^*, \mathbf{y}_{i+1}^*, \mathbf{z}_{i+1}^*)$  is in  $A_{i+1} \times B_{i+1} \times$   
263  $C_{i+1}$ , knowing that  $(\mathbf{x}_i^*, \mathbf{y}_i^*, \mathbf{z}_i^*)$  is in  $A_i \times B_i \times C_i$ . Assuming a uniform distribution of the  
264 columns of type 000, we have (using (1)):

$$\begin{aligned}
265 \quad \pi_i &= \binom{s_i \delta^* n}{\kappa s_i \delta_0 n} / \binom{s_i n}{\kappa s_i \delta_0 n} \geq \binom{s_i \delta_0 n}{\kappa s_i \delta_0} / \binom{s_i n}{\kappa s_i \delta_0 n} \\
266 \quad &\geq cst \cdot \frac{s_i n 2^{\delta_0 s_i n H(\kappa)}}{\delta_0 s_i n 2^{s_i n H(\kappa \delta_0)}} = cst \cdot \frac{2^{s_i n (\delta_0 H(\kappa) - H(\kappa \delta_0))}}{\delta_0} \approx 2^{s_i n (\delta_0 H(\kappa) - H(\kappa \delta_0))}.
\end{aligned}$$

268 At each step  $i$ , if the solution is not found, we restart the procedure up to  $\approx n/\pi_i$  times.  
269 Assuming that  $(x_i^*, y_i^*, z_i^*)$  is in  $A_i, B_i, C_i$ , the probability that we do not find the solution  
270 after this many call to the procedure is

$$271 \quad \mathbb{P}[\text{fail at step } i] \approx (1 - \pi_i)^{\frac{n}{\pi_i}} \leq e^{-n},$$

272 In particular, this is true for  $i = 1$ . This means that the algorithm will return the solution  
273 with overwhelming probability, as long as the 000 columns are uniformly distributed.  $\blacktriangleleft$

274  $\blacktriangleright$  **Lemma 9.** *The expected size of the input lists  $A_i, B_i, C_i$  of PEANUT at step  $i$  is:*

$$275 \quad N_i = 2^n \left( L - R \sum_{j=1}^{i-1} s_j \right).$$

276 **Proof.** The probability that a vector of size greater than  $s_i n$ , whose coefficients are drawn  
277 from  $Ber_p$  is 0 on  $\kappa \delta_0 s_i n$  arbitrary fixed columns is

$$278 \quad q_i = (1 - p)^{\kappa \delta_0 s_i n} = 2^{\kappa \delta_0 s_i n \log(1-p)} = 2^{-R s_i n}.$$

## 10 Faster Algorithms for the Sparse Random 3XOR Problem

279 It follows that the expected size  $N_{i+1}$  of the list  $A_{i+1}, B_{i+1}, C_{i+1}$ , for  $i \geq 1$  is given by

$$280 \quad \mathbb{E}[N_{i+1}] = q_i N_i = N_1 \prod_{j=1}^i q_j N_1 = 2^{-n(L-R \sum_{j=1}^i s_j)}$$

281

282 **► Lemma 10.** *The number of iterations of PISTACHIO is  $\mathcal{O}(n^{(t-1)/2})$ .*

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

$$287 \quad \begin{aligned} \mathbb{P}[Q \text{ good enough}] &= \binom{s_1 n}{s_1 \delta^* n} \cdots \binom{s_t n}{s_t \delta^* n} / \binom{n}{\delta^* n} \\ 288 &\geq cst \cdot \frac{\sqrt{n} 2^{s_1 n H(\delta^*)} \cdots 2^{s_t n H(\delta^*)}}{\sqrt{s_1 n \cdots s_t n} 2^{n H(\delta^*)}} \\ 289 &\geq cst \cdot \frac{\sqrt{n} 2^{n H(\delta^*)}}{\sqrt{(s_1 \cdots s_t) n^t} 2^{n H(\delta^*)}} \\ 290 &\geq cst \cdot n^{(1-t)/2}. \end{aligned}$$

291  
292 It follows that the expected number of iteration of PISTACHIO is  $\mathcal{O}(n^{(t-1)/2})$ . ◀

293 We are now ready to prove Theorem 7.

294 **Proof.** Let us denote by  $C_i$  the time complexity of one iteration  $i$  of PEANUT. In particular,  
295 we have

$$296 \quad \begin{aligned} C_{t+1} &= \mathcal{O}(N_{t+1}^2) \\ 297 \quad C_i &= \mathcal{O}(2^{n s_i I} (N_i + C_{i+1})), \quad \forall i \leq t \end{aligned} \tag{2}$$

298 where  $N_i$  is the size of the lists at the beginning of Step  $i$ . Furthermore, Step  $i$  of the  
299 procedure has to be restarted at most  $2^{n I \sum_{j=1}^{i-1} s_j}$  times. Let us denote by  $T_i$  the value  
300  $2^{n I \sum_{j=1}^{i-1} C_j}$ . From (2), we have

$$301 \quad T_i = \mathcal{O}\left(2^{n I \sum_{j=1}^i s_j} N_i\right) + T_{i+1}.$$

302 From Lemma 9, this is

$$303 \quad T_i = \mathcal{O}\left(2^{n(L + I s_i + (I-R) \sum_{j=1}^{i-1} s_j)}\right) + T_{j+1}.$$

304 For all  $1 \leq i \leq t-1$ , we want the following to hold

$$305 \quad n \left( L + I s_i + (I-R) \sum_{j=1}^{i-1} s_j \right) = n \left( L + I s_{i+1} + (I-R) \sum_{j=1}^i s_j \right).$$

306 This would mean that the time spent filtering the lists is mostly the same in all levels  $i$ .  
307 After simplification, this gives

$$308 \quad s_{i+1} = \frac{R}{I} s_i = \frac{R^i}{I^i} s_1.$$

309

311 The condition  $\sum_i s_i = 1$  ensures that  $s_1 = \frac{1-(R/I)}{1-(R/I)^t}$ . It follows that for all  $i$ ,

$$312 \quad T_i = \mathcal{O}\left(2^{n(L+Is_1)}\right) + T_{i+1} = \mathcal{O}\left(2^{n\left(L+I^t \frac{I-R}{I^t-R^t}\right)}\right) + T_{i+1}.$$

313 In particular, as  $t$  is a constant, this implies that

$$314 \quad C_1 = T_1 = \mathcal{O}\left(2^{n\left(L+I^t \frac{I-R}{I^t-R^t}\right)}\right) + T_{n+1}$$

315 and

$$316 \quad T_{n+1} = \mathcal{O}\left(2^{nI \sum_{i=1}^t s_i} 2^{2n\left(L-R \sum_{i=1}^t s_i\right)}\right) = 2^{n(I+2(L-R))}.$$

317 It follows that

$$318 \quad C_1 = \mathcal{O}\left(2^{n\left(L+I^t \frac{I-R}{I^t-R^t}\right)} + 2^{n(I+2(L-R))}\right).$$

319 We use Lemma 10 to obtain the claimed complexity. ◀

### 320 **Tuning for Maximum Speed**

321 It remains to choose a value of  $\kappa$  that minimizes the running time. This time,  $\kappa$  depends on  
322  $p$  and  $t$ . We can essentially replay the analysis in lemma 6, but everything becomes messier  
323 because of  $t$ .

324 Let  $\kappa_1$  be the value that balances the cost of filtering and solving the subproblems in  
325 PEANUT. It can be seen that this is the solution (in  $\kappa$ ) of:

$$326 \quad (1 - (R/I)^{t-1}) / (1 - (R/I)^t) = 2 - L/R. \quad (3)$$

327 Barring anything else, this equation can be solved numerically. This allows to evaluate  
328 the exponent in the complexity. Let  $\kappa_2$  denote the value that minimizes the total time spent  
329 solving subproblems. As previously, we find that  $\kappa_2 = 4 + 6/(p-2)$ , and that the best value  
330 of  $\kappa$  is the minimum of  $\kappa_1$  and  $\kappa_2$ .

331 To make things more concrete, we will consider two settings: a “small” value of  $t$  (say,  
332  $t = 4$ ), and a “large” value of  $t$  ( $t = +\infty$ ).

333 For  $t = 4$ , We find that the threshold where  $\kappa_1 = \kappa_2$  is attained for  $p^* = 0.1690\dots$   
334 (otherwise we have  $\kappa_1 < \kappa_2 \Leftrightarrow p < p^*$ ). In other terms, for  $p > p^*$ , the best strategy is to  
335 minimize the total time spent in dealing with the subproblems. We find numerically that  
336 while  $p < 0.169$ , algorithm 3 runs in time less than  $N^{5/4}$ .

337 We now consider the case  $t = \infty$ ; arguably, this is not a realistic value, but we think that  
338 it helps understand the global behavior of the algorithm. We pass to the limit by noting  
339 that when  $t \rightarrow +\infty$ , the function  $f(x) = \frac{1-x}{1-x^t}$  becomes

$$340 \quad \hat{f}(x) = \begin{cases} 1-x & \text{when } x \leq 1 \\ 0 & \text{when } x \geq 1 \end{cases}$$

341 Therefore, balancing the cost of filtering with that of solving the subproblems means finding  
342  $\kappa_1$  satisfying:

$$343 \quad L + I\hat{f}\left(\frac{R}{I}\right) = I + 2(L - R)$$

344 This yields a new threshold  $p^* = 0.1265\dots$ : for  $p < p^*$ , the best value of  $\kappa$  is  $\kappa_1$ , while  
 345 for  $p > p^*$  it is again  $\kappa_2$ . For  $p < p^*$ , with  $\kappa = \kappa_1$ , we find that  $R/I > 1$ , and therefore the  
 346 complexity of the whole algorithm is exactly  $N$  (it is *linear* in its input). In fact, we observe  
 347 that even for smaller (“reasonable”) values of  $t$ , the complexity is very close to  $N$  when  $p$  is  
 348 close enough to zero. We summarise this in the following corollary.

- 349 ► **Corollary 11. 1.** For any  $t \geq 2$ , for all  $\epsilon > 0$ , there exist a threshold  $p_0$  such that for  
 350  $p \leq p_0$ , Algorithm 3 runs in time  $\mathcal{O}(n^{(t-1)/2}N^{1+\epsilon})$   
 351 **2.** In particular, for  $t = 12$  and  $\epsilon = 0.01$ ,  $p_0 \geq 1/8$ .  
 352 **3.** For  $t \geq 10$  and  $p \geq 0.130$ , Algorithm 3 runs in time  $\mathcal{O}(n^{(t-1)/2}N^{2-1.97(1-2p)^{2.37}})$ .

353 Tuning the parameters to minimize the concrete running time on an actual computer to  
 354 solve a given instance with specified values of  $n$  and  $p$  is another problem.

355 To conclude, Algorithm 3 spends less time filtering the lists than Algorithm 2. As such,  
 356 solving the subproblems using the quadratic algorithm dominates the running time for smaller  
 357 values of  $p$  with an optimal choice of  $\kappa$ . It follows that for  $p \geq 0.305\dots$ , both algorithms  
 358 exhibit essentially the same behavior. Algorithm 3 shines for small values of  $p$  (as can be  
 359 seen on Fig 1).

## 360 — References —

- 361 **1** Leif Both and Alexander May. The approximate k-list problem. *IACR Transactions on*  
 362 *Symmetric Cryptology*, 2017(1):380–397, 2017.  
 363 **2** Charles Bouillaguet, Claire Delaplace, and Pierre-Alain Fouque. Revisiting and improving  
 364 algorithms for the 3xor problem. *IACR Transactions on Symmetric Cryptology*, 2018(1):254–  
 365 276, 2018.  
 366 **3** Martin Dietzfelbinger, Philipp Schlag, and Stefan Walzer. A subquadratic algorithm for 3xor.  
 367 *arXiv preprint arXiv:1804.11086*, 2018.  
 368 **4** Michael L. Fredman, János Komlós, and Endre Szemerédi. Storing a sparse table with  $o(1)$   
 369 worst case access time. *J. ACM*, 31(3):538–544, June 1984. URL: [http://doi.acm.org/10.](http://doi.acm.org/10.1145/828.1884)  
 370 [1145/828.1884](http://doi.acm.org/10.1145/828.1884), doi:10.1145/828.1884.  
 371 **5** Anka Gajentaan and Mark Overmars. On a class of  $\mathcal{O}(n^2)$  problems in computational geometry.  
 372 *Computational geometry*, 5(3):165–185, 1995.  
 373 **6** Zahra Jafargholi and Emanuele Viola. 3sum, 3xor, triangles. *CoRR*, abs/1305.3827, 2013.  
 374 URL: <http://arxiv.org/abs/1305.3827>, arXiv:1305.3827.  
 375 **7** Pil Joong Lee and Ernest Brickell. An observation on the security of McEliece’s public-key  
 376 cryptosystem. In *Workshop on the Theory and Application of Cryptographic Techniques*,  
 377 pages 275–280. Springer, 1988.  
 378 **8** Gaëtan Leurent and Ferdinand Sibleyras. Low-memory attacks against two-round even-  
 379 mansour using the 3-xor problem. In Alexandra Boldyreva and Daniele Micciancio, editors,  
 380 *Advances in Cryptology - CRYPTO 2019 - 39th Annual International Cryptology Conference*,  
 381 *Santa Barbara, CA, USA, August 18-22, 2019, Proceedings, Part II*, volume 11693 of *Lecture*  
 382 *Notes in Computer Science*, pages 210–235. Springer, 2019. URL: [https://doi.org/10.1007/](https://doi.org/10.1007/978-3-030-26951-7_8)  
 383 [978-3-030-26951-7\\_8](https://doi.org/10.1007/978-3-030-26951-7_8), doi:10.1007/978-3-030-26951-7\_8.  
 384 **9** Alexander May and Ilya Ozerov. On computing nearest neighbors with applications to decoding  
 385 of binary linear codes. In *EUROCRYPT*, pages 203–228, 2015.  
 386 **10** Mridul Nandi. Revisiting Security Claims of XLS and COPA. *IACR Cryptology ePrint Archive*,  
 387 2015:444, 2015.  
 388 **11** Emanuele Viola. Reducing 3xor to listing triangles, an exposition. Technical report, North-  
 389 eastern University, College of Computer and Information Science, May 2012. Available at  
 390 <http://www.ccs.neu.edu/home/viola/papers/xxx.pdf>.