# Segundo trabajo práctico
# Problemas de congruencias en la encriptación de datos

(*En honor a Rivest, Shamir y Adleman, inventores del protocolo RSA)*

Taller de Álgebra 1 - Segundo cuatrimestre de 2021

¿Escucharon alguna vez hablar del algoritmo de encriptación RSA? El sistema criptográfico RSA es un método que sirve para enviar un mensaje de un emisor a un receptor, codificado de tal manera que si el mensaje es interceptado por terceros, no puedan descifrar su contenido. Fue desarrollado por los criptógrafos Rivest, Shamir y Adleman (de allí su nombre) en 1979, y sigue utilizándose ampliamente hoy en día.

Este sistema está basado en la siguiente propiedad matemática:

**Lema:** Sean $p$, $q$ dos primos positivos distintos y sean $n = pq$ y $m = (p-1)(q-1)$. Finalmente, sean $d, e \in \mathbb{N}$ tales que:

$$de \equiv 1 \pmod{m}.$$

Entonces, para todo $a \in \mathbb{Z}$,

$$a^{de} \equiv a \pmod{n}.$$

La demostración de este lema puede encontrarse por ejemplo en las *notas Álgebra I* de T. Krick.

**¿Cómo funciona el sistema criptográfico RSA?** Imaginemos que dos amigos, Alicia y Bob (son los nombres originales utilizados en la presentación del algoritmo en 1979), quieren mandarse mensajes, pero lamentablemente tienen una enemiga: Miranda, que puede leer todo lo que ellos se mandan y publicará en todas las redes sociales cualquier mensaje que encuentre, rompiendo la confidencialidad de la comunicación entre Alicia y Bob.

Por suerte existe el protocolo RSA que nos permite, siguiendo ciertos pasos, encriptar mensajes y salvarse de ajenos que pretendan filtrar nuestros mensajes. Para esto imaginemos que Alicia quiere enviarle el mensaje $M$ a Bob[i]. Estos son los pasos que deberán seguir para mantener una comunicación segura:

[i] El mensaje que le enviará será un número módulo $n$. Cómo traducir texto en un número módulo $n$ y viceversa es un problema aparte, del que *no* se tendrán que ocupar.

1. Bob eligirá dos primos $p$ y $q$ de alguna manera.

2. Luego Bob calculará los siguientes valores:

   - $n = p \times q$
   - $m = (p-1)(q-1)$
   - $e$ y $d$ tales que $0 \leq e < m$ y $e \times d \equiv 1 \mod m$

   y llamará ***clave pública*** a la dupla $(e, n)$ y ***clave privada*** a la dupla $(d, n)$.

3. Bob liberará al mundo su clave pública $(e, n)$, a la que Miranda como cualquier otra persona tendrá acceso.

4. Alicia tomará el mensaje $M$ y $(e, n)$ la clave pública liberada por Bob y hará la siguiente operación:

   - Si $\mathrm{mcd}(M, n) = 1$ codificará (o encriptará) a $M$ como $M^e \mod n$ [ii].
   - Si $\mathrm{mcd}(M, n) \neq 1$, codificará a $M$ como $-M$.

   y le enviará el mensaje codificado (o encriptado) al que llamaremos $C$ a Bob[iii].

5. Bob recibirá el mensaje $C$ enviado por Alicia, procederá de la siguiente manera:

   - Si $C \geq 0$, decodificará el mensaje como $C^d \mod n$ y obtendrá el mensaje original [iv].
   - Si $C < 0$, decodificará $C$ como $-C$, obteniendo el mensaje original.

Asumimos entonces que Miranda conoce los valores de $e, n$ y $C$, por lo que puede recuperar el mensaje $M$ si logra averiguar quién es $d$. Notar que $d$ es el inverso de $e$ módulo $m$, por lo que a Miranda le basta con calcular $m$ para saber quién es $d$. A su vez, para conocer $m$ le alcanza con factorizar $n$. Vemos entonces que si Alicia y Bob quieren que sus mensajes estén a salvo, Bob tiene que elegir primos $p$ y $q$ de manera que su producto $n$ resulte difícil de factorizar. En otras palabras, la fortaleza del protocolo RSA está basada en lo difícil que es factorizar enteros (grandes). En la Figura 1 puede verse un ejemplo de cómo funciona el sistema de encriptación RSA.

[ii] Para que el proceso de codificación y decodificación funcione correctamente, es necesario que $n$ sea lo suficientemente grande como para que dos mensajes distintos $M$ y $M'$ no puedan satisfacer que $M^e \equiv M'^e \mod n$. En nuestro caso, para evitar esto bastará con tomar primos $p$ y $q$ mayores a 300.

[iii] A priori el mensaje $C$ no es público. Pero asumiremos que nos encontramos en el peor escenario posible, Miranda hackeó la comunicación entre Alicia y Bob, gracias a lo cual conoce el valor de $C$.

[iv] No es magia, es una consecuencia del Teorema de Fermat generalizado.

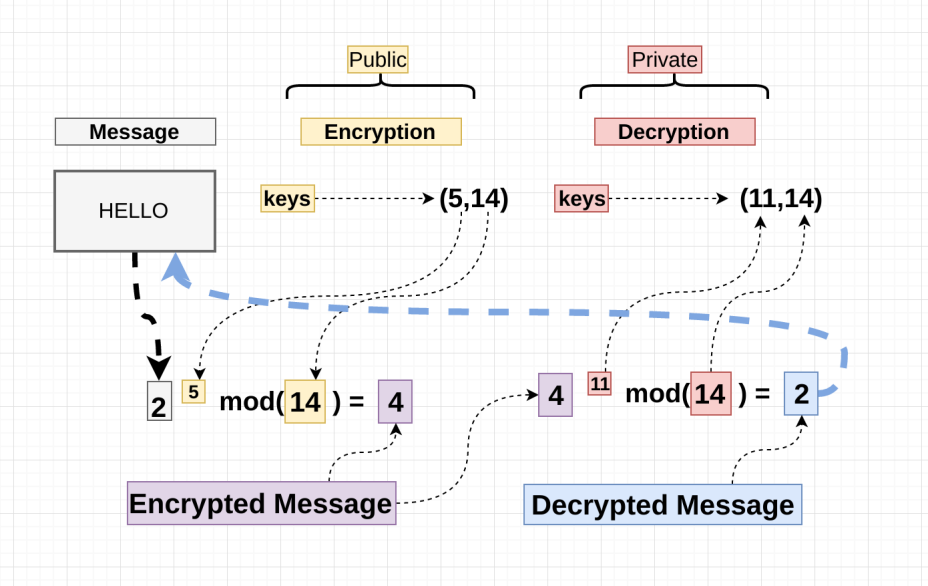


Figura 1: Ejemplo sencillo del funcionamiento del algoritmo de encriptación RSA, tomando $p = 2$, $q = 7$, $n = 14$, $m = 6$, $e = 5$ y $d = 11$.

## Archivos

En la página del Taller encontrarán los archivos necesarios para la realización del Trabajo Práctico.

1. En el archivo `Tipos.hs` se definen nuevos renombres de tipos y otros datos que veremos luego:

```
type Set a      = [a]  -- el tipo conjunto (visto en clase)
type ClPub      = (Integer, Integer) -- (e, n) la clave publica
type ClPri      = (Integer, Integer) -- (d, n) la clave privada
type Mensaje    = [Char]
type Cifrado    = [Integer]
```

   Además, están definidas dos funciones que se explican más adelante.
   **Importante:** Este archivo no deben modificarlo.

2. El archivo `main.hs` es el encargado de unificar todo, cuando quieran probar su código deberán cargarlo en GHCI con el comando `ghci main.hs`, esto incluirá todos los archivos que hayan modificado.

   Luego, podrán utilizar la función `main`, que codifica, decodifica y rompe algunos mensajes dados por nosotros. O también la función `test`, a la cual le pueden pasar una lista de mensajes que codificará, decodificará y romperá.

3. En los archivos `Aritmetica.hs` y `RSA.hs` encontrarán las funciones que deben implementar y, en estos archivos, podrán agregar posibles funciones auxiliares que les parezcan útiles.

   **Importante:** No se admite cambiar las signaturas de las funciones. Tampoco pueden cambiar los nombres de los archivos y todos deben estar en la misma carpeta.

## Funciones implementadas

Las siguientes funciones se darán ya implementadas y no deben modificarlas.

```
elegidor :: Set Integer -> (Integer, Integer)
```

Que dado un conjunto de al menos dos elementos, extrae dos de ellos y los devuelve en una tupla. No es necesario que ustedes utilicen esta función.

```
modExp :: Integer -> Integer -> Integer -> Integer
```

Que dados tres enteros, $x$, $y$ y $z$ devuelve $x^y \bmod z$. Es muy importante usar esta función y no volver a implementarla por cuestiones de tiempo de ejecución.

```
aEnteros :: [Char] -> [Integer]
```

Esta función toma una lista de caracteres y la transforma en una lista de enteros, donde cada caracter lo corresponde con su código ASCII.

```
aChars :: [Integer] -> [Char]
```

Esta función toma una lista de enteros y la transforma en una lista de caracteres, donde cada caracter lo corresponde con su código ASCII. Es la inversa de la anterior. Notar que esta función asume que los enteros pasados pueden ser convertidos a un caracter ASCII y se encuentran entre 0 y 255.

## Ejercicios

### Ejercicio 1

Escribir la función:

```
criba :: Integer -> Set Integer
```

Que, dado un número natural `n`, devuelve un conjunto de enteros con todos los primos positivos menores a `n`. Por ejemplo:

```
*Main> criba 4
[2,3]
*Main> criba 7
[2,3,5]
```

### Ejercicio 2

Escribir la función:

```
coprimoCon:: Integer -> Integer
```

Dado un número entero `n` $> 2$, encuentra otro número `x`, con $1 <$ `x` $<$ `n` $- 1$, de tal forma que `n` y `x` sean coprimos (si es que existe). Notar que puede haber múltiples respuestas válidas, la función puede devolver cualquiera de ellas. Por ejemplo:

```
*Main> coprimoCon 8
5
*Main> coprimoCon 12
7
```

### Ejercicio 3

Escribir la función:

```
inversoMultiplicativo:: Integer -> Integer -> Integer
```

Dados dos números `n` y `m`, calcula el inverso multiplicativo de `n` módulo `m`. Notar que esto sólo se puede hacer si los dos números son coprimos, y que se puede utilizar el algoritmo de Euclides extendido para calcularlo. Por ejemplo:

```
*Main> inversoMultiplicativo 2 5
3
*Main> inversoMultiplicativo 3 11
4
*Main> inversoMultiplicativo 8 25
22
```

### Ejercicio 4

Escribir la función:

```
claves :: Integer -> Integer -> (Integer, Integer, Integer)
```

Que dado dos números primos `p` y `q`, calcula los valores `e`,`d` y `n` utilizados por RSA para generar las claves y los devuelve en una tupla en ese orden. Por ejemplo:

```
*Main> claves 3 5
(5,5,15)
*Main> claves 13 7
(67,43,91)
```

El algoritmo se utiliza con primos grandes, no se preocupen si falla con parejas como $p = 2, q = 3$

### Ejercicio 5

Escribir la función:

```
codificador :: ClPub -> Mensaje -> Cifrado
```

Que dada la clave pública y una secuencia de caracteres, la codifica caracter a caracter y devuelve el mensaje encriptado.

```
*Main> codificador (4323857,5048993) "hola"
[4413982,3675419,2203857,1326513]
```

### Ejercicio 6

Escribir la función:

```
decodificador :: ClPri -> Cifrado -> Mensaje
```

Que dada la clave privada y un mensaje encriptado, la decodifica y devuelve la secuencia original.

```
*Main> decodificador (5044493,5048993) [4413982,3675419,2203857,1326513]
"hola"
```

## Condiciones de entrega

El segundo Trabajo Práctico se debe realizar con el mismo grupo de alumnos con el que se elaboró el primero (o su recuperatorio). Si algún integrante del grupo abandonó la cursada del Taller, se aceptarán trabajos de grupos de dos alumnos.

Está terminantemente prohibido subir el código que implementen a repositorios públicos, así como también usar total o parcialmente soluciones implementadas por otros grupos. No está permitida la interacción con otros grupos para discutir las soluciones de los ejercicios propuestos.

No evaluaremos la eficiencia de los algoritmos que propongan para resolver los ejercicios (y por ende, el tiempo en que tardan en ejecutarse las funciones) pero sí la correctitud del código. Algunas funciones que implementen pueden demorar minutos en arrojar el resultado, no se preocupen por el tiempo sino porque devuelvan el valor correcto.

La entrega consiste en un único archivo `.zip` que contenga los archivos `Aritmetica.hs`

y `RSA.hs` con las funciones de los ejercicios implementadas, junto con todas las funciones auxiliares que sean necesarias para ejecutarlas. Las funciones deben respetar la signatura (nombres y parámetros) especificados en cada ejercicio, dado que serán testeadas automáticamente.

El archivo entregado debe tener la forma "`apellido1-apellido2-apellido3.zip`", respetando el orden alfabético de los apellidos. Por ejemplo, si el grupo está formado por los estudiantes María López, Ariel Gómez, y Juan Pérez que cursan en el turno de los viernes debe entregar un archivo llamado "`Gomez-Lopez-Perez.zip`".

La entrega se debe llevar a cabo a través del campus virtual, subiendo el archivo con el código con el mecanismo disponible dentro del espacio del taller en el campus virtual. Un solo integrante será el encargado de subir el Trabajo Práctico al campus en representación de todo el grupo.

Se deberá utilizar exclusivamente los conceptos vistos en el Taller. Se evaluará la corrección de las funciones implementadas, la declaratividad y claridad del código, y que las funciones auxiliares (si las hay) tengan nombres apropiados.

Si tienen dudas o consultas respecto del trabajo práctico, pueden enviar un mail a la lista de docentes `algebra1-doc (arroba) dc.uba.ar`. No hacer consultas a través del campus virtual ni mandando mails a la lista de alumnos.

Además del código, deberán rendir un coloquio grupal, en el que se conversará **individualmente con cada integrante del grupo** sobre el trabajo realizado, debiendo responder diversas preguntas sobre lo que entregaron. El día y horario del coloquio lo acordarán por email con el docente que haya corregido su trabajo.

**Fecha de entrega:** hasta el lunes 8 de noviembre a las 23:59 hs.