Accréditation anonyme et coloriage de graphe

Yvan Allioux
Abir Hsaine

IAI 3 - 2023

1. Introduction :

Le travail demandé s'articule autour de l'implémentation d'une accréditation anonyme par une preuve de connaissance à divulgation nulle de type 3-coloriage de graphe. Le but est de permettre à un utilisateur de démontrer qu'il connaît une manière de colorier un graphe avec 3 couleurs sans que deux nœuds adjacents aient la même couleur, tout en ne révélant rien d'autre sur ce coloriage.

2. Implémentation du programme :

a. Génération du graphe et coloriage :

La fonction genererGraphe3Coloriable est utilisée pour générer un graphe 3-coloriable.
Le coloriage aléatoire des nœuds est réalisé en choisissant une couleur parmi "rouge", "vert" et "bleu".
La matrice d'adjacence est générée en évitant les arêtes entre les nœuds ayant la même couleur.

b. Mise en gage des couleurs :

La fonction miseEnGageColoriage prend en entrée deux tableaux (l'un contenant les couleurs, l'autre des valeurs aléatoires) et renvoie un tableau contenant les valeurs mises en gage.
Ces valeurs sont obtenues en concaténant chaque couleur avec une valeur aléatoire, puis en hachant le résultat.

c. Preuve de connaissance à divulgation nulle :

La fonction preuveColoriage détermine si un utilisateur connaît le coloriage d'un graphe en se basant sur un protocole de vérification.
Dans ce protocole, après avoir envoyé une mise en gage de son coloriage, l'utilisateur doit révéler les couleurs de deux nœuds demandés par le vérificateur.

3. Analyse du protocole :

a. Propriété de completeness:

Si l'utilisateur et le vérificateur suivent le protocole honnêtement, l'utilisateur réussira toujours à convaincre le vérificateur car :

Le graphe est 3-coloriable par conception.
L'utilisateur détient le coloriage correct et sait comment les couleurs ont été mélangées.
Le vérificateur ne demande que des couleurs de nœuds reliés par une arête, ce qui garantit que les couleurs révélées seront toujours différentes.

b. Propriété de soundness:

Un utilisateur ne possédant pas de preuve d'un 3-coloriage aurait une probabilité négligeable de convaincre le vérificateur car :

Sans la connaissance du coloriage, il ne pourrait pas générer un engagement valide.
Il est improbable de deviner correctement la permutation des couleurs et les valeurs aléatoires utilisées.

c. Propriété de zero-knowledge:

Le protocole est à divulgation nulle car il ne révèle aucune information sur le coloriage réel, sauf que l'utilisateur le connaît. Le vérificateur ne voit que :

Des valeurs mises en gage, qui sont des hachages et ne peuvent donc pas être inversées pour découvrir le coloriage.
Deux couleurs de nœuds adjacents à chaque itération, qui ne révèlent pas le coloriage global, d'autant plus que les couleurs sont mélangées.

4. Conclusion :

Ce TP offre une introduction pratique à l'accréditation anonyme et aux protocoles de preuve à divulgation nulle, des concepts essentiels pour garantir la sécurité et la confidentialité dans les interactions numériques. L'implémentation proposée suit les directives du protocole défini.

5. Annexe & code :

1_generation-graf-et-coloriage.py

```
import random


class Graphe:
    def __init__(self, matrice_adjacence, coloriage):
        self.matrice_adjacence = matrice_adjacence
        self.coloriage = coloriage

def genererGraphe3Coloriable(n):
    couleurs = ["rouge", "vert", "bleu"]

    # Générer un coloriage aléatoire pour chaque nœud
    coloriage = [random.choice(couleurs) for _ in range(n)]

    # Initialiser la matrice d'adjacence avec des zéros
    matrice_adjacence = [[0] * n for _ in range(n)]

    # Remplir la matrice d'adjacence en évitant les arêtes entre nœuds
de même couleur
    for i in range(n):
        for j in range(i + 1, n):
            if coloriage[i] != coloriage[j]:
                matrice_adjacence[i][j] = matrice_adjacence[j][i] =
random.choice([0, 1])

    # Créer un objet Graphe avec la matrice d'adjacence et le coloriage
    graphe = Graphe(matrice_adjacence, coloriage)

    return graphe

#Partie 2:

# Exemple d'utilisation
if __name__ == "__main__":
    n = 5  # Nombre de nœuds dans le graphe
    graphe = genererGraphe3Coloriable(n)

    print("Matrice d'adjacence:")
    for row in graphe.matrice_adjacence:
```

```
        print(row)

    print("Coloriage des nœuds:")
    for i, couleur in enumerate(graphe.coloriage):
        print(f"Nœud {i + 1}: {couleur}")
```

2_mise-en-gage-des-couleures.py

```
import hashlib

def miseEnGageColoriage(couleurs, valeurs_aleatoires):
    if len(couleurs) != 20 or len(valeurs_aleatoires) != 20:
        raise ValueError("Les tableaux doivent avoir une taille de
20.")

    valeurs_mises_en_gage = []

    for i in range(20):
        couleur = couleurs[i]
        valeur_aleatoire = valeurs_aleatoires[i]

        # Concaténer la couleur et la valeur aléatoire en tant que
chaîne de caractères
        # Calculer le hachage SHA-1
        hachage= hashlib.sha1((str(valeur_aleatoire) +
couleur).encode()).hexdigest()

        valeurs_mises_en_gage.append(hachage)

    return valeurs_mises_en_gage

# Exemple d'utilisation
if __name__ == "__main__":
    couleurs = ["rouge", "vert", "bleu", "rouge", "vert", "bleu",
"rouge", "vert", "bleu", "rouge",
                "vert", "bleu", "rouge", "vert", "bleu", "rouge",
"vert", "bleu", "rouge", "vert"]
```

```
    valeurs_aleatoires = [12345, 67890, 98765, 54321, 101010, 111111,
222222, 333333, 444444, 555555,
                          666666, 777777, 888888, 999999, 123456,
654321, 987654, 321098, 13579, 24680]

    valeurs_mises_en_gage = miseEnGageColoriage(couleurs,
valeurs_aleatoires)

    for i, valeur in enumerate(valeurs_mises_en_gage):
        print(f"Valeur mise en gage pour nœud {i + 1}: {valeur}")
```

3_preuve_de_conaisance_a_din_nul.py

```
import random
import hashlib

class Graphe:
    def __init__(self, matrice_adjacence, coloriage):
        self.matrice_adjacence = matrice_adjacence
        self.coloriage = coloriage

def genererGraphe3Coloriable(n):
    couleurs = ["rouge", "vert", "bleu"]

    # Générer un coloriage aléatoire pour chaque nœud
    coloriage = [random.choice(couleurs) for _ in range(n)]

    # Initialiser la matrice d'adjacence avec des zéros
    matrice_adjacence = [[0] * n for _ in range(n)]

    # Remplir la matrice d'adjacence en évitant les arêtes entre nœuds
de même couleur
    for i in range(n):
        for j in range(i + 1, n):
            if coloriage[i] != coloriage[j]:
                matrice_adjacence[i][j] = matrice_adjacence[j][i] =
random.choice([0, 1])

    # Créer un objet Graphe avec la matrice d'adjacence et le coloriage
    graphe = Graphe(matrice_adjacence, coloriage)
```

```
    return graphe


def miseEnGageColoriage(couleurs, valeurs_aleatoires):
    if len(couleurs) != 20 or len(valeurs_aleatoires) != 20:
        raise ValueError("Les tableaux doivent avoir une taille de
20.")

    valeurs_mises_en_gage = []

    for i in range(20):
        couleur = couleurs[i]
        valeur_aleatoire = valeurs_aleatoires[i]

        # Concaténer la couleur et la valeur aléatoire en tant que
chaîne de caractères
        # Calculer le hachage SHA-1

        hachage= hashlib.sha1((str(valeur_aleatoire) +
couleur).encode()).hexdigest()

        valeurs_mises_en_gage.append(hachage)

    return valeurs_mises_en_gage

def preuveColoriage(matrice, valeurs_mises_en_gage, i, j, colors,
random_values):
    # Vérifier les engagements
    if valeurs_mises_en_gage[i] != hashlib.sha1((str(random_values[i])
+ colors[i]).encode()).hexdigest():
        return False
    if valeurs_mises_en_gage[j] != hashlib.sha1((str(random_values[j])
+ colors[j]).encode()).hexdigest():
        return False

    # Vérifier qu'ils sont connectés
    if matrice[i][j] == 0:
        return False

    # Vérifier que les couleurs sont différentes
    if colors[i] == colors[j]:
        return False
```

```
    return True

if __name__ == "__main__":
    # Générer des valeurs aléatoires
 random_values = [random.randint(0, 2**128 - 1) for _ in range(20)]
 n = 20  # Nombre de nœuds dans le graphe
 graphe = genererGraphe3Coloriable(n)
    # Mise en gage des couleurs
 commitments = miseEnGageColoriage(graphe.coloriage, random_values)
 print (commitments)

    # Test du protocole
 authentifie = True
 for _ in range(400):
        i, j = random.choice([(i, j) for i in range(20) for j in
range(20) if graphe.matrice_adjacence[i][j] == 1])
        if not preuveColoriage(graphe.matrice_adjacence, commitments,
i, j, graphe.coloriage, random_values):
            authentifie = False
            break


 if authentifie:
        print("L'utilisateur est authentifié!")
 else:
        print("L'utilisateur n'a pas pu être authentifié.")
```