Una Introducción Matemática a la Criptografía (para mis alumnos de Álgebra I)

Pablo De Nápoli

31 de mayo de 2014

¿ Qué es la Criptografía ?

$$\kappa\rho\upsilon\pi\tau o\varsigma = \text{oculto}$$

$$\gamma\rho\alpha\phi\omega = \text{escribir}$$

- La criptografía es el arte de escribir mensajes ocultos.
- Históricamente la criptografía tiene su origen en aplicaciones militares, pero hoy en día se utiliza habitualmente en aplicaciones de computación.
- Aclaración: La RAE no incluye en su diccionario encriptar/desencriptar , pero estos términos son de uso común en informática (del inglés crypt/decrypt). Se usan como sinónimos de cifrar/descrifrar .

Algoritmos Criptográficos

- Un algoritmo criptográfico es un método matemático para convertir un mensaje en texto plano (que cualquiera puede leer) en un texto cifrado (que sólo es legible para el que sabe como descifrarlo).

- Generalmente los algoritmos de encriptación son públicamente conocidos, pero para poder descifrar el mensaje es necesario conocer además una clave.

Aplicaciones de la Criptografía

Los algoritmos de encriptación se usan todos los días "detrás de escena" en aplicaciones y protocolos tales como

- Secure Shell (SSH): acceso remoto a otras computadoras.
- HTTPS (Sitios web seguros, por ejemplo homebanking)
- GPG: The GNU Privacy Guard: usado por ej. para correo electrónico encriptado.
- IPSec: redes privadas virtuales.

Dos Tipos de Algoritmos

- Algoritmos convencionales o simétricos: quien envía un mensaje encriptado utiliza el mismo procedimiento y la misma clave que su receptor para desencriptarlo.

- Algoritmos de clave pública o asimétricos: se utilizan claves diferentes para encriptar y para desencriptar.

¿Cómo representan las computadoras la información?

- Desde el punto de vista matemático, siempre puede suponerse que el mensaje que se quiere transmitir es un número.

- Internamente las computadoras representan todos los caracteres (letras, números o simbolos especiales) como números binarios de acuerdo al código ASCII:

Caracter	Número Binario	Eq. Decimal
espacio	00100000	32
'!'	00100001	33
'A'	01000001	65
'a'	01100001	97

Funciones de una sóla vía

Los algoritmos de clave pública se basan el hecho de en que existen operaciones matemáticas que se pueden realizar rápidamente en una computadora, para las que la operación inversa (aunque teóricamente es posible) demanda un tiempo de procesamiento tal que la hacen prácticamente imposible.

Ej: dados dos números primos grandes p y q, es muy fácil calcular $N = pq$.

Pero conocido N, es prácticamente imposible calcular p y q si ambos son suficientemente grandes.

Otras aplicaciones de los algoritmos de llave pública

- Los algoritmos de clave pública resuelven el problema de la distribución de claves.
- Los algoritmos de clave pública también pueden usarse para autentificación (firma digital).

Dos algoritmos criptográficos

- En esta clase veremos como funciona el algorítmo de clave pública más utilizado, conocido como el algoritmo RSA (por las siglas de sus creadores).
- El algoritmo RSA fue inventado por Ron Rivest, Adi Shamir y Len Adleman trabajando en el M.I.T. en 1977.
- También veremos como funciona el algoritmo de intercambio de claves de Diffie-Hellman (1976)
- El algoritmo de Diffie-Hellman y el algoritmo RSA emplean varias herramientas de la teoría de números.

Gauss dixit...

"La matemática es la reina de las ciencias, y la teoría de los números es la reina de la matemática." (Carl F. Gauss, 1777–1855)

La Exponencial Modular

Ambos algoritmos utilizan como función de una sóla vía, la función exponencial modular:

$$x \mapsto a^x \pmod{n}$$

que dado x devuelve el resto de a^x módulo n.
Es posible calcular a^x en forma muy eficiente utilizando el método de cuadrados repetidos: por ejemplo calculemos

$$1001^{19} \pmod{301}$$

Para ello, escribimos el exponente en binario:

$$19 = 2^4 + 2 + 1 = 10011_2$$

$$1001^2 \equiv 273 \pmod{301}$$

$$1001^4 \equiv 273^2 \equiv 182 \pmod{301}$$

$$1001^8 \equiv 182^2 \equiv 14 \pmod{301}$$

$$1001^{16} \equiv 14^2 \equiv 196 \pmod{301}$$

$$1001^{18} \equiv 1001^{16} \times 1001^2 \equiv 196 \times 273 \equiv 231 \pmod{301}$$

$$1001^{19} \equiv 1001^{18} \times 1001 \equiv 63 \pmod{301}$$

Raices Primitivas

Si p es un número primo, decimos que g es una raíz primitiva módulo p, si g^n módulo p recorre los valores $1, 2, \ldots, p-1$ cuando n recorre esos mismos valores.
Por ejemplo: si elejimos $p = 23$, $g = 5$ es una raíz primitiva ya que tenemos la siguiente tabla de valores:

n	1	2	3	4	5	6	7	8	9	10	11
g^n	5	2	10	4	20	8	17	16	11	9	22

n	12	13	14	15	16	17	18	19	20	21	22
g^n	18	21	13	19	3	15	6	7	12	14	1

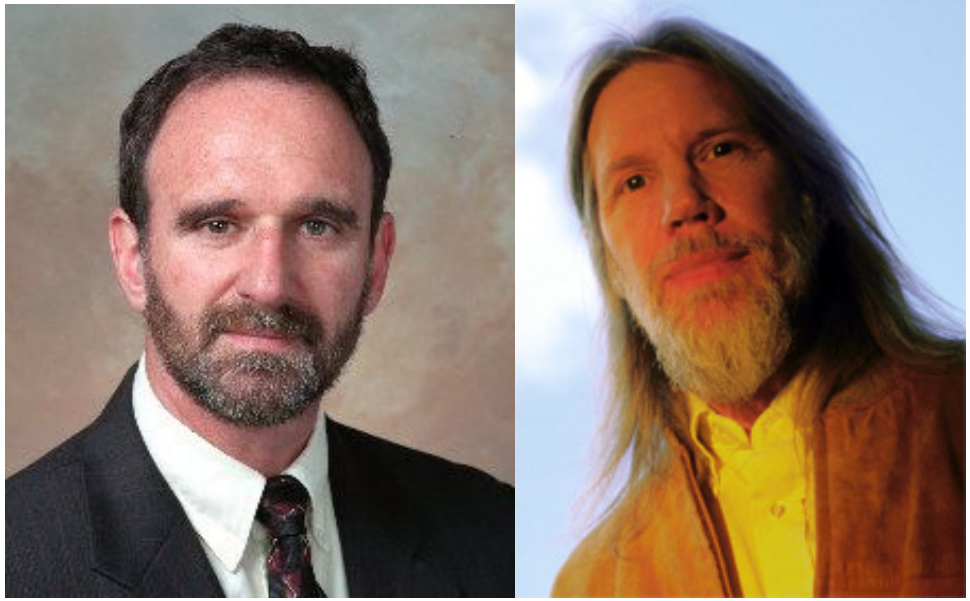
Para todo primo p, existen raices primitivas.

Una cita de Jacobi

"Fourier tenía la opinión de que el principal objetivo de la matemática era la utilidad pública y la explicación de los fenómenos naturales; pero un filósofo como él debería haber sabido que el único objetivo de la ciencia es el honor del espíritu humano, y que bajo este título una pregunta sobre los números es tan valiosa como una pregunta sobre el sistema del mundo."
(Carl Gustav Jacobi,1804–1851)

Parte I

El Algoritmo de Diffie-Hellman



El algoritmo de Diffie-Hellman

Supongamos que dos personas, Alicia y Benito deben ponerse de acuerdo en una clave pero no disponen de un canal seguro para intercambiar la clave.

Entonces pueden hacer lo siguiente:

- Acuerdan (¡en público!) un número primo p (grande) y una base g, idealmente una raíz primitiva de p.

  A modo de ejemplo, supongamos que eligen $p = 23$ y $g = 5$.

Cada uno por su lado . . .

- Alicia elige una clave secreta a (por ej. $a = 6$), calcula g^a (módulo p) y da a conocer el resultado a Benito (¡en público!). En el ejemplo:

$$5^6 \equiv 8 \pmod{23}$$

- Benito hace lo mismo, elije una clave secreta b (por ejemplo $b = 15$) calcula g^b (módulo p) y comunica el resultado a Alicia (¡en público!). En el ejemplo:

$$5^{15} \equiv 19 \pmod{23}$$

Un secreto en común

- Tanto Alicia como Benito pueden calcular g^{ab} módulo p: han quedado con un secreto en común. Alicia (que conoce a y g^b calcula

$$(g^b)^a \equiv g^{ab} \pmod{p}$$

  En el ejemplo: $19^6 \equiv 2 \pmod{23}$
- Mientras que Benito (que conoce g^a y b) calcula:

$$(g^a)^b \equiv g^{ab} \pmod{p}$$

  En el ejemplo: $8^{15} \equiv 2 \pmod{23}$

¿Es seguro?

La seguridad del algoritmo de Diffie-Hellman depende de que conociendo por ejemplo $y = g^a$ (o g^b) no sea posible determinar a (o b), no sea posible (desde el punto de vista práctico) resolver la ecuación de congruencia:

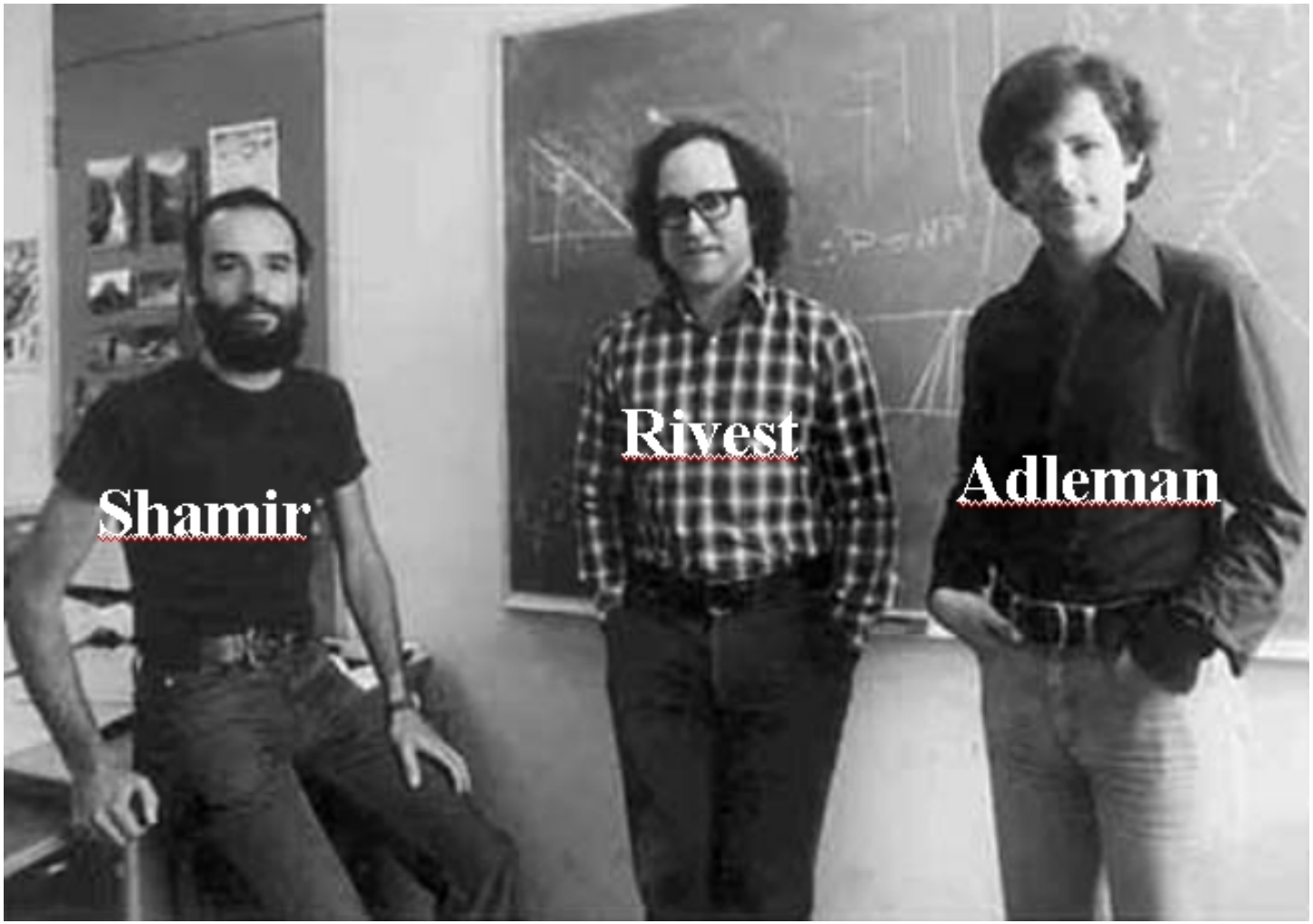
$$g^x \equiv y \pmod{p}$$

(Problema del logaritmo discreto).

Notemos que si g es una raíz primitiva de p este problema siempre tiene una solución, pero no se conoce ningún algoritmo eficiente (esto es: con complejidad polinomial en el número de bits de los datos) para resolverlo, si p y g son grandes).

Parte II

El Algoritmo RSA


Shamir
Rivest
Adleman

Alicia hace pública su clave

- Alicia elige al azar dos primos enormes p y q ($p \neq q$), y calcula $N = pq$.

  A los fijes de un ejemplo, elijamos $p = 17$ y $q = 41$. Entonces $N = 697$.

- También elige (al azar) e que no tenga factores en común con $f = \varphi(N) = (p-1)(q-1)$
  En nuestro ejemplo elegimos
  $e = 231 \quad f = 640$

- Alicia hace públicos N y e: son su clave pública.

  Alicia: Mi clave pública es el par (697,231)

El Algoritmo de Euclides

Para chequear que f y e no tengan factores comunes, se encuentra su máximo común divisor utilizando el algoritmo de Euclides (aprox. 325-265 AC):

$$
\begin{array}{rcrcrcr}
640 & = & 231 & * & 2 & + & 178 \\
231 & = & 178 & * & 1 & + & 53 \\
178 & = & 53 & * & 3 & + & 19 \\
53 & = & 19 & * & 2 & + & 15 \\
19 & = & 15 & * & 1 & + & 4 \\
15 & = & 4 & * & 3 & + & 3 \\
4 & = & 3 & * & 1 & + & 1 \\
3 & = & 1 & * & 3 & + & 0
\end{array}
$$

último resto no nulo = maximo común divisor.

Una consecuencia del algoritmo de Euclides

Como vimos en la materia, el máximo común divisor entre f y e siempre se pueden escribir en la forma

$$af + de$$

donde a y d son dos enteros.

En particular, esto es verdadero para al máximo común divisor entre e y f.

En nuestro ejemplo

$$61 * 640 + (-169) * 231 = 1$$

El secreto de Alicia

Así pues, vemos que existen enteros a y d tales que:

$$af + de = \text{mcd}(e, f) = 1$$

En nuestro ejemplo: $a = 61$, $d = -169$.
Por lo tanto, en particular existirá (¡y podemos calcularlo!) un entero d tal que:

$$de \equiv 1 \pmod{f}$$

Alicia mantiene d en secreto: es su clave privada.

Como preferimos no trabajar con números negativos, hagamos un pequeño truco:

$$-169 \equiv -169 + 640 \equiv 471 \pmod{640}$$

por lo tanto si elegimos $d = 471$ se seguirá verificando la relación

$$de \equiv 1 \pmod{640}$$

Los mensajes son números

Es posible pensar el conjunto de números

$$\{0, 1, 2, \ldots, N-1\}$$

como un alfabeto, y representar el mensaje como un número (o varios) de ese alfabeto. Por lo tanto, podemos pensar que el mensaje a transmitir es un número m de la aritmética modular módulo N.

Encriptando con el RSA

Cuando Benito quiere enviarle un mensaje m a Alicia, busca su clave pública (e, N) y calcula

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

c es el mensaje cifrado.

En el ejemplo, si Benito quiere enviar el mensaje $m = 12$ a Alicia, calcula

$$12^e = 12^{231} \equiv 466 \pmod{697}$$

y envia el mensaje cifrado $c = 466$ a Alicia.

¡Alicia sabe matemática!

Para desencriptar el mensaje, Alicia utilizará el siguiente teorema:

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

Prueba: Como hemos elegido d de modo que

$$de \equiv 1 \pmod{f}$$

existe un entero k tal que

$$de = 1 + kf \text{ donde } f = \varphi(N) = (p-1)(q-1)$$

por lo que

$$(m^e)^d \equiv m^{de} \equiv m(m^f)^k \pmod{N}$$

El teorema del RSA

Para probar el teorema consideramos cuatro casos:

- Si $p \not| m$ y $q \not| m \Rightarrow$ por el teorema de Fermat-Euler

$$m^f = 1 \pmod{N} \Rightarrow (m^e)^d = m \pmod{N}$$

- Si $p|m$ y $q \not| m$,

$$m \equiv 0 \pmod{p} \Rightarrow (m^e)^d \equiv 0 \equiv m \pmod{p}$$

$$m^{q-1} \equiv 1 \pmod{q} (\text{ por el teorema de Fermat})$$

$$\Rightarrow m^f \equiv 1 \pmod{q} \Rightarrow (m^e)^d \equiv m \pmod{q}$$

El teorema del RSA

y como p y q son coprimos, se deduce que

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

- Si $q|m$ pero $p \not| m$ es análogo (intercambiamos los roles)
- Finalmente si $p|m$ y $q|m$, $m \equiv 0 \pmod{N}$ luego

$$(m^e)^d \equiv 0 \equiv m \pmod{N}$$

En cualquiera de los cuatro casos probamos que

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

Como desencripta Alicia

- Deducimos que Alicia puede descifrar el mensaje de Benito, calculando

$$c^d \pmod{N}$$

En ejemplo, Alicia recibe el mensaje $c = 466$ de Benito, y calcula

$$466^{471} \equiv 12 (\text{mod } 697)$$

- Sólo alicia que conoce $f = \varphi(N) = (p-1)(q-1)$, puede desencriptar el mensaje.

¿Es seguro?

La seguridad del Algoritmo RSA depende de que aún conociendo N, no sea posible calcular los factores p y q, y por lo tanto f.

Es decir, depende de que no sea posible (desde el punto de vista práctico) factorizar el número N.

Actualmente no se conoce ningún algoritmo eficiente (o sea, con complejidad polinomial en el número de bits del número N) para factorizar números grandes, y se conjetura que tal algoritmo no existe.

Notemos, sin embargo que sí existe un algoritmo para decidir si un número es primo o no en tiempo polinomial (¡sin calcular sus factores!).

El RSA-576

RSA Security, Inc. organiza un concurso para factorizar números primos, con la finalidad de verificar la seguridad del RSA.
Por ejemplo, el $RSA-576$ (576 bits de longitud, 174 digitos decimales) ha sido factorizado

Y sus factores primos son

p= 39807508642406493739712550055038649119906436234252670840638518957594638895726176858331 7

q= 47277214610743530253622307197304822463291469530209711645985217113052071125636359039752 7

Un comentario final:

- En la práctica, que un programa de criptografía sea seguro o no, no depende sólo del algoritmo matemático que emplea, sino de muchos detalles que hacen a su implementación.

- Sólo pueden considerarse seguras las implementaciones de los algoritmos criptográficos para las que está disponible el código fuente (o sea, el texto del programa en un lenguaje comprensible para los seres humanos), porque sin él no es posible auditar el código para verificar que el programa no tenga puertas traseras, u otros defectos que lo tornen inseguro.

Parte III

Bonus Trac: Una implementación (en C++)

IARSAC

En

https://github.com/pdenapo/iarsac

pueden encontrar el código fuente de IARSAC (una Implementación del Algoritmo RSA en C++) que hice con fines didácticos, utilizando la librería NTL, disponible en

http://www.shoup.net/ntl/

¡No la usen para propósitos serios! Recomiendo para ello GNU Privacy Guard (GPG), Open SSL, etc.

Interface de IARSAC

```
class encriptador_rsa
{

public:

  void generar_claves(long longitud);

  void cifrar(istream& texto_plano,ostream& texto_encriptado
      );

  void descifrar(istream& texto_encriptado,ostream&
      texto_plano);

  void establecer_parametros();

  ZZ N;
  ZZ clave_publica;
  ZZ clave_privada;
};
```

Generación de claves en IARSAC

```
ZZ p,q,f;

p = GenPrime_ZZ(longitud);
do
{
 q = GenPrime_ZZ(longitud);}
while (p==q);

N = p*q;
f = (p-1)*(q-1);

do {
clave_publica = RandomLen_ZZ(longitud)%N;
} while (GCD(clave_publica,f)!=1);

 clave_privada = InvMod(clave_publica,f);
```

Como encripta y desencripta IARSAC

Para encriptar IARSAC usa:

```
numero_encriptado = PowerMod(numero_plano,clave_publica,N);
```

Para desencriptar IARSAC usa:

```
numero_plano = PowerMod(numero_encriptado,clave_privada,N);
```

En acción: generando claves

```
./iarsac -g Alicia -l 80 -v
Generando claves RSA para Alicia
longitud de clave=80 bits
Primos elegidos
p=739997448982420881568093
q=1071100960056827043446387
clave_publica=790786829720893592282321
n=792611978044673896555170296609163838399535329991
f=792611978044673896555168485510754799151610315512
clave_privada=
311356550667082262837401003628093127448107692081
```

En acción: encriptando

```
echo "Alicia te amo!"> mi_texto_plano.txt
./iarsac -c Alicia < mi_texto_plano.txt
>  mi_texto_cifrado.txt
```

y para descifrar usamos

```
cat mi_texto_cifrado.txt | ./iarsac -d Alicia
Alicia te amo!
```

Referencias...

N. Koblitz. A course in Number Theory and Crytography. Springer Verlag, (1987).

W. Stein. Elementary Number Theory.

Disponible en la página de su autor:
`http://sage.math.washington.edu/ent/`

"Introduction to Cryptography", en la documentación de PGP (Pretty Good Privacy), un programa creado por Phil Zimmermann, para permitir el uso masivo de la criptografía de llave pública:
`http://www.pgpi.org/doc/pgpintro/`