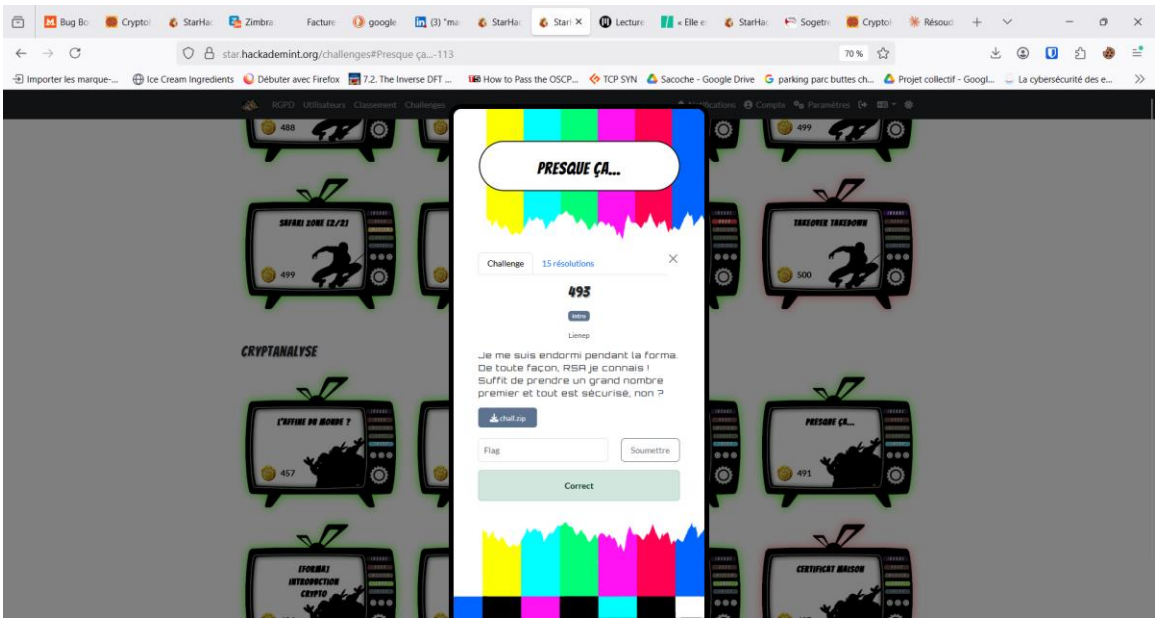
Presque ça



Write-Up

1. Contexte et objectif

Le challenge « Presque ça… » illustre une mauvaise compréhension de RSA. L'énoncé suggère qu'il suffit de choisir « un grand nombre premier » pour être en sécurité. En réalité, RSA standard exige un module N égal au produit de deux grands nombres premiers p et q. Dans ce défi, le module N est directement choisi comme un seul nombre premier fort de 1024 bits, ce qui rend le système trivialement cassable.

L'objectif du challenge est de retrouver le flag au format Star{…} à partir des seules données publiques : le module N, l'exposant public e et le chiffré c = m^e mod N.

2. Rappel sur le schéma RSA correct

2.1 Génération des clés

En RSA standard, on choisit deux grands nombres premiers distincts p et q, puis on calcule :

- • $N = p \cdot q$
- • $\varphi(N) = (p - 1)(q - 1)$, la fonction indicatrice d'Euler.
- • Un exposant public e tel que $\gcd(e, \varphi(N)) = 1$.

On définit alors la clé privée d comme l'inverse modulaire de e modulo φ(N), c'est-à-dire le nombre d satisfaisant $e \cdot d \equiv 1 \pmod{\varphi(N)}$.

2.2 Chiffrement et déchiffrement

Si m est le message clair (interprété comme un entier strictement inférieur à N), le chiffrement RSA standard est :

- • $c \equiv m^e \bmod N$.

Le déchiffrement consiste alors à calculer :

- • m ≡ c^d mod N.

La sécurité repose sur la difficulté de factoriser N pour retrouver p, q puis φ(N), et donc d.

3. Erreur d'implémentation dans le challenge

Dans le script fourni par le challenge, le module N est généré avec getStrongPrime(1024). Cela signifie que N est un unique nombre premier de 1024 bits, et non un produit p·q. Comme N est premier, la fonction d'Euler est alors donnée par la formule simple φ(N) = N − 1.

Puisque N est public, φ(N) est immédiatement connu : il suffit de calculer N − 1. On peut donc en déduire la clé privée d en calculant l'inverse modulaire de e modulo φ(N), sans avoir à factoriser quoi que ce soit. L'implémentation RSA est donc fondamentalement cassée.

4. Preuve que φ(p) = p − 1 pour un nombre premier p

Par définition, la fonction d'Euler φ(n) compte le nombre d'entiers k dans {1,…,n} qui sont premiers avec n : φ(n) = #{k ∈ {1,…,n} | gcd(k, n) = 1}.

Soit p un nombre premier. Les seuls diviseurs positifs de p sont 1 et p. Pour tout entier k tel que 1 ≤ k ≤ p − 1, p ne peut pas diviser k (p > k), donc le plus grand diviseur commun de k et p est 1, ce qui donne gcd(k, p) = 1.

En revanche, pour k = p, on a gcd(p, p) = p ≠ 1. Ainsi, les entiers copremiers avec p dans {1,…,p} sont exactement {1, 2, …, p − 1}. Il y en a p − 1, d'où φ(p) = p − 1.

Dans le challenge, N étant premier, on a donc φ(N) = N − 1.

5. Justification des formules de la clé privée et du déchiffrement

5.1 Pourquoi d = e^{-1} mod φ(N) ?

On impose par définition e·d ≡ 1 (mod φ(N)). Cela signifie qu'il existe un entier k tel que e·d = 1 + k·φ(N). Le nombre d est donc l'inverse modulaire de e modulo φ(N). En Python, la fonction pow(e, -1, φ(N)) calcule précisément cet inverse par l'algorithme d'Euclide étendu.

5.2 Pourquoi m = c^d mod N redonne le message clair ?

On part de c ≡ m^e (mod N) et de la relation e·d = 1 + k·φ(N). On obtient alors :

c^d ≡ (m^e)^d = m^{e·d} = m^{1 + k·φ(N)} = m · (m^{φ(N)})^k (mod N).

Par le théorème d'Euler, si gcd(m, N) = 1, on a m^{φ(N)} ≡ 1 (mod N). On en déduit :

c^d ≡ m · 1^k ≡ m (mod N).

C'est exactement la formule de déchiffrement RSA. En pratique, on la réalise en Python via m = pow(c, d, N).

6. Démarche de résolution du challenge

À partir du fichier output.txt, on récupère les valeurs publiques N, e et c. La démarche pour retrouver le flag est la suivante :

1) Constater que N est premier (par construction du challenge).

2) Calculer φ(N) = N − 1.

3) Calculer la clé privée d = e^{-1} mod φ(N).

4) Déchiffrer le message : m = c^d mod N.

5) Convertir l'entier m en bytes, puis en texte ASCII pour obtenir le flag Star{...}.

7. Script de résolution

Le script suivant illustre l'attaque en Python (avec pycryptodome pour la conversion entier → bytes) :

#!/usr/bin/env python3

from Crypto.Util.number import long_to_bytes

Données publiques extraites de output.txt

N = 177525858978158236964351599640917801414725584912574214601605859362452371576024890042042992697635020680274025347540881914232449944299255293856086203063382527179350592165237483768142615415581520379633997730013970238457527950780852467565508882854136301981320036515393640534328007992627930776727789566639620066803

e = 65537

c = 108260622256457772967783136510011357489954161424060720931542882116755847952610484097893076283088571605243580797900858079451751302172304118890988304923226485473547438250196926994890162681714439377208728562140105159338756266538563702499360776039707885689013818151439367011180609240044436452146901111704401949110

```
def main():
    # 1) N est premier -> ϕ(N) = N - 1
    phi = N - 1
    # 2) d = e^{-1} mod ϕ(N)
    d = pow(e, -1, phi)
    # 3) Déchiffrement RSA
    m = pow(c, d, N)
    # 4) Conversion en texte
    flag_bytes = long_to_bytes(m)
    flag = flag_bytes.decode(errors="replace")
    print("Flag :", flag)
if __name__ == "__main__":
    main()
```

8. Résultat

L'exécution du script de résolution restitue le flag suivant :

Star{F4ut_3c0ut3r_qu4nd_0n_p4rl3}

Ce flag rappelle qu'en cryptographie, il faut effectivement « écouter quand on parle » des hypothèses nécessaires à la sécurité d'un schéma, et ne pas simplifier à l'excès les conditions d'utilisation de RSA.

9. Morale et bonnes pratiques RSA

Ce défi montre qu'utiliser un « grand nombre premier » comme module RSA est une erreur fatale. Si N est premier, alors φ(N) = N − 1 est trivialement connu et la clé privée devient calculable par tout attaquant.

Pour un RSA correct en pratique, il faut notamment :

- • Utiliser un module N = p·q avec p et q grands, distincts et secrets.
- • Choisir un exposant public e adapté (par exemple 65537) avec gcd(e, φ(N)) = 1.
- • Protéger la génération et le stockage des clés privées (d, p, q).
- • Utiliser un padding sûr (OAEP, PSS, etc.) en fonction de l'usage (chiffrement, signature).

Le challenge « Presque ça… » est donc un exemple pédagogique de ce qu'il ne faut pas faire : réduire RSA à « un grand nombre premier et une exponentiation modulaire », sans respecter les hypothèses structurelles sur le module.