Criptografia assimétrica de curva elíptica

É um algoritmo similar ao RSA, mas que é mais simples e consegue igual segurança com variáveis menores (em bits). Usa a função do logaritmo discreto ao invés da exponenciação (como no RSA) seguido por uma operação $mod\ n$ o que garante a irreversibilidade dos cálculos. Ele é a base da arquitetura *bitcoin* e é usado extensivamente em praticamente todas as transações lá.

Lembrando, na criptografia assimétrica, as chaves vem em pares, e tudo que é criptografado com uma chave precisa da outra chave para ser descriptografado. As chaves vão ser chamadas de pública e privada. Primeiro deve ser gerada a chave privada (um número completamente aleatório) e depois a chave pública pode ser facilmente calculada usando a chave privada. O caminho inverso (da pública → privada) revela-se computacionalmente intratável.

Aqui, a principal funcionalidade perseguida não é exatamente a criptografia e sim a assinatura digital, que vem a ser a contrapartida do mesmo método. Ela garante que o possuidor da chave é o único legítimo proprietário dos direitos (neste caso, direito aos bitcoins).

Para os mais paranóicos, a geração da chave privada deve se dar após lançar uma moeda (honesta) 256 vezes. Cara vale 0 e coroa vale 1. Com isso gera-se um número binário de 256 bits, que devidamente convertido a decimal transforma-se em um número imenso. Ele é a chave privada, k, fazendo $k = 2^{moedas}$.

Para os menos paranóicos, pode-se usar um gerador de (pseudo-) aleatórios ou seja, um programa de computador capaz de gerar randômicos. Muito cuidado com este programa. Muitos não usam uma fonte confiável de entropia do sistema operacional. Uma orientação importante aqui é usar métodos testados e reconhecidos como geradores de aleatórios criptograficamente seguros (em inglês: CSPRNG). Mesmo que você seja o cara em termos de programação é muito perigoso usar um método pessoal, por melhor que ele seja.

Em python, isso pode ser conseguido com

```
>>> import os
>>> pkey=os.urandom(32)
>>> pkey_int = int.from_bytes(pkey,
    byteorder='little')
>>> print(pkey_int)
65688278866881945554701861341765131768...
109023524949302063401339857669585074 61
```

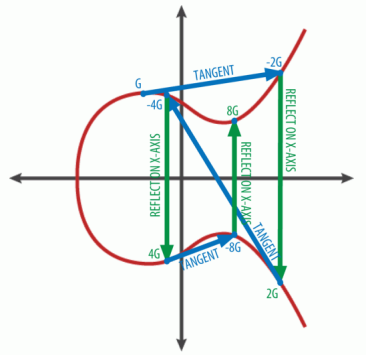
Detalhes Técnicos

A curva elíptica definida no padrão `secp256k1` possui os seguintes parâmetros: $p = 2^{256} - 2^{32} - 2^9 - 2^8 - 2^7 - 2^6 - 2^4 - 1$ que é um número primo. Este número em hexadecimal é p = FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFE FFFFFC2F. O p vai ser usado para fazer $mod\ p$. A curva E é $E = y^2 + x^3 + ax + b$, com as seguintes constantes a=00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 e b=00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000007.

Trocando em miúdos, a função é $(x^3 + 7 - y^2)\ mod\ p$ ou $y^2 = x^3 + 7\ mod\ p$ com $p = 2^{256} - 2^{32} - 977$. Define-se um ponto G chamado ponto gerador cujo valor é G=79BE667E F9DCBBAC 55A06295 CE870B07 029BFCDB 2DCE28D9 59F2815B 16F81798 483ADA77 26A3C465 5DA4FBFC 0E1108A8 FD17B448 A6855419 9C47D08F FB10D4B8

O limite superior de escolhas de chaves secretas é dado por n cujo valor é n = FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFE BAAEDCE6 AF48A03B BFD25E8C D0364141.

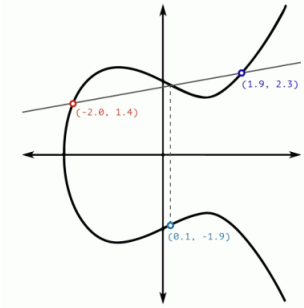
A chave privada k é um número de 256 bits (menor que n). Em decimal este número é aproximadamente igual a $1.157920892 \times 10^{77}$. Lembre-se que segundo os teóricos, o número de elétrons do universo é menor que 10^{80}. A chave pública K pode ser calculada como $K = k * G$. Acompanhe na figura a multiplicação na curva elíptica (na figura, mais simples, no campo dos reais, mas a matemática é a mesma).



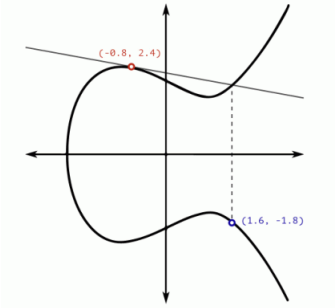
Veja-se um ponto P calculado usando Python

```
Python 3.4.0 (padrão, Mar 30 2014, 19:23:13)
Digite "help", "copyright", "credits" ou
    "license" para maiores informações.
>>> p = 115792089237316195423570985008687907853
        269984665640564039457584007908834671663
>>> x = 550662630222773436695787188951685343262 5
        060345377759417550018736038911672924 0
>>> y = 326705100207588169780830851305070431844 7
        127338065924327593890433575733748242 4
>>> (x ** 3 + 7 - y**2) % p
0
```

Para calcular a curva em nosso esquema de criptografia necessita-se de 2 operações: adição e multiplicação de pontos. Para adicionar o ponto A ao ponto B, ambos na curva, desenha-se o segmento de reta que os une. Este segmento interceptará a curva em um terceiro ponto e o resultado da soma é a reflexão deste ponto através do eixo x. Veja-se o exemplo de somar os pontos (1.9, 2.3) com (-2.0, 1.4), cuja soma é (0.1, -1.9).



Usando esta mesma regra, e querendo somar o ponto a ele mesmo $(A + A)$, basta fazer a tangente à curva no ponto A. Esta linha cruzará a curva em outro ponto e a resposta $(2A)$ é a reflexão deste ponto, como se pode ver em $2 \times (-0.8, 2.4) = (1.6, -1.8)$



Lembrando de fundamentos de matemática que a multiplicação corresponde à sucessivas somas, já se tem tudo o que é necessário.

Para obter a chave pública, far-se-á a multiplicação da chave privada por um ponto inicial fixo, conhecido por todos. Ele se chama ponto gerador, abreviado como G e definido em `secp256k1` como

```
Gx = 550662630222773436695787188951685343 26
     250603453777594175500187360389116729240
Gy = 326705100207588169780830851305070431 84
     471273380659243275938904335757337482424
```

Os números envolvidos precisam ser grandes desse jeito, para garantir segurança ao método. Os programas a usar em Python 3

```
p=2**256 - 2**32 - 977
def inverse(x,p):
  inv1 = 1
  inv2 = 0
  while p != 1 and p!=0:
    inv1, inv2 = inv2, inv1 - inv2 *(x//p)
    x,p = p,x%p
  return inv2
def double_point(point,p):
  (x,y)=point
  if y==0:
    return None
  slope=3*pow(x,2,p)*inverse(2*y,p)
  xsum=pow(slope,2,p)-2*x
  ysum=slope*(x-xsum)-y
  return(xsum%p,ysum%p)
def add_point(p1,p2,p):
  (x1,y1)=p1
  (x2,y2)=p2
  if x1==x2:
    return double_point(p1,p)
  slope=(y1-y2)*inverse(x1-x2,p)
  xsum=pow(slope,2,p)-(x1+x2)
  ysum=slope*(x1-xsum)-y1
  return(xsum%p,ysum%p)
def point_mul(point,a,b):
  scale=point
  acc=None
  while a:
    if a&1:
      if acc is None:
        acc=scale
      else:
        acc=add_point(acc,scale,p)
    scale=double_point(scale,p)
    a>>=1
  return acc


privkey = 0xe87e8404367368d0494480916d2580b
          e6efcdf67894aebfcdf7cc3056863fb9a
g_x =     0x79BE667EF9DCBBAC55A06295CE870B0
          7029BFCDB2DCE28D959F2815B16F81798
g_y =     0x483ADA7726A3C4655DA4FBFC0E1108A
          8FD17B448A68554199C47D08FFB10D4B8
g = (g_x, g_y)
pub_x, pub_y = point_mul(g, privkey, p)
print("x: %x, y: %x" % (pub_x, pub_y))
print('verificacao ')
print('p=',p)
print((pub_x**3+7-pub_y**2)%p)

# veja Bitcoin para programadores, pág 47,
    estes programas estao lá.
```

Exemplo Suponha a chave hexadecimal (começando com 0x)

```
10000000000000000000000000000000
00000000000000000000000000000000
```

Os 4 primeiros dígitos da componente x da chave pública são `8ea9`.

Identicamente para a chave hexadecimal (começando com 0x)

```
10100000000000000000000000000000
00000000000000000000000000000000
```

Os 4 primeiros dígitos da componente x da chave pública são `2e2d`.

☞ Para você fazer

Calcule os primeiros 4 dígitos hexadecimais para a componente x da chave pública para esta chave privada:

```
18900000000000000000000000000000
00000000000000000000000000000000
```