TIPE

Combinatoire et chiffrement : le père Noël secret et ses applications

Stanislas MEZUREUX   Alice ESPINOSA

MP

Lycée Henri Poincaré

13 mai 2022

Table des matières

◁ Chapitre 1 ▷
Présentation

I ▷ Introduction

Célèbre tradition, le Père Noël secret est en apparence un simple tirage au sort pouvant être effectué numériquement. Pourtant, en se penchant sur le sujet, nous avons découvert plusieurs notions de chiffrement et de combinatoire que nous nous proposons d'étudier.

Ainsi, cette tradition de Noël nous a permis d'aborder les thématiques de l'aléa et de la prévention contre la triche. Le sujet s'inscrit donc bien dans le thème de l'année.

II ▷ Mise en cohérence des objectif

II.1 Positionnement thématique

MATHÉMATIQUES *(Algèbre)*, INFORMATIQUE *(Informatique pratique)*, INFORMATIQUE *(Informatique Théorique)*

II.2 Mots-clés

Mots-Clés (en français)	**Mots-Clés** (en anglais)
Preuve à divulgation nulle de connaissance	*Zero-knowledge proof*
Répartition aléatoire	*Random distribution*
Chiffrement homomorphe	*Homomorphic encryption*
Combinatoire	*Combinatorics*
Dénombrement	*Enumeration*

III ▷ Bibliographie commentée

Le numérique ne cessant de prendre de l'ampleur, la prévention contre la triche se doit d'évoluer par la même occasion. En particulier, les canaux de transmissions sécurisés et l'étude de l'aléa sont des éléments de réponses à des problèmes concrets tels que le vote électronique. L'organisation d'un Secret Santa ou père Noël secret en français en est une autre application relativement simple. En effet, dans cette tradition chaque participant tire au hasard un autre participant à qui il devra faire un cadeau. Elle met donc en jeu les différents aspects de cette transmission, du tirage jusqu'à l'envoi des mails en passant par le chiffrement complet des données [1].

L'étude d'un tirage du Père Noël Secret demande la prise en compte de plusieurs aspects : l'existence, l'intérêt, la sécurité et la fiabilité de ce tirage [2]. Cette étude peut se complexifier avec l'ajout de conditions, par exemple si on souhaite que des personnes qui vivent sous un même toit ne puissent pas se faire de cadeaux entre elles.

Un premier point important est de réussir à allier confidentialité et sécurité. Il faut alors trouver une méthode pour prouver que le tirage a été effectué correctement sans avoir besoin de le dévoiler. Pour cela, nous avons utilisé une « preuve à divulgation nulle de connaissance », qui est un processus consistant à prouver la véracité d'une information sans connaître cette information [3]. Il peut se baser sur plusieurs principes, notamment sur le chiffrement homomorphe, qui est un cryptosystème permettant de faire des opérations sur des données chiffrées. De plus, nous profiterons de l'utilisation du chiffrement homomorphe pour pouvoir sécuriser les données des participants et stocker le tirage dans un fichier annexe dans l'optique d'être en capacité de renvoyer le tirage par mail. Le système utilisé ici est celui du cryptographe égyptien Taher ElGamal [4].

Un second point sur lequel nous nous sommes concentrés est la prise en compte de différentes contraintes. Nous les avons modélisés avec des groupes, les membres d'un groupe ne pouvant pas se tirer entre eux. Le nombre de tirages possibles dépend du nombre de groupes mais aussi du nombre de participants dans chaque groupe. L'aspect combinatoire nous a donc permis de définir l'existence ainsi que la rentabilité d'un tirage. Lorsqu'il y a trop peu de participants, le nombre de combinaisons possibles est trop faible pour que le tirage garde son intérêt. Nous avons commencé par étudier le cas de n groupes, en utilisant la formule du crible de Poincaré, puis nous avons essayé de généraliser aux cas où nous avions entre 2 et n-1 groupes [5,6].

IV ▷ Problématique

Il s'agit d'implanter en Python le tirage au sort d'un père Noël secret afin de mettre en avant les différentes technologies de prévention contre la triche et d'en étudier l'efficacité.

V ▷ Objectifs

1. Implantation en Python de l'algorithme effectuant le tirage
2. Étude de la sécurité des données lors de l'envoi de ces dernières
3. Preuve à divulgation nulle de connaissance
4. Condition sur l'existence du tirage
5. Dénombrement du nombre de tirages possibles selon le nombre de groupes et la taille des groupes
6. Étude du problème sous forme de graphe

◁ Chapitre 2 ▷ Algorithme de père Noël secret

I ▷ Tirage au sort

I.1 Stockage des informations sur les candidats

On regroupe les informations sur les candidats dans un tableau de la forme suivante :

Table 2.1 – Format des données d'entrée

Prenom	NOM	Groupe	Email
Stanislas	MEZUREUX	MPSI1	stanmzx@gmail.com
Alice	ESPINOSA	MPSI1	aliceespinosa29@gmail.com
⋮	⋮	⋮	⋮

Le CSV *(Comma-Separated Values)* est un format permettant de représenter des tableaux. Les lignes du tableaux sont les lignes du fichier et les colonnes du tableau sont séparées par des virgules dans le fichier. Le fichier CSV associé au tableau 2.1 est donc :

Code 2.1 – Fichier data.csv

```
Prenom,NOM,Team,email
Stanislas,MEZUREUX,MPSI1,stanmzx@gmail.com
Alice,ESPINOSA,MPSI2,aliceespinosa29@gmail.com
...
```

Ainsi, les informations sur les candidats seront stockées sous la forme d'un fichier `.csv` dans le but d'être converties facilement en tableau Python avec l'algorithme suivant : on transforme le fichier en liste où chaque élément est une ligne et on transforme chaque élément en une liste où on a « découpé » les valeurs selon les virgules. On appelle cette fonction `csv_to_list`.

De plus, une condition du tirage impose que les membres d'un même groupe ne puissent pas s'échanger de cadeau entre eux. Cela conduit donc à regrouper les participants en fonction de leur groupe. À l'issue de la fonction `csv_to_list`, on crée une nouvelle liste dont les éléments sont des listes de candidats d'un même groupe. On appelle cette fonction `group_by_team`.

Exemple I.1: Liste regroupée des participants

Si A et B sont dans le groupe 1 et C et D dans le groupe 2, la liste regroupée vaut [[A, B], [C, D]].

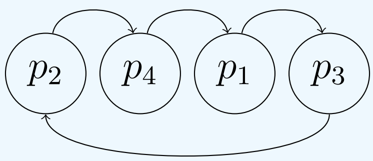
I.2 Création des paires

Remarque : Dans cette section, on suppose que les conditions sur les groupes rendent le tirage possible. Une condition ainsi qu'une preuve seront fournis dans le chapitre 3.

On modélise le tirage au sort par un tableau de couples dont la première composante est le donneur et la deuxième est le receveur du cadeau. On distingue alors plusieurs cas :

- Tous les participants sont dans le même groupe : Chaque participant peut offrir un cadeau à n'importe quel autre participant. Pour effectuer le tirage, on récupère la liste renvoyée par la fonction `group_by_team` qui contient donc un seul élément : la liste correspondant à l'unique groupe. On mélange cette liste et le participant d'indice i offre son cadeau au participant d'indice $i+1$ (modulo le nombre de participants).

Exemple I.2: Tirage pour un seul groupe



Si `group_by_team` renvoie :
`[[`(p_1, p_2)`, `(p_2, p_3)`, `(p_3, p_4)`, `(p_4, p_1)`]]`
alors un tirage possible est :
`[`(p_2, p_4)`, `(p_4, p_1)`, `(p_1, p_3)`, `(p_3, p_2)`]`

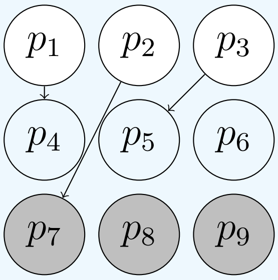
- Les participants sont répartis dans plusieurs groupes : La méthode utilisée consiste à fabriquer les couples en parcourant successivement les autres groupes. On donne cet algorithme sous forme de pseudo-code :

Code 2.2 – Pseudo-code algorithme de création des paires

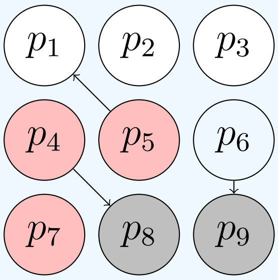
```
L <- liste renvoyée par group_by_team

Mélanger L
L_UPDATED <- copie de L # sera L privée de ceux qui n'ont pas encore de cadeau
R <- [] # liste des paires
Pour chaque GROUPE dans L :
    Pour chaque DONNEUR dans groupe :
        L_PRETENDANTS <- groupe après GROUPE dans L_UPDATED
        RECEVEUR <- Tirer aléatoirement dans L_PRETENDANTS
        Ajouter (DONNEUR, RECEVEUR) à R
        Retirer RECEVEUR de L_UPDATED
    Fin de Pour
Fin de Pour

Renvoyer R
```

Exemple I.3

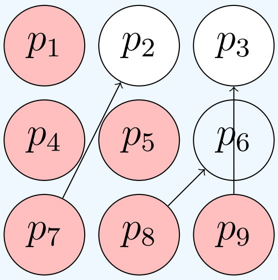
Prenons `[[`p_1, p_2, p_3`]`, `[`p_4, p_5, p_6`]`, `[`p_7, p_8, p_9`]]` la liste renvoyée par `group_by_team`.



Étape 1 : groupe $[p_1, p_2, p_3]$
$p_1 \rightsquigarrow$ tirage aléatoire dans $[p_4, p_5, p_6]$
$p_2 \rightsquigarrow$ tirage aléatoire dans $[p_7, p_8, p_9]$
$p_3 \rightsquigarrow$ tirage aléatoire dans $[p_5, p_6]$



Étape 2 : groupe $[p_4, p_5, p_6]$
$p_4 \rightsquigarrow$ tirage aléatoire dans $[p_8, p_9]$
$p_5 \rightsquigarrow$ tirage aléatoire dans $[p_1, p_2, p_3]$
$p_6 \rightsquigarrow$ tirage aléatoire dans $[p_9]$
participant ayant déjà reçu un cadeau



Étape 3 : groupe $[p_7, p_8, p_9]$
$p_7 \rightsquigarrow$ tirage aléatoire dans $[p_2, p_3]$
$p_8 \rightsquigarrow$ tirage aléatoire dans $[p_6]$
$p_9 \rightsquigarrow$ tirage aléatoire dans $[p_3]$

`[(`p_1, p_4`)`, `(`p_2, p_7`)`, `(`p_3, p_5`)`, `(`p_4, p_8`)`, `(`p_5, p_1`)`, `(`p_6, p_9`)`, `(`p_7, p_2`)`, `(`p_8, p_6`)`, `(`p_9, p_3`)]`.

II ▷ Chiffrement

II.1 Cryptosystème d'ElGamal

Une autre contrainte du tirage est la sécurité, il faut pouvoir garantir la confidentialité des données lorsque les informations sont stockées et manipulées afin d'éviter la triche.

Définition II.1: Algorithme de chiffrement homomorphe

Système possédant des caractéristiques algébriques qui lui permettent de commuter avec certaines opérations mathématiques, c'est-à-dire qu'il permet d'opérer un obscurcissement sur des valeurs numériques tout en conservant les propriétés permettant les dites opérations. [1]

Le chiffrement homomorphe semble être l'outil adapté pour satisfaire la sécurité tout en étant capable de continuer à manipuler les données. Le cryptosystème retenu et présenté ici est celui d'ElGamal, il fonctionne avec une clé publique qui permet de chiffrer les données et une clé privée qui pour les déchiffrer.

Notations : on pose `NBITS` un entier strictement supérieur à 1, q un entier premier de `NBITS` et g un générateur de $\mathbb{Z}_{/q\mathbb{Z}}$ choisi aléatoirement entre 2 et q.

1. https://fr.wikipedia.org/wiki/Chiffrement_homomorphe

Pour générer les deux clés, on prend x un entier quelconque de $[\![2^{\texttt{NBITS}-1}, q-1]\!]$ et avec $h \overset{\text{déf}}{=} g^x \bmod q$ les clés sont données par

$$\boxed{\text{clé privée : } \texttt{sk} \overset{\text{déf}}{=} x \mid \text{clé publique : } \texttt{pk} \overset{\text{déf}}{=} (q, g, h)}$$

Pour chiffrer un entier m, on prend r un entier quelconque de $[\![2^{\texttt{NBITS}-1}, q-1]\!]$ et le message chiffré est

$$\boxed{(c_1, c_2) \overset{\text{déf}}{=} (g^r \bmod q, m \times h^r \bmod q)}$$

Théorème II.1: Déchiffrer un message

Soit (c_1, c_2) un message chiffré.
Le message en clair est $\frac{c_2}{c_1^x} \bmod q$.

Démonstration ▷ (Théorème II.1)
Soit (c_1, c_2) un message chiffré.

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

□

Théorème II.2: Homomorphe multiplicatif

Le cryptosystème d'ElGamal est homomorphe vis-à-vis de la multiplication. C'est-à-dire que si m_1 et m_2 sont deux entiers chiffrés respectivement par $(c_{1,1}, c_{1,2})$ et $(c_{2,1}, c_{2,2})$ alors $(c_{1,1} \times c_{2,1}, c_{1,2} \times c_{2,2})$ est la représentation chiffrée de $m_1 \times m_2$.

Démonstration ▷ (Théorème II.2)
Soit m_1 et m_2 deux entiers chiffrés respectivement par $(c_{1,1}, c_{1,2})$ et $(c_{2,1}, c_{2,2})$.

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

□

II.2 Code source

La dernière version de ce code est disponible à l'adresse https://github.com/StanislasMzx/Secret-Santa-TIPE

Code 2.3 – Fichier elgamal.py

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created in 2021

@author: Stanislas MEZUREUX

```

```
Copyright (c) 2021 Stanislas MEZUREUX. All rights reserved.
"""

import random

NUMBITS = 1024

# q -> cyclic group order
# g -> cyclic group generator
# x -> prvate key
# (q, g, h) where h = g**x mod p -> private key

def gcd(a, b):
    if a < b:
        return gcd(b, a)
    elif a%b == 0:
        return b;
    else:
        return gcd(b, a%b)


# Miller-Rabin
#
↪ medium.com/@prudywsh/how-to-generate-big-prime-numbers-miller-rabin-49e6e6af32fb
def is_prime(n, k=128):
    if n == 2 or n == 3:
        return True
    if n <= 1 or n % 2 == 0:
        return False
    s = 0
    r = n - 1
    while r & 1 == 0:
        s += 1
        r //= 2
    for _ in range(k):
        a = random.randrange(2, n - 1)
        x = pow(a, r, n)
        if x != 1 and x != n - 1:
            j = 1
            while j < s and x != n - 1:
                x = pow(x, 2, n)
                if x == 1:
                    return False
                j += 1
            if x != n - 1:
                return False
    return True


def generate_prime_candidate(length):
    p = random.getrandbits(length)
    p |= (1 << length - 1) | 1
```

```python
    return p


def generate_prime_number(length=1024):
    p = 4
    while not is_prime(p, 128):
        p = generate_prime_candidate(length)
    return p


def keygen():
    q = generate_prime_number(NUMBITS)
    g = random.randint(2, q)
    x = random.randint(2**(NUMBITS-1), q)
    return (x, {'q': q, 'g': g, 'h': pow(g, x, q)})


def encrypt(n, pk):
    q, g, h = pk['q'], pk['g'], pk['h']
    r = random.randint(2**(NUMBITS-1),q)
    return {'c1': pow(g, r, q), 'c2': n*pow(h, r, q)}


def decrypt(n, x, pk):
    return (n['c2']*pow(n['c1'], -x, pk['q']))%pk['q']


def multiply(n1, n2):
    n1c1, n1c2 = n1['c1'], n1['c2']
    n2c1, n2c2 = n2['c1'], n2['c2']
    return {'c1': n1c1*n2c1, 'c2': n1c2*n2c2}
```

III ▷ Preuve à divulgation nulle de connaissance

Définition III.1: Preuve à divulgation nulle de connaissance

Désigne un protocole sécurisé dans lequel une entité, nommée « fournisseur de preuve », prouve mathématiquement à une autre entité, le « vérificateur », qu'une proposition est vraie sans toutefois révéler d'autres informations que la véracité de la proposition.[2]

L'objectif de cette partie est de pouvoir garantir à tous les participant une fois le tirage effectué, sans faire fuiter d'information, que le tirage est conforme. C'est à dire que chaque personne engagée dans le père Noël secret donne exactement un cadeau et reçoit exactement un cadeau.

III.1 Stockage du tirage

Pour pouvoir être en mesure d'implanter la preuve à divulgation nulle de connaissance, la liste de couples qui modélise le tirage doit être stockée. Le cryptosystème introduit dans la section précédent permet de générer un fichier qui contient cette liste sous forme chiffrée lors de l'exécution de l'algorithme.

2. https://fr.wikipedia.org/wiki/Preuve_à_divulgation_nulle_de_connaissance

Le fait de stocker le tirage sous forme chiffrée permet également de pouvoir envoyer l'issue du tirage par e-mail une nouvelle fois. Par exemple en cas d'erreur sur l'e-mail d'un participant ou pour effectuer un nouveau tirage différent du précédent.

III.2 Condition de validité du tirage

Théorème III.1: Condition de validité du tirage

Soit n le nombre de participants.
Si la liste qui représente le tirage est $[(d_1, r_1), \dots, (d_n, r_n)]$ et si les participants sont représentés par les entiers de $[\![1, n]\!]$ alors

$$\text{le tirage est valide} \iff \begin{cases} \prod_{i=1}^{n} d_i = n! \\ \prod_{i=1}^{n} r_i = n! \end{cases}$$

Démonstration ▷ (Théorème III.1)
Soit n le nombre de participants.
Soit $[(d_1, r_1), \dots, (d_n, r_n)]$ la liste qui représente le tirage avec $\forall i \in [\![1, n]\!], d_i \in [\![1, n]\!]$ et $r_i \in [\![1, n]\!]$.

$\boxed{\Rightarrow}$ Si le tirage est valide, chaque participant donne exactement un cadeau et reçoit exactement un cadeau *i.e.* $\forall (i,j) \in [\![1, n]\!]^2, i \neq j \Rightarrow (d_i \neq d_j$ et $r_i \neq r_j)$

Comme $\forall i \in [\![1, n]\!], d_i \in [\![1, n]\!]$ et $r_i \in [\![1, n]\!]$, $\begin{cases} \prod_{i=1}^{n} d_i = n! \\ \prod_{i=1}^{n} r_i = n! \end{cases}$

$\boxed{\Leftarrow}$

$$\prod_{i=1}^{n} d_i = n! \Rightarrow \prod_{i=1}^{n} d_i = \prod_{i=1}^{n} i \Rightarrow \forall (i,j) \in [\![1, n]\!]^2, i \neq j \Rightarrow (d_i \neq d_j \text{ et } r_i \neq r_j)$$

Cela signifie que le tirage est valide. □

IV ▷ Code source

La dernière version de ce code est disponible à l'adresse https://github.com/StanislasMzx/Secret-Santa-TIPE

Code 2.4 – Fichier SecretSanta.py

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created in 2021

@author: Stanislas MEZUREUX

Copyright (c) 2021 Stanislas MEZUREUX. All rights reserved.
"""

```

```python
import smtplib
from random import shuffle
import copy
import secrets
import time
import elgamal as eg
from math import factorial

AMOUNT = 10
NAME = 'MPSI1 227/228'
DATE = '03/01/2022'


class TooMuchInTheTeam(Exception):
    pass


def nb_participants_check(L):
    for i, team in enumerate(L):
        M = [e for A in L[:i]+L[i+1:] for e in A]
        if M != [] and len(M) < len(L[i]):
            raise TooMuchInTheTeam(f"Too much participants in {team[0][3]}")


def csv_to_list(data):
    with open(data) as f:
        L = f.read().splitlines()
    L = L[1:]
    for i, e in enumerate(L):
        L[i] = [i] + e.split(',')
    return L


def group_by_team(L):
    teams = []
    for e in L:
        if e[3] not in teams:
            teams.append(e[3])
    nb_teams = len(teams)
    M = [[] for _ in range(nb_teams)]
    for i, team in enumerate(teams):
        for e in L:
            if e[3] == team:
                M[i].append(e[0])
    return M


def make_pairs(L, pk):
    nb_teams = len(L)
    if nb_teams == 1:
        M = L[0].copy()
        shuffle(M)
```

```
        length = len(M)
        R = [(0, 0)]*length
        for i in range(length):
            R[i] = (eg.encrypt(M[i]+1, pk),
                    eg.encrypt(M[(i+1) % length]+1, pk))
        with open('secret_santa_draw.py', 'w') as f:
            f.write(f'draw = {R}')
        return R
    R = []
    M = copy.deepcopy(L)
    shuffle(M)
    L_new = copy.deepcopy(M)
    for i, team in enumerate(L_new):
        for j, e in enumerate(L_new[i]):
            if len(M) == 1:
                M_next = M[0]
            else:
                M_next = (M[:i]+M[i+1:])[(j+1) % (len(M)-1)]
            M_next_len = len(M_next)
            k = secrets.randbelow(M_next_len)
            gift_to = M_next[k]
            R.append((eg.encrypt(e+1, pk),
                      eg.encrypt(gift_to+1, pk)))
            with open('example/secret_santa_draw.py', 'w') as f:
                f.write(f'draw = {R}\ndraw_len = {len(R)}')
            del M_next[k]
            if M_next_len == 1:
                M = [e for e in M if e != []]
    return R


def send_email(L, data, sk, pk, display_team=True):
    from_addr = 'secret.santa.tipe@gmail.com'

    server = smtplib.SMTP_SSL('smtp.gmail.com', 465)
    server.set_debuglevel(1)
    server.ehlo

    server.login('secret.santa.tipe@gmail.com', 'password')

    for e_encrypted in L:
        e = (eg.decrypt(e_encrypted[0], sk, pk)-1,
             eg.decrypt(e_encrypted[1], sk, pk)-1)
        to_addrs = data[e[0]][4]
        subject = f"Secret Santa - {NAME}"
        text = (
            f'Bonjour {data[e[0]][1]},\nCette année, tu es en charge du '
            f'cadeau de {data[e[1]][1]} {data[e[1]][2]} '
            f'{"("+data[e[1]][3]+")" if display_team else ""}. Je te rappelle '
            f'que le budget est de {AMOUNT}€ et que la célébration aura lieu '
            f'le {DATE}.\nJoyeux Noël à toi !'
        )
```

```python

        message = f"Subject: {subject}\nFrom: {from_addr}\nTo: {to_addrs}\n\n"
        message = message + text
        server.sendmail(from_addr, to_addrs, message.encode("utf8"))

        time.sleep(0.1)

    server.quit()


def zero_knowledge_proof(sk, pk):
    import example.secret_santa_draw as ssd
    if ssd.draw_len == 1:
        return True
    gift_from = ssd.draw[0][0]
    gift_to =  ssd.draw[0][1]
    for i in range(1, ssd.draw_len):
        gift_from = eg.multiply(gift_from, ssd.draw[i][0])
        gift_to = eg.multiply(gift_to, ssd.draw[i][1])
    fact = factorial(ssd.draw_len)
    return eg.decrypt(gift_from, sk, pk) == fact and eg.decrypt(gift_to, sk, pk) == fact


def Secret_Santa(data):
    try:
        sk, pk = eg.keygen()
        info = csv_to_list(data)
        L = group_by_team(info)
        nb_teams = len(L)
        nb_participants_check(L)
        R = make_pairs(L, pk)
        print(f'secret key : {sk}')
        print(f'public key : {pk}')
        send_email(R, info, sk, pk, nb_teams != 1)
    except TooMuchInTheTeam as TeamError:
        print(TeamError)


def resend(sk, pk, data):
    try:
        import example.secret_santa_draw as ssd
        info = csv_to_list(data)
        nb_teams = len(ssd.draw)
        send_email(ssd.draw, info, sk, pk, nb_teams != 1)
    except ModuleNotFoundError as Error:
        print(Error)

# Secret_Santa('example/data.csv')
```

◁ Chapitre 3 ▷ Combinatoire

I ▷ Cas d'un seul groupe

Ce cas est le même que celui où les n participants sont dans n groupes différents, $n \geqslant 2$. On pose alors

- ▷ A : ensemble de n personnes
- ▷ N : nombre de permutations sans point fixe (*i.e.* nombre de tirages possibles)
- ▷ $\overline{N}$: nombre de permutations avec point fixe
- ▷ A_i : ensemble des permutations à n éléments avec i comme point fixe

Théorème I.1: Cardinal de l'ensemble des tirages possibles pour n groupes

$$N = n!\left(1 + \sum_{k=1}^{n} \frac{(-1)^{k+1}}{k!}\right)$$

Démonstration ▷ (Théorème I.1)

$$\begin{aligned}
\overline{N} &= \operatorname{Card}\left(\bigcup_{i=1}^{n} A_i\right) && (1)\\
&= \sum_{k=1}^{n}\left((-1)^{k+1} \sum_{\substack{I\subset[\![1,n]\!]\\ \operatorname{Card}(I)=k}} \operatorname{Card}\left(\bigcap_{i\in I} A_i\right)\right) && (2)\\
&= \sum_{k=1}^{n}\left((-1)^{k+1} \sum_{I\subset[\![1,n]\!]} (n-k)!\right) && (3)\\
&= \sum_{k=1}^{n}(-1)^{k+1}\binom{k}{n}(n-k)! && (4)\\
&= \sum_{k=1}^{n}(-1)^{k+1}\frac{n!}{k!}
\end{aligned}$$

Explications :

- ▷ $(1) \Rightarrow (2)$: formule du crible de Poincaré
- ▷ $(2) \Rightarrow (3)$: k éléments sont fixés, il y a $(n-k)!$ possibilités de placer les $n-k$ derniers
- ▷ $(3) \Rightarrow (4)$: il y a $\binom{k}{n}$ parties de k éléments dans I

$$\begin{aligned} N &= n! - \overline{N} \\ &= n! - \sum_{k=1}^{n} (-1)^{k+1} \frac{n!}{k!} \\ &= n! \left(1 + \sum_{k=1}^{n} \frac{(-1)^{k+1}}{k!} \right) \end{aligned}$$

□

II ▷ Cas de 2 groupes

Cas de $2p$ participants et 2 groupes avec p personnes par groupe, $p \geqslant 1$. On pose

- ▷ G_1 : premier groupe avec $\mathrm{Card}(G_1) = p$
- ▷ G_2 : deuxième groupe avec $\mathrm{Card}(G_2) = p$
- ▷ N : nombre de tirages possibles

Théorème II.1: Cardinal de l'ensemble des tirages possibles pour 2 groupes

$$N = (p!)^2$$

Démonstration ▷ (Théorème II.1)

On pose :

- ▷ $f\colon \begin{array}{ccc} G_1 & \longrightarrow & G_2 \\ x & \longmapsto & y \end{array}$: bijective
- ▷ $g\colon \begin{array}{ccc} G_2 & \longrightarrow & G_1 \\ x & \longmapsto & y \end{array}$: bijective
- ▷ F : ensemble des bijections de G_1 dans G_2
- ▷ G : ensemble des bijections de G_2 dans G_1

$$\begin{aligned} N &= \mathrm{Card}(\{(f, g); f \in F \text{ et } g \in G\}) \\ &= \mathrm{Card}(F \times G) \\ &= \mathrm{Card}(F) \times \mathrm{Card}(G) \\ &= (p!)^2 \end{aligned}$$

□

Bibliographie

[1] Sjouke Mauw, Sasa Radomirovic, Peter Ryan : Security protocols for Secret Santa : *Cambridge International Workshop on Security Protocols (2014)*

[2] Evgeniy Prikhodko : Théorie des graphes : *https://binary-machinery.github.io/2021/02/03/secret-santa-graph.html*

[3] Cécile GONÇALVES : Cryptographie Avancée : *dentification Zéro-Knowledge, Cryptographie distribuée et Chiffrement homomorphe (2015)*

[4] Wikipédia : Cryptosystème d'ElGamal : *https://fr.wikipedia.org/wiki/Cryptosystème_de_ElGamal*

[5] Marc SAGE : Combinatoire : *7-9 (2008)*

[6] Fatima A. Akinola : On Hamilton cycle decompositions of complete multipartite graphs which are both cyclic and symmetric : *(2021)*