

IIC3253

Teoría de números y RSA

RSA: claves públicas y privadas

Las claves pública y privada de un usuario A son construidas de la siguiente forma:

1. Genere números primos distintos P y Q . Defina
$$N := P \cdot Q$$
2. Defina $\phi(N) := (P - 1) \cdot (Q - 1)$
3. Genere un número d tal que $MCD(d, \phi(N)) = 1$
4. Construya un número e tal que
$$e \cdot d \equiv 1 \pmod{\phi(N)}$$
5. Defina $S_A = (d, N)$ y $P_A = (e, N)$

RSA: funciones de cifrado y descifrado

Considere un usuario A con clave pública $P_A = (e, N)$ y clave secreta $S_A = (d, N)$


Se tiene que:

$$Enc_{P_A}(m) = m^e \bmod N$$

$$Dec_{S_A}(c) = c^d \bmod N$$

¿Por qué funciona RSA?

¿Cuáles son los espacios de mensajes, llaves y textos cifrados? 

¿Es cierto que $Dec_{S_A}(Enc_{P_A}(m)) = m$? 

¿Qué algoritmos de tiempo polinomial se debe tener para que se pueda ejecutar el protocolo?

¿De qué depende la seguridad de RSA? ¿Qué problemas no pueden ser resueltos en tiempo polinomial?

¿Qué algoritmos eficientes necesitamos para ejecutar RSA?

Generación aleatoria de números primos

Construcción de claves:

1. Genere números primos distintos P y Q . Defina

$$N := P \cdot Q$$

2. Defina $\phi(N) := (P - 1) \cdot (Q - 1)$

3. Genere un número d tal que $MCD(d, \phi(N)) = 1$

4. Construya un número e tal que

$$e \cdot d \equiv 1 \pmod{\phi(N)}$$

5. Defina $S_A = (d, N)$ y $P_A = (e, N)$

¿Qué algoritmos eficientes necesitamos para ejecutar RSA?

Operaciones básicas (multiplicación, suma, resto, ...)

Construcción de claves:

1. Genere números primos distintos P y Q . Defina

$$N := P \cdot Q$$

2. Defina $\phi(N) := (P - 1) \cdot (Q - 1)$

3. Genere un número d tal que $MCD(d, \phi(N)) = 1$

4. Construya un número e tal que

$$e \cdot d \equiv 1 \pmod{\phi(N)}$$

5. Defina $S_A = (d, N)$ y $P_A = (e, N)$

¿Qué algoritmos eficientes necesitamos para ejecutar RSA?

Máximo común divisor

Construcción de claves:

1. Genere números primos distintos P y Q . Defina

$$N := P \cdot Q$$

2. Defina $\phi(N) := (P - 1) \cdot (Q - 1)$

3. Genere un número d tal que $MCD(d, \phi(N)) = 1$

4. Construya un número e tal que

$$e \cdot d \equiv 1 \pmod{\phi(N)}$$

5. Defina $S_A = (d, N)$ y $P_A = (e, N)$

¿Qué algoritmos eficientes necesitamos para ejecutar RSA?

Generación de un primo relativo

Construcción de claves:

1. Genere números primos distintos P y Q . Defina

$$N := P \cdot Q$$

2. Defina $\phi(N) := (P - 1) \cdot (Q - 1)$

3. Genere un número d tal que $MCD(d, \phi(N)) = 1$

4. Construya un número e tal que

$$e \cdot d \equiv 1 \pmod{\phi(N)}$$

5. Defina $S_A = (d, N)$ y $P_A = (e, N)$

¿Qué algoritmos eficientes necesitamos para ejecutar RSA?

Algoritmo extendido de Euclides

Construcción de claves:

1. Genere números primos distintos P y Q . Defina

$$N := P \cdot Q$$

2. Defina $\phi(N) := (P - 1) \cdot (Q - 1)$

3. Genere un número d tal que $MCD(d, \phi(N)) = 1$

4. Construya un número e tal que

$$e \cdot d \equiv 1 \pmod{\phi(N)}$$

5. Defina $S_A = (d, N)$ y $P_A = (e, N)$

¿Qué algoritmos eficientes necesitamos para ejecutar RSA?

Funciones de cifrado y descifrado:

$$Enc_{P_A}(m) = m^e \bmod N$$

$$Dec_{S_A}(c) = c^d \bmod N$$



Exponenciación rápida

Algoritmos eficientes para ejecutar RSA

Tenemos que resolver dos problemas: cómo generar un número primo al azar, y cómo generar un primo relativo al azar

¿Cuál es la densidad de los números primos?

Sea $\pi(n)$ la cantidad de números primos menores o iguales a n

- Por ejemplo, $\pi(10) = 4$ y $\pi(19) = 8$

Teorema: $\lim_{n \rightarrow \infty} \frac{\pi(n)}{\frac{n}{\ln(n)}} = 1$

Por lo tanto: $\pi(n) \approx \frac{n}{\ln(n)}$

El teorema en la práctica

n	$\pi(n)$	$\frac{n}{\ln(n)}$	Error %
10	4	4.342	8.573
100	25	21.714	13.141
1000	168	144.764	13.830
10000	1229	1085.736	11.656
100000	9592	8685.889	9.446
1000000	78498	72382.413	7.790
10000000	664579	620420.688	6.644

¿Cuál es la probabilidad de obtener un número primo al azar?

$$\Pr_{x \sim \mathbb{U}(1,n)}(x \text{ sea primo}) = \frac{\text{casos favorables}}{\text{casos totales}}$$

¿Cuál es la probabilidad de obtener un número primo al azar?

$$\Pr_{x \sim \mathbb{U}(1,n)}(x \text{ sea primo}) = \frac{\text{casos favorables}}{n}$$

¿Cuál es la probabilidad de
obtener un número primo
al azar?

$$\Pr_{x \sim \mathbb{U}(1,n)}(x \text{ sea primo}) \approx \frac{\frac{n}{\ln(n)}}{n}$$

¿Cuál es la probabilidad de obtener un número primo al azar?

$$\Pr_{x \sim \mathbb{U}(1,n)}(x \text{ sea primo}) \approx \frac{1}{\ln(n)}$$

¿Es esta una probabilidad alta? ¿Podemos obtener un número primo de 400 dígitos en poco tiempo?

¿Cuál es la probabilidad de obtener un número primo al azar?

La probabilidad de obtener un número primo de 400 dígitos:

$$\Pr_{x \sim \mathbb{U}(10^{399}, 10^{400} - 1)} \approx \frac{\frac{10^{400} - 1}{\ln(10^{400} - 1)} - \frac{10^{399} - 1}{\ln(10^{399} - 1)}}{10^{400} - 10^{399}}$$

Esta probabilidad es aproximadamente 0.001085

- Por lo tanto, necesitamos aproximadamente 922 intentos para obtener un número primo



**Conclusión: para generar
números primos al azar nos
basta con encontrar un
algoritmo eficiente para
verificar si un número es
primo**

Antes de estudiar un algoritmo de primalidad

Vamos a ver cómo se puede generar un primo relativo al azar

Y vamos a responder una de las preguntas pendientes:
¿de qué depende la seguridad de RSA?

- ¿Qué problemas no pueden ser resueltos en tiempo polinomial para que este protocolo sea seguro?

Generando un primo relativo al azar

Queremos generar un primo relativo a de un número dado n

Sea $\phi(n)$ la cardinalidad del conjunto

$$\{a \in \{0, \dots, n-1\} \mid \text{MCD}(a, n) = 1\}$$

Generando un primo relativo al azar

Por ejemplo, si $N = P \cdot Q$ con P y Q primos distintos, entonces $\phi(N) = (P - 1) \cdot (Q - 1)$

- Por eso usamos la notación $\phi(N)$ en RSA

$\phi(n)$ es llamada la función ϕ de Euler

Generando un primo relativo al azar

¿Cuán grande es el valor de $\phi(n)$?

Teorema: $\phi(n)$ es $\Omega\left(\frac{n}{\log(\log(n))}\right)$

Podemos entonces generar un primo relativo a n usando el mismo argumento que para los primos

- Generamos números al azar $a \in \{0, \dots, n-1\}$ y verificamos si $MCD(a, n) = 1$

¿De qué depende la seguridad de RSA?

La respuesta depende de la definición de seguridad

¿De qué depende la seguridad de RSA?

Problema de RSA: dado una clave pública (e, N) y un mensaje cifrado $c \in \{0, \dots, N - 1\}$, encontrar $m \in \{0, \dots, N - 1\}$ tal que:

$$c = m^e \bmod N$$

Corresponde a encontrar la e -ésima raíz de c en módulo n

- La e -ésima raíz $m \in \{0, \dots, N - 1\}$ es única dada la definición de e

¿De qué depende la seguridad de RSA?

No se conoce un algoritmo de tiempo polinomial que resuelva el problema de RSA

Si existe un algoritmo polinomial para factorizar un número compuesto, entonces existe un algoritmo polinomial para resolver el problema de RSA

- Pero no se sabe si la dirección inversa es cierta

¿De qué depende la seguridad de RSA?

Problema fuerte de RSA: dado una clave pública (e, N) , encontrar d tal que

$$d \cdot e \equiv 1 \pmod{\phi(N)}$$

Corresponde a encontrar la clave privada a partir de la clave pública

¿De qué depende la seguridad de RSA?

No se conoce un algoritmo de tiempo polinomial que resuelva el problema fuerte de RSA

- Si existe un algoritmo polinomial para el problema fuerte de RSA, entonces existe un algoritmo polinomial para el problema de RSA

Existe un algoritmo polinomial para factorizar un número compuesto $N = P \cdot Q$ si y sólo si existe un algoritmo polinomial para resolver el problema fuerte de RSA

La última pregunta a responder

¿Cómo se puede verificar de manera eficiente si un número es primo?

Vamos a estudiar las ideas detrás de un test aleatorizado de tiempo polinomial para verificar si un número es primo

- Hasta el día de hoy, todos los test de primalidad realmente eficientes son aleatorizados

Test de primalidad y el pequeño teorema de Fermat

Desde ahora en adelante suponga que n es un número impar

Sea $W_n = \{a \in \{1, \dots, n-1\} \mid a^{n-1} \equiv 1 \pmod{n}\}$

El pequeño teorema de Fermat nos dice que si p es primo, entonces $|W_p| = p - 1$

Test de primalidad y el pequeño teorema de Fermat

¿Qué podemos decir sobre W_n si n es un número compuesto?

Test de primalidad y el pequeño teorema de Fermat

¿Qué podemos decir sobre W_n si n es un número compuesto?

Demuestre que si n es compuesto, entonces

$$|W_n| < n - 1$$

¿Pero cuán pequeño es W_n ?

Test de primalidad: primer intento

Suponga que si n es compuesto, entonces

$$|W_n| \leq \frac{n-1}{2}$$

¿Cómo puede usar este resultado para construir un test de primalidad?

Test de primalidad: primer intento

Dados n y k :

1. Elija a_1, \dots, a_k en $\{1, \dots, n - 1\}$ con distribución uniforme y de manera independiente
2. Calcule $b_i = a_i^{n-1} \bmod n$ para cada $i \in \{1, \dots, k\}$
3. Si $b_i \neq 1$ para algún $i \in \{1, \dots, k\}$, entonces retorne **false**. En el caso contrario, retorne **true**.

Test de primalidad: primer intento

¿Con cuáles entradas se puede equivocar el algoritmo? ¿Cuál es la probabilidad de error del algoritmo?

¿Utilizaría este algoritmo en la práctica?

Pero ...

No es cierto que si n es compuesto, entonces

$$|W_n| \leq \frac{n-1}{2}$$

Algunos contraejemplos:

$$|W_{561}| = 320$$

$$|W_{1729}| = 1296$$

$$|W_{2465}| = 1792$$

Pero ...

Se puede demostrar que existe una cantidad infinita de números compuestos n tales que:

$$|W_n| > \frac{n-1}{2}$$

Pero ...

Se puede demostrar que existe una cantidad infinita de números compuestos n tales que:

$$|W_n| > \frac{n-1}{2}$$

¿Cómo podemos solucionar el problema?

Una segunda noción de testigo

Vamos a considerar testigos de la forma

$$a^{\frac{n-1}{2}} \bmod n$$

Estos testigos están bien definidos ya que n es impar

¿Qué valores toman los testigos si n es un número primo? ¿Qué valores toman si n es compuesto?

Un ejemplo para el caso primo

Para $p = 7$:

$$1^3 \bmod 7 = 1 \bmod 7 = 1$$

$$2^3 \bmod 7 = 8 \bmod 7 = 1$$

$$3^3 \bmod 7 = 27 \bmod 7 = 6$$

$$4^3 \bmod 7 = 64 \bmod 7 = 1$$

$$5^3 \bmod 7 = 125 \bmod 7 = 6$$

$$6^3 \bmod 7 = 216 \bmod 7 = 6$$

Vale decir, $a^3 \equiv 1 \bmod 7$ o $a^3 \equiv -1 \bmod 7$

Un ejemplo para el caso primo

¿Se puede generalizar el resultado a cualquier número primo impar?

Definiendo una segunda noción de testigo

Si p es un número primo impar y $a \in \{1, \dots, p-1\}$,
entonces

$$a^{\frac{p-1}{2}} \equiv 1 \pmod{p} \quad \text{o} \quad a^{\frac{p-1}{2}} \equiv -1 \pmod{p}$$

Veamos la demostración en la pizarra

- Usted tiene las herramientas necesarias para demostrar este teorema

Definiendo una segunda noción de testigo

Sea

$$W_n^+ = \{a \in \{1, \dots, n-1\} \mid a^{\frac{n-1}{2}} \equiv 1 \pmod{n}\}$$

$$W_n^- = \{a \in \{1, \dots, n-1\} \mid a^{\frac{n-1}{2}} \equiv -1 \pmod{n}\}$$

Definiendo una segunda noción de testigo

Sea

$$W_n^+ = \{a \in \{1, \dots, n-1\} \mid a^{\frac{n-1}{2}} \equiv 1 \pmod{n}\}$$

$$W_n^- = \{a \in \{1, \dots, n-1\} \mid a^{\frac{n-1}{2}} \equiv -1 \pmod{n}\}$$

Teorema: si p es un número primo impar, entonces

$$|W_p^+| = |W_p^-| = \frac{p-1}{2}$$

¿Cómo se demuestra la caracterización?

Tenemos que ir al mundo de los polinomios en módulo un número dado

Sea $r(x) = x^2 - 1$

¿Cuántas raíces tiene $r(x)$ en módulo 7? **2**

Para $a = 1$ y $a = 6$ se tiene que $r(a) \equiv 0 \pmod{7}$

¿Cómo se demuestra la caracterización?

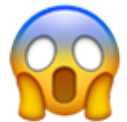
¿Cuántas raíces tiene $r(x)$ en módulo 77?

¿Cómo se demuestra la caracterización?

¿Cuántas raíces tiene $r(x)$ en módulo 77? **4**

1, 34, 43 y 76 son raíces del polinomio

¿Qué pasó? ¿Por qué la diferencia entre 7 y 77?



¿Cómo se demuestra la caracterización?

Los polinomios tienen las propiedades usuales cuando los evaluamos en módulo un primo

Teorema: si p es un número primo y $r(x)$ es un polinomio de grado k , entonces $r(x)$ tiene a lo más k raíces en módulo p

¿Cómo se demuestra la caracterización?

Con esto podemos demostrar la caracterización de W_p^+ y W_p^- para p primo impar

Vamos nuevamente a la pizarra

- Y nuevamente tiene las herramientas necesarias para demostrar el teorema

¿Cómo se ven los conjuntos W_n^+ y W_n^- en general?

	1	-1
641	320	320
561	160	0
1729	1296	0
2465	1792	0
1891	225	225
2047	121	121
2701	324	324

¿Cómo se ven los conjuntos

W_n^+ y W_n^- en general?

	1	-1
641	320	320

número primo p

$\frac{p-1}{2}$ elementos

The diagram illustrates the structure of the sets W_n^+ and W_n^- for a prime $p=641$. It consists of a table with two columns labeled '1' and '-1', and three rows. The first row contains the values 641, 320, and 320. The second and third rows are empty. Red circles highlight the values 641, 320, and 320. Red arrows point from these circles to the text 'número primo p ' and ' $\frac{p-1}{2}$ elementos'.

¿Cómo se ven los conjuntos

W_n^+ y W_n^- en general?

1	-1
---	----

561	160	0
1729	1296	0
2465	1792	0

números compuestos

¿Cómo se ven los conjuntos

W_n^+ y W_n^- en general?

1	-1
---	----

números compuestos n

$$|W_n^+| + |W_n^-| \leq \frac{n-1}{2}$$

1891	225	225
2047	121	121
2701	324	324

¿Cómo se ven los conjuntos

W_n^+ y W_n^- en general?

números compuestos

	1	-1
17098369	16859136	0
43331401	42230160	0
56052361	55566000	0

¿Cómo se ven los conjuntos

W_n^+ y W_n^- en general?

1	-1
---	----

17098369	16859136	0
43331401	42230160	0
56052361	55566000	0

0.986



¿Cómo se ven los conjuntos

W_n^+ y W_n^- en general?

1	-1
---	----

17098369	16859136	0
43331401	42230160	0
56052361	55566000	0

0.991



La formalización de los patrones anteriores

Teorema: Sea $n = n_1 \cdot n_2$, donde $n_1, n_2 \geq 3$ y $MCD(n_1, n_2) = 1$. Si existe $a \in \{2, \dots, n-1\}$ tal que $a^{\frac{n-1}{2}} \equiv -1 \pmod{n}$, entonces:

$$|W_n^+| + |W_n^-| \leq \frac{n-1}{2}$$

Y un último ingrediente

Necesitamos distinguir el caso cuando un número $n \geq 2$ no es primo porque es potencial no trivial de otro número

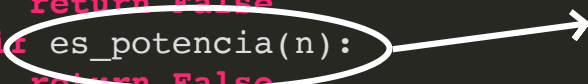
- Vale decir, $n = a^b$ con $b \geq 2$

Por ejemplo, 36 no es primo ya que $36 = 6^2$

Veamos en la pizarra un algoritmo de tiempo polinomial para resolver este problema

Finalmente el código

```
1 def test_primalidad(n: int, k: int) -> bool:
2     """
3     Argumentos :
4         n: int - n >= 1
5         k: int - k >= 1
6     Retorna :
7         bool - True si n es un numero primo, y False en caso contrario.
8         La probabilidad de error del test es menor o igual a 2**(-k),
9         y esta basado en el test de primalidad de Solovay-Strassen
10    """
11    if n == 1:
12        return False
13    elif n == 2:
14        return True
15    elif n % 2 == 0:
16        return False
17    elif es_potencia(n):
18        return False
19    else:
```



Verifica si n es potencia no trivial de un número

Finalmente el código

```
1      neg = 0
2      for i in range(1, k+1):
3          a = random.randint(2, n-1)
4          if mcd(a, n) > 1:
5              return False
6          else:
7              b = pow(a, (n-1)//2, n)
8              if b == n - 1:
9                  neg = neg + 1
10             elif b != 1:
11                 return False
12     if neg > 0:
13         return True
14     else:
15         return False
```

Y en la práctica ...

```
1  if __name__ == "__main__":
2      P =
594363236250881445679738443300610044271230329506694061456935493654987499908267
837823162990672937913416793547138262131162027654525159743671145416885026759510
967807798396037679273587887606706633886423937222779033920335019140885692470045
389062224534954730489613866855218857728804741777937870098279279181986655311360
896681010943076506752842990211660721362674656217273071452543976542283204562818
9761714003
3      Q =
594853230520675412480703452656865320286661579071458125732625920078118304621513
327159370949369506632630618864234462347497091715658932838218340863551767851586
359234950892172187401638485582112316673768053774021786114956665205696743968971
215530824202884686449465635361231768163225461580241273009020222619625670052199
141974408714888109491926351891091769195114042400795276168620861606789205367727
8599271911
4      t1 = time.time()
5      rP = test_primalidad(P, 100)
6      t2 = time.time()
7      rQ = test_primalidad(Q, 100)
8      t3 = time.time()
9      rPQ = test_primalidad(P*Q, 100)
10     t4 = time.time()
11     print("P: ", rP, "- tiempo: ", t2 - t1, " segundos")
12     print("Q: ", rQ, "- tiempo: ", t3 - t2, " segundos")
13     print("P*Q: ", rPQ, "- tiempo: ", t4 - t3, " segundos")
```

Y en la práctica ...

P: True - tiempo: 1.2840402126312256 segundos

Q: True - tiempo: 1.2941510677337646 segundos

P*Q: False - tiempo: 0.343120813369751 segundos

¿Cuál es la probabilidad de error del algoritmo?

Muestre que la probabilidad de error del algoritmo está acotado superiormente por $\left(\frac{1}{2}\right)^k$

¿Utilizaría este algoritmo en la práctica?

¿Cuál es la probabilidad de error del algoritmo?

```
1      neg = 0
2      for i in range(1, k+1):
3          a = random.randint(2, n-1)
4          if mcd(a,n) > 1:
5              return False
6          else:
7              b = pow(a, (n-1)//2, n)
8              if b == n - 1:
9                  neg = neg + 1
10             elif b != 1:
11                 return False
12             if neg > 0:
13                 return True
14             else:
15                 return False
```

El algoritmo solo puede equivocarse en este paso

¿Cuál es la probabilidad de error del algoritmo?

¿Cómo se puede equivocar el algoritmo si n es primo? ¿Cuál es la probabilidad de retornar **false** en este caso?

¿Cómo se puede equivocar el algoritmo si n es compuesto? ¿Cuál es la probabilidad de retornar **true** en este caso?