

# Criptología

## Criptografía

Cifrado simétrico

Cifrado asimétrico

Protocolos

## Criptoanálisis

1 llave

2 llaves

usa 1 o de sim y asim.

## Propósitos:

- ① Confidencialidad: mantener en secreto la información
- ② Integridad de datos: procurar la NO alteración NO autorizada de la información
- ③ Autenticación: Si se da esto se tiene ② casi en automático
- ④ No repudio: Si A → B ent. A no puede negar haber enviado mensaje a B y B no puede negar haberlo recibido

Definición: alfabeto de definición, ej:  $A = \{0,1\}$

① M: espacio de mensajes, consiste de cadenas de símbolos de A.

A los elementos de M se les llama textos planos

③ L: espacio de texto cifrado, a los elementos de L se les llama textos cifrados

⑤ K: espacio de llaves, a los elementos de K se les llama llaves

⑤ Cada elemento  $e \in K$  determina de forma única una función  $E_e: M \rightarrow L$  de cifrado

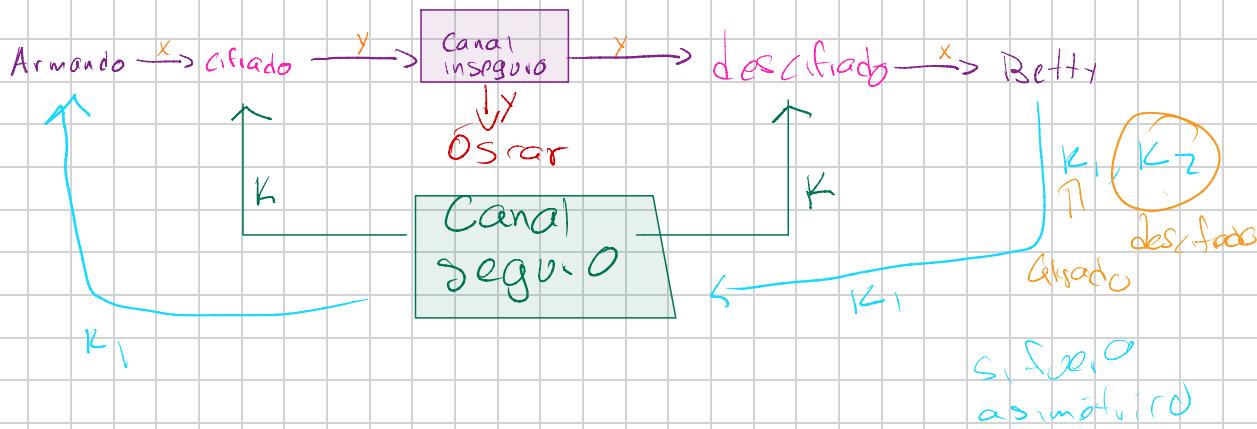
⑥ Para cada  $d \in K$ ,  $D_d: L \rightarrow M$ , función de descifrado

$$x \in M \Rightarrow D_d(E_d(x)) = x$$

$K \subseteq M$

Principio de Kerckhoff: Comprometer el sistema en cuanto a detalles no debe ser un inconveniente para su efectividad  
"conocer el funcionamiento pero no la llave"

## Criptografía Simétrica



## Ejemplos:

① Cifrado de corrimiento:  $\mathbb{Z}/27 \text{ } (\mathbb{Z}/m)$

$$A = K = \mathbb{Z}/27, \quad K \in K$$

$$e_K(x) = (x + K) \bmod 27 \quad x \in M \quad d_K(y) = (y - K) \bmod 27 \quad y \in G$$

Ej: el cifrado César

② Cifrado de sustitución:  $A = \mathbb{Z}/27, \quad K = S_{27}$  (posibles permutaciones)

$$\pi \in K, \quad e_\pi(x) = \pi(x), \quad d_\pi(y) = \pi^{-1}(y)$$

③ Vigenère:  $27 \boxed{\square} \rightarrow$  la llave tiene m columnas

L	L A V E L L A V E
M	M B N F
N	N C X G
↓	↓
S	S Y H
↓	↓

## Criptoanálisis

Criptoanálisis clásico:

Análisis matemático

Fuerza bruta

Ataque a la implementación

↑  
se necesita acceso físico al equipo

Ingeniería social

↑  
Factor humano

La seguridad es por ejemplo similar:

Simétrico 80 bits  $\sim$  RSA - 1024 bits

Llave (bits)

Tiempo (con fuerza bruta)

56 - 64

Horas / días

112 - 128

décadas (sin comp. cuántica)

256

décadas (con "")

Modelos de ataque:

\* Ataque de texto cifrado: solo conocemos un texto cifrado y

\* Ataque de texto plano conocido: se conoce un texto plano  $x$  y su correspondiente cifrado y

\* Ataque de texto plano elegido: escoger un texto plano  $x$  y constuir el resp. cifrado y (se tiene acceso temporal a una sucesión de cifrados)

\* Ataque de texto cifrado elegido: escoger un texto cifrado y construir el correspondiente plano  $x$  (se tiene acceso temporal a la sucesión de descifrados)

Objetivo: Obtener la llave  $K$ .

① Corrimiento: Fuerza bruta

② Sustitución: rankar la probabilidad de las letras

③ Vigenère: similar a ② pero ahora se toman ocurrencias de dos letras, 3 y así ...

④ Hill:  $M = G = (\mathbb{Z}/27)^m, \quad m \geq 2$

$$K = GL(m, \mathbb{Z}/27)$$

$$K \in K \quad e_K(x) = xK, \quad d_K(y) = yK^{-1}$$

Cae con ataques de texto plano conocido

Sop: el oponente conoce  $m$  y que tiene/conoce ciertos parámetros  $x_i, y_i$ :

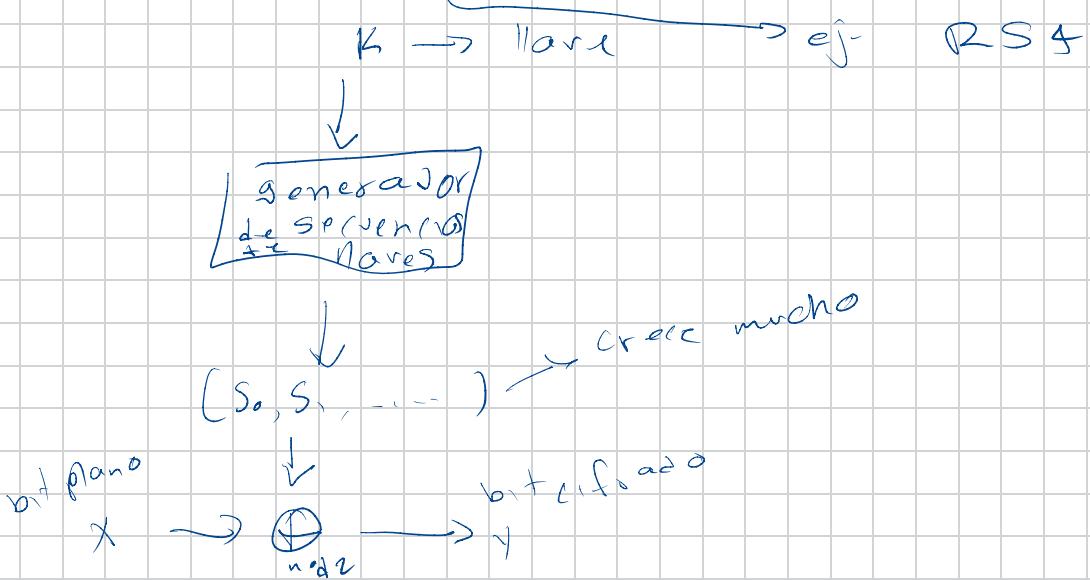
$X = (x_i), \quad y = (y_i), \quad Y = XK$  y sabemos  $X = d_K e_K(X) \Leftarrow$  de aquí sabemos (podemos saber si  $X$  es invertible y así  $X^{-1}Y = K$  y ya tenemos la llave.

24/09/24

## C. f. ados secuenciales

AES  $\rightarrow$  cifrado de bloques (128 bits)  
 DES, 3DES  $\rightarrow$  " (64 bits + 8)

La idea es una secuencia de llaves



1 byte tiene 8 bits (0, 1)

## Pseudo aleatorios

$A, B, M \in \mathbb{Z}$ , fijar semilla  $S_0 = \text{semilla}$

$$s_i = As_{i-1} + B \pmod{M}$$

rand()  $\rightarrow A = 1103515249$        $B = 12349$        $M = 2^{31}$       ) en un tipo C.

son predecibles en general, ya que a veces conociendo los primeros  $n_i$  se pueden conocer los sg.

Si son predecibles ya no sirven para la criptografía

15/10/24

## CSRG Cryptographically Secure Random Number Generator

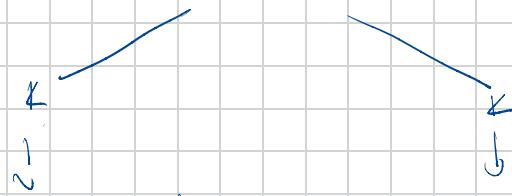
Dados  $N$  bits consecutivos no existe un algoritmo en tiempo polinomial que pueda predecir los siguientes  $m$  bits.

La seguridad de estos depende de la aleatoriedad

Dado un texto de  $x$  cantidad de bits, por ej. 8  $\rightarrow$  tiene 256 formas de cifrarse

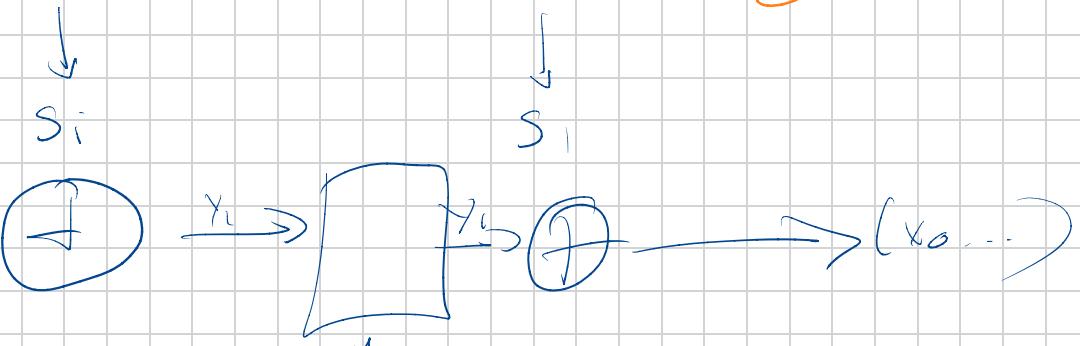
(cada carácter puede ser cifrado de dif. formas en el mismo texto y caracteres diferentes podrían cifrarse igual). Los ( $s_1, \dots, s_n$ ) deben ser suf. aleatorios para que funcione.

Llave inicial  $\rightarrow$  por mecanismo seguro



Generator de secuencia llave = Generator de secuencia llave

← aleatorio  
CSPRNG



y  
canal  
información

LFSR  $\rightarrow$  insendido por si sola

30/01/24

# The Data Encryption Standard DES

- Por más de 30 años ha sido el cifrado cipher más popular
- Ahora se considera inseguro ya que el espacio de llaves es muy pequeño
- Se sigue usando en algunas aplicaciones y más el 3DES que es usar 3 veces seguidas el cifrado
- ¿Por qué estudiarlo?
  - es el cifrado simétrico más estudiado
  - Y ha inspirado muchos otros cifrados cipher.

## 3.1 Introducción a DES

### \* 1972: Creación de un estándar de cifrado

- + La Oficina Nacional de Estandares de EU (NBS, ahora NIST) lanzó una convocatoria para un estándar de cifrado en EU.
- + Objetivo: encontrar un algoritmo seguro y único para diversas aplicaciones comerciales.

### \* Contexto previo:

- + La criptografía era considerada un asunto de seguridad nacional y era mantenida en secreto
- + La demanda comercial de usar cifrados, por ejemplo en los bancos, motivó el desarrollo de un estándar abierto

### \* 1974: Propuesta de IBM

- + IBM presenta un algoritmo basado en Lucifer, un cifrado de bloques desarrollado por Horst Feistel en los 60s.
- + Lucifer opera en bloques de 64 bits con claves de 128 bits.

### \* Revisión de la NSA:

- + Hasta entonces la NSA era secreta y no admitía su existencia.
- + El NBS solicitó la ayuda de la Agencia de Seguridad Nacional (NSA) para evaluar la propuesta del IBM.
- + La NSA influyó modificaciones al algoritmo de IBM, se renombró como DES.
- + Se dice que el DES fue diseñado para resistir ataques de criptanálisis diferencial, un ataque no conocido al público hasta 1990.
- + Presentemente, la NSA redujo la longitud de la llave de 128 bits a 56 bits, lo cual aumenta la vulnerabilidad ante ataques de fuerza bruta.

### \* Preocupaciones sobre la seguridad:

- + Se temía que la NSA incluyera una puerta secreta que permitiera romper el cifrado y que fuera una propiedad matemática que solo conociera la NSA.
- + Hubo críticas a la reducción del tamaño de la llave, argumentando que esto fue debido a que la NSA tenía la habilidad de buscar en el espacio de llaves de  $2^{56}$  y romper así el cifrado por fuerza bruta.

## \* 1977: Publicación del DES

- + A pesar de las críticas la NBS publicó el estándar como DES (FIPS PUB 46)
- + El diseño fue descrito hasta el nivel de bits, pero nunca se publicaron los criterios de diseño, como la elección de las cajas de sustitución (S-boxes).

## \* Análisis y adopción:

- + En los 80's, con el aumento de los computadores personales, el DES fue objeto de intenso escrutinio por la comunidad de criptografía civil
- + No se encontraron serias debilidades hasta 1990.

## \* Estándar temporal y reemplazo:

- + Originalmente el DES fue estandarizado por 10 años hasta 1987.
- + Debido a la falta de seguridad, el NIST reafirmó su uso hasta 1999 y fue reemplazado por el Advanced Encryption Standard (AES).

### 3.1.1 Confusión y Difusión.

Según Claude Shannon, hay dos operaciones primitivas que los cifrados deben considerar para ser fueramente seguros:

#### ① Confusión:

- + El objetivo es ocultar la relación entre la clave y el cifrado.
- + Logra que sea difícil para un atacante rastrear como la clave afectó al resultado.
- + Ejemplo: sustitución usada en DES y AES
- + Ejemplo: Sup. que tenemos el texto plano HELLO y queremos usar sustitución de caracteres para cambiar cada letra por otra según una tabla secreta.  
 $H \rightarrow M, E \rightarrow P, \dots$

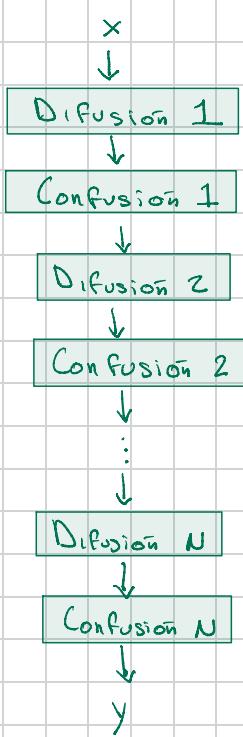
Si cambiamos la llave, todo el mapeo cambia y HELLO podría irse a cifrado completamente diferente como ZYXXO.

#### ② Difusión:

- + Busca disipar la influencia de un símbolo del texto plano en muchos símbolos del texto cifrado.
- + Objetivo es ocultar propiedades estadísticas del texto plano.
- + Ejemplo: Permutación de bits utilizada en DES y operación MixColumn en AES.
- + Despues de aplicar confusión, si usamos una permutación que reorganiza los caracteres o bits, si cambiamos la letra H por G en el texto original, el texto cifrado podría resultar: QRXYP en lugar de ZYXXO.
- + Un ligero cambio en el texto plano se difunde para afectar muchos elementos en el resultado final.

Importancia de combinar difusión y confusión:

- + Cifrar solo con confusión (ej. cifrado por desplazamiento) o solo con difusión no proporciona suficiente seguridad.
- + La combinación de estas dos operaciones conocida como cifrado de producto, ofrece un cifrado fuerte.
- + Los cifrados de bloques modernos son ejemplos de cifrados de producto, con rondas de operaciones aplicadas repetidamente.



Propiedades de difusión en los cifrados de bloques modernos

- + Cambiar un solo bit del texto plano genera un cambio promedio en la mitad de los bits del texto cifrado.
- + Por lo tanto el texto cifrado parece estadísticamente independiente del primero.

Ejemplo: asumamos un cifrado por bloques con bloques de 8 bits de long.

Encriptar dos textos que difieren en un bit:

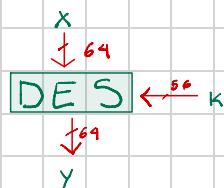
$$\begin{array}{l} X_1 = \underline{00101010} \rightarrow \boxed{\text{Cifrado de}} \\ X_2 = \underline{00001010} \quad \boxed{\text{bloque}} \end{array} \longrightarrow \begin{array}{l} Y_1 = 10111001 \\ Y_2 = 01101100 \end{array}$$

Nota: Los cifrados de bloques modernos tienen bloques de longitud de 64 o 128 bits y pasa lo mismo si se cambia un solo bit.

## 3.2 Un vistazo al algoritmo DES

### \* Bloques y clave:

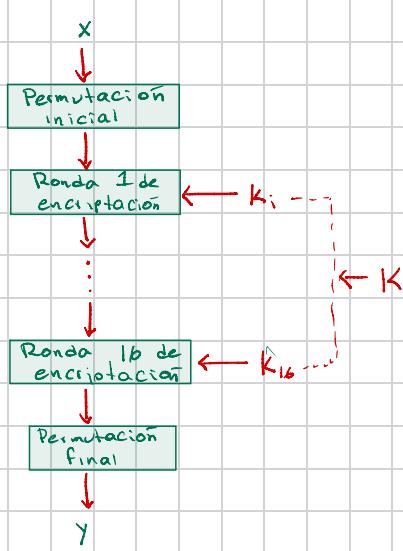
- DES es un cifrado que opera en bloques de 64 bits con una llave de tamaño 56 bits.



- Es un cifrado simétrico, utiliza la misma llave tanto para el cifrado como para el descifrado.

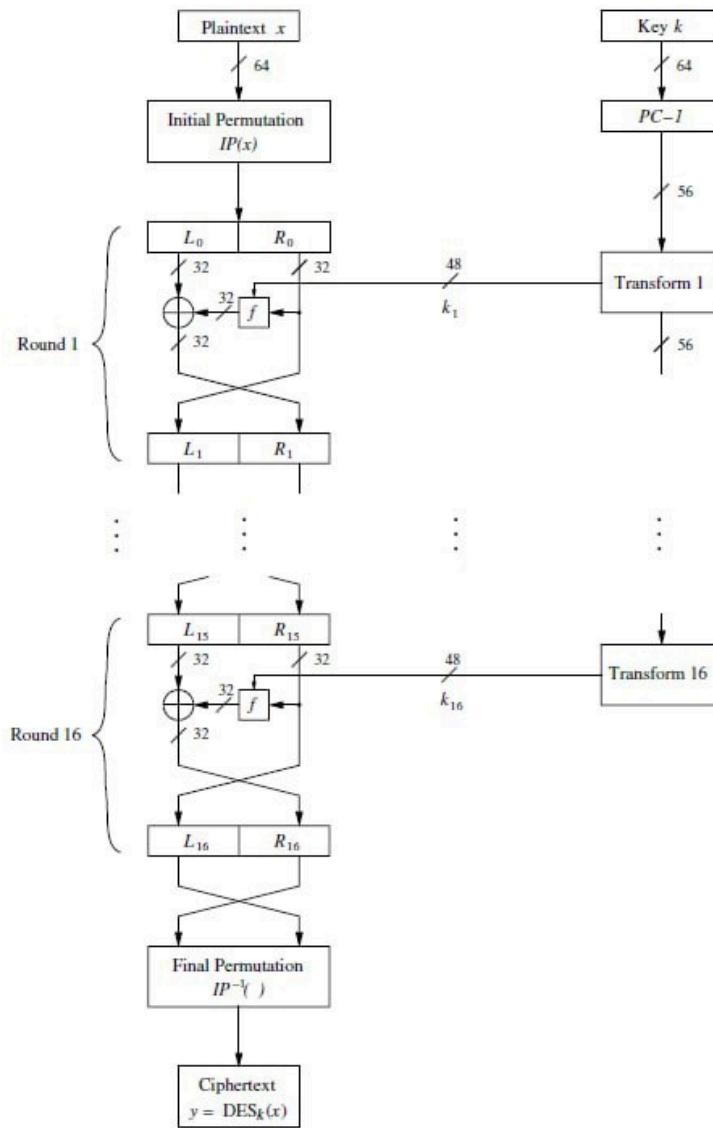
### \* Estructura general

- DES es un algoritmo iterativo que se efectúa en 16 rondas.
- En cada ronda, se utiliza una subllave  $K_i$  diferente derivada de la llave principal  $K$ .
- Es similar a los cifrados modernos de bloques.



## \* Red Feistel

- El corazón de DES es una red Feistel
- Estas redes se usan en muchos códigos de bloques modernos (no en AES)
- Si se usan correctamente puede generar códigos muy seguros.
- Ventaja: La encriptación y desencriptación son esencialmente el mismo proceso, pero con las subllaves aplicadas en orden inverso. Cifrado ... X



## \* Proceso

### ① Permutación inicial

- El bloque de texto plano de 64 bits pasa por una permutación inicial  $P_1$

### ② División de mitades

- El bloque se divide en dos mitades de 32 bits:  $L_0$  (izq) y  $R_0$  (derecha)
- Estas serán el input de la red Feistel

### ③ Rondas (16 en total)

- En cada ronda, la mitad derecha  $R_i$  pasa por una función de cifrado  $f$
- El resultado de  $f$  se combina mediante una operación  $\text{XOR}$  con la mitad izquierda  $L_i$
- Las mitades se intercambian y el proceso se repite.

$$L_i = R_{i-1}$$

$$R_i = L_{i-1} \oplus F(R_{i-1}, k_i)$$

- Al finalizar la ronda 16 las dos mitades de 32 bits  $L_{16}$  y  $R_{16}$  se intercambian de nuevo.
- En cada ronda, una llave  $k_i$  se obtiene la la clave principal de 56 bits usando un "esquema de llaves"

### ④ Permutación Final $(P_1)^{-1}$

- Despues de los 16 rondas se realiza una permutación final inversa  $(P_1)^{-1}$  para obtener el texto cifrado

## \* Función de cifrado $f$ :

- La función  $f$  toma dos entradas  $R_{i-1}$  y la llave  $k_i$
- **Objetivo:** Generar un valor "pseudorandomo" que se combine con la mitad izquierda  $L_i$  mediante  $\text{XOR}$
- **Confusión y difusión:** La función  $f$  es la responsable de la confusión (relación no obvia entre la clave y el texto cifrado) y difusión (propagación del cambio de un bit).
- La función  $f$  se debe diseñar con cuidado, una vez que ha sido creada seguramente, la seguridad del cifrado Feistel aumenta con el número de bits de la llave que se usan y el número de rondas.

## \* Consideraciones finales:

- La seguridad de DES depende del diseño de la función  $F$  y el uso de subllaves y rondas múltiples.
- **Perfeccional es un mapeo biyectivo:** Cada bloque de entrada de 64 bits se transforma de manera única en un bloque de salida de 64 bits. Permanece biyectivo aunque  $F$  no lo sea.
- En el caso de DES, la función  $F$  es sobreyectiva. Usa bloques no lineales y una llave de 48 bits para transformar 32 bits de entrada en 32 de salida.

**Nota:** Feistel solo encripta el lado izquierdo en cada ronda, el derecho permanece igual.

## Ejemplo:

① Entrada: texto plano de 8 bits: 10101010  
clave de 8 bits: 11001100

Actualizamos  $L$  y  $R$ :  $L_0 = R_0 = 1010$

$$R_1 = L_0 \text{ XOR } F(R_0, k_1)$$

$$= 1010 \text{ XOR } 0110 = 1100$$

② IP: "divide en 2 mitades":

$$L_0 = 1010$$

$$R_0 = 1010$$

③  $F$ : tomar  $R$  y llave  $k_1$  y aplicar  $\text{XOR}$

④ Ronda 1: Entrada:  $L_0 = 1010$

$$R_0 = 1010$$

$$\text{Subclave} = 1100$$

$$\text{Aplicar } F: F(R_0, k_1) = R_0 \text{ XOR } k_1$$

$$F(1010, 1100) = 0110$$

⑤ Ronda 2: Entrada  $L_1 = 1010$

$$R_1 = 1100$$

$$\text{Subclave} = 0011$$

$$\text{Aplicar } F: F(R_1, k_2) = F(1100, 0011) = 1111$$

$$\text{Actualizamos } L, R: L_2 = R_1 = 1100$$

$$R_2 = L_1 \text{ XOR } F(R_1, k_2)$$

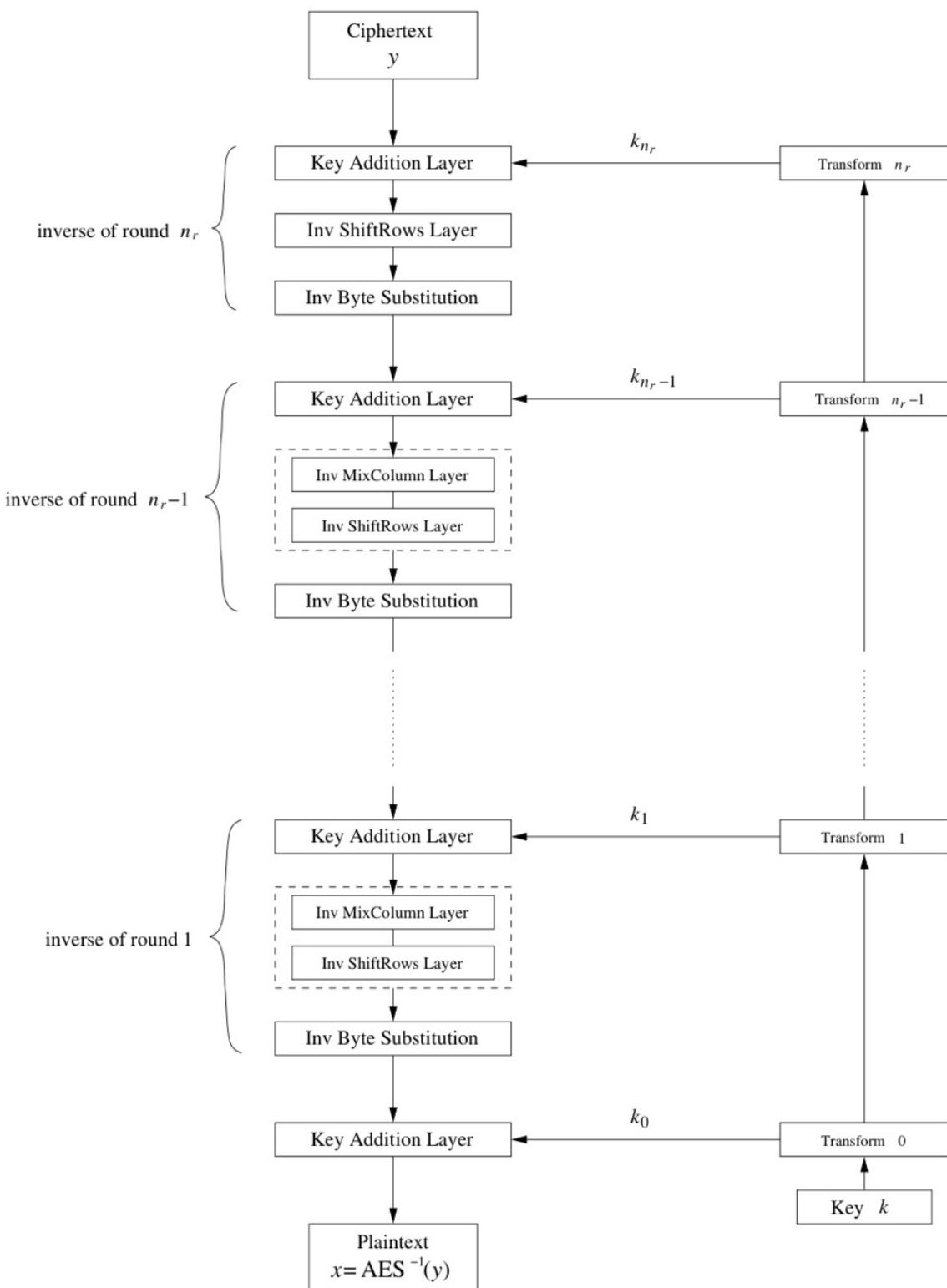
$$1010 \text{ XOR } 1111 = 0101$$

⑥ Permutación Final: ninguna

⑦ Resultado:  $L_2 \cup R_2 = 11000101$

## 4.5 Decryption

\* Al no estar basado en una red Feistel, todas las capas se deben invertir: la sust. de byte se convierte en inversión de S-box por bytes y así. Además, las operaciones inversas son similares a los Operaciones y el orden de las llaves se invierte:



**Fig. 4.8** AES decryption block diagram

$\text{XOR} \longleftrightarrow$  su propia inversa.

La capa de adición de llave en el modo de descifrado es la misma: consiste en una fila de casilleros XOR.

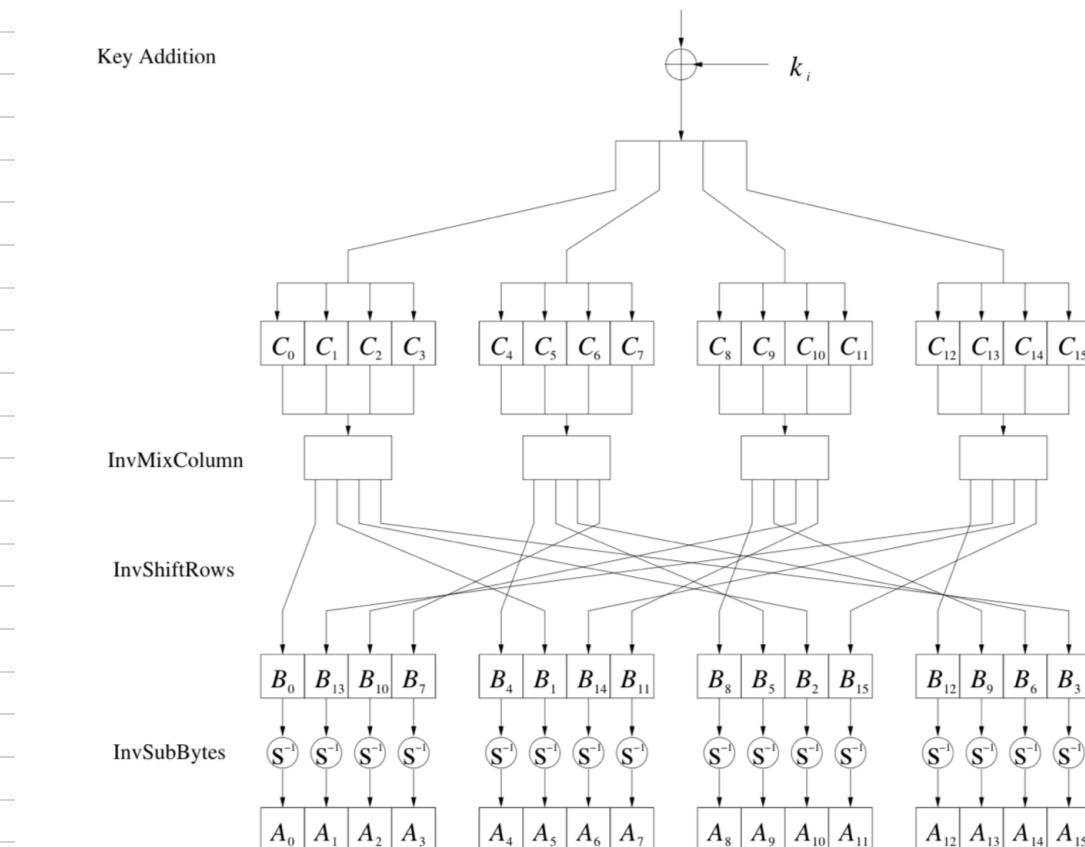


Fig. 4.9 AES decryption round function  $1, 2, \dots, n_r - 1$

### Capa inversa de MixColumn

Después de la adición de llave, se aplica la inversa de MixColumns a la matriz de estado (excepto en la primera ronda). Se utiliza una matriz inversa de  $4 \times 4$  convoluciones constantes. La operación se realiza en  $\text{GF}(2^8)$ . Cada columna de la matriz estado se multiplica por este matriz inversa usando la operación XOR bit a bit.

$$\begin{pmatrix} B_0 \\ B_1 \\ B_2 \\ B_3 \end{pmatrix} = \begin{pmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0E & 0B \\ 0B & 0D & 09 & 0E \end{pmatrix} \begin{pmatrix} C_0 \\ C_1 \\ C_2 \\ C_3 \end{pmatrix}$$

$B_i, C_i \in \text{GF}(2^8)$

$\underbrace{\hspace{2cm}}$

todos los elem. están en  $\text{GF}(2^8)$

La notación para los constantes es hexadecimal:

$$OB = (OB)_{hex} = (00001011)_2 = x^3 + x + 1$$

### Capa inversa de desplazamiento de filas

Para realizar esta operación, las filas de la matriz de estado se desplazan en dirección opuesta. La primera no cambia pero las demás se reordenan para restaurar su posición original.

Si el input esté dado por la matriz estado  $B = (B_0, B_1, \dots, B_{15})$ :

$B_0$	$B_4$	$B_8$	$B_{12}$
$B_1$	$B_5$	$B_9$	$B_{13}$
$B_2$	$B_6$	$B_{10}$	$B_{14}$
$B_3$	$B_7$	$B_{11}$	$B_{15}$

el output de la inversa sería:

$B_0$	$B_4$	$B_8$	$B_{12}$
$B_{13}$	$B_1$	$B_5$	$B_9$
$B_{10}$	$B_{14}$	$B_2$	$B_6$
$B_7$	$B_{11}$	$B_{15}$	$B_3$

no shift

← three positions left shift

← two positions left shift

← one position left shift

### Capa inversa de sustitución de $B_{xy}$

Durante el descifrado, se usa una S-Box inversa (es una función biyectiva).

$$A_i = S^{-1}(B_i) = S^{-1}(S(A_i)), A_i, B_i \text{ elementos de la matriz de estado}$$

**Table 4.4** Inverse AES S-Box: Substitution values in hexadecimal notation for input byte (xy)

	$y$															
$x$	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	52	09	6A	D5	30	36	A5	38	BF	40	A3	9E	81	F3	D7	FB
1	7C	E3	39	82	9B	2F	FF	87	34	8E	43	44	C4	DE	E9	CB
2	54	7B	94	32	A6	C2	23	3D	EE	4C	95	0B	42	FA	C3	4E
3	08	2E	A1	66	28	D9	24	B2	76	5B	A2	49	6D	8B	D1	25
4	72	F8	F6	64	86	68	98	16	D4	A4	5C	CC	5D	65	B6	92
5	6C	70	48	50	FD	ED	B9	DA	5E	15	46	57	A7	8D	9D	84
6	90	D8	AB	00	8C	BC	D3	0A	F7	E4	58	05	B8	B3	45	06
7	D0	2C	1E	8F	CA	3F	0F	02	C1	AF	BD	03	01	13	8A	6B
8	3A	91	11	41	4F	67	DC	EA	97	F2	CF	CE	F0	B4	E6	73
9	96	AC	74	22	E7	AD	35	85	E2	F9	37	E8	1C	75	DF	6E
A	47	F1	1A	71	1D	29	C5	89	6F	B7	62	0E	AA	18	BE	1B
B	FC	56	3E	4B	C6	D2	79	20	9A	DB	C0	FE	78	CD	5A	F4
C	1F	DD	A8	33	88	07	C7	31	B1	12	10	59	27	80	EC	5F
D	60	51	7F	A9	19	B5	4A	0D	2D	E5	7A	9F	93	C9	9C	EF
E	A0	E0	3B	4D	AE	2A	F5	B0	C8	EB	BB	3C	83	53	99	61
F	17	2B	04	7E	BA	77	D6	26	E1	69	14	63	55	21	0C	7D

La inversión se realiza en dos pasos:

(1) Se calcula la trans. de fin inversa, tratando cada byte como elemento en  $GF(2^8)$

$$\begin{pmatrix} b'_0 \\ b'_1 \\ b'_2 \\ b'_3 \\ b'_4 \\ b'_5 \\ b'_6 \\ b'_7 \end{pmatrix} \equiv \begin{pmatrix} 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \end{pmatrix} \begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 1 \end{pmatrix} \text{ mod } 2,$$

T resultados

la representación binaria de  $B_i$

- 2) Se invertir la operación en el campo utilizando el polinomio de reducción  $P(x) = x^8 + x^4 + x^3 + x + 1$   
 Polinomio  $A_i = (A_i^{-1})^{(1)}$ , la op. inversa es "reversa" al calcular la inversa de mueso  $(A_i)$  con la otra  $A_i = (B_i^{-1})^{(1)} \in GF(2^8)$   
 con el pol. de reducción. El  $O \rightarrow O$ . El vector  $A_i \cdot (a_7, \dots, a_0)$  [el elem.  $a_7x^7 + \dots + a_0x + a_0$ ] es el resultado de la sustitución  
 $A_i \cdot S^{-1}(B_i)$

El resultado de la sust. inversa es el valor original de la matriz de estado ante del cifrado

### Ejercicio de clave para el descifrado

La primera ronda de descifrado requiere la última subclave, la segunda la penúltima y así, ant. las subclaves se usan en el orden inverso.

En la práctica, se calculan previamente todo el esquema de llaves y se almacenan las 11, 13 o 15 subclaves.

## 4.6 Implementación en Software y Hardware

### Software

- AES fue diseñado para ser más eficiente en Software que DES.
- No es óptima implementarlo en un procesador moderno de 32 o 64 bits debido a que las operaciones en byte a byte.
- Los creadores de Rijndael propusieron la técnica/usa de T-Boxes para mejorar la eficiencia.
- Las T-boxes agrupan las funciones en tablas de búsqueda de 256 entradas de 32 bits. Esto permite calcular cada ronda con 16 accesos a los tablas, lo que logra velocidades de 16 Gbit/s en procesadores de 64 bits.

### Hardware

- AES requiere más recursos que DES.
- Existen implementaciones de alta velocidad que superan los 10 Gbit/s en chips dedicados.
- Con la paralelización se puede aumentar el rendimiento
- Si se compara con otros algoritmos el AES es extremadamente rápido.

## 4.7 Discusión y Lecturas Adicionales

- \* AES ha sido estudiado intensamente desde los años 90 y no se conocen ataques más eficientes que la fuerza bruta.
- \* Si existen estudios de ataques algebraicos pero no han resultado viables en la práctica.
- \* Implementaciones eficientes utilizan las T-boxes.
- \* Intel introdujo instrucciones AES en 2008 para mejorar la velocidad en hardware.

## 4.8 Conclusiones

- \* AES es un cifrado moderno que soporta llaves de 128, 192, 256 bits y ofrece seguridad contra ataques de fuerza bruta.
- \* No se han encontrado ataques prácticos mejores que la fuerza bruta.
- \* No utiliza redes Feistel, sino aritmética en campo de Galois para lograr una fuerte difusión y confusión.
- \* Es parte de estándares como IPsec y TLS y es el algoritmo obligatorio para aplicaciones gubernamentales.
- \* Su eficiencia en software y hardware lo convirtió/convierte en el algoritmo dominante para el cifrado en años recientes.

$S \sqrt{2} \text{ bits} \Rightarrow 2^{32} \text{ lenguajes}$

## 7.7 RSA en la práctica "Relleno" (padding)

Lo que hemos visto hasta ahora es "schoolbook RSA" que tiene varias "debilidades". La solución a estas es usar RSA junto con un esquema de relleno (padding).

Algunos problemas del RSA básico son:

\* **Determinismo**: para una misma clave, un mismo mensaje siempre genera el mismo cifrado, lo cual permite al atacante inferir información estadística del mensaje original.

\* **Mensajes triviales**: Mensajes como  $x=0, x=1 \text{ o } x=-1$  producen cifrados iguales  $0, 1, -1$ .

\* **Pequeños exponentes públicos**: no se conocen ataques contra exp. pág. como  $2^{16} + 1$  pero los mensajes pág. sin buen relleno podrían ser vulnerables.

\* **Maleabilidad**: RSA permite que un atacante manipule el cifrado sin necesidad de descifrarlo cambiando el mensaje original de manera predecible.

Ej.: Si Oscar multiplica el cifrado y por un número  $s^e$ , el receptor descifra  $sx \bmod n$ . Esto podría permitir por ej. duplicar las cifras de dinero sin ser detectado. !!

$$(s^e y)^d \equiv s^{ed} x^{ed} \equiv sx \bmod n$$

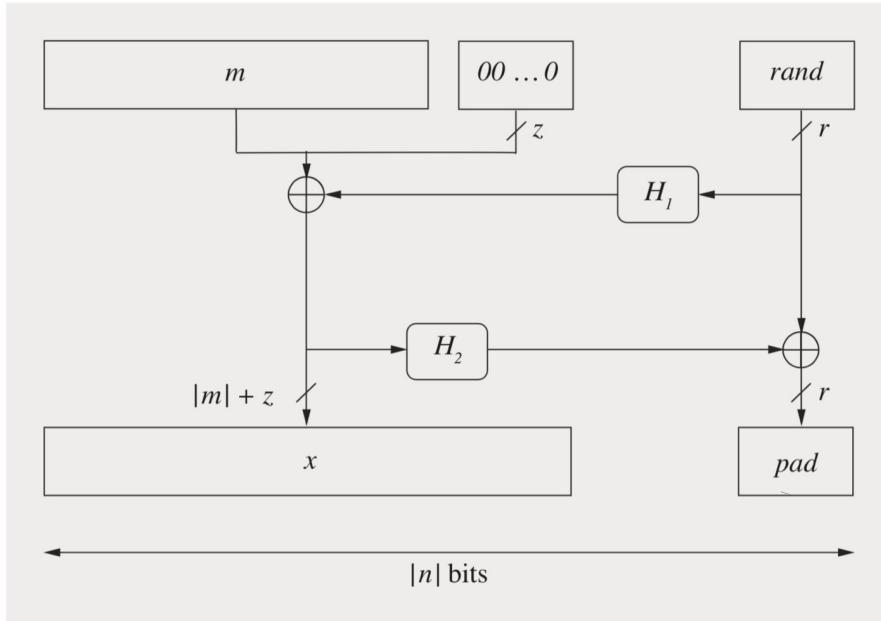
La solución:

Usa un esquema de padding que introduce aleatoriedad en el mensaje antes de cifrarlo.

Técnicas modernas como Optimal Asymmetric Encryption Padding (OAEP) están estandarizadas en Public-Key

# Cryptography Standard #1 (PKCS #1).

## ¿Cómo funciona OAEP?



**Fig. 7.3** Principle of the Optimal Asymmetric Encryption Padding (OAEP) scheme for a message  $m$

Antes de cifrar se añaden dos elementos al mensaje original  $m$ :

- \* Un número aleatorio  $rand$  de  $r$  bits (ej. 128b)
- Esto convierte RSA de determinista a probabilístico
- \* Una cadena de ceros de  $z$  bits. Esta cadena ayuda a prevenir ataques de maleabilidad.

El objetivo es que después del llenado, el tamaño total sea igual a la longitud del módulo RSA  $n$ :

$$|n| = |m| + z + r$$

El mensaje original  $m$  debe ser más corto que  $n$ .

## Proceso del relleno:

Se combina  $m$ , la cadena de ceros y  $rand$  usando dos funciones de "generación de máscaras" (funciones hash como SHA-2, adaptadas para sacar la longitud deseada).

\*  $H_1$  genera una máscara para  $m + \text{ceros}$

\*  $H_2$  genera una máscara para  $rand$

El padding se realiza en dos rondas como en una red Feistel:

1) El valor  $rand$  "cifra" el mensaje  $m + \text{ceros}$  usando  $H_1$ .

2) El resultado anterior "cifra"  $rand$  usando  $H_2$ .

Nota:  $H_1$  y  $H_2$  podrían ser lo mismo.

## Cifrado y Descifrado con RSA-OAEP.

### RSA Encryption with OAEP

Given a message  $m$  and the public key  $k_{pub} = (n, e)$ , the encryption function is:

$$y = e_{k_{pub}}(m) \equiv (x \parallel pad)^e \text{ mod } n$$

where

$$x = (m \parallel 00\dots0) \oplus H_1(rand)$$

$$pad = rand \oplus H_2(x)$$

① Crear  $x = (m \parallel \text{ceros}) \oplus H_1(rand)$

② Crear  $pad = rand \oplus H_2(x)$

③ Cifrar el bloques combinado  $(x \parallel pad)^e \text{ mod } n$

### RSA Decryption with OAEP

Given the private key  $k_{pr} = d$  and the ciphertext  $y$ , the decryption process is:

1. RSA decryption:  $d_{k_{pr}}(y) = y^d \equiv x||pad \bmod n$
2. recompute  $rand = H_2(x) \oplus pad$
3. recompute  $m||00\dots0 = x \oplus H_1(rand)$
4. verify that the zero string from Step 3 in fact contains  $z$  zero bits

Nota: revisar la documentación de OAEP antes de implementarlo porque omitieron detalles en el libro =/

## 7.8 Key Encapsulation

¿Para qué sirve?

Aunque podemos usar cifrado asimétrico (RSA) para mandar mensajes seguros, en la práctica se usa para enviar las llaves simétricas y luego cifrar mensajes grandes con cifrado simétrico que es mucho más rápido.

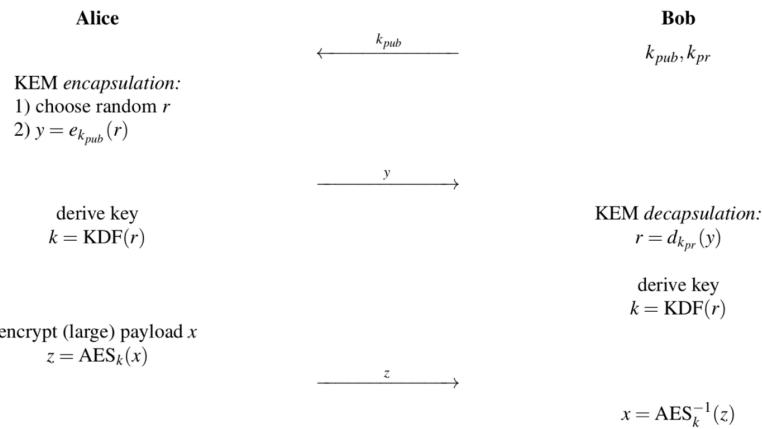
Problema:

Si queremos cifrar una llave simétrica pequeña (ej 128 bits para AES) con RSA-2048 habría que hacer padding pero esto quede complicado / comprometer la seguridad

Solución práctica

Usar Key encapsulation Mechanism (KEM)

# ¿Cómo funciona KEM?



**Fig. 7.4** Key encapsulation mechanism with public-key encryption, with AES being used as an example symmetric cipher

## ① Encapsulación (Alice)

- Elige un valor aleatorio  $r$
- Cifra  $r$  usando cifrado asimétrico  
 $y = e_{k_{pub}}(r)$

- Envía  $y$  a Bob

## ② Decapsulación (Bob)

- Descifra  $r$  usando su clave privada  
 $r = d_{k_{pr}}(y)$

## ③ Generar la clave simétrica

- Tanto Alice como Bob aplican una función de derivación de clave KDF (como una hash) a  $r$ :  $K = \text{KDF}(r)$

## ④ Cifrar y decifrar mensajes grandes:

- Cifrar datos grandes usando AES con  $K$
- Ej.:  $z = \text{AES}_K(x)$  enviar  $z$ .

## Notas:

- El cifrado asimétrico se suele usar solo para enviar  $r$

- la llave final  $K$  se deriva de  $r$  usando una función segura
- El uso de KEM hace que el esquema sea más limpio y fácil de analizar en términos de seguridad.

## 7.9 Ataques a RSA

Aunque desde 1977 se han propuesto muchos ataques contra RSA, ninguno es grave si se implementa correctamente. La mayoría ataca como se usa RSA y no el algoritmo en sí.

Hay 3 tipos de ataques grandes:

- 1) Ataques de protocolo
- 2) Ataques matemáticos
- 3) Ataques de canal lateral

### 1. Ataques de Protocolo

¿Qué hacen? Aprovechan errores en la forma en que se usa RSA

Ejemplos: Ataques que explotan la maleabilidad de RSA

Como prevenirlos: usando padding adecuado y siguiendo los estándares modernos de seguridad

## 2. Ataques Matemáticos

¿Qué hacen? buscan factorizar el módulo  $n$  en sus primos  $p$  y  $q$ .

Svp. que Oscar (atacante) conoce el mód.  $n$ , la llave pública  $e$  y el texto cifrado  $y$ . Su meta es calcular la llave privada  $d$  que cumple  $e \cdot d \equiv 1 \pmod{\phi(n)}$ , podria solo aplicar el alg. extendido de Euclides y calcular  $d$  pero como no conoce  $\phi(n)$ , la mejor forma de hacerlo es descomponer  $n$  en sus primos  $p$  y  $q$ . Si Oscar puede hacer esto, el ataque sería exitoso y trivial en 3 pasos:

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

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

$$x \equiv y^d \pmod{n}$$

Prevención: usar un  $n$  muy grande (mínimo 2048 bits)

Historia: RSA impulsó demasiado la investigación de factorización de enteros y los mayores progresos se han debido a RSA.

**Table 7.3** Some of the RSA factoring records since 1991

Decimal digits	Bit length	Date
100	330	Apr 1991
110	364	Apr 1992
120	397	Jul 1993
129	426	Apr 1994
140	463	Feb 1999
174	576	Dec 2003
200	663	May 2005
212	704	Jul 2012
232	768	Dec 2009
250	829	Feb 2020

La factorización RSA-576 y RSA-768 fueron  
por el Reto de RSA Factoring Challenge que  
se anuncio por RSA laboratorios en 1991  
e incluia un premio en efectivo.

RSA-768 requirió 10 meses y 4 clusters

\* El 129-digt módulos fue publicado en una  
columna por Martin Gardner en Scientific American  
en 1977. Se estimó que el mejor algoritmo  
de factorización en ese tiempo tardaría  
 $4 \cdot 10^{13}$  años ??

\* Al momento de escribir el libro se creía que  
se podía factorizar un número de 1024 bits  
en alrededor de 10 años

Recomendación actual:

Se recomienda usar parámetros en RSA  
en el rango de 2048 - 4096.

Advertencia futura:

Computadoras cuánticas podrían romper  
RSA ... (Cap. 12)

### 3. Ataques de canal lateral

Qué hacen? Esperan información física (como  
consumo de energía o tiempos de ejecución)  
desarrollando el uso de RSA.

Es un campo activo de investigación en la  
criptografía moderna

Ejemplo:

Observar el trazo de consumo de energía de un microprocesador

Al analizar los patrones de actividad alta y baja se puede deducir los bits de la clave privada d.

Detalle técnico

En RSA, cada bit del exp. elevado se procesa con un algoritmo de cuadrar y multiplicar

- Si el bit es 0 → solo cuadrar (actividad breve)

- Si el bit es 1 → cuadrar y multiplicar (act. más larga)

Así al mirar el trazo de energía se pueden leer directamente los bits de la clave.

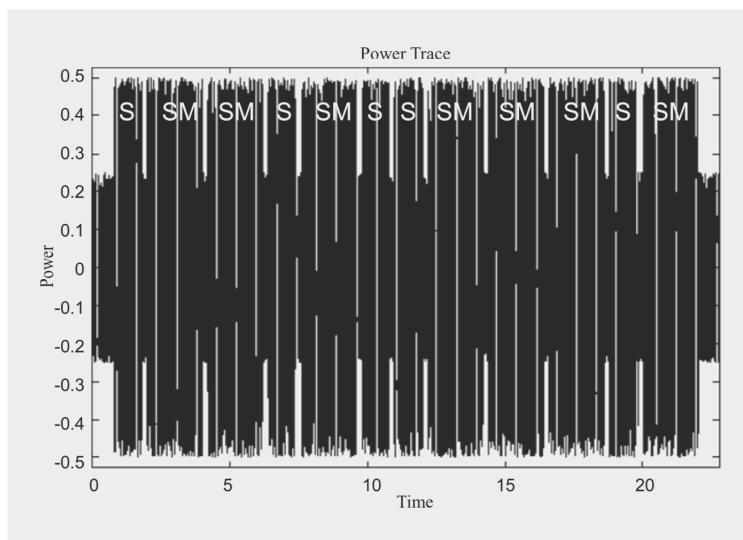


Fig. 7.5 The power trace of an RSA implementation

operations: S SM SM S SM S S SM SM SM S SM  
private key: 0 1 1 0 1 0 0 1 1 1 0 1

Se realizó 2048 bits.  
Obtenido

Este ataque se llama Análisis de Potencia Simple (SPA)

## Contromedidas:

Ejecutar operaciones falsas para que el perfil de energía no dependa del bit.

Nota: Defenderse de ataques más sofisticados del canal lateral es más difícil

## 7.10 Implementación en Software y hardware

RSA es un claro ej. de un algoritmo de clave pública que demanda un alto poder computacional, mucho más que los simétricos como 3DES o AES.

### Estimación del costo computacional

Sup. un módulo RSA de 2048 bits.

Una operación de desifrado requiere unas 3072 multiplicaciones y elevaciones al cuadrado con operandos de 2048 bits.

En una CPU de 32 bits, esto implica representar cada operando con 64 registros ( $2048/32$ )

Una multiplicación de núm. largos requiere multiplicar cada registro contra cada uno del otro operando:

$$64 \times 64 = 4096 \text{ multiplicaciones enteras}$$

Además, cada una de estas debe ser reducida módulo  $n$ , lo cual también requiere 4096 multiplicaciones adicionales.

Así que una sola multiplicación modular implica 8192 multiplicaciones enteras

Como se hacen 3072 de estas, se estima un total de

$$3072 \times 8192 = 25165824 \text{ mult. enteras}$$

Aunque CPUs modernas permiten operar con 64 bits, reduciendo esta cantidad de mult. en 4 sigue siendo una gran carga computacional, los mult. enteras volver ser de las operaciones más lentas.

## Impacto histórico y soluciones

Cuando RSA se inventó, esta exigencia comp. dificultó su adopción práctica: los ordenadores de los 70's no podían realizar millones de mult. en un tiempo razonable.

La solución fue implementar RSA en chips especializadas (hardware) y durante muchos años fue la única forma viable.

Con el tiempo, la mejora de chips (gracias a la ley de Moore) permitió realizar RSA en software desde finales de los 80's.

Hoy en día (momento del libro) una op. de descifrado RSA - 2048 se puede completar en 1.6 mil. segundos en un CPU de 3GHz.

## RSA NO se usa para cifrado de grandes volúmenes

A pesar de las mejoras, RSA sigue siendo demasiado lento con grandes vol. de datos.

Por ej. cifrar 2048 bits en 1.6 ms da una tasa de apenas 160KB/s lo cual es insuficiente para aplicaciones móviles.

Por esto, RSA suele usarse solo para el intercambio de llaves,

## 7.11 Discusion y lecturas complementarias

### RSA y sus variantes

RSA es uno de los sif. más usados y está bien estandarizado.

Con el tiempo se han propuesto variantes con más de dos primos en los módulos, p.ej:

\* Módulo del tipo  $n = p^2q$

\* Módulo multifactor como  $n = pqr$

Estas variantes permiten acelerar las operaciones de descifrado en un factor 2 a 3.

Otros esquemas se basan en el problema de fact. de enteros, por ej.

\* Esquema de Rabin: su seguridad es equivalente a la d. dificultad de factorizar, por lo que es probablemente seguro

\* Cifrado probabilístico de Blum-Goldwasser y el generador de números pseudoraleatorios Blum-Blum-Shub

→ Lectura recomendada Handbook of Applied Cryptography.

## Implementación

- El rendimiento de RSA depende en gran medida de la eficiencia aritmética:
- \* A alto nivel se pueden optimizar los algoritmos, como con la técnica de "ventana deslizante" que mejora en un 25% el alg. de cuadrar y multiplicar. Sin embargo esto que de filtrar información mediante ataques por análisis de potencia o tiempo.
  - \* A bajo nivel: se pueden optimizar las mult. y reducciones modulares. La reducción de Montgomery es la más utilizada tanto en soft. como hardware. También se aplican métodos rápidos de multiplicación como Karatsuba

## Ataques

Desde los 70's, romper RSA ha sido un objetivo de la criptografía:

- \* Progresos en alg. de factorización
- \* Ataques conocidos contra exp. privadas cortas como el de Wiener si  $d < \frac{1}{3}n^{1/4}$
- \* Si los primos p y q están muy cerca se puede aplicar factorización de Fermat
- \* Si se reutiliza uno de los primos entre dif. llaves de RSA, un atacante usando mcd podrá descubrirlo fácilmente. Esto fue descubierto en 2012 y se encontró que 0.75% de los

certificados TSL compartían llaves por falta de entropía al generar primos.

## Ataques de canal lateral

Desde finales de los 90, se han estudiado ataques como:

\* Análisis diferencial de Potencia (DPA) es más poderoso que el SPA. Afortunadamente los contramedidas para DPA suelen ser más fáciles de implementar en algoritmos de llave pública que en cifrados simétricos.

\* Inyección de fallos y ataques de tiempo: Son amenazas relevantes. Un sistema puede ser matemáticamente sólido pero vulnerable a estos ataques físicos.

# Firmas digitales

10.1.

los sistemas de cifrado hasta ahora tienen dos metas principales

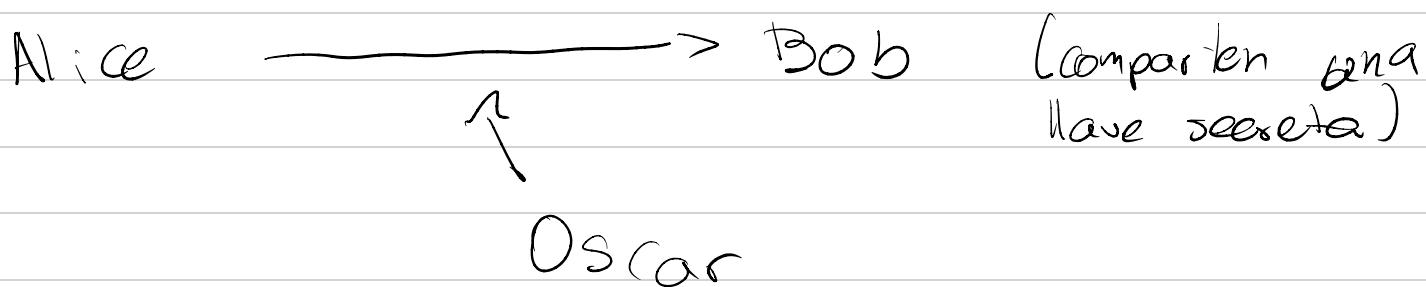
1) Cifrar datos

2) Intercambiar llaves

Podríamos pensar que solo debemos prestar atención a los mecanismos de seguridad

Vamos a ver un caso en el cual los códigos simétricos fallan al proveer la función de seguridad deseada.

Caso típico:



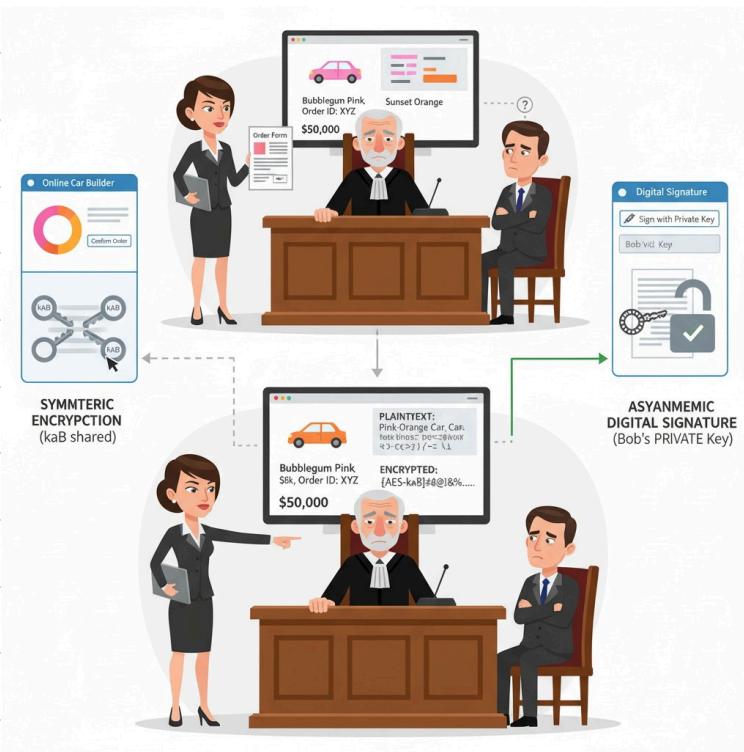
Si Alice y Bob usan un cifrado simétrico (AES) pueden estar seguros de que Oscar no va a aprender el texto plano.

Podrían incluso emplear un código de autenticación de mensajes (cap 13), y Bob estaría seguro de que el mensaje viene de Alice y no de Oscar.

Los códigos de autenticación de mensajes usan cifrados simétricos.

"Todo parece super bien" pero hemos assumido siempre que el **chico malo** es alguien **externo!!!**

Es usual que Alice y Bob se quieran comunicar seguramente y al mismo tiempo engañarse uno al otro!!!



¿Porqué no podemos usar algún esquema de llaves simétrico (diffie-hellman) para verificar o probar que envio Bob?

Alice y Bob tienen la misma información, cualquier persona puede hacer lo mismo.

La solución es usar cripto de llave pública.

El setup asimétrico de los esquemas de llave pública le da al juez la forma de distinguir quién envió.

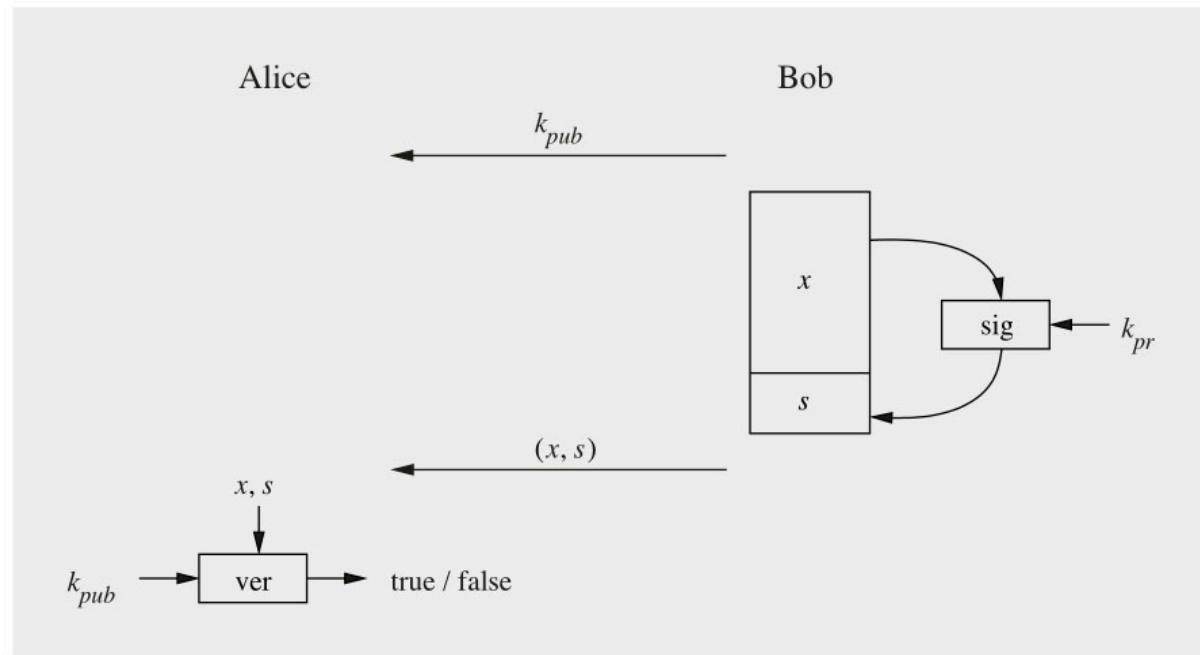
Las firmas digitales son los equivalentes de llaves públicas que resuelve las situaciones de "enganarse" uno a otro.

## Principios de firmas digitales

Firmas digitales es el análogo a firmar en papel un documento.

Solo la persona que crea el mensaje digital puede crear una firma válida.

La idea básica: la persona que firma usa la llave privada y la que recibe usa la llave pública que hace match con la llave privada.



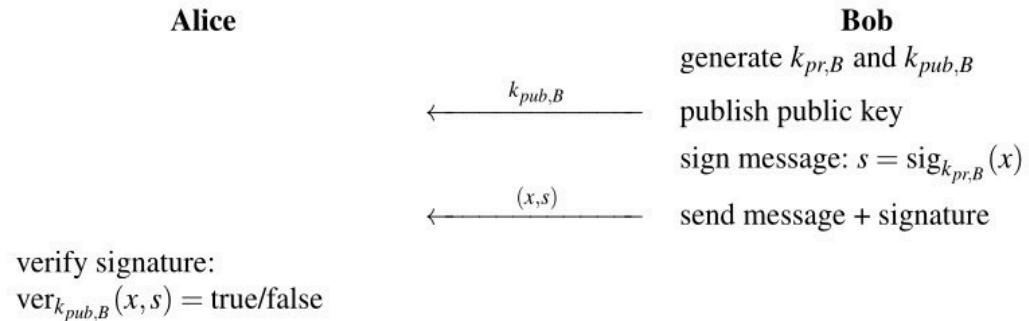
la firma dig  
es una función  
de la llave priv.  
de Bob

la firma dig  
es un entero  
grande, una  
cadena de 2048  
bits

Fig. 10.1 Principle of digital signatures, which involves signing and verifying a message

La única salida es verdadero o falso.

## Basic Digital Signature Protocol



Este esquema no "cifra" el mensaje, para eso se debe usar algún otro cifrado simétrico o otro.

Calcular de las 3 familias de algoritmos de clave pública sirven para construir firmas digitales:

- factorización de enteros
- logaritmos discretos
- curvas elípticas

## Servicios de seguridad

¿Cuáles son los posibles objetivos de seguridad que un sistema de seguridad debe poseer?

- \* Confidencialidad:
- \* Integridad:
- \* Autenticación del mensaje o del origen de los datos
- \* No-repudio:

Confidencialidad, integridad y disponibilidad  
se conocen como el CIA triad ya  
que se consideran importantes para asegurar  
el sistema.

Ejemplos:

\* Emails: si es privado los 3 primeros si es  
corporativo también no repudio

\* Actualización de software en un cel: integridad  
y autenticación de message; el manufacturer quiere  
asegurarse de que solo soft. original se le  
pone al equipo. Confidencialidad no se  
necesita porque el soft. está disponible  
de cualquier forma.

Otros servicios de seguridad:

\* Identificación o autenticación de entidad:  
establecer y verificar la identidad de una  
entidad, ej persona, tarjeta, etc.

\* Control de acceso: acceso restringido a recursos  
de entidades con privilegios.

\* Disponibilidad: asegurar que el syst. elec.  
nico esté disponible fielmente.

\* Auditoria: proporcionar evidencia sobre act. relevantes para la seguridad, como guardar registro de eventos.

\* Seguridad física:

\* Anonimato: proporciona protección contra el descubrimiento y el uso indebido de la identidad.

## APLICACIONES DE FIRMAS DIGITALES

\* Certificados: sirven para garantizar que una clave pública realmente pertenece a la identidad que dice representar, evitando ataques con claves falsas.

\* Secure Boot and Secure firmware update: buscan garantizar integridad y autenticidad del software.

Solo puede ejecutarse soft. confiable del proveedor original

Se usan firmas digitales para verificar el software en cada inicio del sistema (arranque autenticado)

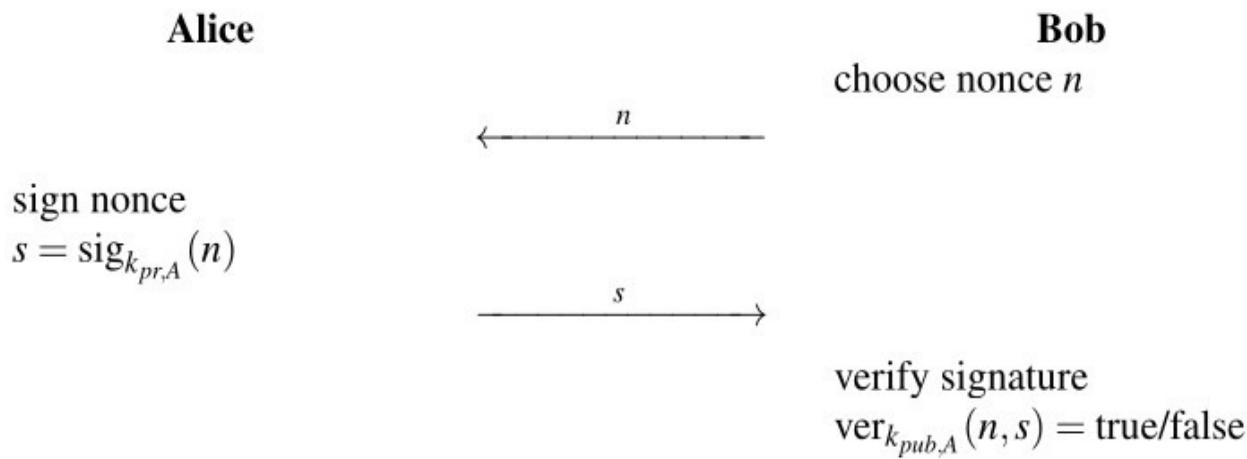
Ventaja principal: los usuarios solo requieren la clave pública del proveedor o el único que puede generar actualizaciones válidas con su clave privada.

Existen estándares de arranque seguro (UEFI) ampliamente adoptados en industria.

\* Proof-of-Knowledge protocols: permiten que una persona demuestre su identidad

sin revelar su decreto.

Ej. Alice necesita convencer a Bob de que posee un secreto sin mostrárselo. Se diferencian de los sistemas basados en contraseñas donde el secreto si se comparte



Bob comprueba la validez de la firma que Alice conoce el secreto "la llave privada"

Algunas aplicaciones son tarjetas inteligentes para autenticación en ATM o acceso a edificios.

## 10.2 (1) esquema de firma RSA

Su seguridad recae en el problema de factorización de enteros.

### 10.2.1 Schoolbook of RSA Dig. Sig.

El padding/belleno debe de usarse para prevenir ataques.

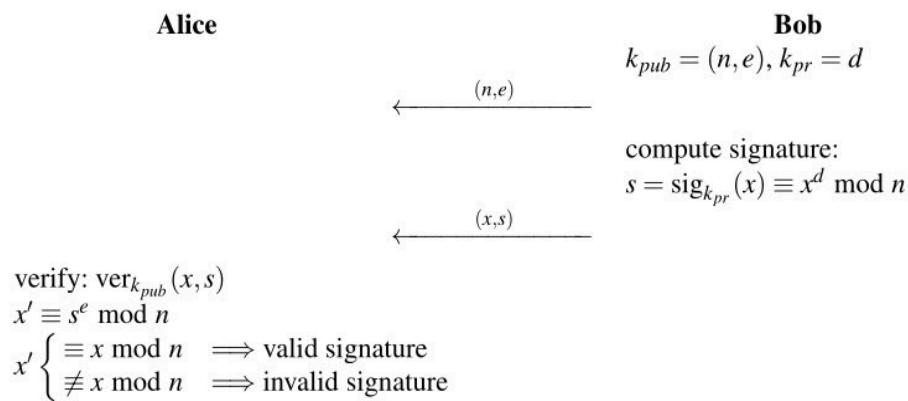
Sup. que Bob le quiere enviar un mensaje a Alice. De sigue el procedimiento de RSA y se tiene

#### RSA Keys

- Bob's private key:  $k_{pr} = (d)$
- Bob's public key:  $k_{pub} = (n, e)$

El protocolo de firma es el siguiente. El mensaje  $x$  que se va a firmar esta en el rango  $(1, 2, \dots, n-1)$ .

#### Basic RSA Digital Signature Protocol



Bob es el único que puede usar  $k_{priv}$  y por lo tanto la actan verifica como el dueño del mensaje firmado

Alice recibe el mensaje firmado y usando la  $k_{pub}$  de Bob le regresa  $x'$ .

S:  $x' = x$  ent. Alice sabe 2 cosas:

- 1) El autor del mensaje tiene la  $k_{pr}$  de Bob y si solo Bob tiene la  $k_{pr}$  ent. es el autor del mensaje. (**Autenticación del mensaje**)
- 2) El mensaje no fue cambiado en el camino (**Integridad**)

Prueba: la verificación regresa "true" si el mensaje y la firma no fueron modificados en el camino.

Verificamos la operación  $s^e \bmod n$

$$s^e \equiv (x^d)^e = x^{de} \equiv x \bmod n$$

Ya que por la relación entre la  $K_{pr}$  y  $K_p$

$$de \equiv 1 \bmod \phi(n)$$

entonces calcular  $x \in \mathbb{Z}_n$  a la  $(de)$ -potencia nos da el mismo entero.



En los esquemas de firmas digitales con RSA, el rol de las llaves se invierte

### \* En RSA:

- La llave pub. se aplica al mensaje
- El receptor usa la llave privada para descifrar

### \* En Firmas digitales

- La  $K_{pr}$  se aplica para generar la firma.
- La  $K_{pub}$  se usa para verificar la firma

Ejemplo: Bob quiere enviar un mensaje firmado ( $x=4$ ) a Alice.

**Alice**

$$\xleftarrow{(n,e)=(33,3)}$$

**Bob**

1. choose  $p = 3$  and  $q = 11$
2.  $n = p \cdot q = 33$
3.  $\Phi(n) = (3-1)(11-1) = 20$
4. choose  $e = 3$
5.  $d \equiv e^{-1} \equiv 7 \pmod{20}$

compute signature for message  
 $x = 4$ :  
 $s = x^d \equiv 4^7 \equiv 16 \pmod{33}$

$$\xleftarrow{(x,s)=(4,16)}$$

verify:

$$x' = s^e \equiv 16^3 \equiv 4 \pmod{33}$$

$x' \equiv x \pmod{33} \Rightarrow$  valid signature

Alice concluye que Bob generó el mensaje y que no fue alterado (integridad y autenticidad)

## 10.2.2 Aspectos Computacionales

La firma digital tiene la misma longitud que el módulo  $n$  ( $\log_2 n$  bits)

\* Se recomienda que  $n \geq 2048$  bits

\* La generación de llaves es similar al cifrado RSA

\* Para calcular y verificar firmas se usa el algoritmo de square-and-multiply

\* Las técnicas de aceleración de RSA también se aplican a firmas digitales

- \* Uso de llaves públicas con los  $(e^{2^k}, f)$  resulta en una verificación muy rápida.
- \* En la práctica, los mensajes se firman una sola vez pero se verifican muchas veces

## 10.2, 3 Seguridad

Lo mismo de RSA:

→ verificar que RSA no debe romperse matemáticamente = dificultad de la factorización

Se requieren primos  $p$  y  $q$  de al menos 1024 bits y ent el módulo  $n \geq 2048$  bits.

Es esencial garantizar que los llaves pub sean auténticas

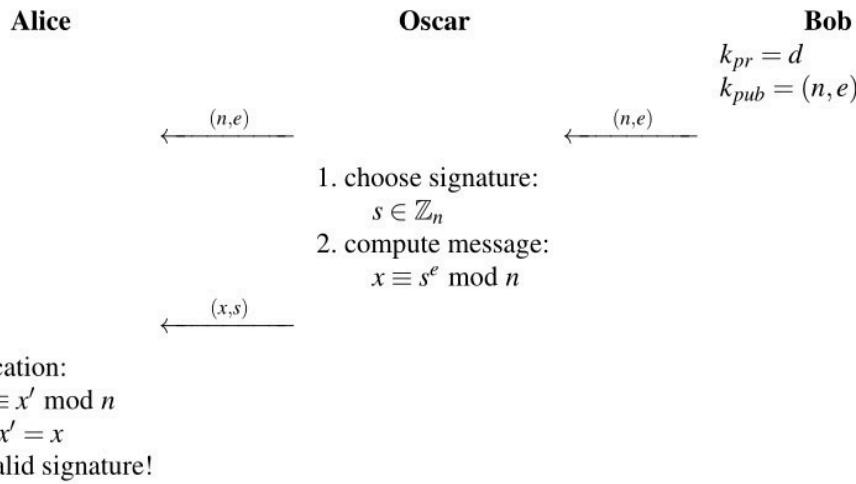
**Riesgo:** si un atacante entrega una llave pública falsa, puede firmar mensajes en nombre del verdadero emisor

**Solución:** uso de certificados digitales para asegurar la autenticidad de los llaves.

## Existental Forgery

Es un ataque posible en los esq. de firmas digitales

### Existential Forgery Attack Against RSA Digital Signature



En el RSA

básico una  
versión puede  
generar una fir-  
ma válida para  
un mensaje de este  
tipo.

### Modo de ataque:

- El atacante se hace pasar por Bob
- Alice verifica la firma como correcta porque realiza los mismos cálculos.

Limitación: el atacante elige primero la firma y luego el mensaje por lo que no controla el contenido (no puede ordenar una transferencia)

Problema: los sistemas automáticos no detectan la falsificación.

Consecuencia: El RSA básico casi no se usa en la práctica

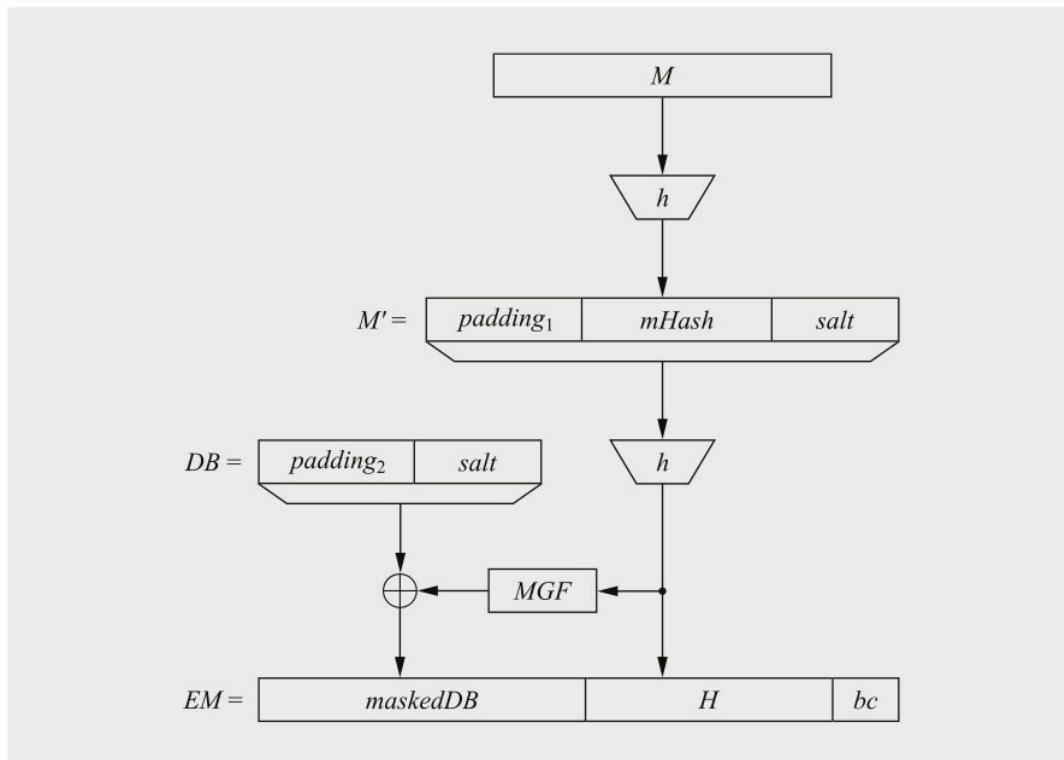
Solución: uso de espacios de padding.

## RSA Padding: The probabilistic Signature Standard (PSS)

El ataque anterior se puede prevenir imponiendo formatos específicos (padding) en los mensajes.

\* Ej: exigir que todos los mensajes tengan 100 bits de ceros al final. La probabilidad de coincidencia por azar es  $2^{-100}$ , prácticamente imposible.

**RSA-PSS (Probabilistic Signature Scheme):** es una extensión del RSA básico que combina firma + padding (encoding).



**Fig. 10.2** Principle of EMSA-PSS encoding

Procedimiento  
de encoding  
conocido como  
Encoding Method  
for Signature  
with Appendix  
(EMSA) Probabi-  
listic Signature  
Scheme (PSS).

## Encoding for the EMSA Probabilistic Signature Scheme

Let  $|n|$  be the size of the RSA modulus in bits. The encoded message  $EM$  has a length  $\lceil(|n| - 1)/8\rceil$  bytes such that the bit length of  $EM$  is at most  $|n| - 1$  bits.

1. Generate a random value  $salt$ .
2. Form a string  $M'$  by concatenating a fixed padding  $padding_1$ , the hash value  $mHash = h(M)$  and  $salt$ .
3. Compute a hash value  $H$  of the string  $M'$ .
4. Concatenate a fixed padding  $padding_2$  and the value  $salt$  to form a data block  $DB$ .
5. Apply a mask generation function  $MGF$  to the hash of the string  $M'$  to compute the mask value  $dbMask$ . In practice, a hash function such as SHA-2 is often used as  $MGF$ .
6. XOR the mask value  $dbMask$  and the data block  $DB$  to compute  $maskedDB$ .
7. The encoded message  $EM$  is obtained by concatenating  $maskedDB$ , the hash value  $H$  and the fixed padding  $bc$ .

En la práctica, no se firma el mensaje directamente, sino su huella digital (hash) de longitud fija (620 bits)

El esquema EMSA-PSS introduce:

- Salt (valor aleatorio) antes del segundo hash → hace que las firmas sean probabilísticas
- Misma entrada firmada dos veces → firmas distintas, lo cual es deseable

Proceso de firma:

mensaje → hash → encoding → Firma RSA  
(con padding + salt)

Proceso de verificación: el receptor reconstruye el encoding (con salt y padding) y valida la firma con la clave pública.

## 12.2 Criptografía basada en retículas

Def: Una retícula  $L$  se define como el conj. de combinaciones lineales enteras de un número de vectores independientes  $a_1, a_2, \dots, a_n$  (base) y con coef. enteros  $x_i \in \mathbb{Z}$ :

$$L = \{x_1 a_1 + x_2 a_2 + \dots + x_n a_n\}$$

$$\dim a_i = m$$

$$n = \text{rango de la retícula}$$

Para esquemas PQC se usa que las coordenadas de los vectores de la base estén en el anillo de enteros  $\mathbb{Z}_q$ .

Por lo general  $q$  se escoge como primo o potencia de dos.

Las operaciones se hacen "mod  $q$ "

Notación:  $a_i \in \mathbb{Z}_q^m$

$$A = \{a_1, a_2, \dots, a_n\} \in \mathbb{Z}_q^{m \times n}$$

### 12.2.1 El problema "Learning With Errors" (LWE)

Recordando: Para resolver el DLP (discrete logarithm problem), un atacante tenía que determinar un entero secreto  $x$  tg  $d^x \equiv \beta \pmod p$  dados  $\mathbb{Z}_p^*$ , generador  $d$  y el target  $\beta$  en el campo.

Podíamos pensar a  $d$  como la base.

Para el LWE se usa un problema similar, pero en una retícula.

La idea base es tomar cierta cant. da de elementos de la base  $a_1, a_2, \dots, a_n \in \mathbb{Z}_q^m$  y multiplicarlos con coeficientes enteros secretos  $s_1, \dots, s_p$  para llegar al punto objetivo  $t$  en la retícula:

$$s_1 a_1 + s_2 a_2 + \dots + s_n a_n = t$$

El problema es similar al de DLP:

- dado un punto de comienzo  $\leftarrow$  base
- $\nearrow$  el punto final  $\leftarrow$  target  $t$
- el adversario quiere encontrar  $(s_1, \dots, s_n)$

En forma matricial  $A \cdot s = t$ , donde  
 $s = (s_1, s_2, \dots, s_n)$

Ej: Retícula sobre  $\mathbb{Z}_{17}$  de rango  $n=4$ . Vectores de dim.  $m=7$ ,  $A \in \mathbb{Z}_{17}^{7 \times 4}$ ,  $t$  vector dim 7  
 El problema es:

$$A \cdot s = \begin{pmatrix} 10 & 13 & 16 & 10 \\ 12 & 5 & 14 & 12 \\ 10 & 11 & 8 & 11 \\ 7 & 7 & 5 & 3 \\ 8 & 13 & 8 & 9 \\ 14 & 13 & 13 & 16 \\ 7 & 6 & 10 & 4 \end{pmatrix} \cdot \begin{pmatrix} s_1 \\ s_2 \\ s_3 \\ s_4 \end{pmatrix} \equiv \begin{pmatrix} 2 \\ 14 \\ 7 \\ 6 \\ 10 \\ 8 \\ 16 \end{pmatrix} = t \bmod 17$$

Es "fácil" encontrar  $s$ , es un sistema de ecuaciones lineales.

la solución es

$$\mathbf{s} = \begin{pmatrix} s_1 \\ s_2 \\ s_3 \\ s_4 \end{pmatrix} = \begin{pmatrix} 6 \\ 2 \\ 12 \\ 3 \end{pmatrix}$$

Pero afortunadamente, podemos convertir este problema en uno difícil:

Consideremos  $t$  un punto **no** de la retícula pero **Cerrano** a un punto en la retícula

Matemáticamente se traduce en introducir un error:

$$A \cdot s + e = t'$$

En el ejemplo se vería así:

$$\begin{pmatrix} 10 & 13 & 16 & 10 \\ 12 & 5 & 14 & 12 \\ 10 & 11 & 8 & 11 \\ 7 & 7 & 5 & 3 \\ 8 & 13 & 8 & 9 \\ 14 & 13 & 13 & 16 \\ 7 & 6 & 10 & 4 \end{pmatrix} \cdot \begin{pmatrix} s_1 \\ s_2 \\ s_3 \\ s_4 \end{pmatrix} + \begin{pmatrix} e_1 \\ e_2 \\ e_3 \\ e_4 \\ e_5 \\ e_6 \\ e_7 \end{pmatrix} \equiv \begin{pmatrix} 1 \\ 13 \\ 8 \\ 6 \\ 10 \\ 9 \\ 15 \end{pmatrix} \pmod{17}$$

Un ataque usando eliminación Gaussiana o cualquier otra técnica para resolver ec. lineal no va a ser capaz de encontrar la solución correcta, ya que  $t'$  no es un punto de la retícula.

los vectores  $t$  y  $t'$  son muy cercanos y podemos ver que pequeñas variaciones de los coeficientes de la retícula, por ej. con errores en  $e_i \in \{-1, 0, 1\}$ , son suficiente para crear un problema no trivial. En el ejemplo:

$$\mathbf{e} = \begin{pmatrix} e_1 \\ e_2 \\ e_3 \\ e_4 \\ e_5 \\ e_6 \\ e_7 \end{pmatrix} = \begin{pmatrix} -1 \\ -1 \\ 1 \\ 0 \\ 0 \\ 1 \\ -1 \end{pmatrix} \equiv \begin{pmatrix} 16 \\ 16 \\ 1 \\ 0 \\ 0 \\ 1 \\ 16 \end{pmatrix} \pmod{17}$$

En conclusión, introducir errores a nuestro sistema nos permite alcanzar la meta:

"Técnicas estándar para resolver sistemas de ec. lineales no pueden ser usadas para recuperar los valores desconocidos de  $s$  y  $e$  eficientemente".

Para dist. en la vida real, al escoger  $n$  y  $m$  suficientemente grandes, nos aseguramos de que la fuerza bruta tampoco pueda resolver el problema.

En realidad se vuelve un problema NP-completo los cuales son incluso difíciles en computadoras cuánticas de larga escala.

Esta def. nos permite crear sistemas que son "quantum computer-resistant":

**Definition 12.2.2** Learning With Errors Problem (LWE)

Given a set of  $n$  basis vectors  $\mathbf{a}_i \in \mathbb{Z}_q^m$  represented by matrix  $\mathbf{A}$  and a point  $\mathbf{t} \in \mathbb{Z}_q^m$ .

The LWE is the problem of determining a set of secret coefficients  $\mathbf{s} = (s_1, s_2, \dots, s_n)$ , with  $s_i \in \mathbb{Z}_q$ , such that:

$$\mathbf{A} \cdot \mathbf{s} + \mathbf{e} \equiv \mathbf{t} \pmod{q}$$

where  $\mathbf{e}$  is an unknown error vector consisting of small integers modulo  $q$ .

## 12.2.2 A simple LWE-Based Encryption System

El problema LWE permite construir un esquema de cifrado de llave pública donde Alice cifra un mensaje y Bob lo descifra. Durante el cifrado se añade un error aleatorio pequeño para ocultar información y este debe ser manejado cuidadosamente para que Bob pueda recuperar los bits originales aún con perturbaciones.

### Codificación de bits

El mensaje consiste en bits  $m: 630, 17$ . Para usar LWE, cada bit debe mapearse a un valor del grupo cíclico  $\mathbb{Z}_q$ .

Solución óptima: usar valores alejados en el circuito modular:

Para maximizar la distancia entre los valores asignados, se propone

$$\bar{m}_i = \text{enc}_G(m_i) = \left\lfloor \frac{q}{2} \right\rfloor \cdot m_i$$

o decir

- Si  $m_i = 0 \rightarrow \bar{m}_i = 0$
- Si  $m_i = 1 \rightarrow \bar{m}_i = \left\lfloor \frac{q}{2} \right\rfloor$

Con  $q=61$ :

- $m_i = 0 \rightarrow \bar{m}_i = 0$
- $m_i = 1 \rightarrow \bar{m}_i = 30$

Esto coloca a los codigos en extremos opuestos del círculo, reduciendo el riesgo de confusión

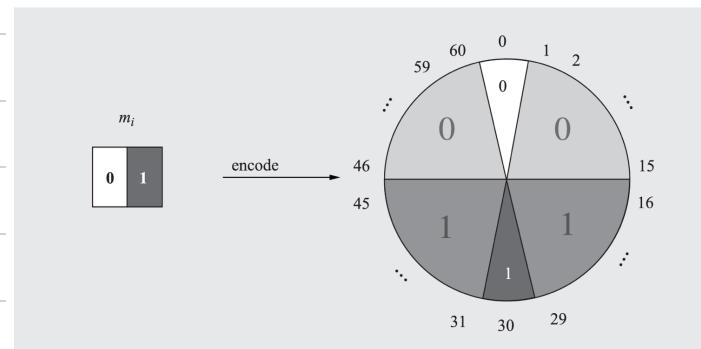


Fig. 12.4 Safe mapping of bits  $m_i$  to coefficients  $\bar{m}_i \in \mathbb{Z}_{61}$

Describir

Para decidir si el valor recibido corresponde a 0 o 1 se usan regiones como rebanadas de pizza alrededor de  $0 \times q/2$ .

$$m_i = \text{dec}(\bar{m}_i) = \begin{cases} 0 & \text{si } -\left\lfloor \frac{q}{4} \right\rfloor \leq \bar{m}_i \leq \left\lfloor \frac{q}{4} \right\rfloor \\ 1 & \text{en otro caso} \end{cases}$$

Con  $q=61$ :

- $\lfloor \frac{q}{4} \rfloor = 15$
- todo valor entre 45 y 60 y entre 0 y 15 se decodifican a 0
- todo valor entre 16 y 45  $\rightarrow 1$

Esto hace la codificación tolerante al error generado en el cifrado LWE.

### Resumen del esquema de cifrado y descifrado Simple-LWE

El esquema Simple-LWE es una versión educativa y simplificada de un esquema basado en LWE. Solo permite cifrar un bit por vez pero nos sirve para entender como funcionan los cifrados basados en retículas.

Consta de 3 fases:

Generación de llaves

↓  
Cifrado

↓  
Descifrado

## ① Generación de llaves

### Simple-LWE Key Generation

**Output:** public key:  $k_{pub} = (\mathbf{t}, \mathbf{A})$  with  $\mathbf{t} \in \mathbb{Z}_q^k$  and  $\mathbf{A} \in \mathbb{Z}_q^{k \times n}$

**private key:**  $k_{pr} = \mathbf{s} \in \mathbb{Z}_q^n$

Pasos:

1. Choose  $n$  random vectors  $\mathbf{a}_i \in \mathbb{Z}_q^k$  and combine them in a matrix  $\mathbf{A} = (\mathbf{a}_1, \mathbf{a}_2, \dots, \mathbf{a}_n) \in \mathbb{Z}_q^{k \times n}$ .
2. Generate a random secret key  $\mathbf{s}$  from "small" integers.
3. Build a random error vector  $\mathbf{e}$  from "small" integers.
4. Compute  $\mathbf{t} = \mathbf{A} \cdot \mathbf{s} + \mathbf{e}$ .
5. Return the public key  $k_{pub} = (\mathbf{t}, \mathbf{A})$  and the private key  $k_{pr} = \mathbf{s}$ .

## ② Cifrado: El remitente usa la llave pública para cifrar un solo bit en $\{0, 1\}$

### Simple-LWE Encryption

**Input:** public key  $k_{pub} = (\mathbf{t}, \mathbf{A})$ , message  $m \in \{0, 1\}$

**Output:** ciphertext  $\mathbf{c} = (\mathbf{c}_{aux}, c_{msg})$  with  $\mathbf{c}_{aux} \in \mathbb{Z}_q^n$  and  $c_{msg} \in \mathbb{Z}_q$

Pasos

1. Sample small random integers into vectors  $\mathbf{r}$ ,  $\mathbf{e}_{aux}$  and a value  $e_{msg}$ .
2. Encode the message  $m$ :  $\bar{m} = \text{enc}(m) \in \mathbb{Z}_q$ .
3. Compute  $\mathbf{c}_{aux} = \mathbf{A}^T \cdot \mathbf{r} + \mathbf{e}_{aux}$ .
4. Compute  $c_{msg} = \mathbf{t}^T \cdot \mathbf{r} + e_{msg} + \bar{m}$ .
5. Return the ciphertext  $\mathbf{c} = (\mathbf{c}_{aux}, c_{msg})$ .

## ③ Descifrado:

El receptor usa la llave privada s para recuperar el bit.

$$\mathbf{t} = \mathbf{A} \cdot \mathbf{s} + \mathbf{e}$$

### Simple-LWE Decryption

**Input:** private key  $k_{pr} = \mathbf{s} \in \mathbb{Z}_q^n$ , ciphertext  $\mathbf{c} = (\mathbf{c}_{aux}, c_{msg})$

**Output:** message  $m \in \{0, 1\}$

Pasos:

1. Return message  $m = \text{dec}(c_{msg} - \mathbf{s}^T \cdot \mathbf{c}_{aux})$ .

usa la función que

es tolerante a errores para recuperar el bit

$\mathbf{s}^T \cdot \mathbf{A}^T \cdot \mathbf{r} + \mathbf{s}^T \cdot \mathbf{e}_{aux}$  se elimina porque conoce s y deja el mensaje más un ruido manejable

## Prueba de corrección y seguridad

Partan de la ecuación de generación de llave  
 $t = A \cdot s + e$

### Seguridad

Se consideran dos tipos de ataque

① El atacante quiere calcular la llave privada a partir de la pública:

El atacante quiere obtener  $s$ , lo cual sería resolver el problema LWE que se supone es difícil si los parámetros los escogimos bien.

② Recuperar el mensaje a partir del cifrado

El cifrado es el par

$$c_{aux} = A^T r + e_{aux}, \quad c_{msg} = t^T r + e_{msg} + m$$

• entender  $c_{aux}$ : el atacante no conoce

$r$  ni  $e_{aux} \Rightarrow$  problema LWE

• entender  $c_{msg} \Rightarrow$  problema LWE

Por lo tanto, tanto romper la llave como descifrar el mensaje son problemas LWE.

### Corrección:

Se puede demostrar que con alta probabilidad y errores suficientemente pequeños, al calcular  $c_{msg} - s^T c_{aux}$  se obtiene un valor cercano a  $m$  y la función de descodificación tamaña al bit correcto.

# LIMITACIONES DEL ESQUEMA SIMPLE-LWE

Solo es pedagógico no es práctico.

## ① Cifra solo 1 bit:

Para aplicaciones reales, por ej. cifrar una llave simétrica de 256 bits, hay que extender el esquema a múltiples bits de manera eficiente.

## ② Expansión del cifrado

Un solo bit se convierte en un cifrado con  $n+1=4$  elementos de  $\mathbb{Z}_6$ . (el ejemplo)

Cada valor necesita 6 bits para representar todos los posibles valores  $c_i \in \{0, \dots, 60\}$ . Entonces necesitamos un total 24 bits. Considerando que 1 bit  $\rightarrow$  24 bits en aplicaciones reales, esto es demasiado.

Mejora: reducir esta expansión.

## ③ Llaves muy grandes

El usa de matrices para llaves públicas involucra complejidad cuadrática en términos de almacenamiento y ancho de banda.

El esquema FROB KEM basado en LWE usa  $A \in \mathbb{Z}_q^{640 \times 640}$  por muy pequeño y  $\mathbb{Z}_q^{1344 \times 1344}$  para su conj. de parámetros más grande.

## ④ Distribución de los errores.

Nunca se especifica lo de "errores" pequeños.

Este es un problema no trivial y bastante estudiado.  
Se usa por ej. las distribuciones Gaussiana discreta  
o binomial.

## 12.2.3 The Ring Learning Problem with Errors

los esquemas modernos sustituyen las matrices por polinomios, lo cual reduce drásticamente el almacenamiento y la complejidad computacional.

**Definition 12.2.3** The ring  $R_q = \mathbb{Z}_q[x]/(x^n + 1)$

The polynomial ring  $\mathbb{Z}_q[x]/(x^n + 1)$  consists of all polynomials with a maximum degree of  $n - 1$  with coefficients from  $\mathbb{Z}_q$  and  $n$  being a power of two, i.e.,  $n = 2^i$ .

The ring operations addition, subtraction and multiplication are performed as regular polynomial arithmetic, with the results being reduced modulo the cyclotomic polynomial  $x^n + 1$ . All integer coefficients are reduced modulo  $q$ .

### Definición del problema Ring-LWE

Reemplazamos la matriz grande de LWE  
por un polinomio único  $a(x)$ .

**Definition 12.2.4** Ring-LWE Problem

Let  $R_q$  denote the ring  $\mathbb{Z}[x]_q/(x^n + 1)$ , where  $q$  is a prime and the positive integer  $n$  is a power of two. Given are polynomials  $\mathbf{a}$  and  $\mathbf{t} \in R_q$ .

Ring-LWE is the problem of determining a secret polynomial  $s \in R_q$  such that:

$$\mathbf{a}(x) \cdot s(x) + e(x) = \mathbf{t}(x)$$

where the error vector  $e$  is a polynomial in the ring  $R_q$  with small integer coefficients obtained from a discrete distribution  $D$ .

$a, t$   
tienen coef.  
grandes)

$s$  y  $e$  valores  
pequeños

# Relación entre LWE estándar y Ring-LWE

## Similitudes

- \* ambos generan retículos con buenas propiedades
- \* Resolver Ring-LWE se hace tan difícil como LWE

## Diferencias clave

### LWE:

- usa matriz  $A$  grande  $K \times n$
- muy costoso computacionalmente

### Ring-LWE

- Solo usa un polinomio  $a(x)$  de grado  $n-1$
- Reduce masivamente el costo computacional
- Ocupa menos memoria y ancho de banda

## Desventaja del Ring-LWE

Introduce más estructura algebraica, lo que abre la posibilidad teórica de ataques que la exploten.

No se conocen ataques prácticos que lo rompan con parámetros seguros.

→ Modulo-LWE: equilibrio, eficiencia y seguridad. Es la base de muchos esq post cuánticos.

