COMBINATOIRE ET CHIFFREMENT : LE PÈRE NOËL SECRET ET SES APPLICATIONS

Stanislas MEZUREUX | 33920

2021|22

Lycée Henri POINCARÉ

SOMMAIRE

POSITION DU PROBLÈME

PÈRE NOËL SECRET ?

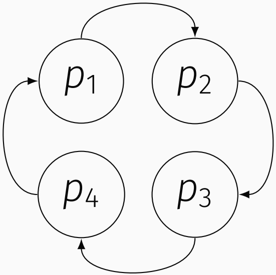


Figure 1 – Un groupe

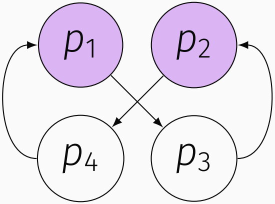


Figure 2 – Deux groupes

OBJECTIFS

- Entrée : liste des participants (nom + email + groupe)

OBJECTIFS

- Entrée : liste des participants (nom + email + groupe)
- Sortie : email

OBJECTIFS

- Entrée : liste des participants (nom + email + groupe)
- Sortie : email
- Chiffrement des données

OBJECTIFS

- Entrée : liste des participants (nom + email + groupe)
- Sortie : email
- Chiffrement des données
- Stockage du tirage

OBJECTIFS

- Entrée : liste des participants (nom + email + groupe)
- Sortie : email
- Chiffrement des données
- Stockage du tirage
- Preuve à divulgation nulle de connaissance

OBJECTIFS

- Entrée : liste des participants (nom + email + groupe)
- Sortie : email
- Chiffrement des données
- Stockage du tirage
- Preuve à divulgation nulle de connaissance
- Dénombrement des tirages

TIRAGE AU SORT

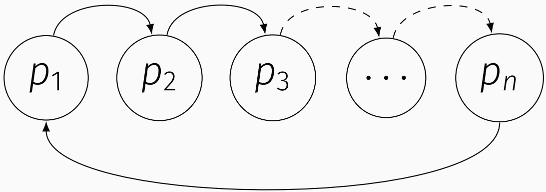


Figure 3 – Un seul groupe, pas de problème d'existence

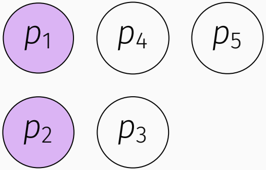


Figure 4 – Cas pathologique

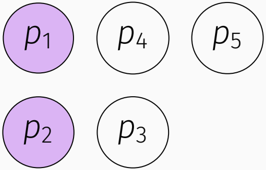


Figure 4 – Cas pathologique

Condition d'existence du tirage

Soit $G_1, \dots, G_n$ les différents groupes.

$$\forall i \in [\![1, n]\!], \mathrm{Card}(G_i) \leqslant \sum_{k \in [\![1,n]\!] \setminus \{i\}} \mathrm{Card}(G_k)$$

CRÉATION DES PAIRES

Soit $G_1, \ldots, G_n$ les différents groupes.

- Mélanger l'ordre des groupes

Soit $G_1, \dots, G_n$ les différents groupes.

- Mélanger l'ordre des groupes
- **Pour chaque groupe $G_i = [p_{i,1}, \dots, p_{i,k}]$:**

Soit $G_1, \ldots, G_n$ les différents groupes.

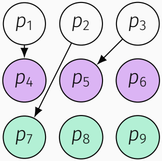
- Mélanger l'ordre des groupes
- Pour chaque groupe $G_i = [p_{i,1}, \ldots, p_{i,k}]$:
    1. $\forall \ell \in \llbracket 1, k \rrbracket, p_{i,\ell} \rightsquigarrow p_r$ **avec** $p_r \hookleftarrow \mathcal{U}(G_{(i+\ell)\%n})$ **et** $\ell \not\equiv 0[n]$

CRÉATION DES PAIRES

Soit $G_1, \ldots, G_n$ les différents groupes.

- Mélanger l'ordre des groupes
- Pour chaque groupe $G_i = [p_{i,1}, \ldots, p_{i,k}]$:
  1. $\forall \ell \in [\![1, k]\!]\,, p_{i,\ell} \rightsquigarrow p_r$ avec $p_r \hookleftarrow \mathcal{U}(G_{(i+\ell)\%n})$ et $\ell \not\equiv 0[n]$
  2. **Supprimer p_r de $G_{(i+\ell)\%n}$**

EXEMPLE



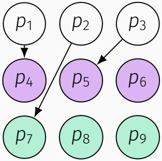
Étape 1 : groupe $[p_1, p_2, p_3]$

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

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

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

EXEMPLE

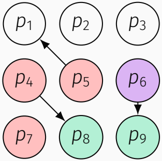


Étape 1 : groupe $[\,p_1, p_2, p_3\,]$

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

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

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



Étape 2 : groupe $[\,p_4, p_5, p_6\,]$

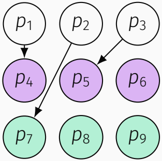
$p_4 \rightsquigarrow \mathcal{U}([p_8, p_9])$

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

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

a déjà reçu un cadeau

EXEMPLE

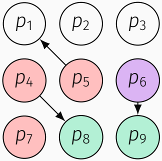


Étape 1 : groupe $[\, p_1, p_2, p_3 \,]$

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

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

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



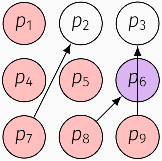
Étape 2 : groupe $[\, p_4, p_5, p_6 \,]$

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

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

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

a déjà reçu un cadeau



Étape 3 : groupe $[\, p_7, p_8, p_9 \,]$

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

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

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

GESTION DES DONNÉES

Table 1 – Format des données d'entrée

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

REPRÉSENTATION DES DONNÉES

Table 1 – Format des données d'entrée

	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 5 – Format des données manipulées

```
[[Stanislas, Alice], [Bob, David], ...]

Stanislas = ['Stanislas', 'MEZUREUX', 'MPSI1', 'stanmzx@gmail.com']
```

DÉFINITIONS

Chiffrer (C) Transformer un message afin qu'il ne soit lisible qu'à l'aide d'une clé

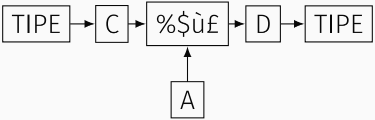


Figure 6 – Chiffrement symétrique

Chiffrer (C) Transformer un message afin qu'il ne soit lisible qu'à l'aide d'une clé

Déchiffrer (D) Opération inverse du chiffrement à l'aide de la clé

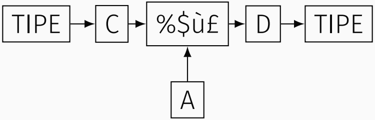


Figure 6 – Chiffrement symétrique

Chiffrer (C) Transformer un message afin qu'il ne soit lisible qu'à l'aide d'une clé

Déchiffrer (D) Opération inverse du chiffrement à l'aide de la clé

Décrypter (A) Opération inverse du chiffrement sans la clé

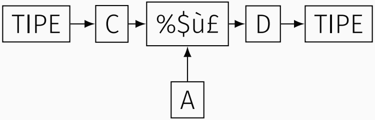


Figure 6 – Chiffrement symétrique

Cryptosystème asymétique

Chiffrement $\rightsquigarrow$ clé publique
Déchiffrement $\rightsquigarrow$ clé privée

Cryptosystème asymétique

Chiffrement $\rightsquigarrow$ clé publique

Déchiffrement $\rightsquigarrow$ clé privée

`NBITS` Entier naturel strictement supérieur à 1

Cryptosystème asymétique

Chiffrement $\rightsquigarrow$ clé publique
Déchiffrement $\rightsquigarrow$ clé privée

`NBITS` Entier naturel strictement supérieur à 1

q Entier premier de `NBITS` bits

Cryptosystème asymétique

Chiffrement $\rightsquigarrow$ clé publique

Déchiffrement $\rightsquigarrow$ clé privée

`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 : $x \hookleftarrow \mathcal{U}(\llbracket 2^{\texttt{NBITS}}, q-1 \rrbracket)$

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

Générer les clés

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

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

Chiffer $m \in \mathbb{N}$

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

le message chiffré est $(g^r \bmod q, mh^r \bmod q)$

Générer les clés

clé privée : $x \hookleftarrow \mathcal{U}([\![2^{\texttt{NBITS}}, q-1]\!])$

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

Chiffer $m \in \mathbb{N}$

$r \hookleftarrow \mathcal{U}([\![2^{\texttt{NBITS}}, q-1]\!])$

le message chiffré est $(g^r \bmod q, mh^r \bmod q)$

Déchiffrer (c_1, c_2)

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

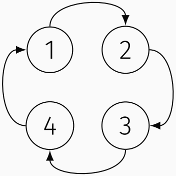
CHIFFREMENT DU TIRAGE

n participants représentés par les entiers de $[\![1, n]\!]$

$\rightsquigarrow$ tirage représenté par une liste de couples (donneur, receveur) puis chiffrée

n participants représentés par les entiers de $[\![1, n]\!]$

$\rightsquigarrow$ 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

Apprès chiffrement :

$[((c_1,c_2),(c_3,c_4)),((c_3,c_4),(c_5,c_6)),((c_5,c_6),(c_7,c_8)),((c_7,c_8),(c_1,c_2))]$

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

Condition de validité du tirage

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

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

Définition

Le cryptographe peut effectuer des opérations entre entiers chiffrés.
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 $(c_{2,1}, c_{2,2})$.

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

DÉNOMBREMENT DES TIRAGES

p : nombre de personnes par groupe

n : nombre de groupes

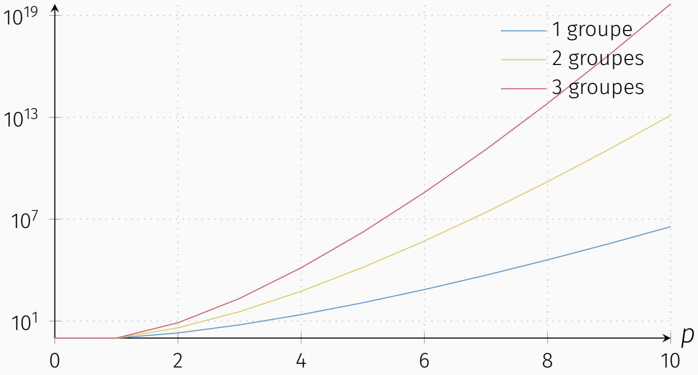


Figure 8 – Nombre de tirages possibles

CONCLUSION

CONCLUSION

- Algorithme opérationnel

CONCLUSION

- Algorithme opérationnel
- Prévention contre la triche

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

Questions ?

ANNEXES

CODE CHIFFREMENT HOMOMORPHE

```python
#!/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 q -> 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
# https://medium.com/@prudywsh/how-to-generate-big-prime-
↪  numbers-miller-rabin-49e6e6af32fb
def is_prime(n, k=128):
    if n == 2 or n == 3:
```

CODE CHIFFREMENT HOMOMORPHE

```python
        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:
```

CODE CHIFFREMENT HOMOMORPHE

```python
                    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
    return p


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

CODE CHIFFREMENT HOMOMORPHE

```
            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)}

```

```python

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}
```

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

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

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

CODE SECRET SANTA

```python

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]}”)
```

```python
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
```

```python
        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'
```

CODE SECRET SANTA

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

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

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 Nöel à toi !'
        )

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

CODE SECRET SANTA

```python
            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)
```

CODE SECRET SANTA

```python
    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:
```

```python
            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')
```

CODE DÉNOMBREMENT TIRAGES

```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created in 2022

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

from itertools import permutations


"""
Presumed formula : (nb_people_in_one_team !)^(nb_teams)
"""

```

```python

def check_draw(D, L, p, n):
    for e in D:
        if e[1] not in L[((e[0]-1)//p + 1)%n]:
            return False
    return True


def gen_draws(p, n):
    M = [i for i in range(1, n*p+1)]
    P = list(permutations(M))
    R = [[]]*len(P)
    for i, e in enumerate(P):
        D = [()]*(n*p)
        for j in range(len(e)):
            D[j] = (j, e[j])
```

```python
            R[i] = D
    return R


def count(p, n):
    L = [[p*i+j+1 for j in range(p)] for i in range(n)]
    R = gen_draws(p, n)
    res = 0
    for e in R:
        if check_draw(e, L, p, n):
            res += 1
    return res


# print(count(4, 1))
```