ECDHE_ECDSA_wikipedia_ESTUDIANTES_-PYTHON

December 4, 2020

[1]:
```
import hashlib
import sympy
import ecpy
from ecpy.curves import Curve,Point
```

1 Verificación firma

1.1 Curva $E : y^2 \equiv x^3 + a \cdot x + b \mod p$

n Orden de la curva

$G = E([G_X, G_Y])$ punto base de la curva

1.1.1 Curve P-256 https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-4.pdf Apéndice D

$p = 115792089210356248762697446949407573530086143415290314195533631308867097853951 = 2^{256} - 2^{224} + 2^{192} + 2^{96} - 1$

$n = 115792089210356248762697446949407573529996955224135760342422259061068512044369$

$a = -3$

$b =$ 0x5ac635d8aa3a93e7b3ebbd55769886bc651d06b0cc53b0f63bce3c3e27d2604b

$G_X =$ 0x6b17d1f2e12c4247f8bce6e563a440f277037d812deb33a0f4a13945d898c296

$G_Y =$ 0x4fe342e2fe1a7f9b8ee7eb4a7c0f9e162bce33576b315ececbb6406837bf51f5

1.1.2 Clave privada $r \in [0, n-1]$

1.1.3 Clave pública $Q = r \cdot G$

1.1.4 Firma

$$k \in \mathbb{Z} \quad \text{aleatorio}$$
$$k \cdot G = (x_1, y_1)$$
$$f_1 \equiv x_1 \mod n,$$
$$w = k^{-1} \mod n$$
$$f_2 \equiv w \cdot (h + r \cdot f_1) \mod n,$$

si $f_1 = 0$ o $f_2 = 0$ se toma otro k

La firma es (f_1, f_2)

1.1.5 Verificación firma (f_1, f_2)

$$
\begin{gathered}
aux = f_2^{-1} \mod n \\
w_1 \equiv h \cdot aux \mod n \\
w_2 \equiv f_1 \cdot aux \mod n \\
Q = E([publicKey_X, publicKey_Y]) \\
P = w_1 \cdot G + w_2 \cdot Q
\end{gathered}
$$

Firma correcta si $P_X \equiv f_1 \mod n$

[ ]:

```

```

2 Conexión TLS 1.3

https://tools.ietf.org/html/rfc8446

https://tls13.ulfheim.net/

https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-4.pdf Apéndice D

[2]:

```
p =␣
 ↪115792089210356248762697446949407573530086143415290314195533631308867097853951␣
 ↪ # p=2**256-2**224+2**192+2**96-1
orden =␣
 ↪115792089210356248762697446949407573529996955224135760342422259061068512044369
a=-3
b = 0x5ac635d8aa3a93e7b3ebbd55769886bc651d06b0cc53b0f63bce3c3e27d2604b
Gx=0x6b17d1f2e12c4247f8bce6e563a440f277037d812deb33a0f4a13945d898c296
Gy=0x4fe342e2fe1a7f9b8ee7eb4a7c0f9e162bce33576b315ececbb6406837bf51f5

print('Es el orden primo?',sympy.isprime(orden))


# clave publica Wikipedia
publicKey_WikipediaX=0x057947b19ac448be2b2e0ace4019ca4b9af401aa7c41b4d6147c4d124e5a7846
publicKey_WikipediaY=0xef2cb6e579c3cce2e6d81c813c342cb312fdcac0035078af6f31c7462bee981d

# Defino la curva
E  = Curve.get_curve('secp256r1')

# Defino dos puntos
G=Point(Gx,Gy,E)
publicKey_Wikipedia=Point(publicKey_WikipediaX, publicKey_WikipediaY, E)


print('generador es de la curva? ', E.is_on_curve(G))
```

```
print('La clave pública es de la curva? ', E.is_on_curve(publicKey_Wikipedia))



print('Punto del infinito?', orden*G)
print('Punto del infinito?', orden*publicKey_Wikipedia)


print('----------------------------')
```

```
Es el orden primo? True
generador es de la curva?  True
La clave pública es de la curva?  True
Punto del infinito? inf
Punto del infinito? inf
----------------------------
```

[ ]:

```

```

The digital signature is then computed over the concatenation of (https://tools.ietf.org/html/rfc8446#page-69):

- A string that consists of octet 32 (0x20) repeated 64 times
- The context string
- A single 0 byte which serves as the separator
- The content to be signed

This structure is intended to prevent an attack on previous versions of TLS in which the ServerKeyExchange format meant that attackers could obtain a signature of a message with a chosen 32-byte prefix (ClientHello.random). The initial 64-byte pad clears that prefix along with the server-controlled ServerHello.random.

The context string for a server signature is "TLS 1.3, server CertificateVerify". The context string for a client signature is "TLS 1.3, client CertificateVerify". It is used to provide separation between signatures made in different contexts, helping against potential cross-protocol attacks.

For example, if the transcript hash was 32 bytes of 01 (this length would make sense for SHA-256), the content covered by the digital signature for a server CertificateVerify would be:

```
2020202020202020202020202020202020202020202020202020202020202020
2020202020202020202020202020202020202020202020202020202020202020

544c5320312e332c207365727665722043657274696669636174655665726966
79

00

0101010101010101010101010101010101010101010101010101010101010101
```

The content to be signed se refiere al *The Transcript Hash* (ver sección 4.4.1 del RFC):

For concreteness, the transcript hash is always taken from the following sequence of handshake messages, starting at the first ClientHello and including only those messages that were sent: ClientHello, HelloRetryRequest, ClientHello, ServerHello, EncryptedExtensions, server CertificateRequest, server Certificate, server CertificateVerify, server Finished, EndOfEarlyData, client Certificate, client CertificateVerify, client Finished.

En nuestro caso: *ClientHello and including only those messages that were sent: ClientHello, HelloRetryRequest, ClientHello, ServerHello, EncryptedExtensions, server CertificateRequest, server Certificate*

Nota: Los paquetes son los correspondientes a "Handsake protocol" no "TLsv1.3 Record Layer: Hansake Protocol" hay una diferencia de 5 bytes en cada uno.

La función hash usada para el *The Transcript Hash* es la elegida en el cipher_suite

[ ]:

```

```

[3]:

```
octet_0x20_repeated_64_times='20'*64

#The_context_string="TLS 1.3, server CertificateVerify"
#bytes(bytearray('TLS 1.3, server CertificateVerify',encoding='ascii')).hex()
The_context_string='544c5320312e332c20736572766572204365727469666963617465566572696679'

byte_separator='00'

# cipher_suite TLS_AES_256_SHA384 ===> SHA384
The_Transcript_Hash='0702c4ebdeb337df49706c328a437c9fdde2b3779ffaabbe6711335bfa8bc20dff2c2ca49

to_hash=octet_0x20_repeated_64_times+The_context_string+byte_separator+The_Transcript_Hash

# Algoritmo de firma para Certificate Verify esdas_secp256r1_sha256 ===> SHA256
h_aux=hashlib.sha256(bytes(bytearray.fromhex(to_hash)))
h=int(h_aux.hexdigest(),16)


#Signature=30.45.02.20.
↪7ff0099745312bf89bf88248d8778d01b9f140f4e82cba7fe584e10c90a29dd9.
#                02.21.
↪00eff29b882b325e65cc5708f95e15ee2a1b4761d04d5f0603735f22853fb6179c
# 30 indica que es una secuencia
# 45 indica la longitud de la secuencia en bytes (69 bytes)
# 02 indica un entero
# 20 indica la longitud del entero en bytes (32 bytes)
# los siguientes 32 bytes son f1 (el byte inicial es 00 por el tema del␣
↪complemento a 2
```

```python
f1=0x7ff0099745312bf89bf88248d8778d01b9f140f4e82cba7fe584e10c90a29dd9
# 02 indica un entero
# 21 indica la longitud del entero en bytes (33 bytes)
# los siguientes 33 bytes son f2
f2=0x00eff29b882b325e65cc5708f95e15ee2a1b4761d04d5f0603735f22853fb6179c

aux=pow(f2,-1,orden)
w1=(h*aux)%orden
w2=(f1*aux)%orden
Q = publicKey_Wikipedia
result = w1*G+w2*Q
print('firma correcta?', result.x%orden==f1)
```

```
firma correcta? True
```

[ ]:

```python

```