ACADÉMIE D'AIX-MARSEILLE
Liberté Égalité Fraternité

TraAM

ACADÉMIE DE NICE
Liberté Égalité Fraternité

Esprit Critique :

Vers le hashage :

objectif du TP : Comprendre une "signature" pour vérifier l'intégrité d'un message, d'une photo

Cybersécurité:

La valeur de hachage (ou hashage) est une valeur numérique d'une longueur fixe qui identifie de manière unique les données. Une valeur de hachage est le résultat d'un algorithme de hachage. Voici les plus courants:

1. **MD5** (Message Digest, cf la RFC 1321 (https://www.ietf.org/rfc/rfc1321.txt)), conçue par Ron Rivest en 1992, est une fonction de hachage cryptographique largement utilisée avec une valeur de hachage de 128 bits. Les hachages MD5 ne sont PAS considérés comme cryptographiquement sécurisés. En 2011, l'IETF a publié le RFC 6151, « Updated Security Considerations for the MD5 Message-Digest and the HMAC-MD5 Algorithms », qui a mentionné un certain nombre d'attaques contre les hachages MD5, y compris la collision de hachage.

2. **SHA-1** (Secure Hash Algorithm 1, cf la RFC 3174 (https://www.rfc-editor.org/rfc/rfc3174.html)), inventée par la National Security Agency (NSA) des États-Unis en 1995. SHA-1 prend une entrée et produit une chaîne de valeur de hachage de 160 bits sous la forme d'un nombre hexadécimal à 40 chiffres. Le NIST a déprécié l'utilisation de SHA-1 en 2011 et a interdit son utilisation des signatures numériques à la fin de 2013, en raison de sa sensibilité aux attaques par force brute. Au lieu de cela, le National Institute of Standards and Technology (NIST) recommande de migrer du SHA-1 vers des algorithmes de hachage plus forts dans les familles SHA-2 et SHA-3.

3. **SHA-2** (Secure Hash Algorithm 2), conçue le NIST et la NSA en 2001 pour remplacer le SHA-1. SHA-2 a de nombreuses variantes, et sans doute la plus courante est SHA-256. L'algorithme SHA-256 retourne une valeur de hachage de 256 bits sous la forme d'un nombre hexadécimmal à 64 chiffres.

Un hachage n'est pas considéré comme sécurisé si deux fichiers ont la même valeur de hachage ou le même condensat.

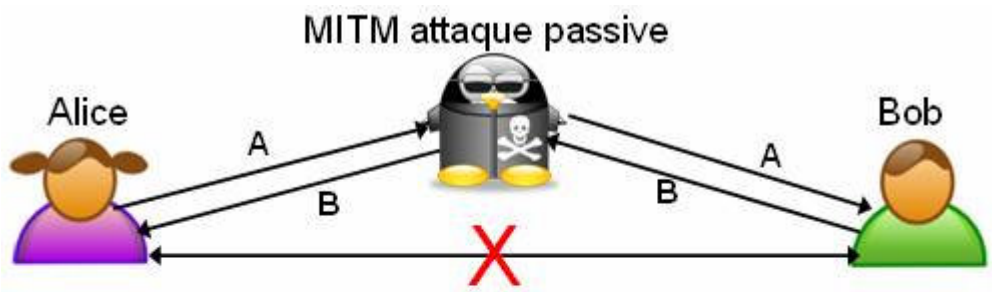
Les professionnels de la sécurité utilisent généralement les valeurs de hachage pour obtenir des informations sur un échantillon de logiciel malveillant spécifique, un fichier malveillant ou suspect, et comme un moyen d'identifier et de référencer de manière unique l'artefact malveillant.

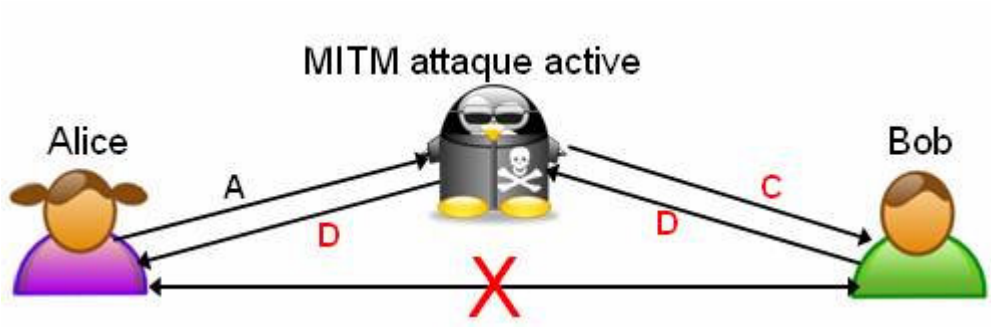
Entrée[1]:
```
from IPython.display import HTML
HTML("""<iframe width="560" height="315" src="https://www.youtube.com/embe
```

Sortie[1]:

Niveau SNT / Collège

Alice souhaite envoyer un message à Bob, mais celle-ci craint que le message puisse être modifié pendant le transfert. Elle a entendu parlé de l'attaque de l'homme du milieu (Man In The Middle, en abrégé MITM).





Comment Bob peut-il être sûr de l'intégrité du message?

#Double cliquez dans cette cellule pour votre proposition

Attentes professeurs , idées on l'envoie plusieurs fois, on peut aussi dire qu'on procède à la manière de la double authentification des banques avec l'envoi d'un code, donc Alice doit être capable de calculer le code et de l'envoyer. Puis Bob doit être capable de calculer le code du message reçu et de le comparer à celui qu'Alice a envoyé.

L'idée est d'envoyer le message, et d'envoyer l'empreinte séparement en utilisant un chiffrement asymétrique (plus lent, plus de ressources)

Alice envoie "mot" mais Bob reçoit "mat", comment pourrait-on faire pour que Bob soit sûr de l'intégrité du message reçu?

#Double cliquez dans cette cellule pour votre proposition

Alice : "Mot" et 138097 Bob reçoit "Mat", calcule empreinte("Mat").

Entrée[3]:

```
# pour SNT et collège :
def empreinte(data):
    empreinte_valeur = 1
    for char in str(data):
        empreinte_valeur = (empreinte_valeur * 31 + ord(char))
    return empreinte_valeur%10**9
```

Utilisation de la méthodologie PRIMM (Predict Run Investigate Modify Make)

Expliquez ce que cette fonction est sensée effectuer.

#Double cliquez dans cette cellule pour votre proposition

Exécutez la cellule ci-dessous

Entrée[ ]:
```
# Exemple d'utilisation :
exemple = "mot"
résultat = empreinte(exemple)
print(résultat)
```

La sortie correspond elle à ce que vous aviez prédit?

#Double cliquez dans cette cellule pour votre proposition

Expliquez pas à pas ce qu'effectue cette fonction. A quoi correspond la fonction ord ?

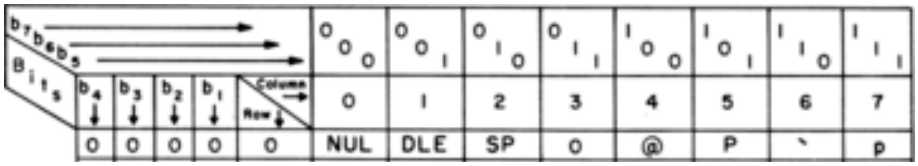
#Double cliquez dans cette cellule pour votre proposition

Modifiez cette fonction pour obtenir un condensat (une empreinte) plus longue

#Double cliquez dans cette cellule pour votre proposition

Avec la table ASCII pour première NSI:

Retour sur table ASCII.



0	0	0	1	1	SOH	DC1	!	1	A	Q	a	q
0	0	1	0	2	STX	DC2	"	2	B	R	b	r
0	0	1	1	3	ETX	DC3	#	3	C	S	c	s
0	1	0	0	4	EOT	DC4	$	4	D	T	d	t
0	1	0	1	5	ENQ	NAK	%	5	E	U	e	u
0	1	1	0	6	ACK	SYN	&	6	F	V	f	v
0	1	1	1	7	BEL	ETB	'	7	G	W	g	w
1	0	0	0	8	BS	CAN	(	8	H	X	h	x
1	0	0	1	9	HT	EM	)	9	I	Y	i	y
1	0	1	0	10	LF	SUB	*	:	J	Z	j	z
1	0	1	1	11	VT	ESC	+	;	K	[	k	{
1	1	0	0	12	FF	FS	,	<	L	\	l	¦
1	1	0	1	13	CR	GS	—	=	M	]	m	}
1	1	1	0	14	SO	RS	.	>	N	^	n	~
1	1	1	1	15	SI	US	/	?	O	_	o	DEL

Entrée[ ]:
```
#créer fonction ord avec les élèves NSI
def empreinte(data):
    empreinte_valeur = 1
    for char in str(data):
        empreinte_valeur = (empreinte_valeur * 31 + ord(char)) & 0xFFF
    return empreinte_valeur
# & 0b11111111
# Exemple d'utilisation :
exemple = "Fer dde ddkufuheiorghrtgh dskfihreuigzhre"
résultat = empreinte(exemple)
print("Empreinte des données :", hex(résultat))
```

Combien d'empreintes différentes peut-on obtenir avec la fonction précédente?

#Double cliquez dans cette cellule pour votre proposition

Entrée[ ]:
```
Quel problème cela peut-il générer?
```

#Double cliquez dans cette cellule pour votre proposition

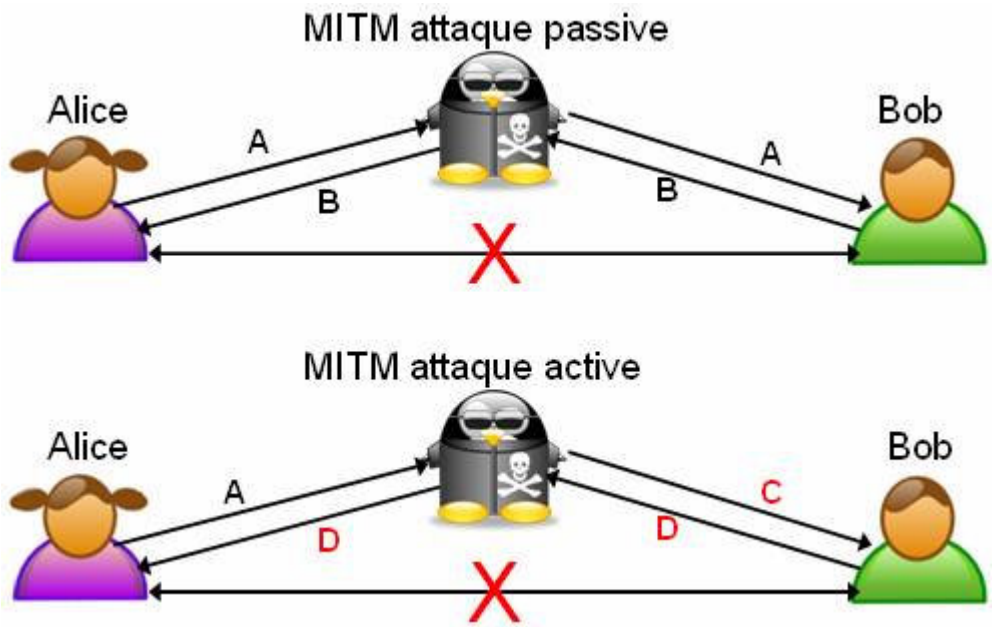
Les collisions de hachage : c'est lorsque deux entrées différentes produisent le même résultat de hachage. Cette probabilité est extrêmement faible dans la plupart des algorithmes de hachage.

Proposer, ci-dessous, une modification du code précédent pour décupler ce nombre

#Double cliquez dans cette cellule pour votre proposition

Type *Markdown* and LaTeX: α^2

Communication facile ou pas de l'empreinte des données?



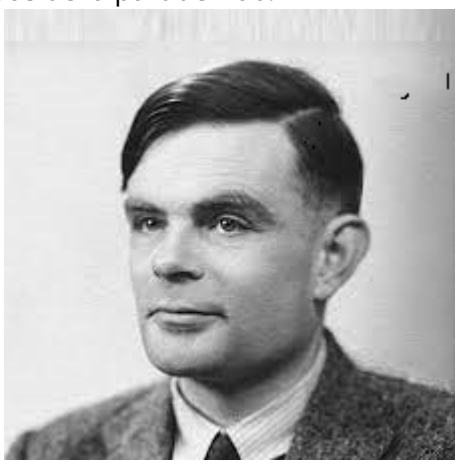
Alice : "Mot" et 138097 Bob reçoit "Mat", calcule empreinte("Mat"). Que peut-on en déduire?

Entrée[ ]:
```
#your code
```

Modification de photo ?

Bob envoie la photo, ci-dessous, d'Alan Turing à Alice. Mais comment être sûr qu'Alice reçoive la "bonne" photo ?

Alice reçoit la photo ci-dessous de la part de Bob.



. Comment faire pour vérifier que la photo reçu est bien celle que Bob a envoyé?

#Double cliquez dans cette cellule pour votre proposition

Partie à retravailler

Entrée[7]:

```
import numpy as np
import matplotlib.pyplot as plt
img = np.array("turing1.jpg")

print(img.size)
plt.imshow(img) ;
#plt.show() # On peut continuer à utiliser la fonction imshow pour visuali

#print("Dimensions de l'image : ",img.shape) # Dimensions x, y et nombre d

#print("Pixel 500 sur 200 : ",img[200,500])

#print("Composante rouge de ce dernier pixel :",img[200,500][0])

# Tous le rouge de tous les pixels

#print("Rouge : ",img[:,:,0])
```

```
1
```

Entrée[ ]:

Entrée[18]:

```
from PIL import Image
import numpy as np

# Charger l'image JPEG
image_path = "turing1.jpg"
image = Image.open(image_path)
print(image.size)
# Vérifier que l'image a bien les dimensions 255x255
if image.size != (225, 225):
    print("L'image n'a pas les dimensions 255x255 pixels.")
    exit()

# Convertir l'image en une matrice de pixels
pixel_matrix = np.array(image)

# Afficher les dimensions de la matrice
print(f"Dimensions de la matrice : {pixel_matrix.shape}")
```

```
(225, 225)
Dimensions de la matrice : (225, 225, 3)
```

Entrée[19]:

```
from PIL import Image
import numpy as np

# Charger l'image JPEG
image_path = "turing1.jpg"
image = Image.open(image_path)

# Convertir l'image en niveaux de gris
image_grayscale = image.convert("L")

# Convertir l'image en niveaux de gris en une matrice de pixels
pixel_matrix = np.array(image_grayscale)

# Afficher les dimensions de la matrice
print(f"Dimensions de la matrice : {pixel_matrix.shape}")
```

```
Dimensions de la matrice : (225, 225)
```

Entrée[ ]:
```python
from PIL import Image

def griser_image(image_path):
    image = Image.open(image_path)
    # Convertir l'image en niveaux de gris
    grayscale_image = image.convert("L") # L=luminance = 0.2126 * R + 0.71

    # Récupérer les données de l'image en niveaux de gris sous forme de ma
    grayscale_matrix = list(grayscale_image.getdata())
    return grayscale_matrix

# Exemple d'utilisation :
image_path = 'turing1.jpg'
matrice = griser_image(image_path)
print(matrice)
```

Entrée[ ]:
```python
from PIL import Image

def image_to_grayscale(image_path):
    # Charger l'image
    image = Image.open(image_path)

    # Convertir l'image en niveaux de gris
    grayscale_image = image.convert("L")

    # Récupérer les données de l'image en niveaux de gris sous forme de ma
    grayscale_matrix = list(grayscale_image.getdata())

    return grayscale_matrix

# Exemple d'utilisation :
image_path = 'turing1.jpg'
grayscale_matrix = image_to_grayscale(image_path)
print(grayscale_matrix)
```

Avec du pixelart noir et blanc

Entrée[6]:

```
def pixelart_hash(pixelart):
    # Convertir la représentation du pixelart en chaîne
    pixelart_str = "\n".join("".join(map(str, row)) for row in pixelart)
    # Utiliser la fonction de hachage simple
    hash_value = empreinte(pixelart_str)
    return hash_value

# Exemple d'utilisation :
exemple_pixelart = [
    [1, 0, 1],
    [1, 0, 1],
    [0, 1, 1]
]

hash_result = pixelart_hash(exemple_pixelart)
print("Hash du pixelart :", hash_result)
```

```
Hash du pixelart : 516629759
```

L'utilisation d'une fonction de hachage permet de ne pas stocker les mots de passe en clair dans la base mais uniquement de stocker une empreinte de ces derniers. Il est important d'utiliser un algorithme public réputé fort afin de calculer les dites empreintes. (CNIL)

Entrée[3]:

```
from ipythonblocks import BlockGrid
```

Entrée[22]:

```

grille=BlockGrid(5, 1, block_size=30)
grille[0,0]=(0,0,0)
grille[0,1]=(255, 255, 255)
grille[0,2]=(255,0,0)
grille[0,3]=(0,255,0)
grille[0,4]=(0,0,255)
grille
```

Sortie[22]:

Les seuls couleurs utilisables sont Noir, Blanc, Rouge, Vert et Bleu. On associe respectivement à ces couleurs les entiers 1, 2, 3, 4 et 5.

Entrée[23]:

```
N = (0, 0, 0)
P = (255, 0, 0)
matrice = [
    [N, N, N, N, N, N],
    [N, N, P, P, N, P],
    [N, P, P, P, P, P],
    [N, P, P, P, P, P],
    [N, N, P, P, P, P],
    [N, N, N, P, P, P]
        ]
def dessiner(mat):
    nb_lignes = len(mat)
    nb_colonnes = len(mat[0])
    grille = BlockGrid(nb_colonnes, nb_lignes, block_size=30)
    for ligne in range(nb_lignes):
        for colonne in range(nb_colonnes):
            grille[ligne, colonne] = mat[ligne][colonne]
    return grille
dessiner(matrice)
```

Sortie[23]:

Entrée[26]:

```
def hash_pixelart(pixelart):
    hash_value = 0
    for row in pixelart:
        for pixel in row:
            hash_value = hash_value * 31 + pixel  # Utilisation d'une fonc

    return hash_value

# Exemple d'utilisation :
pixelart = [
    [1, 0, 0],    # Première ligne de pixels (rouge)
    [0, 1, 0],    # Deuxième ligne de pixels (vert)
    [0, 1, 1]     # Troisième ligne de pixels (bleu)
]

hash_value = hash_pixelart(pixelart)
print("Hash du pixelart:", hash_value)
```

```
Hash du pixelart: 852891960994
```

Attaque à l'aide des dictionnaires arc-en-ciel