

COMBINATOIRE ET CHIFFREMENT : LE PÈRE NOËL SECRET ET SES APPLICATIONS

Stanislas MEZUREUX | 33920

2021|22

Travail d'Initiative Personnelle Encadré



1.

POSITION DU PROBLÈME

- ▶ Communications électroniques en plein essor
- ▶ Sécurité du canal de transmission
- ▶ Générateur aléatoire de paires
- ▶ Même nature que vote électronique, reviewing

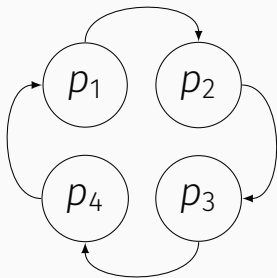


Figure 1 – Un groupe

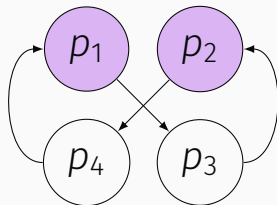


Figure 2 – Deux groupes

- ▶ **Entrée** : liste des participants (nom + contact + groupe)
- ▶ **Sortie** : issue du tirage
- ▶ **Chiffrement** des données



2.

TIRAGE AU SORT

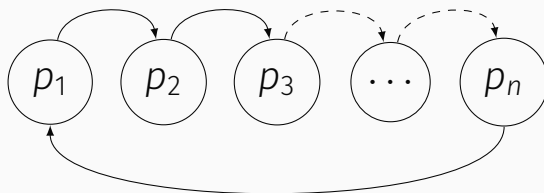


Figure 3 – Un seul groupe, pas de problème d'existence

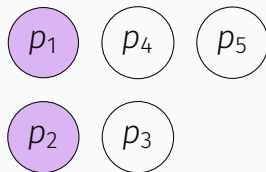


Figure 4 – Cas pathologique

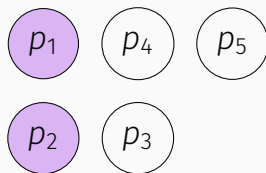


Figure 4 – Cas pathologique

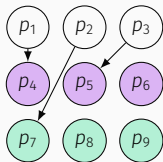
Condition d'existence du tirage

Soit G_1, \dots, G_n les différents groupes.

$$\forall i \in \llbracket 1, n \rrbracket, \text{Card}(G_i) \leq \sum_{k \in \llbracket 1, n \rrbracket \setminus \{i\}} \text{Card}(G_k)$$

Soit G_1, \dots, G_n les différents groupes.

- ▶ **Mélanger** l'ordre des groupes
- ▶ Pour chaque groupe $G_i = [p_{i,1}, \dots, p_{i,k}]$:
 1. $\forall \ell \in \llbracket 1, k \rrbracket, p_{i,\ell} \rightsquigarrow p_r$ avec $p_r \leftarrow \mathcal{U}(G_{(i+\ell)\%n})$ et $\ell \neq 0[n]$
 2. **Supprimer** p_r de $G_{(i+\ell)\%n}$

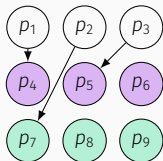


Étape 1 : groupe $[p_1, p_2, p_3]$

$p_1 \rightsquigarrow \mathcal{U}([p_4, p_5, p_6])$

$p_2 \rightsquigarrow \mathcal{U}([p_7, p_8, p_9])$

$p_3 \rightsquigarrow \mathcal{U}([p_5, p_6])$

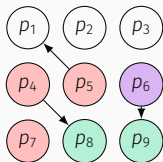


Étape 1 : groupe $[p_1, p_2, p_3]$

$$p_1 \rightsquigarrow \mathcal{U}([p_4, p_5, p_6])$$

$$p_2 \rightsquigarrow \mathcal{U}([p_7, p_8, p_9])$$

$$p_3 \rightsquigarrow \mathcal{U}([p_5, p_6])$$



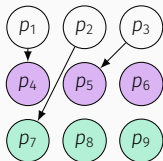
Étape 2 : groupe $[p_4, p_5, p_6]$

$$p_4 \rightsquigarrow \mathcal{U}([p_8, p_9])$$

$$p_5 \rightsquigarrow \mathcal{U}([p_1, p_2, p_3])$$

$$p_6 \rightsquigarrow \mathcal{U}([p_9])$$

a déjà reçu un cadeau

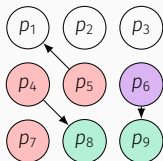


Étape 1 : groupe $[p_1, p_2, p_3]$

$p_1 \rightsquigarrow \mathcal{U}([p_4, p_5, p_6])$

$p_2 \rightsquigarrow \mathcal{U}([p_7, p_8, p_9])$

$p_3 \rightsquigarrow \mathcal{U}([p_5, p_6])$



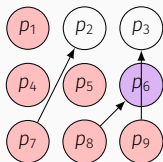
Étape 2 : groupe $[p_4, p_5, p_6]$

$p_4 \rightsquigarrow \mathcal{U}([p_8, p_9])$

$p_5 \rightsquigarrow \mathcal{U}([p_1, p_2, p_3])$

$p_6 \rightsquigarrow \mathcal{U}([p_9])$

a déjà reçu un cadeau



Étape 3 : groupe $[p_7, p_8, p_9]$

$p_7 \rightsquigarrow \mathcal{U}([p_2, p_3])$

$p_8 \rightsquigarrow \mathcal{U}([p_6])$

$p_9 \rightsquigarrow \mathcal{U}([p_3])$

p : nombre de personnes par groupe

n : nombre de groupes

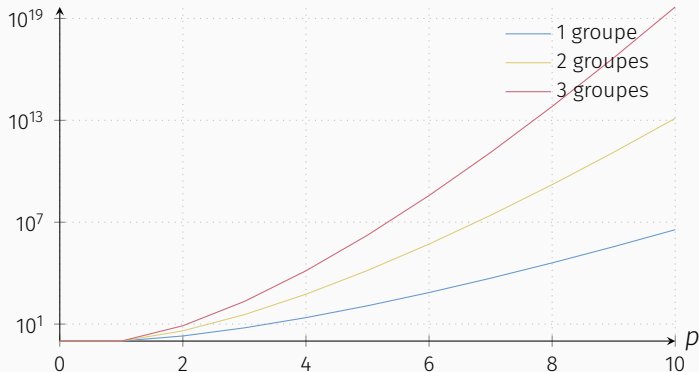


Figure 5 – Nombre de tirages possibles

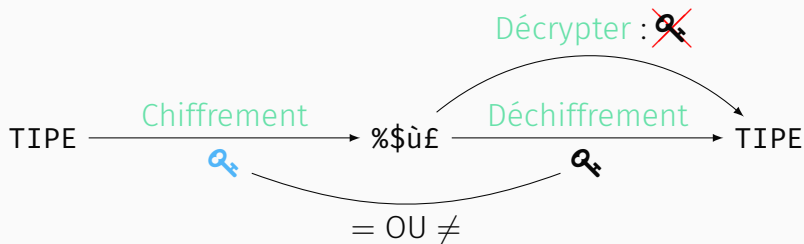


3. GESTION DES DONNÉES

	Prenom	NOM	Groupe	Email
1	Stanislas	MEZUREUX	MPSI1	stanmzx@gmail.com
2	Alice	ESPINOSA	MPSI1	aliceespinosa29@gmail.com
3	Bob	...	MPSI2	...
4	David	...	MPSI2	...
⋮	⋮	⋮	⋮	⋮

Figure 6 – Format des données manipulées

```
1  [[1, 2], [3, 4], ...]
2
3  1 = ['Stanislas', 'MEZUREUX', 'MPSI1', 'stanmzx@gmail.com']
```

- Chiffrement symétrique : $Q_k = Q_k$
- Chiffrement asymétrique : $Q_k \neq Q_k$

NBITS Entier naturel strictement supérieur à 1

q Entier premier de **NBITS** bits

g Générateur quelconque de $\mathbb{Z}/q\mathbb{Z}$

Générer les clés

clé privée (sk) : $x \leftarrow \mathcal{U}(\llbracket 2^{\text{NBITS}} - 1, q - 1 \rrbracket)$

clé publique (pk) : (q, g, h) avec $h \stackrel{\text{déf}}{=} g^x$

Exemple

Avec **NBITS** = 4; $q = 11$; $g = 4$

\leadsto sk = 8 et pk = (11, 4, 9)

Connu NBITS et pk

Secret sk

Chiffer $m \in \mathbb{N}$

$r \leftarrow \mathcal{U}(\llbracket 2^{\text{NBITS}-1}, q-1 \rrbracket)$

Nouveau r à chaque tirage

Le message chiffré est $(g^r \bmod q, m(h^r \bmod q))$

Exemple

NBITS = 4

sk = 8 et pk = (q = 11, g = 4, h = 9)

$m = 2 \rightsquigarrow (4^8 \bmod 11, 2(9^8 \bmod 11)) = (9, 6)$ avec $r = 8$

Déchiffrer (c_1, c_2)

$$\frac{c_2}{c_1^x \bmod q} = \frac{mh^r}{(g^r)^x} = \frac{m(g^x)^r}{(g^r)^x} q = \frac{mg^{xr}}{g^{xr}} = m$$

Exemple

NBITS = 4

sk = 8 et pk = (q = 11, g = 4, h = 9)

$$(9, 6) \rightsquigarrow \frac{6}{9^8 \bmod 11} = \frac{6}{3} = 2$$

Définition

Le cryptographe peut effectuer des opérations entre entiers chiffrés.
ELGAMAL est homomorphe vis-à-vis de la multiplication.

NBITS = 4

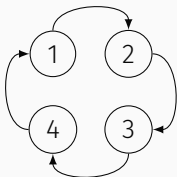
sk = 8 et pk = (q = 11, g = 4, h = 9)

Si 2 est chiffré par (9,6) et 3 par (3, 15) alors 6 est chiffré par (27, 60)

$$\begin{aligned} & \frac{90}{27^8 \bmod 11} \bmod 11 \\ &= 6 \end{aligned}$$

n participants représentés par les entiers de $\llbracket 1, n \rrbracket$

\leadsto tirage représenté par une liste de **couples (donneur, receveur)**
puis chiffrée



$\Rightarrow [(1, 2), (2, 3), (3, 4), (4, 1)]$

Figure 7 – Exemple de tirage

ZKP : Zero Knowledge Proof

Validité du tirage : chaque participant donne et reçoit exactement 1 cadeau

Condition de validité du tirage

Soit $[(d_1, r_1), \dots, (d_n, r_n)]$ le tirage.

$$\begin{cases} \prod_{i=1}^n d_i = n! \\ \prod_{i=1}^n r_i = n! \end{cases} \quad \text{et} \quad \forall (i, j) \in \llbracket 1, n \rrbracket^2 / i < j \quad \begin{cases} d_i \neq d_j \\ r_i \neq r_j \\ d_j \neq r_j \end{cases}$$

	1 groupe	2 groupes =	2 groupes \neq	n groupes
Condition \exists	✓	✓	✓	✓
Tirage	✓	✓	✓	✓
Chiffrer	✓	✓	✓	✓
Dénombrer	✓	✓	✗	✗
ZKP	✓	✗	✗	✗



4.

CONCLUSION

- ▶ **Algorithme** opérationnel
- ▶ Prévention contre la **triche**
- ▶ Nombreux domaines d'**application**

ANNEXES

Démonstration

ELGAMAL est homomorphe vis-à-vis de la multiplication.

Soit m_1 et m_2 chiffrés respectivement par $(c_{1,1}, c_{1,2})$ et $(c_{2,1}, c_{2,2})$.

$$\begin{aligned}\frac{c_{1,2}c_{2,2}}{(c_{1,1}c_{2,1})^x \bmod q} &= \frac{m_1 h^{r_1} m_2 h^{r_2}}{(g^{r_1} g^{r_2})^x} \\ &= \frac{m_1 m_2 h^{r_1+r_2}}{(g^{r_1+r_2})^x} \\ &= \frac{m_1 m_2 (g^x)^{r_1+r_2}}{(g^{r_1+r_2})^x} \\ &= m_1 m_2 \bmod q\end{aligned}$$

CODE CHIFFREMENT HOMOMORPHE

```
1  #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3  """
4  Created in 2021
5
6  @author: Stanislas MEZUREUX
7  Copyright (c) 2021 Stanislas MEZUREUX. All rights reserved.
8  """
9
10 import random
11
12 NUMBITS = 1024
13
14 # q -> cyclic group order
15 # g -> cyclic group generator
16 # x -> private key
```

```

17  # (q, g, h) where  $h = g^{**x} \bmod q \rightarrow$  private key
18
19  def gcd(a, b):
20      if a < b:
21          return gcd(b, a)
22      elif a%b == 0:
23          return b;
24      else:
25          return gcd(b, a%b)
26
27
28  # Miller-Rabin
29  # medium.com/@prudywsh/how-to-generate-big-prime-numbers-
    ↪ miller-rabin-49e6e6af32fb
30  def is_prime(n, k=128):
31      if n == 2 or n == 3:
32          return True
33      if n <= 1 or n % 2 == 0:
34          return False
35
    s = 0

```

```
36     r = n - 1
37     while r & 1 == 0:
38         s += 1
39         r //= 2
40     for _ in range(k):
41         a = random.randrange(2, n - 1)
42         x = pow(a, r, n)
43         if x != 1 and x != n - 1:
44             j = 1
45             while j < s and x != n - 1:
46                 x = pow(x, 2, n)
47                 if x == 1:
48                     return False
49             j += 1
50         if x != n - 1:
51             return False
52     return True
53
54
55 def generate_prime_candidate(length):
```

```

56     p = random.getrandbits(length)
57     p |= (1 « length - 1) | 1
58     return p
59
60
61 def generate_prime_number(length=1024):
62     p = 4
63     while not is_prime(p, 128):
64         p = generate_prime_candidate(length)
65     return p
66
67
68 def keygen():
69     q = generate_prime_number(NUMBITS)
70     g = random.randint(2, q)
71     x = random.randint(2**(NUMBITS-1), q)
72     return (x, {'q': q, 'g': g, 'h': pow(g, x, q)})
73
74
75 def encrypt(n, pk):

```



```

76     q, g, h = pk['q'], pk['g'], pk['h']
77     r = random.randint(2**(NUMBITS-1),q)
78     return {'c1': pow(g, r, q), 'c2': n*pow(h, r, q)}
79
80
81 def decrypt(n, x, pk):
82     return (n['c2']*pow(n['c1'], -x, pk['q']))%pk['q']
83
84
85 def multiply(n1, n2):
86     n1c1, n1c2 = n1['c1'], n1['c2']
87     n2c1, n2c2 = n2['c1'], n2['c2']
88     return {'c1': n1c1*n2c1, 'c2': n1c2*n2c2}
89
90
91 def is_equal(n1, n2, sk, pk):
92     d = {'c1': n1['c1']*pow(n2['c1'], -1, pk['q']),
93          'c2': n1['c2']*pow(n2['c2'], -1, pk['q'])}
94     return decrypt(d, sk, pk) == 1

```

```
1  #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3  """
4  Created in 2021
5
6  @author: Stanislas MEZUREUX
7  Copyright (c) 2021 Stanislas MEZUREUX. All rights reserved.
8  """
9
10 import smtplib
11 from random import shuffle
12 import copy
13 import secrets
14 import time
15 import elgamal as eg
16 from math import factorial
```

```

17
18 AMOUNT = 10
19 NAME = 'MPSI1 227/228'
20 DATE = '03/01/2022'
21
22
23 class TooMuchInTheTeam(Exception):
24     pass
25
26
27 def nb_participants_check(L):
28     for i, team in enumerate(L):
29         M = [e for A in L[:i]+L[i+1:] for e in A]
30         if M != [] and len(M) < len(L[i]):
31             raise TooMuchInTheTeam(f"Too much participants in
    ↪ {team[0][3]}")
32
33
34 def csv_to_list(data):
35     with open(data) as f:

```

```
36         L = f.read().splitlines()
37     L = L[1:]
38     for i, e in enumerate(L):
39         L[i] = [i] + e.split(',')
40     return L
41
42
43 def group_by_team(L):
44     teams = []
45     for e in L:
46         if e[3] not in teams:
47             teams.append(e[3])
48     nb_teams = len(teams)
49     M = [[] for _ in range(nb_teams)]
50     for i, team in enumerate(teams):
51         for e in L:
52             if e[3] == team:
53                 M[i].append(e[0])
54     return M
55
```

```

56
57 def make_pairs(L, pk):
58     nb_teams = len(L)
59     if nb_teams == 1:
60         M = L[0].copy()
61         shuffle(M)
62         length = len(M)
63         R = [(0, 0)]*length
64         for i in range(length):
65             R[i] = (eg.encrypt(M[i]+1, pk),
66                     eg.encrypt(M[(i+1) % length]+1, pk))
67         with open('secret_santa_draw.py', 'w') as f:
68             f.write(f'draw = {R}')
69         return R
70     R = []
71     M = copy.deepcopy(L)
72     shuffle(M)
73     L_new = copy.deepcopy(M)
74     for i, team in enumerate(L_new):
75         for j, e in enumerate(L_new[i]):

```

```

76         if len(M) == 1:
77             M_next = M[0]
78         else:
79             M_next = (M[:i]+M[i+1:])[ (j+1) % (len(M)-1)]
80         M_next_len = len(M_next)
81         k = secrets.randbelow(M_next_len)
82         gift_to = M_next[k]
83         R.append((eg.encrypt(e+1, pk),
84                     eg.encrypt(gift_to+1, pk)))
85     with open('example/secret_santa_draw.py', 'w') as
    ↪ f:
86         f.write(f'draw = {R}\ndraw_len = {len(R)}')
87     del M_next[k]
88     if M_next_len == 1:
89         M = [e for e in M if e != []]
90     return R
91
92
93 def send_email(L, data, sk, pk, display_team=True):
94     from_addr = 'secret.santa.tipe@gmail.com'

```

```

95
96 server = smtplib.SMTP_SSL('smtp.gmail.com', 465)
97 server.set_debuglevel(1)
98 server.ehlo
99
100 server.login('secret.santa.tipe@gmail.com',
    ↪ 'pamnkvaauruvqndga')
101
102 for e_encrypted in L:
103     e = (eg.decrypt(e_encrypted[0], sk, pk)-1,
104          eg.decrypt(e_encrypted[1], sk, pk)-1)
105     to_addrs = data[e[0]][4]
106     subject = f"Secret Santa - {NAME}"
107     text = (
108         f'Bonjour {data[e[0]][1]},\nCette année, tu es en
        ↪ charge du '
109         f'cadeau de {data[e[1]][1]} {data[e[1]][2]} '
110         f'{"(" + data[e[1]][3] + ")"}' if display_team else "").
        ↪ Je te rappelle '

```

```

111         f'que le budget est de {AMOUNT}€ et que la
           ↳ célébration aura lieu '
112         f'le {DATE}.\nJoyeux Noël à toi !'
113     )
114
115     message = f"Subject: {subject}\nFrom: {from_addr}\nTo:
           ↳ {to_addrs}\n\n"
116     message = message + text
117     server.sendmail(from_addr, to_addrs,
           ↳ message.encode("utf8"))
118
119     time.sleep(0.1)
120
121     server.quit()
122
123
124     def zero_knowledge_proof(sk, pk):
125         import example.secret_santa_draw as ssd
126         if ssd.draw_len == 1:
127             return True

```



```

128 gift_from = ssd.draw[0][0]
129 gift_to = ssd.draw[0][1]
130 for i in range(1, ssd.draw_len):
131     gift_from = eg.multiply(gift_from, ssd.draw[i][0])
132     gift_to = eg.multiply(gift_to, ssd.draw[i][1])
133 fact = factorial(ssd.draw_len)
134 for j in range(1, ssd.draw_len):
135     for i in range(1, j):
136         c_i = ssd.draw[i]
137         c_j = ssd.draw[j]
138         if (eg.is_equal(c_i[0], c_j[0], sk, pk) or
139             eg.is_equal(c_i[1], c_j[1], sk, pk) or
140             eg.is_equal(c_j[0], c_j[1], sk, pk)):
141             return False
142 return eg.decrypt(gift_from, sk, pk) == fact and
    ↪ eg.decrypt(gift_to, sk, pk) == fact
143
144
145 def Secret_Santa(data):
146     try:

```

```

147         sk, pk = eg.keygen()
148         info = csv_to_list(data)
149         L = group_by_team(info)
150         nb_teams = len(L)
151         nb_participants_check(L)
152         R = make_pairs(L, pk)
153         print(f'secret key : {sk}')
154         print(f'public key : {pk}')
155         # send_email(R, info, sk, pk, nb_teams != 1)
156     except TooMuchInTheTeam as TeamError:
157         print(TeamError)
158
159
160 def resend(sk, pk, data):
161     try:
162         import example.secret_santa_draw as ssd
163         info = csv_to_list(data)
164         nb_teams = len(ssd.draw)
165         send_email(ssd.draw, info, sk, pk, nb_teams != 1)
166     except ModuleNotFoundError as Error:

```

```
167         print(Error)
168
169     # Secret_Santa('example/data.csv')
```

CODE DÉNOMBREMENT TIRAGES

```
1  #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3  """
4  Created in 2022
5
6  @author: Stanislas MEZUREUX
7  Copyright (c) 2021 Stanislas MEZUREUX. All rights reserved.
8  """
9
10 from itertools import permutations
11
12
13 """
14 Presumed formula : (nb_people_in_one_team !)^ (nb_teams)
15 """
16
```

```
17
18 def check_draw(D, L, p, n):
19     for e in D:
20         if e[1] not in L[((e[0]-1)//p + 1)%n]:
21             return False
22     return True
23
24
25 def gen_draws(p, n):
26     M = [i for i in range(1, n*p+1)]
27     P = list(permutations(M))
28     R = [[]]*len(P)
29     for i, e in enumerate(P):
30         D = [()]*(n*p)
31         for j in range(len(e)):
32             D[j] = (j, e[j])
33         R[i] = D
34     return R
35
36
```

```
37 def count(p, n):
38     L = [[p*i+j+1 for j in range(p)] for i in range(n)]
39     R = gen_draws(p, n)
40     res = 0
41     for e in R:
42         if check_draw(e, L, p, n):
43             res += 1
44     return res
45
46     # print(count(10, 3))
```