HW1: algoritmo de Gordon

Definição do Algoritmo de Gordon

In [39]:

```
def gordonAlgorithm(nbits):

    nbits2 = nbits/2
    nbitsLbound = nbits2-1

    global s
    global t

    t = random_prime(2^nbits2, 2^nbitsLbound)
    s = random_prime(2^nbits2, 2^nbitsLbound)

    global i
    global r
    i = 0
    while (not is_prime(2*i*t + 1)):
        i+=1
    r = 2*i*t +1

    global p0
    p0 = (2 * power_mod(s,r-2,r) *s) - 1

    global j
    j = 0
    while (not is_prime(p0 + 2*j*r*s) ):
        j+=1
    p = p0 + 2*j*r*s


    return p
```

Verificar se é Strong Prime

First Verification:

$$s^{r-1} \equiv 1 \pmod{r}$$

Second Verification:

$$p_0 \equiv 1 \pmod{r}$$

Third Verification:

$$p_0 \equiv -1 \pmod{s}$$

Fourth Verication:

a) $p - 1 == p_0 + 2 * j * r * s - 1$

b) $p - 1 \equiv 0 \pmod{r}$

c) $p_0 + 2 * j * r * s - 1 \equiv 0 \pmod{r}$

d) $p - 1$ has the prime factor r

Fifth Verification:

e) $p + 1 == p_0 + 2 * j * r * s + 1$

f) $p + 1 \equiv 0 \pmod{s}$

g) $p_0 + 2 * j * r * s + 1 \equiv 0 \pmod{s}$

h) $p + 1$ has the prime factor s

Sixth Verification:

k) $r - 1 == 2 * i * t$

l) $r - 1 \equiv 0 \pmod{t}$

m) $2 * i * t \equiv 0 \pmod{t}$

n) $r - 1$ has the prime factor t

In [40]:

```
def strongest_prime_verification(p,r,s,t,i,j):

    # first verification
    print( "First Verification: " + str((power_mod(s,r-1,r) == 1 )))

    # second verification
    print("Second Verification: " + str((mod(p0,r) == 1)))

    # third verification
    print("Third Verification: " + str((mod(p0,s) == -1)))

    # fourth verification
    print("Fourth Verification:")
    a = p - 1 == p0 + 2 * j * r * s - 1
    b = mod(p-1,r) == 0
    c = mod(p0 + 2 * j * r * s - 1,r) == 0
    d = factor(p - 1)

    print("     a) " + str(a))
    print("     b) " + str(b))
    print("     c) " + str(c))
    print("     d) " + str(d))

    # fifth verification
    e = p + 1 == p0 + 2 * j * r * s + 1
    f = mod(p + 1,s) == 0
    g = mod(p0 + 2 * j * r * s + 1,s) == 0
    h = factor(p + 1)

    print("Fifth Verification:")
    print("     e) " + str(e))
    print("     f) " + str(f))
    print("     g) " + str(g))
    print("     h) " + str(h))

    # sixth verification
    k = r - 1 == 2 * i * t
    l = mod(r - 1,t) == 0
    m = mod(2 * i * t,t) == 0
    n = factor(r - 1)

    print("Sixth Verification:")
    print("     k) " + str(k))
    print("     l) " + str(l))
    print("     m) " + str(m))
    print("     n) " + str(n))
```

Exemplo 1

In [41]:

```
nbits = 10
p = gordonAlgorithm(nbits)
nbits2 = nbits/2
print("Número de bits: " + str(nbits))
print("nbits: " + str(nbits2))
print("nbitsLbound: " + str(nbits2-1))
print("\n")
print("t: " + str(t))
print("s: " + str(s))
print("\n")
print("i: " + str(i))
print("r: " + str(r))
print("\n")
print("p0: " + str(p0))
print("\n")
print("j: " + str(j))
print("p: " + str(p))
print("Número de bits do strong prime: " + str(p.nbits()))
```

```
Número de bits: 10
nbits: 5
nbitsLbound: 4


t: 31
s: 7


i: 5
r: 311


p0: 1245


j: 4
p: 18661
Número de bits do strong prime: 15
```

In [42]:

```
strongest_prime_verification(p, r, s, t, i, j)
```

```
First Verification: True
Second Verification: True
Third Verification: True
Fourth Verification:
     a) True
     b) True
     c) True
     d) 2^2 * 3 * 5 * 311
Fifth Verification:
     e) True
     f) True
     g) True
     h) 2 * 7 * 31 * 43
Sixth Verification:
     k) True
     l) True
     m) True
     n) 2 * 5 * 31
```

Exemplo 2

In [43]:

```
nbits = 126
p = gordonAlgorithm(nbits)
nbits2 = nbits/2
print("Número de bits: " + str(nbits))
print("nbits: " + str(nbits2))
print("nbitsLbound: " + str(nbits2-1))
print("\n")
print("t: " + str(t))
print("s: " + str(s))
print("\n")
print("i: " + str(i))
print("r: " + str(r))
print("\n")
print("p0: " + str(p0))
print("\n")
print("j: " + str(j))
print("p: " + str(p))
print("Número de bits do strong prime: " + str(p.nbits()))
```

```
Número de bits: 126
nbits: 63
nbitsLbound: 62


t: 7119124826779017113
s: 3499998670523038511


i: 7
r: 99667747574906239583


p0: 488812231604663801030514607650864021603


j: 31
p: 22116705239988917121523773465469162038209
Número de bits do strong prime: 135
```

In [44]:

```
strongest_prime_verification(p, r, s, t, i, j)
```

```
First Verification: True
Second Verification: True
Third Verification: True
Fourth Verification:
     a) True
     b) True
     c) True
     d) 2^6 * 31 * 1039 * 3275411 * 32865691 * 99667747574906239583
Fifth Verification:
     e) True
     f) True
     g) True
     h) 2 * 3 * 5 * 11 * 43 * 4243 * 104953547671283 * 3499998670523038511
Sixth Verification:
     k) True
     l) True
     m) True
     n) 2 * 7 * 7119124826779017113
```