D[0x]

Devel R0X

Criptografía RSA en CTFs

Daniel Espinoza
División de Ciberseguridad
Consultant/Advocate/Cryptographer

D[0x] ¿Por qué esta charla?/._

D[0x] Agenda/._

La matemática
detrás RSA

D[0x] Agenda/._

La matemática
detrás RSA

Ejemplo de
vulnerabilidad en RSA

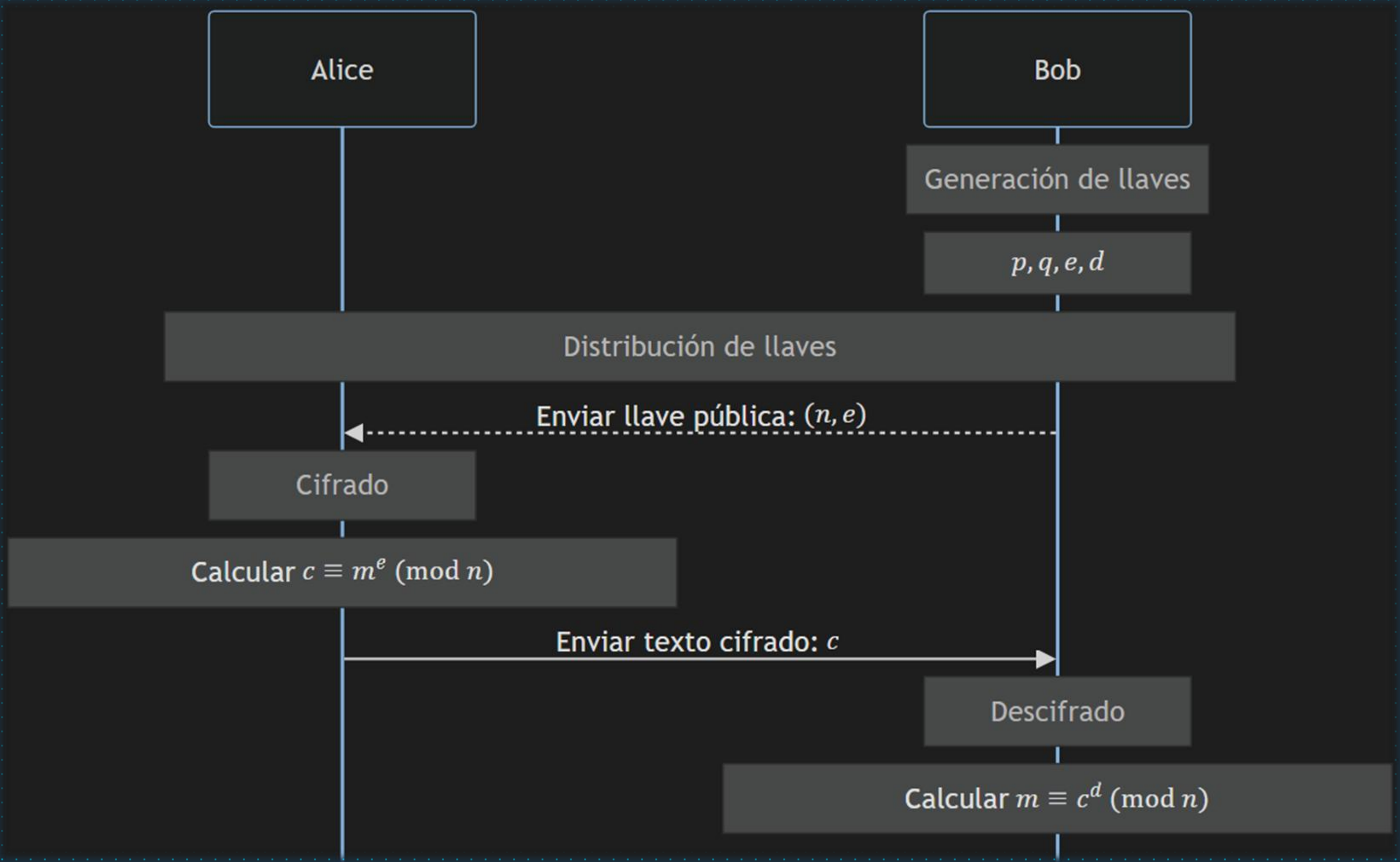
D[0x] Agenda/._

La matemática detrás RSA

Ejemplo de vulnerabilidad en RSA

Consejos y conclusiones

D[0x] Rivest-Shamir-Adleman (RSA): Confidencialidad



D[0x] Rivest-Shamir-Adleman (RSA): Cifrado

Mensaje encriptado

Exponente público

$$c \equiv m^e \pmod{n}$$

Mensaje

Módulo público

D[0x] Rivest-Shamir-Adleman (RSA): Cifrado (Ejemplo)

$$m = 5, e = 3, n = 33$$

Rivest-Shamir-Adleman (RSA): Cifrado (Ejemplo)

$$m = 5, e = 3, n = 33$$

$$m^e = 5^3 = 125$$

 Rivest-Shamir-Adleman (RSA): Cifrado (Ejemplo)

$$m = 5, e = 3, n = 33$$

$$m^e = 5^3 = 125$$

$$c = 125 \ \% \ 33$$

$$m = 5, e = 3, n = 33$$

$$m^e = 5^3 = 125$$

$$c = 125 \ \% \ 33$$

$$c = 26$$

D[0x] Rivest-Shamir-Adleman (RSA): Descifrado

Exponente privado

Mensaje

$$m \equiv c^d \pmod{n}$$

Mensaje encriptado

Módulo público

 Rivest-Shamir-Adleman (RSA): Descifrado (Ejemplo)

$$c = 26, d = 7, n = 33$$

D[0x] Rivest-Shamir-Adleman (RSA): Descifrado (Ejemplo)

$$c = 26, d = 7, n = 33$$

$$c^d = 26^7 = 8031810176$$

D[0x] Rivest-Shamir-Adleman (RSA): Descifrado (Ejemplo)

$$c = 26, d = 7, n = 33$$

$$c^d = 26^7 = 8031810176$$

$$m = 8031810176 \,\%\, 33$$

 Rivest-Shamir-Adleman (RSA): Descifrado (Ejemplo)

$$c = 26, d = 7, n = 33$$

$$c^d = 26^7 = 8031810176$$

$$m = 8031810176 \ \% \ 33$$

$$m = 5$$

$$m \equiv (m^e)^d \pmod{n}$$

$$n = p \cdot q$$

$$n = p \cdot q$$

$$\phi(n) = (p - 1) \cdot (q - 1)$$

$$n = p \cdot q$$

$$\phi(n) = (p-1) \cdot (q-1)$$

$$d \equiv e^{-1} \pmod{\phi(n)}$$

$$n = p \cdot q$$

$$n = 33$$

$$n = p \cdot q$$

$$n = 33$$

$$p = 3, q = 11$$

D[0x] Rivest-Shamir-Adleman (RSA): Seguridad

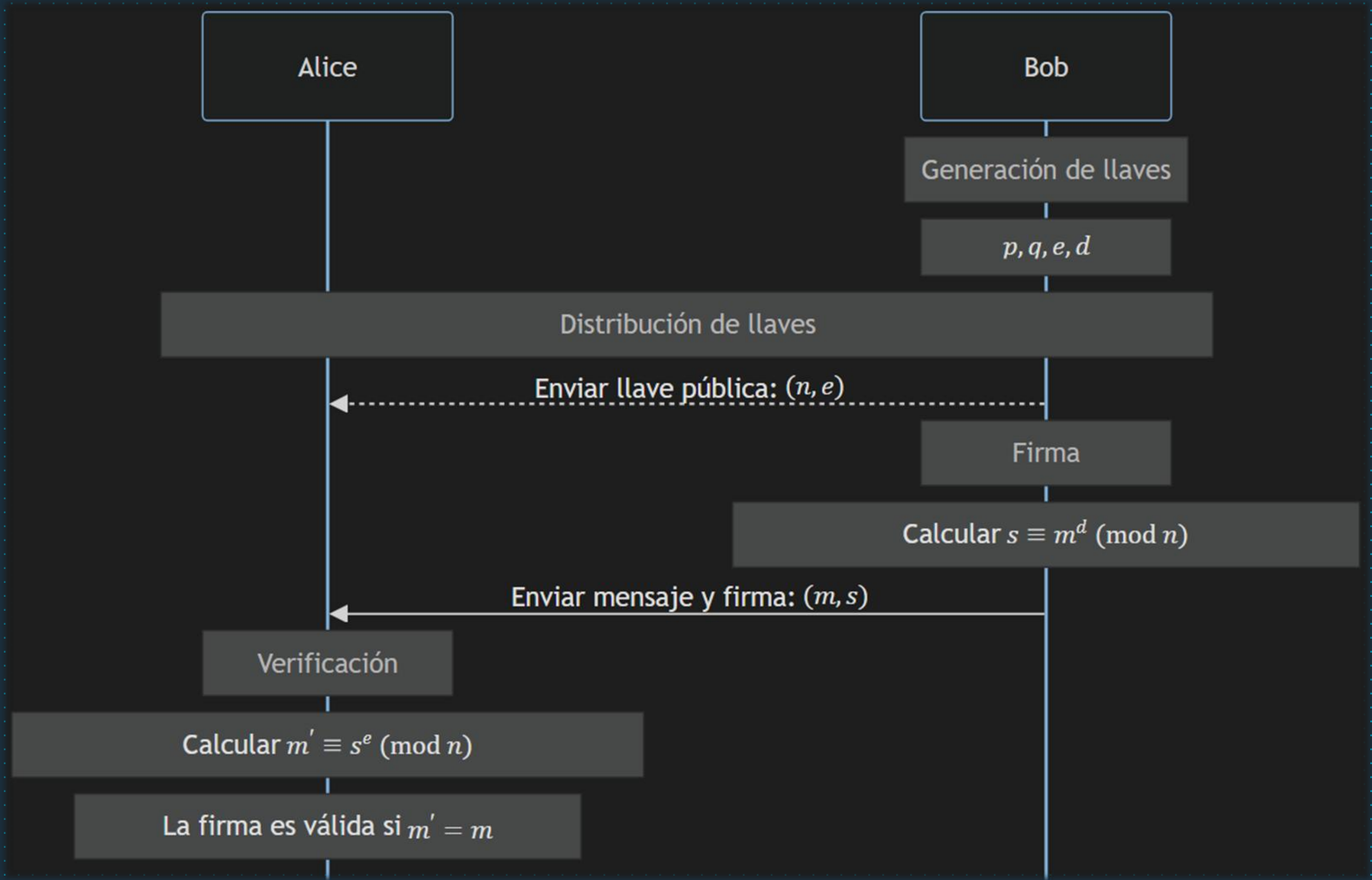
$$
\begin{aligned}
n = {} & 16189755987346577147637270807409027159945 \\
& 2394102879587698760299998868009534827363997854 15 \\
& 193591855231346146824436008737237497296466246331 \\
& 785684373965665561617411593886701558234340345997 \\
& 083102416929340687328489609077856704302187476168 \\
& 992491321784109236127556792457927035393435856362 \\
& 361835889793149976039916425088179916815502736207 \\
& 957854128176756832870793198051387546611387769139 \\
& 247996820809993358936781329480391492111044444926 \\
& 647900347095210097618435446771695937725167992183 \\
& 568477542258892472791281446776405498445354254167 \\
& 917884002118754959121077634096463876343935811646 \\
& 548565870626230476122169197163992324309941125821
\end{aligned}
$$

2048 bits para n, es decir, 1024 bits para p y q

2048 bits para n, es decir, 1024 bits para p y q

$$e = 0x10001 = 65537$$

D[0x] Rivest-Shamir-Adleman (RSA): Autenticación



D[0x] Rivest-Shamir-Adleman (RSA): Firmar

Firma digital

$$s \equiv m^d \pmod{n}$$

$$m' \equiv s^e \pmod{n}$$

 Rivest-Shamir-Adleman (RSA): Verificar

$$m' \equiv s^e \pmod{n}$$

- Si $m' = m$, la firma es válida

$$m' \equiv s^e \pmod{n}$$

- Si $m' = m$, la firma es válida
- Si $m' \neq m$, la firma no es válida

$$s_p \equiv m^{d_p} \pmod{p}$$

$$s_p \equiv m^{d_p} \pmod{p}$$

$$s_q \equiv m^{d_q} \pmod{q}$$

$$s_p \equiv m^{d_p} \pmod{p}$$

$$s_q \equiv m^{d_q} \pmod{q}$$

$$s = s_q + h \cdot q$$

$$d_p = d \ \% \ (p - 1)$$

$$d_p = d \ \% \ (p - 1)$$

$$d_q = d \ \% \ (q - 1)$$

$$d_p = d \ \% \ (p - 1)$$

$$d_q = d \ \% \ (q - 1)$$

$$h \equiv q^{-1} \cdot (s_p - s_q) \pmod p$$

D[0x] Crypto/Small StEps (HTB CyberApocalypse CTF 2023)

```python
from Crypto.Util.number import getPrime, bytes_to_long

FLAG = b"HTB{???????????????}"
assert len(FLAG) == 20

class RSA:
    def __init__(self):
        self.q = getPrime(256)
        self.p = getPrime(256)
        self.n = self.q * self.p
        self.e = 3

    def encrypt(self, plaintext):
        plaintext = bytes_to_long(plaintext)
        return pow(plaintext, self.e, self.n)

def menu():
    print('[E]ncrypt the flag.')
    print('[A]bort training.\n')
    return input('> ').upper()[0]
```

```python
def main():
    print('This is the second level of training.\n')
    while True:
        rsa = RSA()
        choice = menu()

        if choice == 'E':
            encrypted_flag = rsa.encrypt(FLAG)
            print(f'\nThe public key is:\n\nN: {rsa.n}\ne: {rsa.e}\n')
            print(f'The encrypted flag is: {encrypted_flag}\n')
        elif choice == 'A':
            print('\nGoodbye\n')
            exit(-1)
        else:
            print('\nInvalid choice!\n')
            exit(-1)

if __name__ == '__main__':
    main()
```

Ref: 7Rocky

```python
def main():
    print('This is the second level of training.\n')
    while True:
        rsa = RSA()
        choice = menu()

        if choice == 'E':
            encrypted_flag = rsa.encrypt(FLAG)
            print(f'\nThe public key is:\n\nN: {rsa.n}\ne: {rsa.e}\n')
            print(f'The encrypted flag is: {encrypted_flag}\n')
        elif choice == 'A':
            print('\nGoodbye\n')
            exit(-1)
        else:
            print('\nInvalid choice!\n')
            exit(-1)

if __name__ == '__main__':
    main()
```

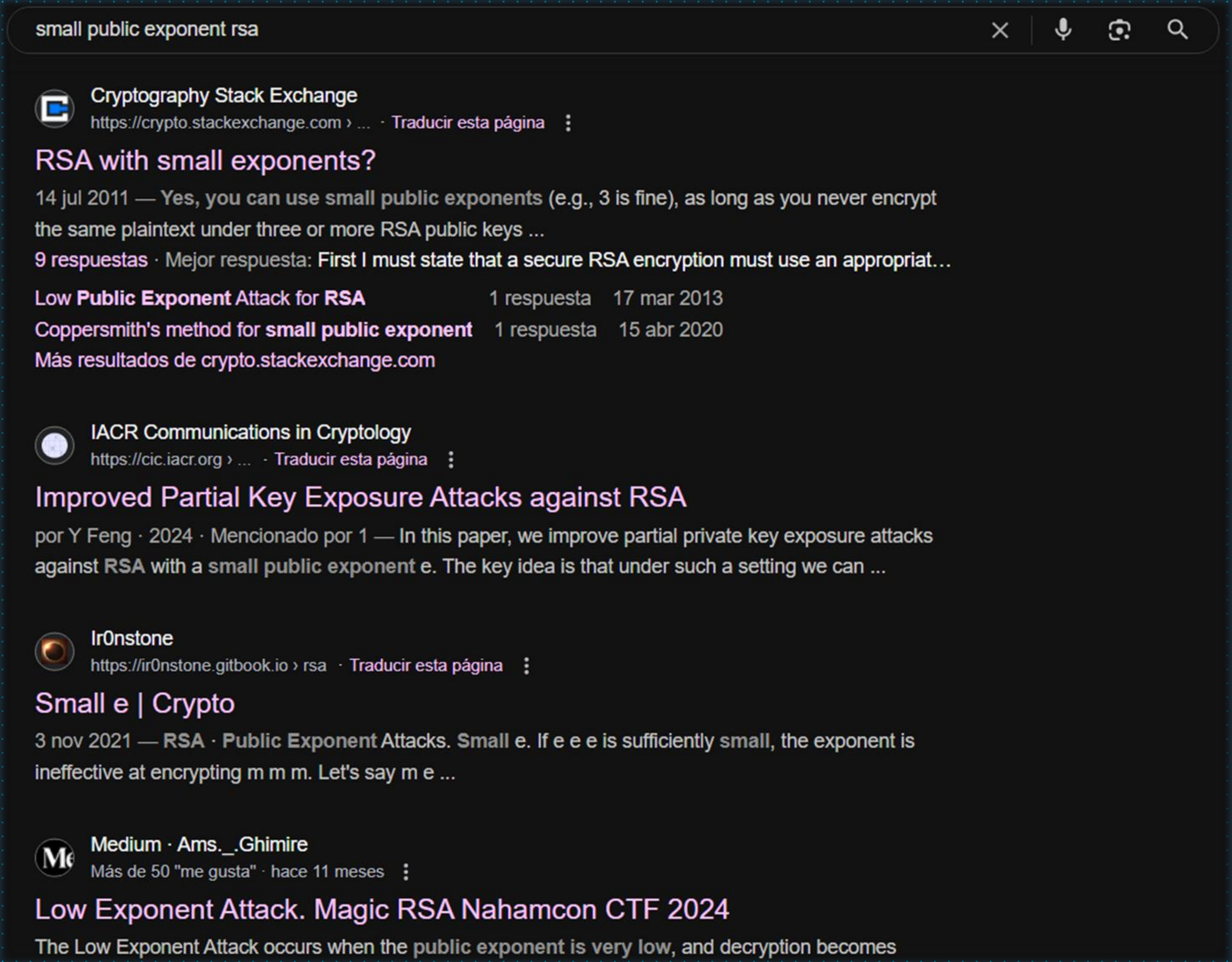
```
$ nc 188.166.152.84 32213
This is the second level of training.

[E]ncrypt the flag.
[A]bort training.

> E

The public key is:

N:
780267078770898233881040126527082163217777480112116907048841806630485
118812004339764274248824041906946095908012487180348656284995480572920
1222419964291931
e: 3

The encrypted flag is:
704073366705359338196741042088902542400637815384603946629989028609523
664391764674479477376809522776373305238189621046855532504025129898978
86053
```

Ref: 7Rocky

Detección de “cosas extrañas”

```python
from Crypto.Util.number import getPrime, bytes_to_long

FLAG = b"HTB{???????????????}"
assert len(FLAG) == 20

class RSA:
    def __init__(self):
        self.q = getPrime(256)
        self.p = getPrime(256)
        self.n = self.q * self.p
        self.e = 3

    def encrypt(self, plaintext):
        plaintext = bytes_to_long(plaintext)
        return pow(plaintext, self.e, self.n)

def menu():
    print('[E]ncrypt the flag.')
    print('[A]bort training.\n')
    return input('> ').upper()[0]
```

D[0x] Búsqueda (“small public exponent rsa”)



D[0x] Búsqueda ("small public exponent rsa")

RSA > PUBLIC EXPONENT ATTACKS

Small e

If e is sufficiently small, the exponent is ineffective at encrypting m.

Let's say $m^e < N$; in this case, we can simply take the eth root of c. For example, if $e = 3$, then we can calculate $m = \sqrt[3]{c}$.

If $m^e > N$ then this is a *bit* more secure, but we can progressively add more multiples of N until the cube root gives us a valid answer:

$$m = \sqrt[3]{c + kn}$$

Python

In Python we can use the `gmpy3` `iroot` function:

```
from gmpy2 import iroot

m = iroot(ct, e)
```

Low Exponent Attack

Ams._Ghimire Follow 4 min read · May 29, 2024

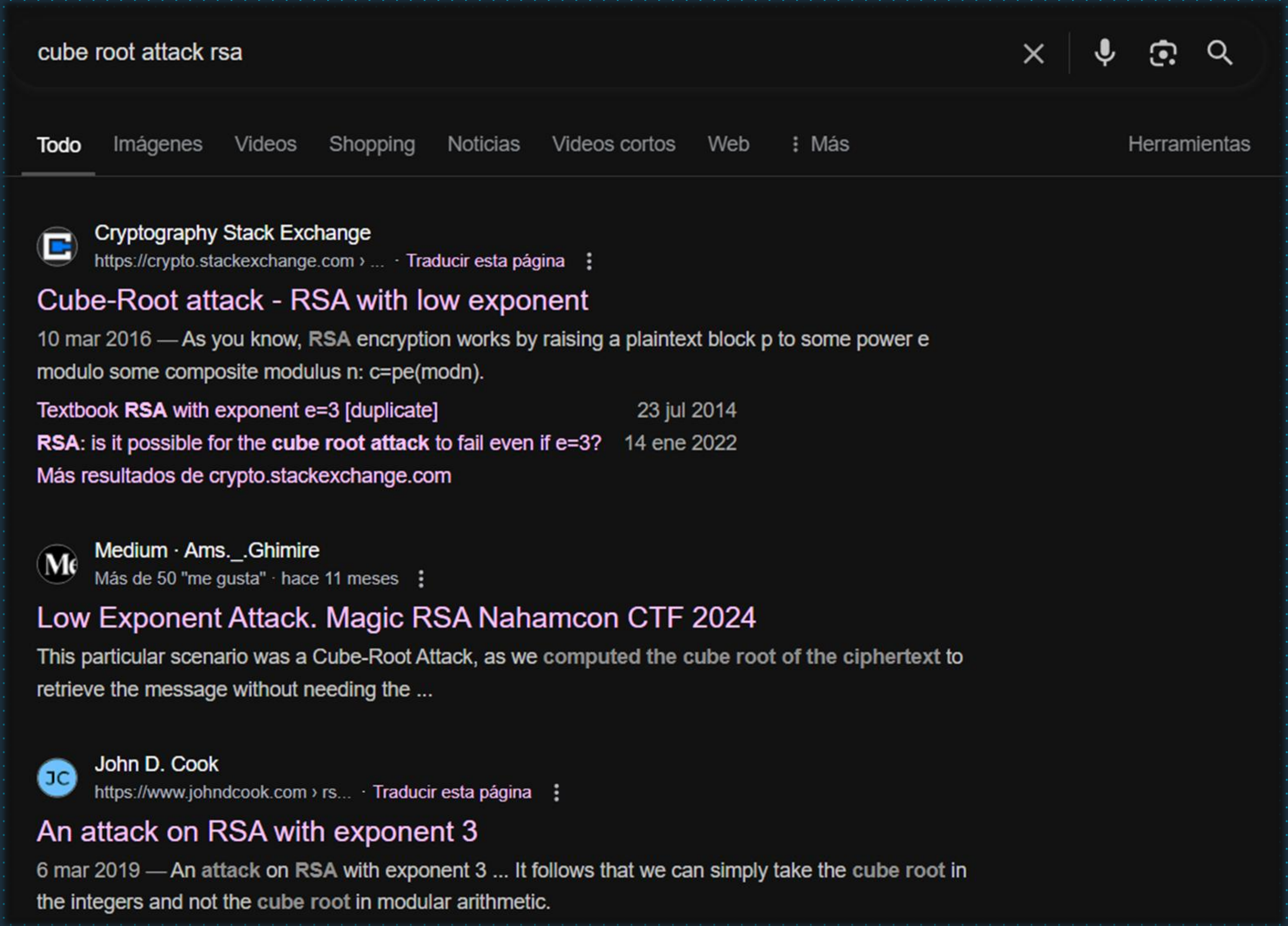
50 1

Magic RSA Nahamcon CTF 2024

INTRODUCTION

In this blog, we will be discussing about the RSA cryptosystem and a flaw in its implementation that arises when the value of the exponent is set very low. This attack is referred to as the **Low Exponent Attack** or the **Cube-Root Attack.**

D[0x] Búsqueda ("cube root attack rsa")



$$c \equiv m^3 \pmod{n}$$

$$c \equiv m^3 \pmod{n}$$

Si $m^3 < n$, entonces el módulo no se aplica!

$$c = m^3$$

$$c = m^3$$

$$m = \sqrt[3]{c}$$

D[0x] Implementación del ataque

```
$ python3 -q
>>> from gmpy2 import iroot
>>>
>>> c = 
7040733667053593381967410420889025424006378153846039466299890286095236643917646744794773768095227763733052381896210468555325040251298989788605
3
>>> e = 3
>>>
>>> m = iroot(c, e)[0]
>>> bytes.fromhex(hex(m)[2:])
b'HTB{5ma1l_E-xp0n3nt}'
```

Ref: 7Rocky

Búsqueda: artículos científicos

3 Low Private Exponent

To reduce decryption time (or signature-generation time), one may wish to use a small value of d rather than a random d. Since modular exponentiation takes time linear in $\log_2 d$, a small d can improve performance by at least a factor of 10 (for a 1024 bit modulus). Unfortunately, a clever attack due to M. Wiener [22] shows that a small d results in a total break of the cryptosystem.

Theorem 2 (M. Wiener) *Let $N = pq$ with $q < p < 2q$. Let $d < \frac{1}{3}N^{1/4}$. Given $\langle N, e\rangle$ with $ed = 1 \bmod \varphi(N)$, Marvin can efficiently recover d.*

Proof The proof is based on approximations using continued fractions. Since $ed = 1 \bmod \varphi(N)$, there exists a k such that $ed - k\varphi(N) = 1$. Therefore,

$$\left|\frac{e}{\varphi(N)} - \frac{k}{d}\right| = \frac{1}{d\varphi(N)}.$$

Hence, $\frac{k}{d}$ is an approximation of $\frac{e}{\varphi(N)}$. Although Marvin does not know $\varphi(N)$, he may use N to approximate it. Indeed, since $\varphi(N) = N-p-q+1$ and $p+q-1 < 3\sqrt{N}$, we have $|N-\varphi(N)| < 3\sqrt{N}$.

4

Ref: Twenty Years of Attacks on the RSA Cryptosystem

D[0x] Consejos y conclusiones

Buscar con Keywords según la lectura del código fuente. Utilizar variaciones para encontrar información relevante.

Leer artículos científicos sin miedo. Buscar sólo el resultado o ejemplo clave.

Pensar lateralmente. Factorizar directamente es el camino más difícil. Quizás exista información relevante que permita factorizar más fácil.

D[0x]

¡Mucho éxito!

¡Aprendan y disfruten de un nuevo Campo de Marte!

D[0x]

Colaborative Security: Hacking Together for a Stronger Defense._

www.develrox.com
info@develrox.com

DevelR0X