Trabalho prático 1 - Estruturas Criptográficas

Autores: Ariana Lousada (PG47034), Cláudio Moreira (PG47844)

Grupo 12

Problema 3

Para este problema desenvolveu-se a classe Ed25519 que implementa a curva Ed25519. De modo a ser possível preencher os parâmetros da curva de forma correta, consultou-se a documentação do FIPS186-5. Estes valores são utilizados para a inicialização das variáveis correspondentes aos parâmetros.

O segundo método to_hash faz hash da mensagem pretendida, com a função hash adequada. No caso da curva Ed25519, essa função coresponde ao SHA512.

O terceiro e quarto método (to_bytes e as_key) são apenas métododos auxiliares de conversão e cálculo de chaves através de um hash, respetivamente.

Os métodos public_key e private_key geram a chave pública e a chave privada, respetivamente. Para o cálculo da chave pública, em primeiro lugar é calculado o hash da chave privada utilizando o SHA512, armazenando apenas os primeiros 32 bytes. De seguida, é calculado um escalar utilizando o método as_key para posteriormente gerar a chave pública com utilização do ponto B da curva, copiando o bit menos significativo da coordenada x do ponto B (após o enconding da coordenada y) e colocando-o no bit mais significativo o octeto final.

O método sign é responsável pelas assinaturas. Em primeiro lugar, é calculado o hash da chave privada (utilizando o SHA512), construindo posteriormente um escalar utilizando a primeira metade do digest final. De seguida, é calculado o SHA512(dom2 || prefix || PH(M)), no qual M é a mensagem pretendida. Uma vez que se trata da curva Ed25519, dom2 corresponde à string vazia e PH corresponde à função de identidade. Portanto, é apenas necessário calcular o que está apresentado na variável m_h. De seguida, calcula-se o ponto R de modo a conseguir obter a assinatura.

Por fim, o método verify é responsável pela verificação das assinaturas. Em primeiro lugar, a assinatura é separada em duas partes de 32 octetos. A primeira metade é decoded como um ponto R e a segunda como um ponto S, fazendo decode também da chave pública. De seguida é calculado o SHA512(dom2 || R || A || PH(M)), no qual A se trata da chave pública. Como já abordado anteriormente, como a curva a ser implementada corresponde à Ed25519. dom2 corresponde à string vazia e PH à função identidade. Recorrendo aos métodos inner e outer, é calculada a equação [8][S]B = [8]R + [8][k]A', onde B é o parâmetro da curva e A' é a chave descodificada.

In [3]:

```
import collections
import hashlib
import os
import random

Point = collections.namedtuple('Point', ['x', 'y'])

key_mask = int.from_bytes(b'\x3F' + b'\xFF' * 30 + b'\xF8', 'big', signed=False)

class Ed25519() :

        length = 256 #b

        def __init__(self) :
                self.q = 2**255 - 19 #p
                self.l = 2**252 + 27742317777372353535851937790883648493
                self.d = -121665 * self.inverse(121666)

                self.i = pow(2, (self.q - 1)//4, self.q)

                self.B = self.point(4 * self.inverse(5))

        def to_hash(self, m) :
                return hashlib.sha512(m).digest()

        def from_bytes(self, h) :
                """ selecionar 32 bytes e retornar inteiro de 256 bit """
                return int.from_bytes(h[0:self.length//8], 'little', signed=False)

        def to_bytes(self, k) :
                return k.to_bytes(self.length//8, 'little', signed=False)

        def as_key(self, h) :
                return 2**(self.length-2) + (self.from_bytes(h) & key_mask)

        def private_key(self) :
                """ gerar uma chave privada pseudo aleatoria """
                m = os.urandom(1024)
                h = self.to_hash(m)
```

```
                k = self.as_key(h)
                return self.to_bytes(k)

        def public_key(self, sk) :
                """ calculo da chave publica atraves da chave privada"""
                h = self.to_hash(sk)
                a = self.as_key(h)
                c = self.outer(self.B, a)
                return self.point_to_bytes(c)

        def inverse(self, x) :
                return pow(x, self.q - 2, self.q)

        def sign(self, message, private_key, public_key) :
                s_h = self.to_hash(private_key)
                s_d = self.as_key(s_h)

                m_h = self.to_hash(s_h[self.length//8:self.length//4] + message)
                m_d = self.from_bytes(m_h)

                R = self.outer(self.B, m_d)

                r_h = self.to_hash(self.point_to_bytes(R) + public_key + message)
                r_d = m_d + self.from_bytes(r_h) * s_d

                return self.point_to_bytes(R) + self.to_bytes(r_d % self.l)

        def verify(self, message, signature, public_key) :
                r_b = signature[0:self.length//8]
                r_h = self.to_hash(r_b + public_key + message)
                r_d = self.from_bytes(r_h)

                s_d = self.from_bytes(signature[self.length//8:self.length//4])
                b_j = self.outer(self.B, s_d)

                P = self.bytes_to_point(public_key)
                p_j = self.outer(P, r_d)

                R = self.bytes_to_point(r_b)

                return b_j == self.inner(R, p_j)

        def recover(self, y) :
                """ com um valor y, recupera a pre imagem x """
                p = (y*y - 1) * self.inverse(self.d*y*y + 1)
                x = pow(p, (self.q + 3)//8, self.q)
                if (x*x - p) % self.q != 0:
                        x = (x * self.i) % self.q
                if x % 2 != 0 :
                        x = self.q - x
                return x

        def point(self, y) :
                """ atraves de um valor y, recupera x e retorna o ponto P(x, y) """
                return Point(self.recover(y) % self.q, y % self.q)

        def is_on_curve(self, P) :
                return (P.y*P.y - P.x*P.x - 1 - self.d*P.x*P.x*P.y*P.y) % self.q == 0

        def inner(self, P, Q) :
                """ produto interno na curva de dois pontos """
                x = (P.x*Q.y + Q.x*P.y) * self.inverse(1 + self.d*P.x*Q.x*P.y*Q.y)
                y = (P.y*Q.y + P.x*Q.x) * self.inverse(1 - self.d*P.x*Q.x*P.y*Q.y)
                return Point(x % self.q, y % self.q)

        def outer(self, P, n) :
                """ outer product on the curve, between a point and a scalar """
                if n == 0:
                        return Point(0, 1)
                Q = self.outer(P, n//2)
                Q = self.inner(Q, Q)
                if n & 1:
                        Q = self.inner(Q, P)
                return Q

        def point_to_bytes(self, P) :
                return (P.y + ((P.x & 1) << 255)).to_bytes(self.length//8, 'little')

        def bytes_to_point(self, b) :
                i = self.from_bytes(b)
                y = i % 2**(self.length - 1)
                x = self.recover(y)
                if (x & 1) != ((i >> (self.length-1)) & 1) :
                        x = self.q - x
                return Point(x, y)

if __name__ == '__main__' :
        def hexit(s) :
                return ''.join("{0:02X}".format(i) for i in reversed(s))
```

```
        ecc = Ed25519()

        alice_sk = b'alicealicealicealicealicealiceal' # ecc.private_key()
        alice_pk = ecc.public_key(alice_sk)

        assert(hexit(alice_pk) == 'AFA095BF733298216D0E88A0F2A4FEB15E5FEB73E7FA7522B67594FD2EF770D6')

        message = 'foo bar'.encode('utf8')
        signature = ecc.sign(message, alice_sk, alice_pk)

        import cProfile
        cProfile.run("ecc.sign(message, alice_sk, alice_pk)")

        assert(ecc.verify(message, signature, alice_pk))
        cProfile.run("ecc.verify(message, signature, alice_pk)")

        print("code test: OK")
```

```
        3050 function calls (2794 primitive calls) in 0.153 seconds

   Ordered by: standard name

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
    257/1    0.000    0.000    0.153    0.153 <ipython-input-3-48a9265e377e>:105(outer)
        2    0.000    0.000    0.000    0.000 <ipython-input-3-48a9265e377e>:115(point_to_bytes)
        3    0.000    0.000    0.000    0.000 <ipython-input-3-48a9265e377e>:23(to_hash)
        3    0.000    0.000    0.000    0.000 <ipython-input-3-48a9265e377e>:26(from_bytes)
        1    0.000    0.000    0.000    0.000 <ipython-input-3-48a9265e377e>:30(to_bytes)
        1    0.000    0.000    0.000    0.000 <ipython-input-3-48a9265e377e>:33(as_key)
      790    0.000    0.000    0.148    0.000 <ipython-input-3-48a9265e377e>:50(inverse)
        1    0.000    0.000    0.153    0.153 <ipython-input-3-48a9265e377e>:53(sign)
      395    0.004    0.000    0.152    0.000 <ipython-input-3-48a9265e377e>:99(inner)
        1    0.000    0.000    0.153    0.153 <string>:1(<module>)
      396    0.000    0.000    0.000    0.000 <string>:1(__new__)
      396    0.000    0.000    0.000    0.000 {built-in method __new__ of type object at 0x3f863ac40}
        3    0.000    0.000    0.000    0.000 {built-in method _hashlib.openssl_sha512}
        1    0.000    0.000    0.153    0.153 {built-in method builtins.exec}
      790    0.148    0.000    0.148    0.000 {built-in method builtins.pow}
        3    0.000    0.000    0.000    0.000 {built-in method from_bytes}
        3    0.000    0.000    0.000    0.000 {method 'digest' of '_hashlib.HASH' objects}
        1    0.000    0.000    0.000    0.000 {method 'disable' of '_lsprof.Profiler' objects}
        3    0.000    0.000    0.000    0.000 {method 'to_bytes' of 'int' objects}


        5903 function calls (5397 primitive calls) in 0.296 seconds

   Ordered by: standard name

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
    508/2    0.001    0.000    0.295    0.147 <ipython-input-3-48a9265e377e>:105(outer)
        2    0.000    0.000    0.001    0.000 <ipython-input-3-48a9265e377e>:118(bytes_to_point)
        1    0.000    0.000    0.000    0.000 <ipython-input-3-48a9265e377e>:23(to_hash)
        4    0.000    0.000    0.000    0.000 <ipython-input-3-48a9265e377e>:26(from_bytes)
     1534    0.009    0.000    0.287    0.000 <ipython-input-3-48a9265e377e>:50(inverse)
        1    0.000    0.000    0.296    0.296 <ipython-input-3-48a9265e377e>:67(verify)
        2    0.000    0.000    0.001    0.000 <ipython-input-3-48a9265e377e>:82(recover)
      766    0.007    0.000    0.294    0.000 <ipython-input-3-48a9265e377e>:99(inner)
        1    0.000    0.000    0.296    0.296 <string>:1(<module>)
      770    0.000    0.000    0.001    0.000 <string>:1(__new__)
      770    0.000    0.000    0.000    0.000 {built-in method __new__ of type object at 0x3f863ac40}
        1    0.000    0.000    0.000    0.000 {built-in method _hashlib.openssl_sha512}
        1    0.000    0.000    0.296    0.296 {built-in method builtins.exec}
     1536    0.279    0.000    0.279    0.000 {built-in method builtins.pow}
        4    0.000    0.000    0.000    0.000 {built-in method from_bytes}
        1    0.000    0.000    0.000    0.000 {method 'digest' of '_hashlib.HASH' objects}
        1    0.000    0.000    0.000    0.000 {method 'disable' of '_lsprof.Profiler' objects}


code test: OK
```