Universidad Nacional Autónoma de México
Facultad de Ciencias
Criptografía y Seguridad

Tarea 2

Ángel Iván Gladín García
No. cuenta: 313112470
angelgladin@ciencias.unam.mx

29 de Abril 2019

1. Sea $\mathbb{Z}_{89627}$

a) Muestre que 2 es generardor de $\mathbb{Z}^*_{89627}$.
Solución: Se procederá a dar la la definición de generador:

Definición 1. *Una unidad $g \in \mathbb{Z}^*_n$ es llamado un **generador** o **raíz primitiva** de $\mathbb{Z}^*_n$ si cada $a \in \mathbb{Z}^*_n$ se tiene $g^k = a$ para algún entero k.*

Teniendo esa definición se hizo un programa en Python para dado un generador g y p primo tal que $|\mathbb{Z}_p| = p$ verificar si g genera a $\mathbb{Z}^*_n$.

```
# Generador del grupo
g = 2
# Orden del grupo
p = 89627

def g_es_generador(g, p):
    # Se crea conjunto para no tener repetidos y se guardarán los elementos de Z_p
    s = set()
    # Se itera de 1 hasta p-1 las potencias denotado por k
    for k in range(p):
        # Agregamos g^k mod p al conjunto
        s.add(pow(g, k, p))
    # Regresamos la cardinalidad del conjunto.
    return len(s)

# Verificamos si en efecto es un generador
print(g_es_generador(g, p) == p-1)
```

Con el *script* previo se puede corroborar que en efecto 2 genera a $\mathbb{Z}^*_n$.

b) Mediante cálculo de índices encontrar $\log_2(88777)$.
Solución:

Figura 1: Algoritmo de cálculo de índices para logaritmos discretos en grupos cíclicos.

3.68 Algorithm Index-calculus algorithm for discrete logarithms in cyclic groups

INPUT: a generator α of a cyclic group G of order n, and an element $\beta \in G$.
OUTPUT: the discrete logarithm $y = \log_\alpha \beta$.

1. (*Select a factor base* S) Choose a subset $S = \{p_1, p_2, \ldots, p_t\}$ of G such that a "significant proportion" of all elements in G can be efficiently expressed as a product of elements from S.
2. (*Collect linear relations involving logarithms of elements in* S)

2.1 Select a random integer $k, 0 \leq k \leq n-1$, and compute α^k.
2.2 Try to write α^k as a product of elements in S:

$$\alpha^k = \prod_{i=1}^{t} p_i^{c_i}, \quad c_i \geq 0. \tag{3.5}$$

If successful, take logarithms of both sides of equation (3.5) to obtain a linear relation

$$k \equiv \sum_{i=1}^{t} c_i \log_\alpha p_i \pmod{n}. \tag{3.6}$$

2.3 Repeat steps 2.1 and 2.2 until $t + c$ relations of the form (3.6) are obtained (c is a small positive integer, e.g. $c = 10$, such that the system of equations given by the $t + c$ relations has a unique solution with high probability).

3. (*Find the logarithms of elements in* S) Working modulo n, solve the linear system of $t + c$ equations (in t unknowns) of the form (3.6) collected in step 2 to obtain the values of $\log_\alpha p_i, 1 \leq i \leq t$.
4. (*Compute* y)
   4.1 Select a random integer $k, 0 \leq k \leq n-1$, and compute $\beta \cdot \alpha^k$.
   4.2 Try to write $\beta \cdot \alpha^k$ as a product of elements in S:

$$\beta \cdot \alpha^k = \prod_{i=1}^{t} p_i^{d_i}, \quad d_i \geq 0. \tag{3.7}$$

If the attempt is unsuccessful then repeat step 4.1. Otherwise, taking logarithms of both sides of equation (3.7) yields $\log_\alpha \beta = (\sum_{i=1}^{t} d_i \log_\alpha p_i - k) \bmod n$; thus, compute $y = (\sum_{i=1}^{t} d_i \log_\alpha p_i - k) \bmod n$ and return(y).

Con ayuda de un programa que encontré en internet `https://github.com/simonedaniello/Index-Calculus-in-C` verifiqué que $\log_2 88777 = 137$ en $\mathbb{Z}^*_{89267}$.

c) Mediante paso grande paso chico encontrar $\log_2(88777)$.
Solución: Se realizo un programa en *Python* siguiendo el algoritmo descrito abajo para facilitar las cuentas:

```python
from math import ceil, sqrt


def bsgs(g, beta, p):
    # Raíz de p
    M = ceil(sqrt(p - 1))

    # La tabla con las entradas (alpha^j, j)
    tabla = {pow(g, j, p): j for j in range(M)}

    # alpha^{-m}
    c = pow(g, M * (p - 2), p)

    for j in range(M):
        gamma = (beta * pow(c, j, p)) % p
        if gamma in tabla:
            return j * M + tabla[gamma]

    # No se encontró solución.
    return None

# Generador del grupo
g = 2
# Beta
beta = 88777
# Orden del grupo más uno
p = 89627

print(bsgs(g, beta, p))
```

Figura 2: Algoritmo de paso grande paso chico

3.56 Algorithm Baby-step giant-step algorithm for computing discrete logarithms

INPUT: a generator α of a cyclic group G of order n, and an element $\beta \in G$.
OUTPUT: the discrete logarithm $x = \log_\alpha \beta$.

1. Set $m \leftarrow \lceil \sqrt{n} \rceil$.
2. Construct a table with entries (j, α^j) for $0 \leq j < m$. Sort this table by second component. (Alternatively, use conventional hashing on the second component to store the entries in a hash table; placing an entry, and searching for an entry in the table takes constant time.)
3. Compute α^{-m} and set $\gamma \leftarrow \beta$.
4. For i from 0 to $m - 1$ do the following:
   4.1 Check if γ is the second component of some entry in the table.
   4.2 If $\gamma = \alpha^j$ then return($x = im + j$).
   4.3 Set $\gamma \leftarrow \gamma \cdot \alpha^{-m}$.

Ejecución del algoritmo:

- Sea $m \leftarrow 300$.
- Construir una tabla con las entradas (j, α^j), (en el programa se hizo al revez).

"1": 0	"2817": 201	"49779": 181,	"70752": 69
"1024": 10	"28254": 72	"49951": 88,	"70843": 250
"10275": 89	"28365": 108	"50238": 44,	"7101": 297
"1034": 207	"28404": 299	"50329": 262,	"71462": 22
"10364": 30	"28982": 253	"50763": 237,	"71814": 37
"10849": 45	"29371": 283	"50859": 293,	"72139": 57
"10968": 275	"29498": 106	"50915": 157,	"72202": 82
"11031": 263	"30211": 95	"512": 9,	"72919": 217
"11130": 242	"30796": 131	"51281": 145,	"72998": 197
"11268": 203	"3106": 222	"51337": 101,	"73500": 41
"11410": 113	"31154": 258	"517": 206,	"74563": 281
"11899": 238	"31217": 97	"51740": 148,	"74773": 93
"12091": 294	"31432": 79	"5182": 29,	"75438": 169
"12203": 158	"31988": 163	"51877": 70,	"76027": 141
"12424": 224	"32": 5	"52059": 251,	"76153": 19
"128": 7	"32330": 123	"52188": 104,	"76197": 34
"12935": 146	"32768": 15	"52602": 256,	"76345": 194
"13047": 102	"32780": 190	"52896": 121,	"76650": 166
"13224": 119	"32871": 171	"53297": 23,	"7699": 129
"1371": 272	"3306": 117	"54001": 38,	"77801": 175
"13853": 149	"33088": 212	"54651": 58,	"78120": 230
"14127": 71	"33557": 133	"54777": 83,	"78287": 50
"14202": 298	"33934": 25	"5484": 274,	"7858": 77
"14491": 252	"34989": 260	"55412": 151,	"78595": 269
"14749": 105	"35227": 143	"5565": 241,	"78700": 61
"15398": 130	"35241": 99	"55714": 286,	"79386": 126
"1553": 221	"35376": 68	"56211": 218,	"79448": 185
"15577": 257	"35731": 21	"5634": 202,	"79665": 179
"15716": 78	"35907": 36	"56369": 198,	"79708": 86
"15994": 162	"36101": 81	"56508": 73,	"79911": 235
"16": 4	"36499": 196	"56730": 109,	"79935": 291
"16165": 122	"36750": 40	"5705": 112,	"79949": 155
"16384": 14	"37719": 168	"57373": 42,	"7997": 161
"16390": 189	"38325": 165	"57964": 254,	"8": 3
"1653": 116	"39060": 229	"58742": 284,	"80235": 249
"16544": 211	"3929": 76	"58996": 107,	"80883": 56
"16967": 24	"39350": 60	"59499": 282,	"81273": 216
"17688": 67	"39693": 125	"59919": 94,	"8192": 13
"18375": 39	"39724": 184	"60422": 96,	"8195": 188
"19530": 228	"39854": 85	"61249": 170,	"82095": 280
"19675": 59	"4": 2	"61592": 132,	"82200": 92
"19862": 183	"4096": 12	"6212": 223,	"8272": 210
"19927": 84	"41100": 91	"62308": 259,	"82827": 140
"2": 1	"4136": 209	"62427": 142,	"82890": 18
"2048": 11	"41445": 17	"62434": 98,	"82912": 33
"20550": 90	"41456": 32	"62679": 20,	"82986": 193
"2068": 208	"41493": 192	"62767": 35,	"83714": 174
"20728": 31	"41857": 173	"62864": 80,	"83957": 49
"21197": 152	"42323": 177	"63063": 195,	"84111": 268
"21698": 46	"42394": 153	"63673": 167,	"84646": 178
"21801": 287	"42725": 214	"63976": 164,	"84769": 234

- Calcular $\alpha^{-m} = 59166$
- Regresar $x = im + j = 137$

Por tanto $\log_2 88777 = 137$ en $\mathbb{Z}^*_{89267}$.

d) Mediante ρ-Pollard para logaritmos encontrar $\log_2(88777)$.
Solución:

Figura 3: Algoritmo de la Rho de Pollard para calcular logaritmo discreto

3.6.3 Pollard's rho algorithm for logarithms

Pollard's rho algorithm (Algorithm 3.60) for computing discrete logarithms is a randomized algorithm with the same expected running time as the baby-step giant-step algorithm (Algorithm 3.56), but which requires a negligible amount of storage. For this reason, it is far preferable to Algorithm 3.56 for problems of practical interest. For simplicity, it is assumed in this subsection that G is a cyclic group whose order n is prime.

The group G is partitioned into three sets S_1, S_2, and S_3 of roughly equal size based on some easily testable property. Some care must be exercised in selecting the partition; for example, $1 \notin S_2$. Define a sequence of group elements $x_0, x_1, x_2, \ldots$ by $x_0 = 1$ and

$$x_{i+1} = f(x_i) \stackrel{\text{def}}{=} \begin{cases} \beta \cdot x_i, & \text{if } x_i \in S_1, \\ x_i^2, & \text{if } x_i \in S_2, \\ \alpha \cdot x_i, & \text{if } x_i \in S_3, \end{cases} \tag{3.2}$$

for $i \geq 0$. This sequence of group elements in turn defines two sequences of integers $a_0, a_1, a_2, \ldots$ and $b_0, b_1, b_2, \ldots$ satisfying $x_i = \alpha^{a_i}\beta^{b_i}$ for $i \geq 0$: $a_0 = 0, b_0 = 0$, and for $i \geq 0$,

$$a_{i+1} = \begin{cases} a_i, & \text{if } x_i \in S_1, \\ 2a_i \bmod n, & \text{if } x_i \in S_2, \\ a_i + 1 \bmod n, & \text{if } x_i \in S_3, \end{cases} \tag{3.3}$$

and

$$b_{i+1} = \begin{cases} b_i + 1 \bmod n, & \text{if } x_i \in S_1, \\ 2b_i \bmod n, & \text{if } x_i \in S_2, \\ b_i, & \text{if } x_i \in S_3. \end{cases} \tag{3.4}$$

Floyd's cycle-finding algorithm (Note 3.8) can then be utilized to find two group elements x_i and x_{2i} such that $x_i = x_{2i}$. Hence $\alpha^{a_i}\beta^{b_i} = \alpha^{a_{2i}}\beta^{b_{2i}}$, and so $\beta^{b_i - b_{2i}} = \alpha^{a_{2i} - a_i}$. Taking logarithms to the base α of both sides of this last equation yields

$$(b_i - b_{2i}) \cdot \log_\alpha \beta \equiv (a_{2i} - a_i) \pmod{n}.$$

Provided $b_i \not\equiv b_{2i} \pmod{n}$ (note: $b_i \equiv b_{2i}$ occurs with negligible probability), this equation can then be efficiently solved to determine $\log_\alpha \beta$.

3.60 Algorithm Pollard's rho algorithm for computing discrete logarithms

INPUT: a generator α of a cyclic group G of prime order n, and an element $\beta \in G$.
OUTPUT: the discrete logarithm $x = \log_\alpha \beta$.

1. Set $x_0 \leftarrow 1, a_0 \leftarrow 0, b_0 \leftarrow 0$.
2. For $i = 1, 2, \ldots$ do the following:
   2.1 Using the quantities $x_{i-1}, a_{i-1}, b_{i-1}$, and $x_{2i-2}, a_{2i-2}, b_{2i-2}$ computed previously, compute x_i, a_i, b_i and x_{2i}, a_{2i}, b_{2i} using equations (3.2), (3.3), and (3.4).
   2.2 If $x_i = x_{2i}$, then do the following:
   Set $r \leftarrow b_i - b_{2i} \bmod n$.
   If $r = 0$ then terminate the algorithm with failure; otherwise, compute $x = r^{-1}(a_{2i} - a_i) \bmod n$ and return(x).

Script en Python que hace ρ-Pollard.

```
def ext_euclid(a, b):
    if b == 0:
```

```
        return a, 1, 0
    else:
        d, xx, yy = ext_euclid(b, a % b)
        x = yy
        y = xx - (a / b) * yy
        return d, x, y


def inverse(a, n):
    return ext_euclid(a, n)[1]


def xab(x, a, b, (G, H, P, Q)):
    sub = x % 3

    if sub == 0:
        x = x * G % P
        a = (a + 1) % Q

    if sub == 1:
        x = x * H % P
        b = (b + 1) % Q

    if sub == 2:
        x = x * x % P
        a = a * 2 % Q
        b = b * 2 % Q

    return x, a, b

def pollard(G, H, P):
    Q = (P - 1) / 2

    x = G * H
    a = 1
    b = 1

    X = x
    A = a
    B = b

    for i in range(1, P):
        # Tortuga
        x, a, b = xab(x, a, b, (G, H, P, Q))

        # Liebre
        X, A, B = xab(X, A, B, (G, H, P, Q))
        X, A, B = xab(X, A, B, (G, H, P, Q))
```

```
            if x == X:
                break

        nom = a - A
        denom = B - b

        res = (inverse(denom, Q) * nom) % Q

        if pow(G, res, P) == H:
            return res

        return res + Q


generador = 2
beta = 88777
p = 89627

res = pollard(generador, beta, p)
print(res)
```

Por tanto $\log_2 88777 = 137$ en $\mathbb{Z}^*_{89267}$.

e) Con el método de su preferencia calcular $\log_2(54539)$.
Solución: Usando el algoritmo de paso chico paso grande queda como:
Ejecución del algoritmo:

- Sea $m \leftarrow 300$.

"1": 0	"23798": 239	"41456": 32	"56730": 109	"76197": 34
"1024": 10	"23833": 110	"41493": 192	"5705": 112	"76345": 194
"10275": 89	"24182": 295	"41857": 173	"57373": 42	"76650": 166
"1034": 207	"24406": 159	"42323": 177	"57964": 254	"7699": 129
"10364": 30	"24848": 225	"42394": 153	"58742": 284	"77801": 175
"10849": 45	"25119": 43	"42725": 214	"58996": 107	"78120": 230
"10968": 275	"256": 8	"43396": 47	"59499": 282	"78287": 50
"11031": 263	"25870": 147	"43599": 232	"59919": 94	"7858": 77
"11130": 242	"2591": 28	"43602": 288	"60422": 96	"78595": 269
"11268": 203	"26094": 103	"43872": 277	"61249": 170	"78700": 61
"11410": 113	"26301": 255	"44124": 265	"61592": 132	"79386": 126
"11899": 238	"26448": 120	"4422": 65	"6212": 223	"79448": 185
"12091": 294	"2742": 273	"44267": 52	"62308": 259	"79665": 179
"12203": 158	"27706": 150	"44520": 244	"62427": 142	"79708": 86
"12424": 224	"27857": 285	"44601": 135	"62434": 98	"79911": 235
"128": 7	"2817": 201	"45072": 205	"62679": 20	"79935": 291
"12935": 146	"28254": 72	"45499": 271	"62767": 35	"79949": 155
"13047": 102	"28365": 108	"45590": 220	"62864": 80	"7997": 161
"13224": 119	"28404": 299	"45640": 115	"63063": 195	"8": 3
"1371": 272	"28982": 253	"45919": 63	"63673": 167	"80235": 249
"13853": 149	"29371": 283	"46109": 27	"63976": 164	"80883": 56
"14127": 71	"29498": 106	"46222": 200	"64": 6	"81273": 216
"14202": 298	"30211": 95	"46778": 75	"64660": 124	"8192": 13
"14491": 252	"30796": 131	"47596": 240	"65536": 16	"8195": 188
"14749": 105	"3106": 222	"47666": 111	"65560": 191	"82095": 280
"15398": 130	"31154": 258	"48364": 296	"65742": 172	"82200": 92
"1553": 221	"31217": 97	"48663": 128	"65975": 176	"8272": 210
"15577": 257	"31432": 79	"48812": 160	"6612": 118	"82827": 140
"15716": 78	"31988": 163	"48911": 187	"66176": 213	"82890": 18
"15994": 162	"32": 5	"49696": 226	"66613": 231	"82912": 33
"16": 4	"32330": 123	"49779": 181	"66947": 51	"82986": 193
"16165": 122	"32768": 15	"49951": 88	"67114": 134	"83714": 174
"16384": 14	"32780": 190	"50238": 44	"67563": 270	"83957": 49
"16390": 189	"32871": 171	"50329": 262	"67773": 62	"84111": 268
"1653": 116	"3306": 117	"50763": 237	"67868": 26	"84646": 178
"16544": 211	"33088": 212	"50859": 293	"69145": 127	"84769": 234
"16967": 24	"33557": 133	"50915": 157	"69269": 186	"84781": 290
"17688": 67	"33934": 25	"512": 9	"69703": 180	"84788": 154
"18375": 39	"34989": 260	"51281": 145	"69789": 87	"84931": 248
"19530": 228	"35227": 143	"51337": 101	"69978": 261	"85255": 55
"19675": 59	"35241": 99	"517": 206	"70195": 236	"85450": 215
"19862": 183	"35376": 68	"51740": 148	"70243": 292	"85861": 279
"19927": 84	"35731": 21	"5182": 29	"70271": 156	"86227": 139
"2": 1	"35907": 36	"51877": 70	"70454": 144	"86792": 48
"2048": 11	"36101": 81	"52059": 251	"70482": 100	"86869": 267
"20550": 90	"36499": 196	"52188": 104	"70752": 69	"87198": 233
"2068": 208	"36750": 40	"52602": 256	"70843": 250	"87204": 289
"20728": 31	"37719": 168	"52891": 121	"7101": 297	"87279": 247
"21197": 152	"38325": 165	"53297": 23	"71462": 22	"87441": 54
"21698": 46	"39060": 229	"54001": 38	"71814": 37	"87744": 278
"21801": 287	"3929": 76	"54651": 58	"72139": 57	"87927": 138

- Calcular $\alpha^{-m} = 59166$.
- Regresar $x = im + j = 953$.

Por tanto $\log_2 54539 = 953$ en $\mathbb{Z}^*_{89267}$.

f) Si los parámetros públicos son $(89627, 2, 88777)$, descifrar el siguiente mensaje el cual está encriptado con Gamal justifica tu desifrado.
Si $\gamma = 54539$.

$(\gamma, 81315)(\gamma, 87570)$ $(\gamma, 31275)(\gamma, 50040)$ $(\gamma, 31275)$
$(\gamma, 18764)(\gamma, 500040)$ $(\gamma, 31275)(\gamma, 50040)$ $(\gamma, 12510)$
$(\gamma, 50040)(\gamma, 68805)$

Solución: La llave pública A es de la forma (p, α, α^a), fijándose en el tercer parámetro de la llave que es 88777 la cual ya se había calculado su logaritmo (base 2) que corresponde con el generador a la potencia k, es decir, sea g un generador de de $\mathbb{Z}^*_{89627}$ y k un natural, entonces $g^k \in \mathbb{Z}^*_{89627}$. Por consiguiente $\alpha^a \mod 89627$, que es $2^{137} \mod 89627$. Por lo que $a = 137$ donde a es la **llave privada**.

Figura 4: Algoritmo de ElGamal

8.18 Algorithm ElGamal public-key encryption

SUMMARY: B encrypts a message m for A, which A decrypts.

1. *Encryption*. B should do the following:
   (a) Obtain A's authentic public key (p, α, α^a).
   (b) Represent the message as an integer m in the range $\{0, 1, \ldots, p-1\}$.
   (c) Select a random integer $k, 1 \leq k \leq p-2$.
   (d) Compute $\gamma = \alpha^k \bmod p$ and $\delta = m \cdot (\alpha^a)^k \bmod p$.
   (e) Send the ciphertext $c = (\gamma, \delta)$ to A.
2. *Decryption*. To recover plaintext m from c, A should do the following:
   (a) Use the private key a to compute $\gamma^{p-1-a} \bmod p$ (note: $\gamma^{p-1-a} = \gamma^{-a} = \alpha^{-ak}$).
   (b) Recover m by computing $(\gamma^{-a}) \cdot \delta \bmod p$.

Para agililizar las cuentas y descrifrar el texto más rápido, se hizo un *script* en Python para descifrar el texto:

```
gamma = 54539

mensaje = [[(gamma, 81315,), (gamma, 87570,)],
           [(gamma, 31275,), (gamma, 35473,)],
           [(gamma, 25020,)],
           [(gamma, 18765,), (gamma, 50040,)],
           [(gamma, 31275,), (gamma, 50040,)],
           [(gamma, 12510,)],
```

```
            [(gamma, 50040,), (gamma, 68805,)]]

# Llave privada `a`
a = 137
# Z_p
p = 89627

def descifra_elgamal(g, d, a, p):
    # \gamma^{p-1-a} = \gamma^{-a}
    gamma_inv_a = pow(g, p - 1 - a, p)
    # (\gamma) \cdot \delta mod p
    m = pow(gamma_inv_a * d, 1, p)
    return m

# Cadena para guardar el texto descigrado
s = ''
# Iterar la lista
for l in mensaje:
    # Iterar las tuplas
    for t in l:
        # Tomamos la posición 0 y 1 de la tupla porque corresponden
        # a gamma y delta respectivamente.
        num_letra = descifra_elgamal(t[0], t[1], a, p)
        # Se hace un mapeo del número resultante al alfabeto
        s += chr(num_letra + 65)

# Texto descifrado
print(s)
```

Dando como resultado el texto descifrado:

$$NOFUEDIFICIL$$

2. Sea $n = 4633$

a) Descomponer n con el algoritmo de la criba cuadrática.
Solución: Antes de proceder con el algoritmo, mostraremos el algoritmo y unas definiciones que serán usadas.
Decimos que un número es liso (smooth number en inglés) si se factoriza en número enteros *pequeños*. Por ejemplo un número 7-*smooth* es un número cuyos factores son a lo más 7 como $49 = 7^2$.
La fórmula explícita de símbolo de Lagrange es la siguiente:

$$(\frac{a}{p}) \equiv a^{\frac{p-1}{2}} \pmod{p} \qquad (\frac{a}{p}) \in \{-1, 0, 1\}$$

Ejecución del algoritmo.

Figura 5: Algoritmo de la criba cuadrática

3.21 Algorithm Quadratic sieve algorithm for factoring integers

INPUT: a composite integer n that is not a prime power.
OUTPUT: a non-trivial factor d of n.

1. Select the factor base $S = \{p_1, p_2, \ldots, p_t\}$, where $p_1 = -1$ and p_j ($j \geq 2$) is the $(j-1)^{\text{th}}$ prime p for which n is a quadratic residue modulo p.
2. Compute $m = \lfloor \sqrt{n} \rfloor$.
3. (*Collect $t+1$ pairs (a_i, b_i). The x values are chosen in the order $0, \pm 1, \pm 2, \ldots$.*) Set $i \leftarrow 1$. While $i \leq t+1$ do the following:
   3.1 Compute $b = q(x) = (x+m)^2 - n$, and test using trial division (cf. Note 3.23) by elements in S whether b is p_t-smooth. If not, pick a new x and repeat step 3.1.
   3.2 If b is p_t-smooth, say $b = \prod_{j=1}^{t} p_j^{e_{ij}}$, then set $a_i \leftarrow (x+m)$, $b_i \leftarrow b$, and $v_i = (v_{i1}, v_{i2}, \ldots, v_{it})$, where $v_{ij} = e_{ij} \bmod 2$ for $1 \leq j \leq t$.
   3.3 $i \leftarrow i+1$.
4. Use linear algebra over $\mathbb{Z}_2$ to find a non-empty subset $T \subseteq \{1, 2, \ldots, t+1\}$ such that $\sum_{i \in T} v_i = 0$.
5. Compute $x = \prod_{i \in T} a_i \bmod n$.
6. For each j, $1 \leq j \leq t$, compute $l_j = (\sum_{i \in T} e_{ij})/2$.
7. Compute $y = \prod_{j=1}^{t} p_j^{l_j} \bmod n$.
8. If $x \equiv \pm y \pmod n$, then find another non-empty subset $T \subseteq \{1, 2, \ldots, t+1\}$ such that $\sum_{i \in T} v_i = 0$, and go to step 5. (In the unlikely case such a subset T does not exist, replace a few of the (a_i, b_i) pairs with new pairs (step 3), and go to step 4.)
9. Compute $d = \gcd(x-y, n)$ and return(d).

1. Tomamos $S = \{-1, 2, 3, 17, 19, 29\}$[1] de tamaño $t = 6$ porque según el algoritmo se toman los que $(\frac{n}{p}) = (\frac{4633}{p}) = 1$.
2. Calculamos $m = 68$.
3. Se hace una tabla con los $t+1$ valores de x para los cuales $q(x)$ es 29-liso.

i	x	$q(x)$	Factorización de $q(x)$	a_i	v_i
1	0	-9	-2^3	68	(1,0,0,0,0,0)
2	-1	-144	$-2^4 \cdot 3^2$	67	(1,0,0,0,0,0)
3	1	128	2^7	69	(0,1,0,0,0,0)
4	-3	-408	$-2^3 \cdot 3 \cdot 17^1$	65	(1,1,1,1,0,0)
5	3	408	$2^3 \cdot 3^1 \cdot 17^1$	71	(0,1,1,1,0,0)
6	4	551	$19 \cdot 29$	72	(0,0,0,0,1,1)
7	5	696	$2^3 \cdot 3 \cdot 29$	73	(0,1,1,0,0,1)

4. Tomando $T = \{v_1 + v_2\}$ tales que $v_1 + v_2 = 0$.
5. Calcular $x = (a_1 a_2 \mod n) = 4489$.
6. Calcular $l_1 = 1$, $l_2 = 2$, $l_3 = 2$, $l_4 = 0$, $l_5 = 0$, $l_6 = 0$.

[1] Se calculó cada dígito con la ayuda de esta página web `https://adrianstoll.com/number-theory/jacobi.html` (bien se pudo programar pero se encontró más rápido de hacer con ésa herramienta).

7. Calcular $y = -2^2 \cdot 3^2 = -36$
8. Como $4489 \not\equiv \pm 36 \pmod{n}$.
9. Entonces calculados el mínimo común múltiplo $gcd(x - y, n) = gcd(4592, 4633) = 41$. Por tanto 4633 tiene dos factores no triviales que son 41 y 113.

b) Cálcular $\phi(n)$ y descomponerla como producto de potencias de primos.
Solución: Recordando de defnición de la función aritmética ϕ que es:

$$\phi(n) = n \prod_{p|n} (1 - \frac{1}{p})$$

Como por el inciso anterior sabemos que $n = 4633$ es un número compuesto de la forma $n = 41 \cdot 113$ podemos utilizar la definición de ϕ.
Con ayuda de *Python* se creo un programa para calcular $\phi(n)$,

```
# Para mejor manejo de números racionales
from fractions import Fraction


def phi(n, factores_primos):
    # Incializamos el resultado con el valor de `n`
    r = Fraction(n, 1)
    # Iteramos sobra cada valor primo y lo multiplicamos
    # según la definición de phi
    for p in factores_primos:
        r *= (1 - Fraction(1, p))
    # Como es un entero, regresamos el numerador
    return r.numerator

# Número al que se le aplicara la phi
N = 4633
# Lista de factores de N
L = [41, 113]

# Phi de Euler
print(phi(N, L))
```

Ergo $\phi(\mathbf{n}) = \mathbf{4480}$.

Para encontrar la descomposisción de primos de $\phi(n)$ primero se hizo un analisis previo con Factor[2] y se percató que sus factores primos no son muy grandes, para evitar *repercusiones* en la calificación de esta tarea se procedió a realizxar un programa que realizará la factorización en primos del número.

```
# Para mejor manejo de nímeros racionales
from fractions import Fraction
```

[2]Un programa incluido en los binarios en la mayoría de las distribuciones Linux que dado un número lo descompone en primos. Para más información ver la siguiente página web `https://www.howtoforge.com/linux-factor-command/`

```python
# Lista de los primeros 6 primos
primos = [2, 3, 5, 7, 11, 13]

def descoposicion_primos(N):
    factores_primos = []
    for p in primos:
        # Verificar si el número primo en turno es factor del número
        if N % p == 0:
            # Potencia del primo
            e = 1
            # Se Verifica hasta que potencia del primo divide al número
            while N % pow(p, e) == 0:
                # Se agrega el factor primo
                factores_primos.append(p)
                e += 1
    return factores_primos

N = 4480
print(descoposicion_primos(N))
```

Una vez hecho el *script* previo, se puede apreciar que los factores primos son los siguientes:

$$[2, 2, 2, 2, 2, 2, 2, 5, 7]$$

Ergo la descomposición en producto de primos queda como:

$$\phi(n) = 4480 = 2^7 \cdot 5^1 \cdot 7^1$$

c) Mostrar que $(11, \phi(n-1)) = 1$.
Solución: Teniendo $\phi(n-1) = \phi(4632)$, se procederá a obtener sus factorización canónica en primos para después aplicar la definición de la función aritmética de phi de Euler. Como $4623 = 2^3 \cdot 3 \cdot 193$, utilizando el programa previamente hecho en el inciso $2b$) se tiene que $\phi(4632) = 1536$.
Ahora para exhibir que 11 y 1536 son primos relativos, se deberan ver sus dividores y por definición deben de tener al 1 como único factor común. Siendo 11 un primo y 1536 un número compuesto de la forma $1536 = 2^9 \cdot 3$. Ergo, el único número que los divide es 1.

d) Encontrar d tal que $d(11) \cong 1 \mod (\phi(n-1))$.
Solución: Recordando de defínción de la función aritmética d que es:

$$d(n) = \sum_{d|n} 1$$

Sea n la descomposición canónica de un número, podemos descomponerlo de la siguiente manera $d(n) = \prod_{i=1}^{k} p_i^{e_i}$. Como 11 ser un número primo podemos aplicar la función aritmetica quedando de la forma $d(11) = (e_1 + 1) = 1 + 1 = 2$.
Quedando la congruencia de la siguiente forma (recordando que $\phi(n-1) = \phi(4632) = 1536$):

$$2 \cong 1 \mod 1536$$

e) Si la llave pública es $(n, 11)$ descifrar el siguiente mensaje:

930
1439
3707
484
2048
484
1093

Solución: Teniendo calculado previamente $\phi(4633) = \phi(41)\phi(113) = 40 \cdot 112 = 4480$ Se procederá con la ejecución del algoritmo.

- $n = 4633 = 41 \cdot 113$.
- Calcular $\phi(4633) = 4480$.
- Nos dan $e = 11$ como parámetro de nuestra llave pública, lo único es que se debe corroborar que $(4633, e) = (4633, 11) = 1$ (lu cual es verdad).
- Encontrar una d (llave privada) tal que se satisfaga la siguiente congruencia $de \cong 1$ mód $\phi(n)$ que sustituyendo queda de la forma $d11 \cong 1 \mod 4480$ que con ayuda de un programa[3] se determinó dando $d = 2851$.
- Sea $d = 2851$.

Hecho el análisis previo, y teniendo la llave privada d, se sigue que $n = 4633$ y $d = 2851$. Ahora se debe seguir la función de decifrado que es:

$$m = c^d \mod n$$

Con ayuda de un programa en *Python* se hizo la exponenciación modular, que su forma de empleo se traduce como $f(x, e, m) = x^e \mod m$

```
def f(x, e, m):
    X = x
    E = e
    Y = 1
    while E > 0:
        if E % 2 == 0:
            X = (X * X) % m
            E = E / 2
        else:
            Y = (X * Y) % m
            E = E - 1
    return Y
```

Sabiendo eso, se tiene que aplicar la función a cada renglón del mensaje, igual con ayuda de Python

```
l = [930,1439,3707,484,2048,484,1093]

m = [f(x, 2851, 4633) for x in l]
```

[3] http://www.a-calculator.com/congruence/

Dando $m = [5, 4, 11, 8, 2, 8, 3]$ haciendo un mapeo a cada letra del alfabeto indicando su posición $[0 \to A]...[25 \to z]$ se tiene que el mensaje descifrado es:

FELICID

3. Ejercicio 3

a) Mostrar que el problema del logaritmo discreto no depende del generador.
Solución: Sea α and γ dos generadores de un grupo cíclico G de orden n, y sea $\beta \in G$. Sea $x = \log_a \beta$, $y = \log_\gamma \beta$, y $z = \log_\alpha \gamma$. Entonces $\alpha^x = \beta = \gamma^y = (\alpha^z)^y$. Por consiguiente $x = zy$ mód n, y

$$\log_\gamma \beta = (\log_\alpha \beta)(\log_\alpha \gamma)^{-1} \quad \text{mód } n$$

Esto significa que cualquier algoritmo de que calcule logarítmos a una base α puede ser usado para calcular logaritmos a cualquier otra base γ que también es un generador de G.

b) Sea $n = 10942095573514557503$ decir si es primo o compuesto en caso de ser compuesto descomponerlo. (usando métodos vistos en clase).
Solución: Usando el algoritmo de la criba cuadrática se procederá a factorizar el número. Gracias a la ayuda del ayudantede laboratorio que nos proporcionó[4] y explicó el código en clase el código de éste, se utilizará para realizar su factorización y determinar si es compuesto. Después de ejectutar el código, vimos que el número es compuesto de la forma:

$$10942095573514557503 = 23 \cdot 494664743 \cdot 961748927$$

Referencias

[1] Alfred J. Menezes, Scott A. Vanstone, and Paul C. Van Oorschot. 1996. *Handbook of Applied Cryptography (1st ed.)*. CRC Press, Inc., Boca Raton, FL, USA.

[2] Smooth number `https://en.wikipedia.org/wiki/Smooth_number`

[3] Notas tomadas en clases, ayudantías y laboratorio.

[4] RSA Algorithm `https://simple.wikipedia.org/wiki/RSA_algorithm`

[5] Generators `https://crypto.stanford.edu/pbc/notes/numbertheory/gen.html`

[4] `https://github.com/kanekko/cryptography-and-security/blob/master/Lab07_Quadratic%20Sieve/QuadraticSieve.java`