



I. Criptografía avanzada

Iván García y Hugo Leal

1. RC4

2. AES

3. RSA

CRIPTOGRAFÍA SIMÉTRICA

¿Qué es la criptografía simétrica?

Es un tipo de cifrado en el que se utiliza la misma clave tanto para cifrar como para descifrar un mensaje.

VENTAJAS

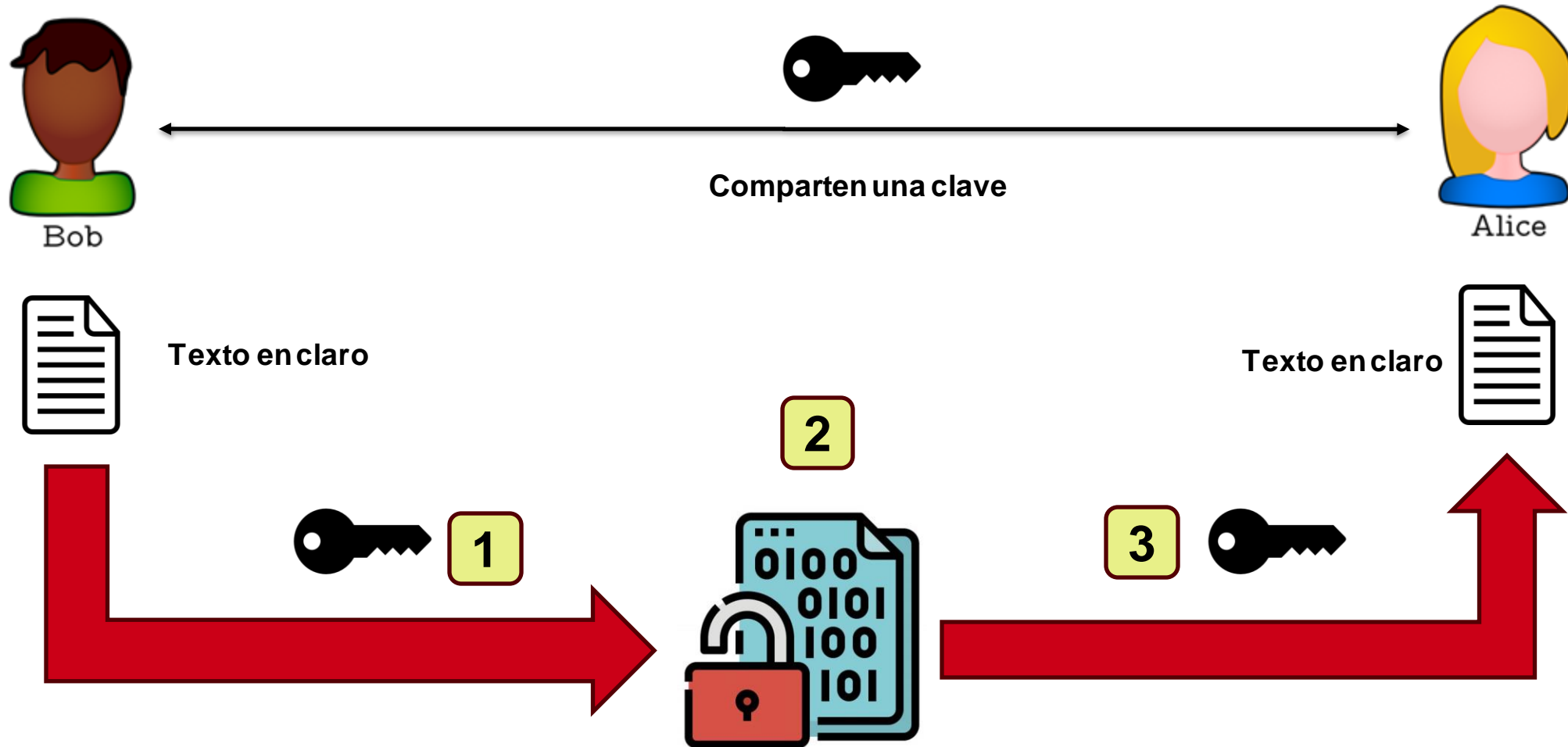
- Muy fácil de usar
- Muy útil
- Rápida y eficiente
- Segura

DESVENTAJAS

- ¿Cómo compartimos la clave?
- Demasiadas claves

CRIPTOGRAFÍA SIMÉTRICA

¿Cómo funciona?



¿Qué es?

- Algoritmo de cifrado
- No se recomienda su uso en la actualidad
- TLS y WEP
- Muy rápido y eficiente
- Adaptable a longitud de claves variable

¿Cómo funciona RC4?

Creamos el vector S

0	1	2	3	4
5	6	7	8	9
10	11	12	...	255

```
//Creamos el vector S con todos los valores de 0 a 255
for (int i = 0; i < 256; i++)
{
    s[i] = i;
}
```

¿Cómo funciona RC4?

Creamos el vector T

Clave = "Cl4v3Sup3rSegur4"

T

67	108	52	118	51	83	117	112	51	114	83	101	103	117	114	52
67	108	52	118	51	83	117	112	51	114	83	101	103	117	114	52
67	108	52	118	51	83	117	112	51	114	83	101	103	117	114	52

¿Cómo funciona RC4?

Creamos el vector T

```
for (int i = 0; i < 256; i++)  
{  
    t[i] = clave[i % (strlen(clave) - 1)];  
}
```


¿Cómo funciona RC4?

Algoritmo KSA

Cálculos

i	j	S[i]	T[j]	j
0	0	0	67	67
1	67	1	108	176
2	176	2	52	230

```

//Aplicamos el algoritmo KSA
j = 0;
int aux = 0;
for (int i = 0; i < 256; i++)
{
    j = (j + s[i] + t[i]) % 256;
    //Intercambiamos los dos valores
    aux = s[i];
    s[i] = s[j];
    s[j] = aux;
}
  
```

¿Cómo funciona RC4?

Algoritmo PRGA

Cálculos

K	i	j	S[i]	S[j]	h	S[h]
0	1	20	20	124	144	177
1	2	122	102	28	130	25
2	3	208	86	234	64	129

```

//Aplicamos el algoritmo PRGA
int i = 0;
int k = 0;
j = 0;
while (k < len)
{
    i = (i + 1) % 256;
    j = (j + s[i]) % 256;
    //Intercambiamos los valores
    aux = s[i];
    s[i] = s[j];
    s[j] = aux;
    h = (s[i] + s[j]) % 256;
    keyStream[k] = s[h];
    k++;
}
  
```

¿Cómo funciona RC4?

Hacemos el XOR

```
int *cifrado = malloc(sizeof(int) * len);  
for (i = 0; i < len; i++)  
{  
    cifrado[i] = keyStream[i] ^ input[i];  
}
```

Clave: Cl4v3Sup3rSegur4

Mensaje: Texto de prueba

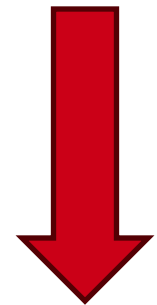


Mensaje cifrado: E5 7C F9 AD 15 AB 9E 18 89 A8
BB C7 44 5F F

Ataque de maleabilidad

Texto cifrado (URJC): 11011010 01000111 01111010 00101111



Texto cifrado': 11011010 01011101 01100001 00111110








DESCIFRAMOS

Mensaje descifrado: UHQR



RC4 en Cyberchef

Last build: 2 months ago - Version 10 is here! Read about the new features [here](#) Options  About / Support 



Recipe   

RC4  






Passphrase: contraseña UTF8 ▾ Input format: Latin1 Output format: Latin1

To Binary  





Delimiter: Space Byte Length: 8 ▾

To Hex  



Delimiter: Space Bytes per line: 0 ▾

Input     

URJQ|

Output    

|ÚGz/

STEP  **BAKE!**  Auto Bake

rec 4 1 Tr Raw Bytes ← LF

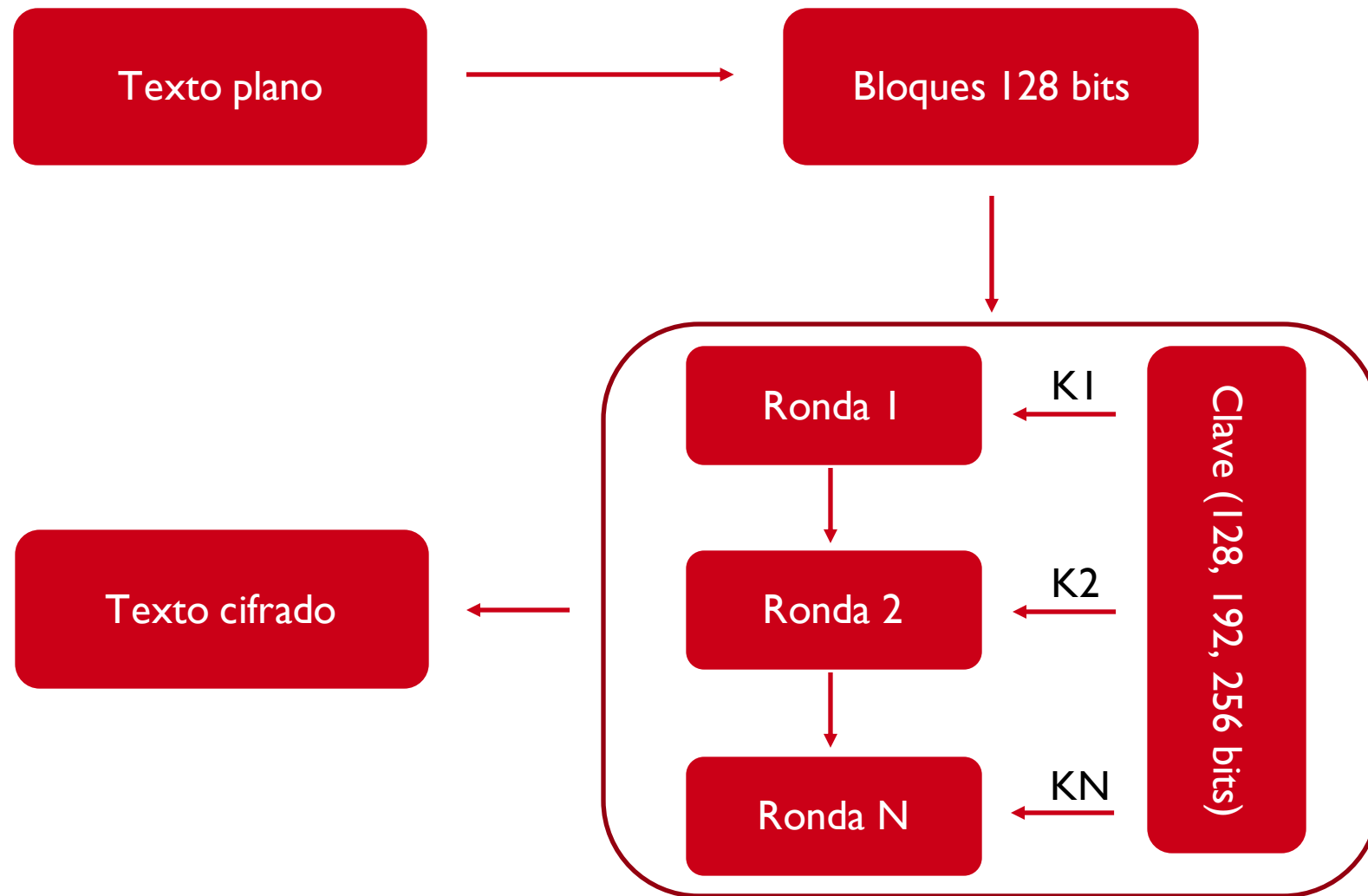
rec 4 1 0ms Tr Raw Bytes ← LF

AES – (Advanced Encryption Standard)

¿Qué es?

- Algoritmo de cifrado
- Se utiliza como estándar global de encriptación
- Se utiliza en aplicaciones como WhatsApp y Signal
- 128, 192 y 256 bits

¿Cómo funciona AES?



¿Cómo funciona AES?

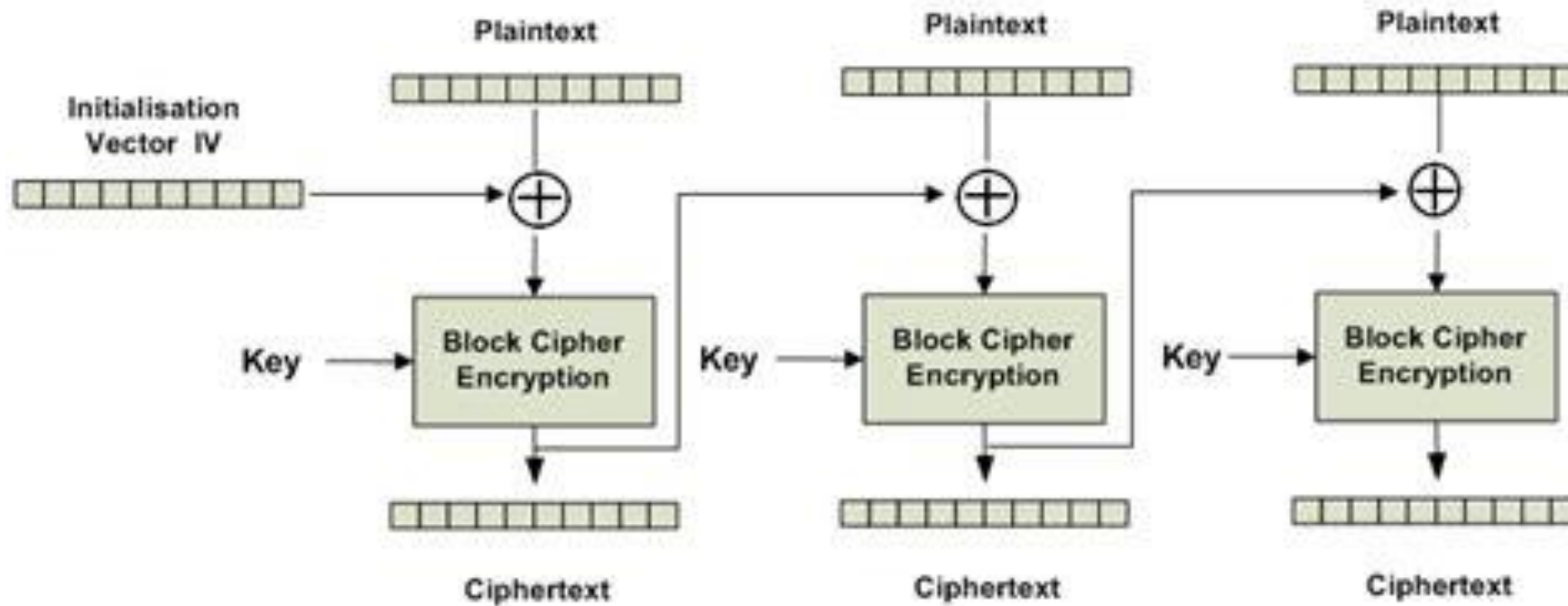
- En cada ronda, se calcula una **nueva clave** a partir de la **original**
- El **número de rondas** depende de la **longitud de la clave**

LONGITUD DE LA CLAVE (bits)	RONDAS
128	10
192	12
256	14

- Lo que ocurre en cada una de esas rondas, queda a vuestra curiosidad

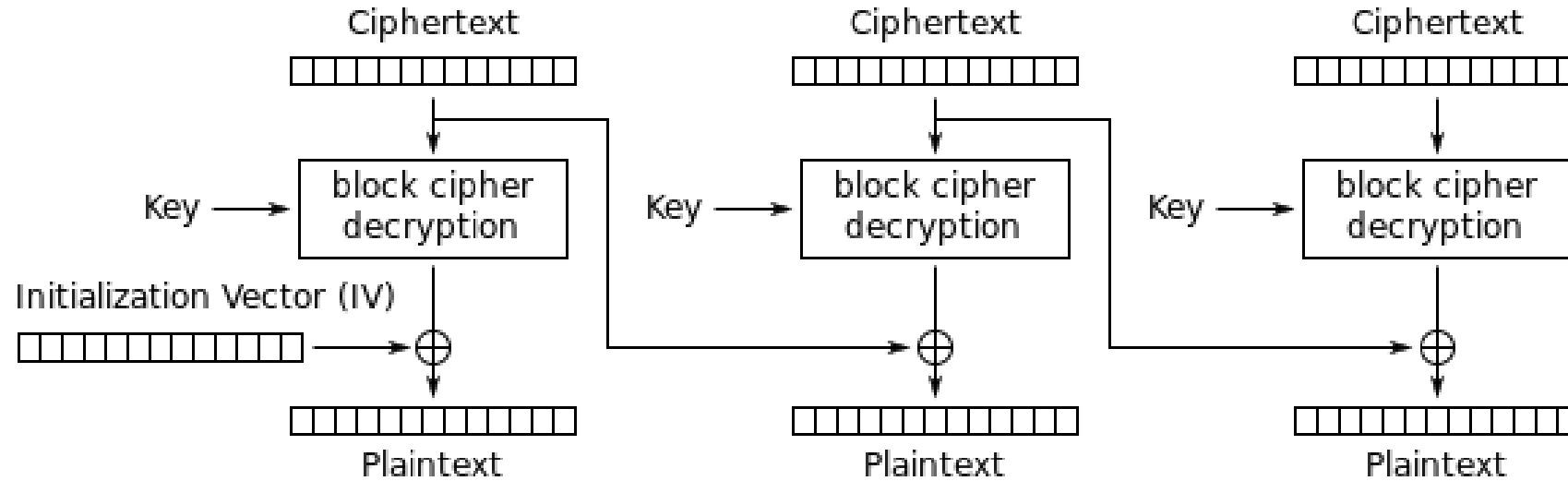
¿Qué es el modo CBC?

Cifrado:

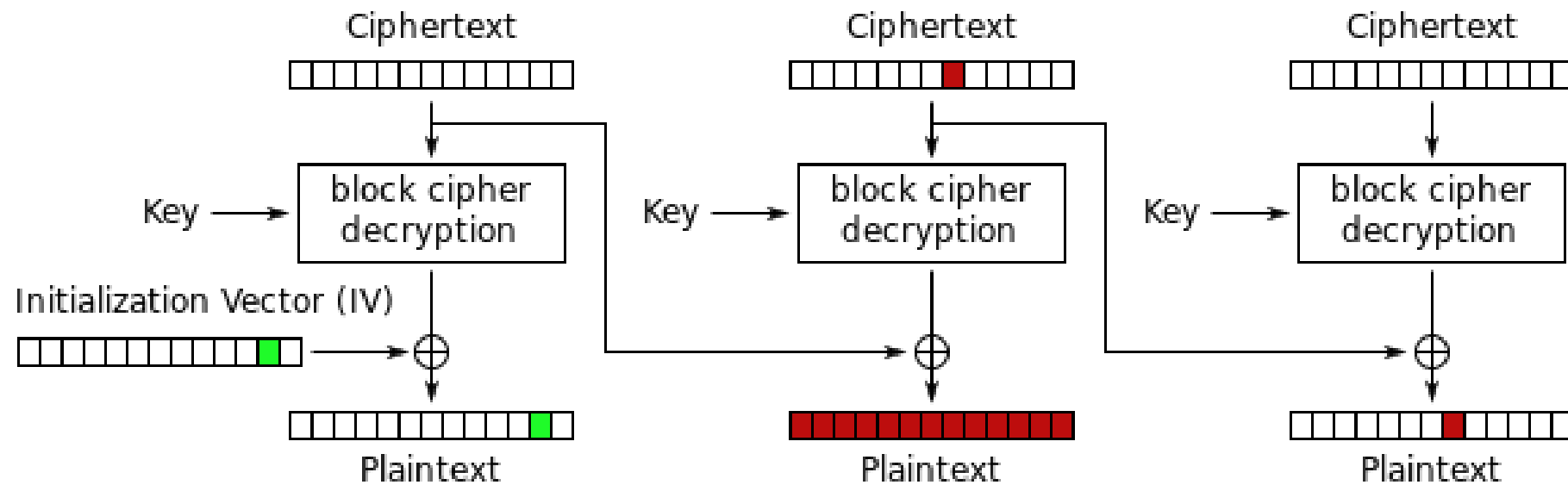


¿Qué es el modo CBC?

Descifrado:



Bit flipping



Bit flipping

$$C'_{i-1} = C_{i-1} \oplus x$$

$$P'_i = D_K(C_i) \oplus C'_{i-1}$$

$$P'_i = D_K(C_i) \oplus C_{i-1} \oplus x$$

$$P'_i = P_i \oplus x$$

$$\text{Let } x = P_i \oplus y$$

$$P'_i = P_i \oplus P_i \oplus y$$

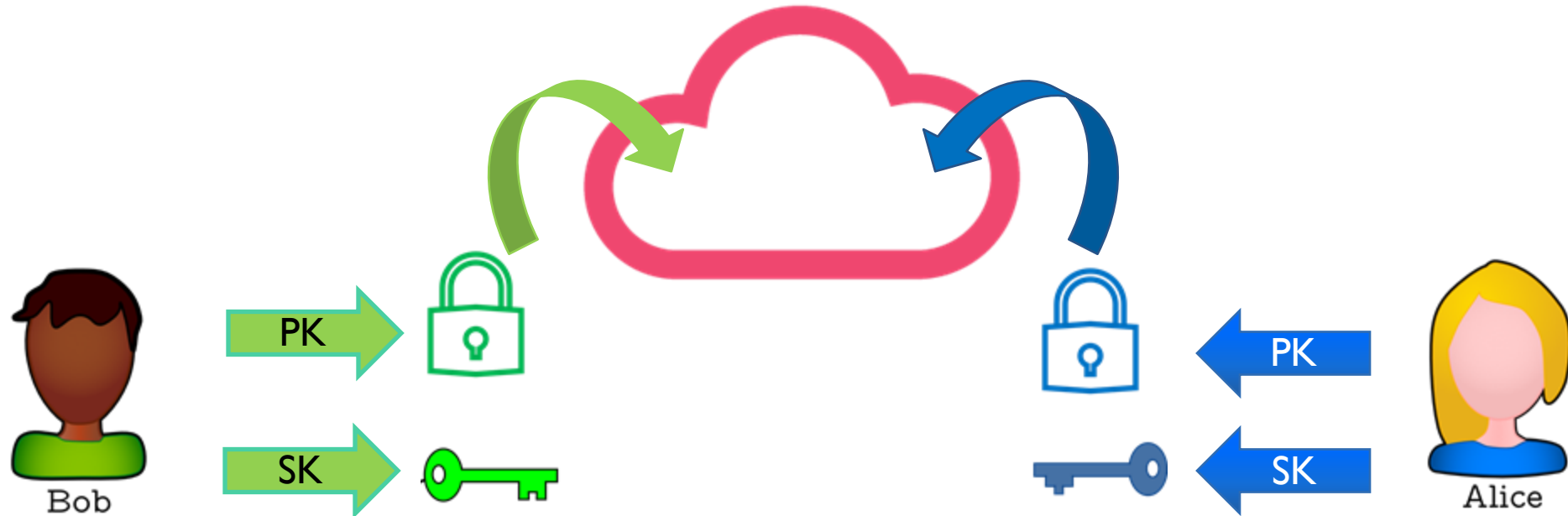
$$P'_i = y$$

CRIPTOGRAFÍA ASIMÉTRICA

¿Qué es la criptografía asimétrica?

Es un tipo de cifrado que utiliza una clave pública para cifrar y otra privada para descifrar.

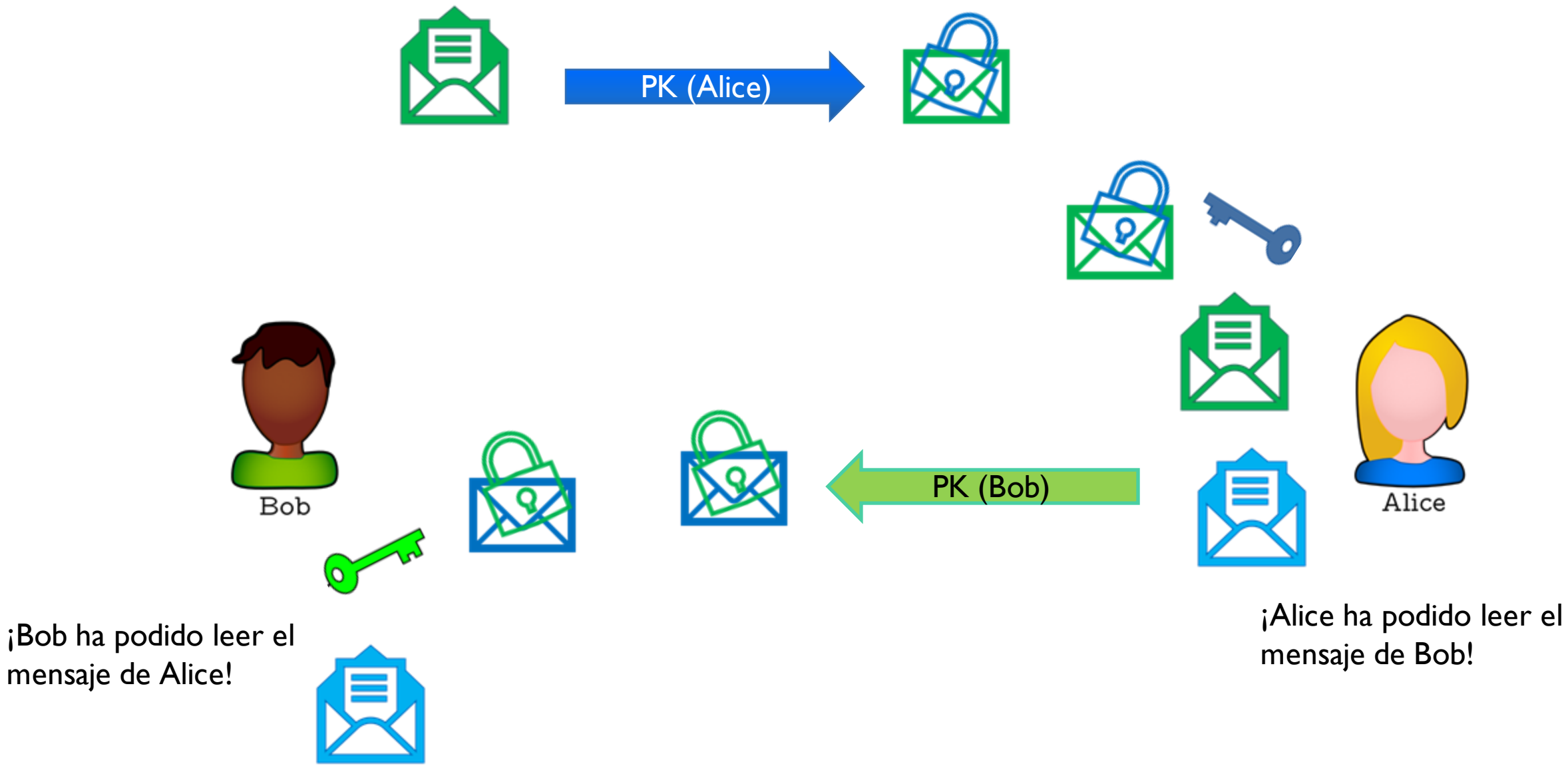
RSA o el Gamal



**Bob quiere mandar un mensaje seguro a Alice
pero no han acordado ninguna clave secreta
previamente**

¿Cómo lo hacen?

CRIPTOGRAFÍA ASIMÉTRICA



¡Bob ha podido leer el mensaje de Alice!

¡Alice ha podido leer el mensaje de Bob!

BASES DE LA ARITMÉTICA MODULAR

Modular/Clock Arithmetic



Modulus 12

$$a \equiv b \pmod{n}$$

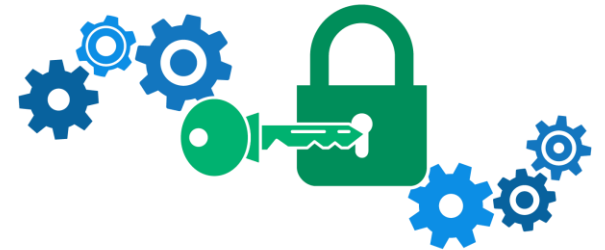
$$63 \equiv 83 \pmod{10}$$

$$\underline{64 \bmod 5 = 4}$$

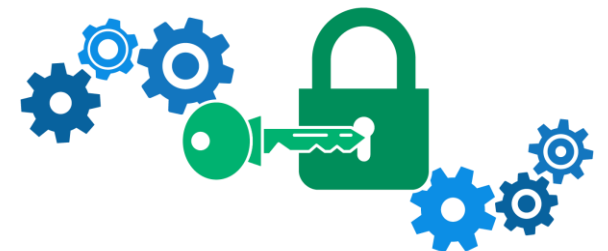
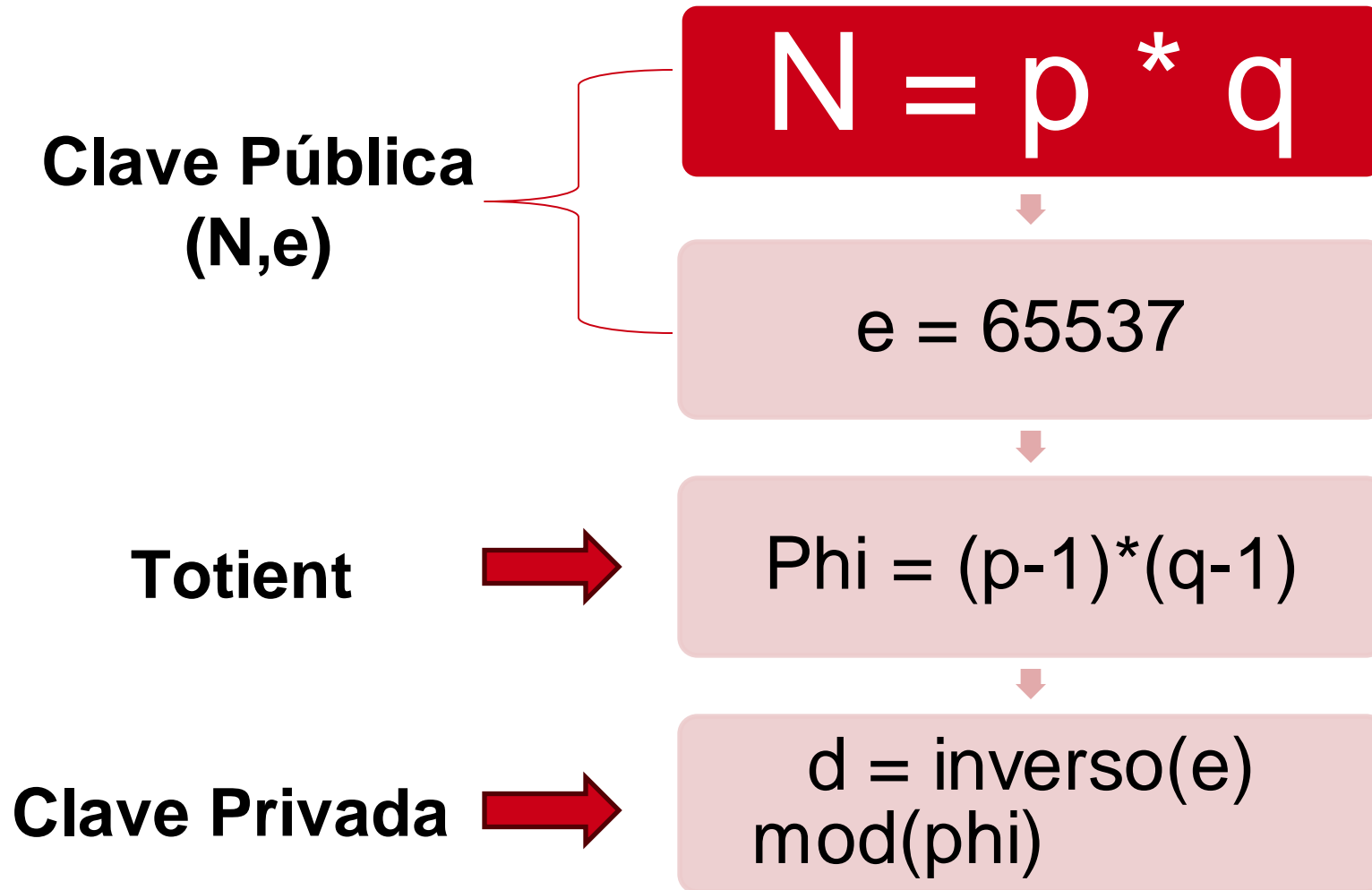
$$\begin{array}{r}
 64 \\
 14 \\
 4 \\
 \hline
 5 \\
 12
 \end{array}$$

Inverso: $a^{-1} * a = 1 \pmod{N}$

¡¡SOLO TIENE INVERSO SI $\text{GCD}(a,N) = 1$!!



PARÁMETROS RSA



RSA – Encriptar y Desencriptar

1. $\text{Texto_encriptado} = \text{mensaje}^e \pmod{N}$
2. $\text{Texto_desencriptado} = \text{Texto_encriptado}^d \pmod{N}$

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

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

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

Python Useful Functions

Instalar módulos extra de python --> `python3 -m pip install pycryptodome gmpy2`

```
pow(base,exponente,modulo) -->  $x^e \text{ mod } N$   
pow(base,-1,modulo) --> calcular el inverso de "base" modulo  
gmpy2.iroot(x,i) --> raiz i de x  
long_to_bytes(mensaje) --> transforma un numero grande en bytes  
bytes_to_long(mensaje) --> transforma un mensaje a un numero entero grande
```

RSA – Encriptar y Desencriptar

```
from Crypto.Util.number import bytes_to_long, long_to_bytes

p = getPrime(512) #Asigna un numero primo de 512 bits
q = getPrime(512)
N = p*q          #Se calcula el modulo
e = 3

#ENCRIPtar UN MENSAJE
mensaje = b"rsa es facil" #mensaje en bytes
mensaje_long = bytes_to_long(mensaje) # Se transforma el mensaje a un numero entero
mensaje_encrypted = pow(mensaje_long, e, N)
print(mensaje_encrypted)

#DESENCRIPtar UN MENSAJE
phi = (p-1)*(q-1) #Se calcula el totient
d = pow(e,-1,phi) #Se calcula el inverso de e modulo phi
mensaje_desencrypted = pow(mensaje_encrypted, d , N) #Calculas el valor original del mensaje
print(long_to_bytes(mensaje_desencrypted)) #Pasas el resultado a bytes para poder ser interpretado
```

Extraer parámetros de claves PEM:

```
-----BEGIN RSA PRIVATE KEY-----
Proc-Type: 4, ENCRYPTED
DEK-Info: AES-128-CBC,2AF25344B8391A25A9B318F3FD767D6D
```

```
kG0UYIcGyaxupjQqaS2e1HqbwRLlNctW2HfJeaKUjWZH4usiD9AtTnIKVU0pZN8
ad/StMWJ+MkQ5MnAMJglQeUbRxcBP6++Hh251jMcg8ygYcx1UMD03ZjaRuwcF0Y0
ShNbbx8Euvr2agjbF+ytimDyWhoJXU+UpTD58L+SIsZzal9U8f+Txbhg9K2KQHBE
6xaubNKhdJKs/6YJVEHTyYfYsbtYt4LsoAyM8w+pTPVa3LRWnGykVVR5g79b7lsJ
ZnEPK07fJk8JCdb0wPnlNy9LsyNxXRfV3tX4MRcj0XYZnG2Gv8KEIeIXzNiD5/Du
y8byJ/3I3/EsqHphIHGD3UfvHy9naXc/nLUup7s0+WAZ4AUx/MJnJV2nN8o69JyI
9z7V9E4q/aKCh/xpJmYLj7AmdVd4DL00ByVdy0SjKRXFaAiSVNQJY8hRHZSS7+k4
piC96HnJU+Z8+1XbvzR93Wd3kLRM07EesIQ5KKNNU8PpT+0lv/dEVEppVIDE/8h/
/U1cPvX9Aci0EUys3naB6pVW8i/IY9B6Dx6W4JnnSUFsyhR63WNusk9QgvtikH
40Znca5xHPij8hvUR2v5jGM/8bvr/7QtJFRcmMkYp7FMUB0sQ1NLhcjTTVAFN/AZ
fnWkJ5u+To0qzuPBWGPzsoZx5AbA4Xi00ppqkekeLali95mKKPecjUgpm+wsx8epb
9FtpP4aNR8LYlpKSDiiYzNiXEMQiJ9MSK9na10B5FFPsjr+yYefMyLPgogDpES80
X1VZ+N7S8ZP+7djB22vQ+/pUQap3PdXEpg3v6S4bfXkYKvFkcocqs8IivdK1+UFg
S33lgrCM4/ZjXYP2bpuE5v6dPq+hZvnmKkzcmT1C7YwK1XEyBan8flvIey/ur/4F
FnonsEl16TZvolSt9RH/19B7wfUHXXCyp9sG8iJGklZvteiJDG45A4eHhz8hxSzh
Th5w5guPynFv610HJ6wcNVz2MyJsmTyi8WuVxZs8wxrH9kEzXYD/GtPmcviGCexa
RTKYbgVn4WkJQYncyC0R1Gv308bEigX4SYKqIitMDnixjm6xU0URbnT1+8VdQH7Z
uhJVn1fzdRkZHWwLT+d+oqiIiSrvd6nWhttoJrjrAQ7YWGAm2MBdGA/MxLYJ9FNDr
1kxuSODQNGtGnWZPieLvDkwotqZkd0g7fimGRWiRv6yXo5ps3EJFuSU1fSCv2q2
XGdfc80bLC7s3KZwkYjG82tjMZU+P5PifJh6N0PqpxUCxDqAfY+RzcTcM/SLhS79
yPzCZH8uWIrjaNaZmDSPC/z+bWwJKuu4Y1GCXCqkWvwuaGmYeEnXD0xGupUchkrM
+4R21WQ+eSaULd2PDzLClmYrplnmpbD7C7/ee6KDTL7JmDV25DM9a16JYOneRtMt
qLNgzj0Na4ZNMMyRAHEL1SF8a72umG02xLWebDoYf5VSSSYtCNJdwt3LF7I8+adt
z0gLMmjr2L5c2HdlTut5MgiY8+qkHlsL6M91c4diJoEXVh+8YpblAoogOHHBlQe
KI1I1cqiDbVE/bmiERK+G4rqa0t7VQN6t2VWetWrGb+Ahw/imKhpITWLWApA3k9EN
-----END RSA PRIVATE KEY-----
```

```
openssl rsa -pubin -inform PEM -text -noout < public.key
```

```
openssl rsa -text -noout -in ~/.ssh/id_rsa
```

Modulus (2048 bit):

```
00:98:10:23:16:ff:b6:f4:26:a2:42:a6:19:23:0e:
0f:27:4a:b9:43:3d:a0:4b:b9:1b:1a:57:92:dd:a8:
bc:5d:b8:6e:e6:7f:0f:2e:89:a5:77:16:d1:cf:44:
69:74:2b:b1:a9:dd:72:bd:a8:9c:aa:90:ca:7b:f4:
d3:d3:db:11:98:bd:61:f1:2c:77:41:ad:c4:42:6a:
88:d1:37:04:12:a9:36:ec:09:34:0d:31:71:b9:5a:
ea:ed:ce:61:1c:1e:5f:6c:9e:28:ee:21:2a:e4:c6:
1f:75:29:78:a5:96:b1:53:17:4d:bf:88:d1:12:5c:
a6:75:aa:7c:fe:23:a8:dd:25:35:46:c6:8a:eb:2e:
e4:a3:1d:7f:b6:6d:9c:7d:66:59:84:c9:51:15:82:
67:a6:85:e9:c8:d6:2b:a7:e6:28:08:d2:b1:99:92:
67:32:c4:ba:f7:c9:1a:16:30:e5:cb:39:cb:96:28:
70:32:ba:18:d2:64:2f:74:3e:dd:09:e0:68:56:57:
cf:50:63:c0:95:a9:b0:5b:2a:ad:21:4f:bd:e7:15:
64:4a:9d:e4:c5:c3:5c:35:bf:e6:78:f4:8a:40:83:
da:7d:0d:6c:02:60:4a:3f:0c:9c:03:fd:48:e6:72:
f3:0d:5b:90:6b:de:59:58:c9:f4:26:4a:61:b4:52:
21:1d
```

Exponent: 65537 (0x10001)

RSA – Ataque Exponente pequeño y Modulo Grande

e \Rightarrow 3

N \Rightarrow

17920302547451456255260628601727787230172785014692541565373018303626923200144183
95774329698747061027147739143566410586761477690456773509856029277075469025914383
28530840229118555963282468638588243456874311075732609267271548495053483974016854
08690680392631146761855793479884844297557829348192912981737152331029681972235731
933154889782680523867681705997376633312277

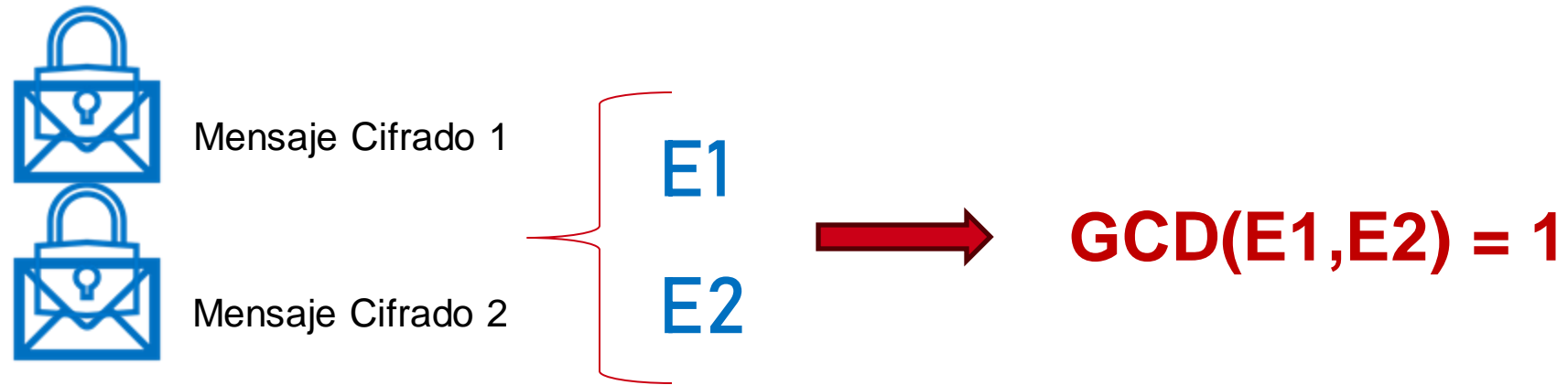
mens \Rightarrow

123920310321213321233213231212672136867326723673126732

mens^e \Rightarrow

190295043635636787334664801249623051718205871126260174796304068080172301486
402873771262184351748127806309044976187422625748234599636095254735855176834
3285695168

RSA – Diferente Exponente y Mismo Modulo



Si se cifra el mismo mensaje con distintos exponentes y con el mismo modulo se puede recuperar el mensaje original

https://asecuritysite.com/rsa/rsa_e2

RSA – Encriptar y Desencriptar

Recursos Útiles

- <http://factordb.com/>
- <https://aurea.es/demos/criptografia/pag/calculadoraRSA.html>
- <https://github.com/jvdsn/crypto-attacks>
- <https://asecuritysite.com/rsa/>
- [RSA Calculator \(tausquared.net\)](https://tausquared.net/)
- [RsaCtfTool: https://github.com/Ganapati/RsaCtfTool](https://github.com/Ganapati/RsaCtfTool)

Recursos de consulta y práctica

- CryptoHack. Retos de criptografía:

<https://cryptohack.org/challenges/>

- CrypTool. RSA paso a paso:

<https://www.cryptool.org/en/cto/rsa-step-by-step.html>

- Vídeo AES:

<https://www.youtube.com/watch?v=tzjIRoqRnv0>



I. Criptografía avanzada

Alumnos Ciberseguridad



Universidad
Rey Juan Carlos