Criptografia i Seguretat

Hash Lab

En aquesta pràctica buscarem col·lisions a funcions hash. Per aquest propòsit definirem una funció hash simplificada per simplificar el cost computacional.

1 Objectiu

En aquesta pràctica veurem d'una forma molt simplificada i generica la facilitat o dificultat en trobar col·lisions en funcions hash. Les colisions les buscarem sense fer servir cap altra tècnica que la proba de valors aleatoris (força bruta).

Realitzeu els exercicis que es detallen. Heu de respectar el nom de les funcions així com els seus paràmetres i valors de sortida. Al final teniu uns testos que us poden ajudar en el desenvolupament. Podeu definir totes les funcions auxiliars que considereu oportunes.

En el cas de la cerca de col·lisions peseu que en alguns casos el temps d'execució pot ser elebat. Penseu en incloure alguna condició per si de cas com un nombre màxim d'interacions.

Implementarem una funció hash y perque la compleixtat del calculs a realitzar no sigui desproporcionada, utilitzarem una simplificacio de la funcio MD5.

```
import hashlib
from typing import Optional, Tuple
```

2 Exercici 1

Heu de programar una funcio 'uab_md5' que realitzara un hash MD5 y retornara el primers bits de la sortida del MD5. Com a funció hash MD5 farem servir la implementció MD5 del mòdul 'hashlib' de Python.

Parametres:

- **message**: missatge al que aplicar la funció hash. Serà una cadena de caràcters de mida arbitrària.
- **num_bits**: nombre de bits de sortida que serà un valor entre 1 i 128.

La funció retorna com a sortida els 'num_bits' **més significatius** de la sortida de la funció MD5, representats en **format decimal**. Es a dir com un únic nombre decimal. En cas no de poder fer el càlcul pel motiu que sigui retornarà 'None'.

```
def uab_md5(message: str, num_bits: int) -> Optional[int]:
```

3 Exercici 2

Programeu ara una funció que trobi segones preimatges per a la funció 'uab_md5'. Aquesta funció rebrà som a paràmetres un missatge i el nombre de bits del hash.

Paràmetres:

- **message**: cadena de caràcters representant el missatge del que volem trobar una preimatge del seu hash.
- **num_bits**: nombre de bits del hash, entre 1 i 128.

La funció ha de retornar dos valors:

- un missatge, diferent del proporcionat a l'entrada, que tingui el mateix valor hash per la funcio 'uab_md5' amb el nombre de bit 'num_bits'.
- el nombre de iteracions que s'han hagut de fer per trobar la segona preimatge. Es a dir, el nombre de missatges que heu hagut de probar.

En cas de no trobar un resultat, la funció ha de retornar 'None'.

```
def second_preimage(message: str, num_bits: int) -> Optional[Tuple[str, int]]:
```

4 Exercici 3

Programeu una funció que trobi col·lisions per a la funció 'uab_md5'.

La funció rebrà com a argument el nombre de bit del hash, 'num_bits', y retornarà dos missatges diferents que tingui el mateix valor hash per la funció 'uab_md5' amb el nombre de bits especificat. Com en el cas anterior, també ha de retornar el nombre de iteracions fetes.

En cas de no trobar un resultat, la funció ha de retornar 'None'.

```
def collision(num_bits: int) -> Optional[Tuple[str, str, int]]:
```

5 Exercici 4

Ara analitzeu la dificultat d'aconseguir segones preimatges i col·lisions de la funció 'uab_md5' amb un nombre de bits determinat. Per aixo mesurarem el temps i el nombre de iteracions (valors hash que hem hagut de probar).

Per mesurar el temps d'execució de la funció es recomana fer servir la funció 'perf_counter()' del mòdul 'time' de Python.

5.1 Part A

Evaluarem la funció 'uab_md5' per le diferentes mides de bits entre **1 i 24**. Per cada mida buscarem:

- una segona preimatge amb la funció 'second_preimage', i guardarem el temps d'execució i nombre d'iteracions en cada cas.

- una col·lisió amb la funció 'collision' i guardarem el temps d'execució i nombre d'iteracions en cada cas.

Un cop teniu les dades les heu de mostrar les dades:

- En forma de taula
- En forma de gràfica:
- Gràfica de temps: mostra el temps que es triga per obtenir un resultat en funció del nombre de bits en cada cas.
- Gràfica de iteracions: mostra el nombre de iteracions o valors que s'han hagut de generar en funció del nombre de bits per cada cas.

Per les gràfiques podeu fer una gràfica per cada cas, no cal que mostreu tot junt. Feu servir les cel·les de codi i text que considereu oportunes

5.2 Part B

Finalment, calculeu el nombre teoric aproximat de valors hash necesaris per trobar colisions fortes per cada mida de bits i compareu aquest valor amb el nombre de iteracions obtingut de forma empírica amb la funció 'collision'. La comparació la podeu fer de forma gràfica per visualitzar millor la posible diferencia entre els dos valors. Raoneu si els valors obtinguts son els que esperarieu a nivell teoric o no i per què.

Tingueu en compte que el càlcul de tots els casos pot arribar a tenir un cost temporal elevat.

6 Proves

Per tal de facilitar la implementació de les funcions proporcionem a continuació uns testos unitaris.

```
import unittest

class TestLab1(unittest.TestCase):

    def test_uab_md5(self):
        test_vectors_ok = (
            ["hola", 100, 381757249806289069081790873225],
            ["hola", 1, 0],
            ["dfk3874", 68, 229291433845740375560],
            ["dfk3874", 64, 14330714615358773472],
            ["Alexandria", 128, 221630910082124901698625759824682079437],
            ["Alexandria", 129, None],
            ["Alexandria", 0, None])

        for t in test_vectors_ok:
            my_value = uab_md5(t[0], t[1])
            self.assertEqual(my_value, t[2])

    def test_second_preimage(self):
```

```
        msg = "find a second preimage"
        for n in range(1, 15):
            new_msg, _ = second_preimage(msg, n)
            self.assertEqual(uab_md5(new_msg, n), uab_md5(msg, n))
            self.assertNotEqual(new_msg, msg)

    def test_collision(self):
        for n in range(1, 15):
            msg1, msg2, _ = collision(n)
            self.assertEqual(uab_md5(msg1, n), uab_md5(msg2, n))
            self.assertNotEqual(msg1, msg2)


unittest.main(argv=[''], verbosity=2, exit=False, buffer=True)
```

Lliurament

Cal lliurar un pdf amb tot el que heu fet i a més a més el codi que heu programat pel campus virtual en la data establerta. Recordeu posar el vostre nom al pdf lliurat.

Referencies

Enllaços addicionals que poden ser útils:

- Paar, Christof, and Jan Pelzl. Understanding Cryptography: A Textbook for Students and Practitioners. 2010. **Chapter 11**
- Wikipedia contributors, "Birthday attack,"Wikipedia, The Free Encyclopedia, `https://en.wikipedia.org/w/index.php?title=Birthday_attack&oldid=1013321734` (accessed Feb 14, 2022).