IA012 - Tarefa 4

Rodrigo Seiji Piubeli Hirao (186837)

16 de dezembro de 2021

01) O algoritmo de RSA é muito pesado computacionalmente, exigindo muito tempo para realizar a operação e muito espaço para armazenar suas chaves, então foi trocado pelo ECC que é mais rápido e exige chaves menores para mesma segurança. Sendo bom para aparelhos com menos recursos.

02) O ECC são o conjunto de grupos elípticos mod p, ou seja, eles descrevem a equação 1 que retorna um conjunto de curvas elípticas por conta do módulo. Assim tendo também que seguir a condição da equação 2. Além de usar apenas números inteiros.

$$y^2 mod p = x^3 + ax + b mod p \tag{1}$$

$$4a^3 + 27b^2 mod p \neq 0 \tag{2}$$

03) O grupo elíptico é o conjunto de curvas elípticas criadas pela equação 1 usada ter um módulo, o que faz com que ela tenha infinitas soluções.

04) O ECC depende da equação 3, que é muito fácil de ser caclulado sabendo k e P, mas difícil sabendo apenas R e P. O que torna a função uma função de mão única.

$$R = kP \tag{3}$$

05) Devemos calcular $E_{23}(1, 12)$, e testar se P(8,7) + Q(21,5) = R(6,21)

```
sage: p = 23
sage: a = 1
sage: b = 12
sage: E = EllipticCurve(GF(p), [a,b])
sage: E
Elliptic Curve defined by y^2 = x^3 + x + 12 over Finite Field of size 23
sage: P = E(8, 7); P
(8 : 7 : 1)
sage: 2*P
(8 : 16 : 1)
sage: Q = E(21,5); Q
(21 : 5 : 1)
sage: P+Q
(6 : 21 : 1)
```

O que deu correto.
Além disso devemos ver se P(0,9) + Q(12,21) = R(12, 2)

```
sage: P = E(0, 9); P
(0 : 9 : 1)
sage: Q = E(12, 21); Q
(12 : 21 : 1)
sage: P+Q
(12 : 2 : 1)
```

O que também se provou correto.

06) O resultado obtido foi o esperado

```
sage: Px = 8
sage: Py = 7
sage: Qx = 21
sage: Qy = 5
sage: L = (Qy - Py)/(Qx - Px) ; L
-2/13
sage: Rx = (L^2 - Px - Qx) % p; Rx
6
sage: Ry = (L * (Px - Rx) - Py) % p; Ry
21
```

07) Pode se ver pelo resultado que $Kpr_b \times Kpu_a = Kpr_a \times Kpu_b = K_s$

```
sage: p = 0xA9FB57DBA1EEA9BC3E660A909D838D726E3BF623D52620282013481D1F6E5377
sage: a = 0x7D5A0975FC2C3057EEF67530417AFFE7FB8055C126DC5C6CE94A4B44F330B5D9
sage: b = 0x26DC5C6CE94A4B44F330B5D9BBD77CBF958416295CF7E1CE6BCCDC18FF8C07B6
sage: x = 0x8BD2AEB9CB7E57CB2C4B482FFC81B7AFB9DE27E1E3BD23C23A4453BD9ACE3262
sage: y = 0x547EF835C3DAC4FD97F8461A14611DC9C27745132DED8E545C1D54C72F046997
sage: q = 0xA9FB57DBA1EEA9BC3E660A909D838D718C397AA3B561A6F7901E0E82974856A7
sage: h = 1

sage: E = EllipticCurve(GF(p), [a,b])
sage: P = E(x, y)

sage: n_a = 0x51897b64e85c3f714bba707e867914295a1377a7463a9dae8ea6a8b914246319
sage: n_b = 0x1897b64e85c3f714bba707e867914295a1377a7463a9dae8ea6a8b9142463195

sage: Kpr_a = n_a; Kpr_a
3688025041838493828599179414144182848095428653212946104253407438339852549404 1
sage: Kpr_b = n_b; Kpr_b

sage: Kpu_a = n_a * P; Kpu_a
(70690535889208483973246486221299188049850189864816032111026075288961660510 64 :
11051574877168605934234633436003731172108196531199368213425220426109695874 67 : 1)
sage: Kpu_b = n_b * P; Kpu_b
(56423593456817369703118768906784508970104950167857058858361437621806115659082 :
33441588546436459456293388911431474176771812101846068510477079692866073503861 : 1)

sage: K_s = Kpr_a * Kpu_b; K_s
(64736710232610134458185960766823644578957694822397984483337542929826313693080 :
45930701194457663081955491551190257505029656361127717930652190342507693453610 : 1)
sage: K_s = Kpr_b * Kpu_a; K_s
(64736710232610134458185960766823644578957694822397984483337542929826313693080 :
45930701194457663081955491551190257505029656361127717930652190342507693453610 : 1)
```

08) Para gerar o certificado s com a chave privada k foi usado

```
sage: Kpr
43705063589837965376603771803517676698531451532489112150657350662076145163166
sage: Kpu = Kpr * P ; Kpu
(75754610981435305088380623431777046463291274879745580761955141492212690712535 : 172868630718638887098
```

```
sage: def sign(m, k, P, q):
....:     e = int(hashlib.sha256(m).hexdigest(), 16)
....:     while e > q:
....:         e /= 2
....:
```

```
....:     R = k*P
....:     r = int(R[0]) % q
....:     s = (e + r * Kpr)/k % q
....:     return (r, s)
....:
....:
sage: sign(b'I am Seiji', k, P, q)
(39473398605507859322755385576405408742073325072825768608367214199448981391834,
 28720078325953420757935057137616745101314925350367838017563912950073780527241)
```

Para confirmar se a chave privada k é válida foi usado

```
sage: R
(39473398605507859322755385576405408742073325072825768608367214199448981391834 :
125197600794437478159868132151620741675571579585257177038575791791581995255 31 : 1)
sage: E(R[0], R[1])
(39473398605507859322755385576405408742073325072825768608367214199448981391834 :
125197600794437478159868132151620741675571579585257177038575791791581995255 31 : 1)
sage: q*R
(0 : 1 : 0)
```

E por fim para validar o certificado s

```
sage: q
768849563970453442208097466290016490927375317844145295387555190630635363590 79
sage: r
39473398605507859322755385576405408742073325072825768608367214199448981391834
sage: s
28720078325953420757935057137616745101314925350367838017563912950073780527241

sage: def verify(m, s, r, Kpu, P, q):
....:     e = int(hashlib.sha256(m).hexdigest(), 16)
....:     while e > q:
....:         e /= 2
....:     u_1 = (e / s) % q ; u_1
....:     u_2 = (r / s) % q
....:     R_2 = u_1 * P + u_2 * Kpu
....:     x = int(R_2[0]) % q
....:     return x == r
....:
sage: verify(b'I am Seiji', S1[1], S1[0], Kpu, P, q)
True
```

Assim pudemos criar e verificar o certificado pois $x = r$.

09) Fazendo os cálculos temos os 2 pares (r, s) que percebemos que temos r iguais, mas s diferentes. Então podemos calcular k com a equação 4 e a partir de k calcular Kpr com a equação 5

$$k = \frac{e' - e}{s' - s} \tag{4}$$

$$Kpr = \frac{sk - e}{r} \tag{5}$$

```
sage: S1 = sign(b'I am Seiji', k, P, q)
sage: S2 = sign(b'I am still Seiji!', k, P, q)

sage: S1
(39473398605507859322755385576405408742073325072825768608367214199448981391834,
```

```
 28720078325953420757935057137616745101314925350367838017563912950073780527241)
sage: S2
(39473398605507859322755385576405408742073325072825768608367214199448981391834,
 62456821151373123504418928872517364437536391765376823316657408691266942826211)

sage: # cálculos dos hashs e1 e e2

sage: (e1 - e2)/(s1 - s2) % q
362458308885430376767621468771348368242077706111094735050970747049325878708 11
sage: k
362458308885430376767621468771348368242077706111094735050970747049325878708 11

sage: (s1*k - e1)/r % q
437050635898379653766037718035176766985314515324891121506573506620761451631 66
sage: Kpr
437050635898379653766037718035176766985314515324891121506573506620761451631 66
```

10) A Curve25519 é uma $E_{2^{255}-19}(486662, 1)$ que usa chaves de 256bits oferecendo segurança de 128bits feita para ser usada com Diffie-Helman. Ela é muito popular por ser uma curva considerada segura [1], rápida e, principalmente, domínio público.

[1] https://safecurves.cr.yp.to/index.html