TP2-2

March 28, 2023

1 Trabalho Prático 2

Trabalho realizado pelo grupo 11:

- Beatriz Fernandes Oliveira, PG50942
- Bruno Filipe Machado Jardim, PG49997

1.1 Problema 2

2. Construir uma classe Python que implemente o EdCDSA a partir do "standard" FIPS186-5.
    - A implementação deve conter funções para assinar digitalmente e verificar a assinatura.
    - A implementação da classe deve usar uma das "Twisted Edwards Curves" definidas no standard e escolhida na iniciação da classe: a curva "edwards25519" ou "edwards448".
    - Por aplicação da transformação de Fiat-Shamir construa um protocolo de autenticação de desafio-resposta.

[141]:
```python
from pure25519.basic import bytes_to_scalar, Base, bytes_to_clamped_scalar,␣
↪bytes_to_element , scalar_to_bytes, random_scalar
import hashlib, binascii
class EdDSA:
    def __init__(self):
        self.order = 2^252 + 27742317777372353535851937790883648493

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

    def Hint(self,m):
        h = self.HASH(m)
        return int(binascii.hexlify(h[::-1]), 16)

    def genKeys(self):
        def publickey(sk):
            assert len(sk) == 32
            a = bytes_to_clamped_scalar(sk)
            A = Base.scalarmult(a)
            return A.to_bytes()

        sk = os.urandom(32)
```

```python
        hashed_sk = hashlib.sha512(sk).digest()
        pk = publickey(hashed_sk[:32])
        return sk, pk

    def sign(self,m,sk,pk):
        assert len(sk) == 32 and len(pk) == 32

        h = self.HASH(sk)
        hdigest1, hdigest2 = h[:32], h[32:]
        a = bytes_to_clamped_scalar(hdigest1)
        r = self.Hint(hdigest2 + m)
        R = Base.scalarmult(r)
        R_bytes = R.to_bytes()
        S = r + self.Hint(R_bytes + pk + m) * a
        return R_bytes + scalar_to_bytes(S)

    def verify(self,s, m, pk):
        assert len(s) == 64 and len(pk) == 32

        R = bytes_to_element(s[:32])
        A = bytes_to_element(pk)
        S = bytes_to_scalar(s[32:])
        h = self.Hint(s[:32] + pk + m)
        a = Base.scalarmult(S)
        b = R.add(A.scalarmult(h))
        return a==b

    # DOES NOT WORK
    def challenge(self, m, sk, pk):
        c = random_scalar(entropy_f=os.urandom)
        C = Base.scalarmult(c)
        C_bytes = C.to_bytes()
        e = self.Hint(C_bytes + m)
        sk_int = int(binascii.hexlify(sk[::-1]), 16)
        z = (c + e * sk_int) % self.order
        return z,e

    def prove(self,message, challenge, pk):
        z,e = challenge
        Z = Base.scalarmult(z)
        Q = bytes_to_element(pk)
        Se = Q.scalarmult(e)
        soma = Z.add(Se)

        result = self.Hint(soma.to_bytes()+message)

        print(e, result)
```

[144]:

```python
ed = EdDSA()
sk,pk = ed.genKeys()
print("[Alice]: My private key is: ", sk)
print("[Alice]: My public key is: ", pk)
sig = ed.sign(b'TP2-2 - EdDSA',sk,pk)

if ed.verify(sig,b'TP2 - EdDSA',pk): print("Signature is valid")
else: print("Signature is not valid")
if ed.verify(sig,b'TP2-2 - EdDSA',pk): print("Signature is valid")
else: print("Signature is not valid")
```

```
[Alice]: My private key is:  b'$\xe5\x8eB\xc5\x94*\x9d\xcd\xcaZV\x8a@\x15Y\x96>\
xaa\xf6}]\xd0\xdc\xd1\x16v\x14\x0c\x84\xce6'
[Alice]: My public key is:
b'\x00\x9c\x12\xb1\xab\xf03b\x1c\x94&8\xd7\xb0\xbd<\xe6e\xeel#\xa1\x00R\xf8!
\xee\xcae\xf1\x84'
Signature is not valid
Signature is valid
```

[143]:

```python
challenge = ed.challenge(b'TP2-2 - EdDSA',sk,pk)
challenge

proof = ed.prove(b'TP2-2 - EdDSA',challenge,pk)
```

```
9255394769851793343406735774905502016481117042327437064555048488754499184143573З
636753243369900898521966102324182521018263561750117805923366063708144861б7 42012
381481795856366266023377368223876832283063703009387244215332597447237722159894б6
867984627973332689067091974593071555103070405378750895294390715391438
```