Diffie-Hellman with a bad prime

João Nuno da Silva Luís, NMEC 107403

Para perceber melhor como funcionava o problema, comecei por tentar resolver os casos que o professor apresenta no Slide 42.

O primeiro programa que fiz era apenas um *for loop* que testava valores de alfa e via se a sua exponenciação era igual a A. Rapidamente esse algoritmo deixava de servir para números com mais de 7 algarismos.

Depois de alguma pesquisa, encontrei a implementação já em python do algoritmo **baby_step_giant_step** (https://gist.github.com/0xTowel/b4e7233fc86d8bb49698e4f1318a5a73) para o cálculo do logaritmo discreto, o que permitiu calcular o valor de S= 26991399064 em pouco tempo. No entanto, quando o fiz para os números do challenge, o programa acabou por dar *killed*.

Entretanto encontrei a função ***discrete_log*** do ***sympy.ntheory***, que automaticamente escolhe o melhor algoritmo para diferentes tamanhos do problema (https://docs.sympy.org/latest/modules/ntheory.html#sympy.ntheory.residue_ntheory.discrete_log). Esta função permitiu resolver o exercício de S=1267222359226852228 bastante rápido, mas para os números do challenge deu *killed*.

Depois de mais alguma pesquisa, encontrei que usando a fatorização de p-1, podíamos usar o algoritmo **Pohlig–Hellman** de forma eficiente. Para obter a fatorização, fui ao Wolfram Alpha, e fiz **factorize(p-1)** , obtendo os seguintes fatores primos:
2×5531790401440711×7456838241168989×8515259982369671×8745786245782841×9499742089305047×9700182083860057×10364391344477629×11426445140881751×11607899603171899×12153450212629753 (11 distinct prime factors).
No entanto, não consegui encontrar forma de passar diretamente os fatores à função *discrete_log* do sympy, pelo que com a ajuda do ChatGPT implementei o algoritmo de *pohlig_hellman*.

Esta é a versão final do programa, que me permitiu calcular o valor de S do challenge, tendo demorado cerca de 1h20 a calcular esse valor.

Valores obtidos:

alfa:
76185728014758291666092773912518825132405298878619634878181323024810271658723533
18054822588329712963365645885630406864545451241036914676146689344261587389223434

Shared key 'S':
19355214398071404590557919552376263455946950071305344323010571682019298343504563
44276580108149259190367874157372580456932714792940465750572323647741529430303243

Código que resolveu o challenge:

sympy_cha_imp.py > pohlig_hellman

```python
from sympy.ntheory import discrete_log
from sympy.ntheory.modular import crt

# Values for Diffie-Hellman
p = 9459040338068898689261456459355656735760126665215411768729930709057649580926326656219024573411869732770921533628486688534108189813533765746159096301632668722327
r = 5
A = 4273882268770451042535779967439697351016686492206027478809543375227950930138924865543470033644860424586440579542980690863469253402996371905611340139885738693080
B = 3847640689624364711664886026800290793265399414700736841705321414309213654447740358930872088921683993690069628098925077346749794347163383463395658935649932413838

# Manually input the factorization of p-1 as a dictionary
factorization = {
    2: 1,
    5531790401440711: 1,
    7456838241168989: 1,
    8515259982369671: 1,
    8745786245782841: 1,
    9499742089305047: 1,
    9700182083860057: 1,
    10364391344477629: 1,
    11426445140881751: 1,
    11607899603171899: 1,
    12153450212629753: 1
}

def pohlig_hellman(p, g, A, factorization):
    residues = []
    moduli = []

    for q, e in factorization.items():
        q_power = q**e
        g_q = pow(g, (p-1)//q_power, p)  # g^((p-1)/q) mod p
        A_q = pow(A, (p-1)//q_power, p)  # A^((p-1)/q) mod p

        # Solve discrete log for g_q^x = A_q (mod p)
        x_q = discrete_log(p, A_q, g_q, order=q_power)

        residues.append(x_q)
        moduli.append(q_power)

    # Combine results using Chinese Remainder Theorem (CRT)
    x, _ = crt(moduli, residues)
    return x

alpha = pohlig_hellman(p, r, A, factorization)

print(f"Private key 'alpha': {alpha}")

shared_key = pow(B, alpha, p)

print(f"Shared key: {shared_key}")
```