

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?

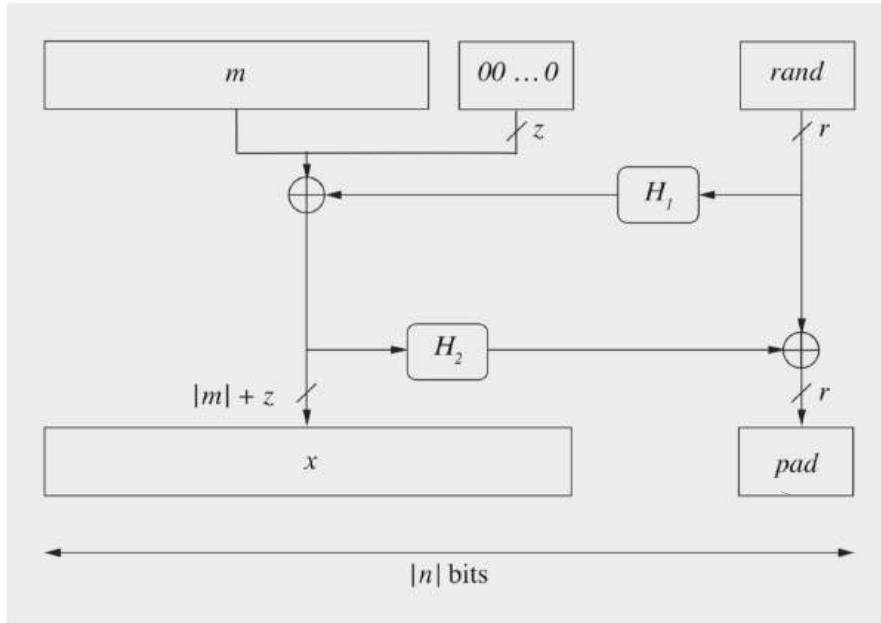


Fig. 7.3 Principle of the Optimal Asymmetric Encryption Padding (OAEP) scheme for a message m

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 collage, 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: $d_{k_{pr}}(y) = y^d \equiv x||pad \pmod n$
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?

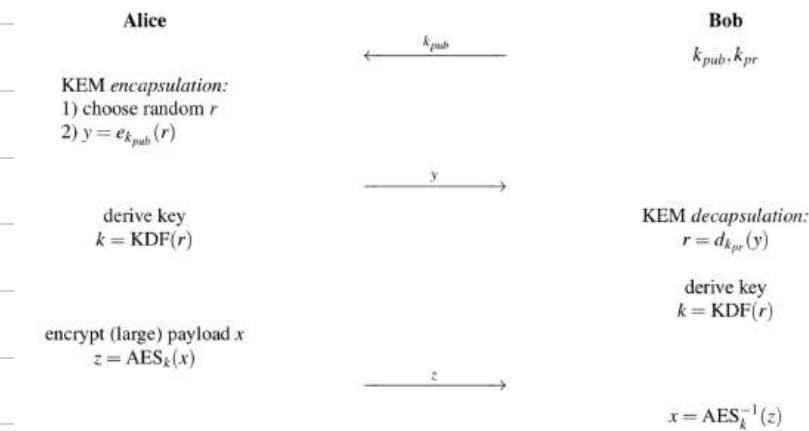


Fig. 7.4 Key encapsulation mechanism with public-key encryption, with AES being used as an example symmetric cipher

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)
describiendo 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 + 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 + multiplicar

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

- Si el bit es 1 \rightarrow cuadrar + multiplicar (act. más larga)

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

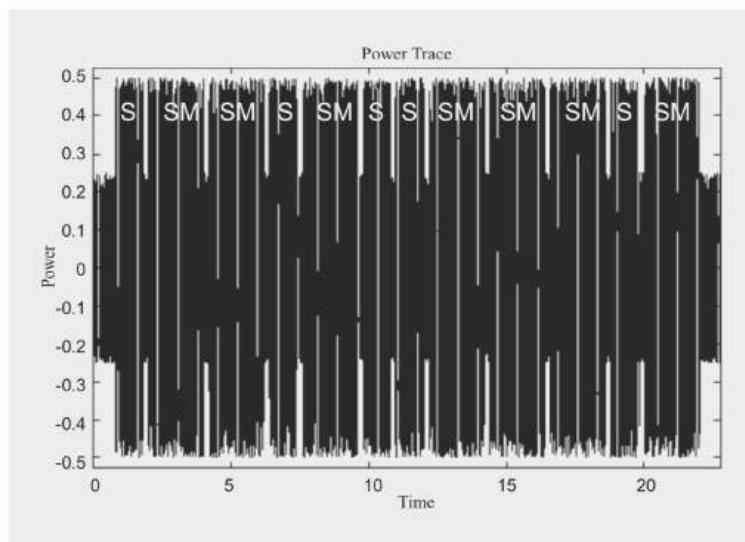


Fig. 7.5 The power trace of an RSA implementation

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 con 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 atajos para análisis de potencia o tiempo.
 - * A bajo nivel: se pueden optimizar los 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.