One prime to rule them all

W pierwszych krokach tworzenia pary kluczy dla algorytmu RSA, trzeba wygenerować dwie duże różne liczby pierwsze – zwyczajowo nazywane p i q. W kolejnym kroku mnoży się te liczby przez siebie i tak powstaje n:

$$n = p \cdot q$$

Wartość n jest później częścią zarówno klucza prywatnego, jak i publicznego, więc zasadniczo jest publicznie znana. Całe bezpieczeństwo algorytmu RSA opiera się na trudności odwrócenia procesu mnożenia (rozłożenia na czynniki pierwsze), tj. mając dane n trudno jest ustalić, jakie były wejściowe wartości p i q.

Oszczędni Elektrycerze popełnili błąd.

Jeśli wygeneruje się jedną liczbę pierwszą i pomnoży ją przez siebie, by otrzymać n, tj.:

$$n = p \cdot p$$

to matematyczne zasady działania RSA pozostaną bez zarzutu. Nadal będzie się dało zaszyfrować i odszyfrować tekst w ten sam sposób, co przy dwóch różnych liczbach pierwszych.

Jednocześnie bezpieczeństwo takiego rozwiązania jest żadne, ponieważ rozłożenie n na czynniki pierwsze staje się trywialnie proste – wystarczy obliczyć pierwiastek kwadratowy z n.

Należy pamiętać, że funkcja ϕ musi być obliczona trochę inaczej niż w „tradycyjnej" wersji RSA:

$$\text{Jeżeli } p \text{ jest liczbą pierwszą, to } \phi(p^k) = p^{k-1} \cdot (p-1).$$

W naszym przypadku $n = p \cdot p = p^2$, więc $k = 2$, a zatem:

$$\phi(p^2) = p^{2-1} \cdot (p-1) = p^1 \cdot (p-1) = p \cdot (p-1)$$

Pozostałe kroki, czyli wybranie drugiego elementu klucza publicznego (e), obliczenie drugiego elementu klucza prywatnego (d) oraz szyfrowanie i deszyfrowanie, przebiegają tak samo jak w normalnym RSA.

Wiedząc to wszystko, flagę można obliczyć skryptem w języku Python, podstawiając odpowiednie wartości w zaznaczonych miejscach:

```python
# python3 solution.py

import math
from Crypto.Util.number import long_to_bytes

# encryption/public key
n = #####
e = #####

# ciphertext
ct = #####

# computing the square root of n
p = math.isqrt(n)

# computing phi for p * p
phi = p * (p - 1)

# computing decryption/private key
d = pow(e, -1, phi)

# computing plaintext = decrypting ciphertext
pt = pow(ct, d, n)
decrypted = long_to_bytes(pt)

print(decrypted)
```