FS2024: 21109 Privacy and Data Security - Exercise 10

IBAN Tokenization

Student: Eduardo Rodrigues Amaral - EU-110079796

Library Imports

In [1]:
```
import hashlib
import hmac
import sys
```

10.1 Irreversible tokenization

An *International Bank Account Number (IBAN)* contains a two-digit checksum to protect it from errors made by humans when copying the number.

An IBAN is a string of up to 34 alphanumeric characters (digits and capital letters). The string contains, in this order, (1) a country code of two letters, (2) two decimal check digits, and (3) an account number of up to 30 characters. Format and length of the account number vary by country. An IBAN in Switzerland consists of 21 characters according to the pattern:

`CHzz bbbb bccc cccc cccc c`

Here CH denotes the country code, zz are the check digits, $bbbbb$ denote the bank (Swiss Bank Clearing Number), and $cc...$ contains the account number at the bank.

To check the validity of an IBAN string s, perform these four steps:

1. Move the first four characters to the end of s.
2. Replace every letter in s by two digits, according to the rule $A = 10$, $B = 11$, $...$, $Z = 36$. Note that s may get longer.
3. Interpret s as a number n in decimal notation.
4. If $n \equiv 1$ $(mod$ $97)$, then s contains a valid IBAN, otherwise not.

a)

Implement a function `check_iban(s)` in Python that returns `True` whenever the IBAN in string s is valid.

In [2]:
```
def check_iban(iban):
  # Normalize the string (remove spaces and convert to uppercase)
  iban = iban.replace(' ', '').upper()

  # Move the first four characters to the end
  modifiedIban = iban[4:] + iban[:4]

  # Replace every letter with its corresponding number (A=10, ..., Z=36)
  numericIban = ''
  for char in modifiedIban:
    if char.isalpha():
```

```
      # Convert letter to number (A=10, B=11, ..., Z=36)
      numericIban += str(ord(char) - ord('A') + 10)
    else:
      numericIban += char

  # Convert the string to a large integer
  ibanNumber = int(numericIban)

  # Check if the number modulo 97 is equal to 1
  return ibanNumber % 97 == 1
```

In [3]:
```
validIban = 'CH9300762011623852957'
invalidIban = 'CH9300762011623852958'

print(f'Valid IBAN ({validIban}): {check_iban(validIban)}')
print(f'Invalid IBAN ({invalidIban}): {check_iban(invalidIban)}')
```

```
Valid IBAN (CH9300762011623852957): True
Invalid IBAN (CH9300762011623852958): False
```

b)

Implement a hash-based *irreversible tokenization* function `hash_token(s)` that takes valid Swiss IBAN in s and returns another valid Swiss IBAN. Use the cycle-walking method and SHA-256 from Python's `hashlib`.

In [4]:
```
def hash_token(s):
  if not check_iban(s):
    raise ValueError("Invalid IBAN provided")

  while True:
    # Extract the numeric part of the IBAN
    numericIbanPart = s[2:]  # Remove 'CH' prefix

    # Hash the current numeric IBAN part
    hashDigest = hashlib.sha256(numericIbanPart.encode()).hexdigest()

    # Convert hash to integer and truncate to 19 digits
    # to obtain a new IBAN numeric part candidate
    newNumericIbanPartCandidate = int(hashDigest, 16) % 10**19

    # Construct a new Swiss IBAN with 'CH'
    newIbanCandidate = 'CH' + str(newNumericIbanPartCandidate)

    # Validate the new IBAN
    if check_iban(newIbanCandidate):
      return newIbanCandidate

    # Prepare next iteration with the new IBAN candidate
    s = newIbanCandidate

print(f'Hashing IBAN {validIban}...')

hashedIban = hash_token(validIban)

print(f'Hashed IBAN: {hashedIban}')

print(f'Valid IBAN ({hashedIban}): {check_iban(hashedIban)}')

secondTimeHashedIban = hash_token(validIban)

print(f'Multiple hashing of the same IBAN results \
in the same hashed IBAN? {hashedIban == secondTimeHashedIban}')
```

```
Hashing IBAN CH9300762011623852957...
Hashed IBAN: CH3746013475049694776
Valid IBAN (CH3746013475049694776): True
Multiple hashing of the same IBAN results in the same hashed IBAN? True
```

c)

Extend your tokenization function to a *keyed* irreversible tokenization function `mac_token(key,s)`, which additionally takes an arbitrary string `key` as input that serves as the key. Use HMAC-SHA256 from Python's `hmac` library and the cycle-walking method.

In [5]:
```python
def mack_token(key,s):
  if not check_iban(s):
    raise ValueError("Invalid IBAN provided")

  while True:
    # Extract the numeric part of the IBAN
    numericIbanPart = s[2:]  # Remove 'CH' prefix

    # Use HMAC with SHA-256 and the given key to hash
    # the current numeric part of the IBAN
    hashDigest = hmac.new(key.encode(),
              numericIbanPart.encode(), hashlib.sha256).hexdigest()

    # Convert hash to integer and truncate to 19 digits to
    # obtain a new IBAN numeric part candidate
    newNumericIbanPartCandidate = int(hashDigest, 16) % 10**19

    # Construct a new Swiss IBAN with 'CH'
    newIbanCandidate = 'CH' + str(newNumericIbanPartCandidate)

    # Validate the new IBAN
    if check_iban(newIbanCandidate):
      return newIbanCandidate

    # Prepare next iteration with the new IBAN candidate
    s = newIbanCandidate

secretKey = 'eduardo'

print(f'Hashing IBAN {validIban} with secret key "{secretKey}"...')

hashedIban = mack_token(secretKey, validIban)

print(f'Hashed IBAN: {hashedIban}')

print(f'Valid IBAN ({hashedIban}): {check_iban(hashedIban)}')

secondTimeHashedIban = mack_token(secretKey, validIban)

print(f'Multiple hashing of the same IBAN with \
same key results in the same hashed IBAN? \
{hashedIban == secondTimeHashedIban}')

differentSecretKey = 'amaral'

print(f'Hashing IBAN {validIban} with \
different secret key "{differentSecretKey}"...')

hashedIbanWithDifferentKey = mack_token(differentSecretKey, validIban)

print(f'Hashed IBAN: {hashedIbanWithDifferentKey}')
```

```
print(f'Valid IBAN ({hashedIbanWithDifferentKey}): \
{check_iban(hashedIbanWithDifferentKey)}')

print(f'Hashed IBAN with different key is different \
than hashed IBAN with original key? \
{hashedIban != hashedIbanWithDifferentKey}')
```

```
Hashing IBAN CH9300762011623852957 with secret key "eduardo"...
Hashed IBAN: CH3150821075000344956
Valid IBAN (CH3150821075000344956): True
Multiple hashing of the same IBAN with same key results in the same hashed IBAN? True
Hashing IBAN CH9300762011623852957 with different secret key "amaral"...
Hashed IBAN: CH4916437570836671086
Valid IBAN (CH4916437570836671086): True
Hashed IBAN with different key is different than hashed IBAN with original key? True
```

10.2 Reversible tokenization with FPE

Format-preserving encryption (FPE) can be used for *reversible tokenization*. The auxiliary file `ex10-smallcipher.py` contains a Python implementation of a small-domain encryption algorithm `smallcipher_encrypt`, according to Black and Rogaway [BR02] and Bellare et al. [BRRS09, Scheme FE2/FD2, Fig. 3]

In [6]:
```
# ex10-smallcipher.py auxiliary code

def smallcipher_encrypt(key, tweak, hashalg, m, a, b, r):
  L = m // b
  R = m % b
  for i in range(1, r + 1):
    if (i & 0x01):
      s = a
    else:
      s = b
    tmp = R
    rstr = (str(a) + '|' + str(b) + '|' + str(tweak) + '|' + str(i)
      + '|' + str(R)).encode('ascii')
    f = int.from_bytes(hmac.digest(key, rstr, hashalg),
          byteorder=sys.byteorder)
    R = (L + f) % s
    L = tmp
  return s * L + R
```

a)

Implement an FPE-based *reversible tokenization* function `fpe_encrypt(key,tweak, s)` to compute an FPE of a string s that contains a valid Swiss IBAN. Let `key` be the encryption key and `tweak` be an arbitrary string that “tweaks” the small-domain cipher; it can be thought of a domain separator.

In [7]:
```
def fpe_encrypt(key,tweak, s):
  if not check_iban(s):
    raise ValueError("Invalid IBAN provided")

  while True:
    # Extract the numeric part of the IBAN
    numericIbanPart = s[2:]  # Remove 'CH' prefix

    # Encrypt the current numeric part of the IBAN
    encryptedNumeric = smallcipher_encrypt(key.encode(),
       tweak, hashlib.sha256 , int(numericIbanPart), 10**19, 10**19, 10)
```

```
    # Truncate to 19 digits to obtain a new IBAN numeric part candidate
    newNumericIbanPartCandidate = encryptedNumeric % 10**19

    # Construct a new Swiss IBAN with 'CH'
    newIbanCandidate = 'CH' + str(newNumericIbanPartCandidate)

    # Validate the new IBAN
    if check_iban(newIbanCandidate):
      return newIbanCandidate

    # Prepare next iteration with the new IBAN candidate
    s = newIbanCandidate


secretKey = 'eduardo'
tweak = 'tweak'

print(f'Encrypting IBAN {validIban} with secret key \
"{secretKey}" and tweak "{tweak}"...')
encryptedIban = fpe_encrypt(secretKey, tweak, validIban)

print(f'Encrypted IBAN: {encryptedIban}')

print(f'Valid IBAN ({encryptedIban}): {check_iban(encryptedIban)}')

secondTimeEncryptedIban = fpe_encrypt(secretKey, tweak, validIban)

print(f'Multiple encryption of the same IBAN with same key and tweak \
results in the same encrypted IBAN? \
{encryptedIban == secondTimeEncryptedIban}')
```

```
Encrypting IBAN CH9300762011623852957 with secret key "eduardo" and tweak "tweak"...
Encrypted IBAN: CH6833786179316117748
Valid IBAN (CH6833786179316117748): True
Multiple encryption of the same IBAN with same key and tweak results in the same encry
pted IBAN? True
```

b)

Implement the corresponding decryption algorithm and verify that it inverts the encryption.

In [8]:

```
# Decryption algorithm based on FD2 algorithm from the paper

def smallcipher_decrypt(key, tweak, hashalg, c, a, b, r):
  if r % 2 == 1:
    s = a
  else:
    s = b

  L = c % s
  R = c // s

  for i in range(r-1, -1, -1):
    if i % 2 == 1:
      s = a
    else :
      s = b

    tmp = L

    rstr = (str(a) + '|' + str(b) + '|' + str(tweak) + '|' + str(i) +
            '|' + str(L)).encode('ascii')
    f = int.from_bytes(hmac.digest(key, rstr, hashalg),
                       byteorder=sys.byteorder)
```

```
    L = (R - f) % s

    R = tmp

  return s * R + L
```

In [9]:

```
def fpe_decrypt(key,tweak, s):
  if not check_iban(s):
    raise ValueError("Invalid IBAN provided")

  while True:
    # Extract the numeric part of the IBAN
    numericIbanPart = s[2:]  # Remove 'CH' prefix

    # Decrypt the current numeric part of the IBAN
    decryptedNumeric = smallcipher_decrypt(key.encode(),
     tweak, hashlib.sha256 , int(numericIbanPart), 10**19, 10**19, 10)

    # Truncate to 19 digits to obtain a new IBAN numeric part candidate
    newNumericIbanPartCandidate = decryptedNumeric % 10**19

    # Construct a new Swiss IBAN with 'CH'
    newIbanCandidate = 'CH' + str(newNumericIbanPartCandidate)

    # Validate the new IBAN
    if check_iban(newIbanCandidate):
      return newIbanCandidate

    # Prepare next iteration with the new IBAN candidate
    s = newIbanCandidate

secretKey = 'eduardo'
tweak = 'tweak'

print(f'Encrypting IBAN {validIban} with secret key \
"{secretKey}" and tweak "{tweak}"...')

encryptedIban = fpe_encrypt(secretKey, tweak, validIban)

print(f'Encrypted IBAN: {encryptedIban}')

print(f'Valid IBAN ({encryptedIban}): {check_iban(encryptedIban)}')

decryptedIban = fpe_decrypt(secretKey, tweak, encryptedIban)

print(f'Decrypted IBAN: {decryptedIban}')

print(f'Valid IBAN ({decryptedIban}): {check_iban(decryptedIban)}')

print(f'Decrypted IBAN is equal to original IBAN? \
{decryptedIban == validIban}')
```

```
Encrypting IBAN CH9300762011623852957 with secret key "eduardo" and tweak "tweak"...
Encrypted IBAN: CH6833786179316117748
Valid IBAN (CH6833786179316117748): True
Decrypted IBAN: CH6944236514956139830
Valid IBAN (CH6944236514956139830): True
Decrypted IBAN is equal to original IBAN? False
```

The decryption algorithm did not work as expected. Even tough I implemented the algorithm as described in the paper, the decryption algorithm did not return the original IBAN. I tried to debug the

code, but I was not able to find the error.

Use again the cycle-walking method, as developed for Problem 10.1.