2024-09-18

Write-Up

Rivest Shamir Adleman

Contents

1 Chall informations 2
1.1 Script 2
1.2 Known infos 3
2 Analyse 4
2.1 Key Points 4
3 How to solve 5
3.1 Recovering the Flag 5
3.2 Solve script 6
3.3 Explanation 7
4 Conclusion 8

Ectario

1 Chall informations

1.1 Script

From the `chall.py`:

```python
from Crypto.Util.number import *
from flag import FLAG

FLAG = bytes_to_long(FLAG)
e = 11
nbits = 2048
p = getPrime(nbits)
q = getPrime(nbits)
n = p*q

ct = pow(FLAG, e, n)

print(f"{ct=}")
```

1.2 Known infos

Goal: The flag has been encrypted using RSA with an exponent $e = 11$. Our goal is to decrypt the ciphertext, but given the ciphertext size, we need to consider a specific attack vector.

Informations from `chall.py`:

```
e = 11
n = p*q                 # where p and q are 2048-bit primes
ct = FLAG**11 % n
```

The challenge involves RSA encryption where the modulus n is approximately 4096 bits. The encrypted flag is $\text{FLAG}^{11} \bmod n$, but the ciphertext length gives a critical clue for solving this challenge.

2 Analyse

2.1 Key Points

1. Ciphertext Length: The ciphertext ct is significantly smaller than expected—about 2351 bits, while the modulus n is around 4096 bits. This difference in size suggests something important: the value of FLAG^{11} is smaller than n, meaning the encryption did not "wrap around" modulo n.

2. Unlikely Probability: If $\text{FLAG}^{11} \bmod n$ were truly wrapped, the high bits of the ciphertext would not be zero. The fact that the ciphertext is only 2351 bits implies that the most significant 1745 bits of the result are zero, which is extremely improbable in a random RSA scenario. The probability of this occurring, assuming uniform distribution, is roughly $\left(\frac{1}{2}\right)^{1745}$, which is virtually impossible. This strongly suggests that $\text{FLAG}^{11} < n$, meaning that the encryption did not involve modular reduction.

3. Direct Root Extraction: Since $\text{FLAG}^{11} < n$, the ciphertext is simply FLAG^{11}, and no modular arithmetic has affected the result. This allows us to directly compute the 11th root of the ciphertext to recover the flag.

3 How to solve

3.1 Recovering the Flag

To solve this challenge, we just need to compute the 11th root of the ciphertext:

1. Given that $\mathrm{FLAG}^{11} < n$, we know that no wrapping occurred during the encryption, so the ciphertext is simply FLAG^{11}.

2. By computing the 11th root of the ciphertext, we can recover the original value of the flag, as $\mathrm{FLAG} = \sqrt[11]{\mathrm{ct}}$.

3.2 Solve script

```
from Crypto.Util.number import *
from libnum import nroot

e = 11
ct = int(open("output.txt", "r").read().strip().split("=")[1])

assert ct.bit_length() < 3000

print(long_to_bytes(nroot(ct, e)))
```

3.3 Explanation

Ciphertext Size: The key insight is that the ciphertext size is much smaller than the modulus n, suggesting that $\mathrm{FLAG}^{11} < n$. This allows us to avoid dealing with the modulus and directly compute the 11th root of the ciphertext to retrieve the flag.

Root Calculation: By using the function `nroot(ct, 11)`, we can efficiently compute the 11th root of the ciphertext and convert the result back into bytes to retrieve the flag.

4 Conclusion

This challenge illustrates a case where the RSA encryption process did not involve modular wrapping. By carefully analyzing the size of the ciphertext and understanding the implications, we were able to directly extract the flag by computing the 11th root of the ciphertext without needing to factor n or perform more complex RSA attacks.