

Exercício 2 - Uso de Curvas Elípticas

Sessão igual ao esquema do exercício anterior, mas agora com o uso de curvas elípticas.

Em vez do protocolo de acordo chaves **Diffie–Hellman** usou-se **Elliptic-curve Diffie–Hellman** e em vez do **Digital Signature Algorithm** usou-se **Elliptic Curve Digital Signature Algorithm** .

In [94]:

```
from cryptography.hazmat.backends import default_backend
from cryptography.hazmat.primitives.ciphers import Cipher, algorithms, modes
from cryptography.hazmat.primitives import hashes, hmac, cmac
from cryptography.hazmat.primitives.asymmetric import dh, dsa
from cryptography.hazmat.primitives import serialization, hashes
from getpass import getpass
from cryptography.exceptions import *
from cryptography.hazmat.primitives.asymmetric import ec

default_algorithm = hashes.SHA256
# seleciona-se um dos vários algoritmos implementados na package

def my_mac(key):
    return hmac.HMAC(key,default_algorithm(),default_backend())
```

In [95]:

```

from BiConn import BiConn
from Auxs import hashshs, mac, kdf
import getpass, os, io

default_curve = ec.SECP256R1 #curva SECP256R1
def ECDH(conn):
    # agreement
    pk = ec.generate_private_key(default_curve, default_backend()) #ao gerar a c
have privada,
    pub = pk.public_key().public_bytes(
#recebe como
argumento a curva definida
        encoding=serialization.Encoding.PEM,
        format=serialization.PublicFormat.SubjectPublicKeyInfo)

    #gerar as chaves privada e pública
    private_key_dsa = ec.generate_private_key(default_curve, default_backend())
    pub_dsa = private_key_dsa.public_key().public_bytes(
        encoding=serialization.Encoding.PEM,
        format=serialization.PublicFormat.SubjectPublicKeyInfo)

    #envia a chave pública
    conn.send(pub_dsa)

    #cálculo da assinatura
    signature = private_key_dsa.sign(pub, ec.ECDSA(hashshs.SHA256())) #ECDSA rece
be como argumento a hash256

    peer_pub_dsa =serialization.load_pem_public_key(
        conn.recv(),
        backend=default_backend())

    conn.send(pub)
    conn.send(signature)

    #ASSINAR

    try:
        peer_pub = conn.recv()
        sig = conn.recv()
        peer_pub_dsa.verify(sig, peer_pub, ec.ECDSA(hashshs.SHA256()))
        print("ok ecdh")
    except InvalidSignature:
        print("fail ecdh")

    # shared_key calculation
    peer_pub_key = serialization.load_pem_public_key(
        peer_pub,
        backend=default_backend())
    shared_key = pk.exchange(ec.ECDH(), peer_pub_key) #em vez de se trocar apena
s a chave, tambem se troca ECDH

    # confirmation
    my_tag = hashshs(bytes(shared_key))
    conn.send(my_tag)
    peer_tag = conn.recv()
    if my_tag == peer_tag:
        print('OK ECDH')

```

```

        return my_tag
    else:
        print('FAIL ECDH')

#eliminar dados
pk = None
pub = None
peer_pub = None
peer_pub_key = None
shared_key = None
my_tag = None
peer_tag = None

```

In [96]:

```

message_size = 2**10

def Emitter(conn):
    # Acordo de chaves DH e assinatura DSA
    key = ECDH(conn)

    # Mensagem
    inputs = io.BytesIO(bytes('1'*message_size, 'utf-8'))

    # iv para a cifra
    iv = os.urandom(16)

    # Cifra
    cipher = Cipher(algorithms.AES(key), modes.CFB(iv),
                    backend=default_backend()).encryptor()

    # HMAC
    mac = my_mac(key)

    conn.send(iv) # Envio do iv
    buffer = bytearray(32) # Buffer onde vão ser lidos os blocos

    # lê, cifra e envia sucessivos blocos do input
    try:
        while inputs.readinto(buffer):
            ciphertext = cipher.update(bytes(buffer))
            mac.update(ciphertext)
            conn.send((ciphertext, mac.copy().finalize()))

        conn.send((cipher.finalize(), mac.finalize())) # envia a finalização
    except Exception as err:
        print("Erro no emissor: {0}".format(err))

    inputs.close() # fecha a 'input stream'
    conn.close() # fecha a conexão

```

In [97]:

```

def Receiver(conn):
    # Acordo de chaves DH e assinatura DSA
    key = ECDH(conn)

    # Inicializa um output stream para receber o texto decifrado
    outputs = io.BytesIO()

    # Recebe o iv
    iv = conn.recv()

    # Cifra
    cipher = Cipher(algorithms.AES(key), modes.CFB(iv),
                    backend=default_backend()).decryptor()

    # HMAC
    mac = my_mac(key)

    # operar a cifra: ler da conexão um bloco, autenticá-lo, decifrá-lo e escre
    ver o resultado no 'stream' de output
    try:
        while True:
            try:
                buffer, tag = conn.recv()
                ciphertext = bytes(buffer)
                mac.update(ciphertext)
                if tag != mac.copy().finalize():
                    raise InvalidSignature("erro no bloco intermédio")
                outputs.write(cipher.update(ciphertext))
            if not buffer:
                if tag != mac.finalize():
                    raise InvalidSignature("erro na finalização")

                outputs.write(cipher.finalize())
                break

            except InvalidSignature as err:
                raise Exception("autenticação do ciphertext ou metadados: {}".fo
rmat(err))
                print(outputs.getvalue())      # verificar o resultado

    except Exception as err:
        print("Erro no receptor: {0}".format(err))

    outputs.close()      # fechar 'stream' de output
    conn.close()         # fechar a conexão

```

```
BiConn(Emitter, Receiver, timeout=30).auto()
```

[illegible]