Universidad Nacional Autónoma de México
Facultad de Ciencias - Ciencias de la Computación
Criptografía y seguridad 2018-1

Práctica 3

Nota: se proporcionan algunos algoritmos en Python. Si decides utilizar Java o C, tendrás que implementar estos mismos algoritmos.

En esta primera parte se revisan algunos algoritmos sobre teoría de números que son útiles para la implementación de esquemas de llave pública.

1. Aritmética modular

En el archivo `arit_modular.py` se encuentran algunas funciones que servirán posteriormente, es importante revisarlas y saber qué hacen para cuando sean requeridas en otros algoritmos.

Vamos a necesitar una función para calcular a^b mód N de forma eficiente, para lograrlo usaremos la estrategia «square and multiply», que se presenta a continuación.

Algoritmo 1. POTENCIA_MOD

Entrada: módulo N, base $a \in \mathbb{Z}_N$, exponente $b > 0$
Salida: a^b mód N
$x := a$
$t := 1$
while $b > 0$ **do**
  if b es impar **then**
    $t := tx$ mód N
    $b := b - 1$
  $x := x^2$ mód N
  $b := b/2$
return t

Escribe este seudocódigo en la función correspondiente.

2. Generación de primos

Para generar primos al azar usaremos la estrategia más simple: obtener un entero aleatorio y luego comprobar si es primo, repetir hasta encontrar uno. No podemos obtener un entero aleatorio de cualquier tamaño, así que tenemos que acotar nuestro espacio a un rango que nos sea útil. Este rango además define el tamaño (en bits) del primo; aunque esto es importante para la seguridad, para nuestros fines basta definir un rango arbitrario.

Para comprobar si un número es primo usaremos el test de Miller-Rabin. Antes de eso, investiga (o inventa) un algoritmo de complejidad $O(\log^2 N)$ para verificar si un entero N es una potencia de otro entero, es decir, si $N = M^e$, donde M y $e \geq 2$ son enteros. Esto último corresponde a la función ES_POTENCIA que se encuentra en el archivo.

Con todo lo anterior ya puedes implementar el test de Miller-Rabin.

Algoritmo 2. MILLER_RABIN

Entrada: Entero $N > 2$ y un parámetro $t > 0$
Salida: Mensaje indicando si N es primo o compuesto
if N es par **then**
    return COMPUESTO
if N es una potencia de otro número **then**
    return COMPUESTO
Obtener $r \geq 1$ y u impar tales que $N - 1 = 2^r u$
for $j = 1$ hasta t **do**:
    $a \leftarrow \{1, \ldots, N-1\}$ ▷ indica que a se escoge al azar
    if $a^u \neq \pm 1$ mód N y $a^{2^i u} \neq -1$ mód N para $i \in \{1, \ldots, r-1\}$ **then**
        return COMPUESTO
return PRIMO

Una vez que tienes el test de Miller-Rabin, crea una función GENERA_PRIMO que reciba un entero $n > 1$ y que devuelva un primo aleatorio en el rango $[2^{n-1}, 2^n - 1]$.

Finalmente, escribe una función GENERA_MODULO que acepta un parámetro entero $n > 1$ y hace lo siguiente:

1. Calcula dos primos aleatorios p, q en el rango $[2^{n-1}, 2^n - 1]$.
2. Calcula $N = p \cdot q$, $m = \varphi(N)$ y se define $e = 65537$.
3. Calcula $d = e^{-1}$ mód m.
4. Devuelve (N, e, d).

3. EXTRA EXTRA EXTRA

¿Para qué es el siguiente código? En particular, ¿cuál es el propósito de las líneas 4-7?

```
1 def funcion(n=16):
2     r = open('/dev/urandom')
3     p = 4
4     while not test_miller_rabin(p):
5         s1 = r.read(n).encode('hex')
6         s2 = int(s1, 16)
7         p = s2 | (1<<(n*8-1))
8     return p
```

4. Cifrado RSA de escuela

La función GENERA_MODULO nos sirve para crear una pareja de claves. La clave privada o secreta es (N, d) y la clave pública es (N, e). Recuerda que la clave pública se usa para cifrar, mientras que la clave privada sirve para descifrar. Un mensaje válido es un entero $2 \leq m < n$. Para cifrar simplemente se calcula m^e mód N, y para descifrar un mensaje c se calcula c^d mód N.

Crea una función llamada CIFRAR_RSA que reciba dos parámetros, una clave pública (N, e) y un mensaje m, y que devuelva el valor cifrado. También haz la función correspondiente DESCIFRAR_RSA.

Esta es una versión muy simplificada de RSA. En una implementación real de RSA deben cuidarse varios detalles, o de otro modo será susceptible a varios ataques que permiten recuperar la clave privada o el mensaje claro. En la siguiente práctica se revisarán algunos de estos ataques.

5. Diffie-Hellman

La criptografía de clave pública se originó con este esquema que resuelve el siguiente problema: dos personas que pueden comunicarse por un canal público quieren generar una clave secreta para proteger la comunicación posterior. El esquema de Diffie-Hellman consiste en lo siguiente:

1. Alicia escoge un primo grande p y un entero $g > 1$ (comúnmente 2). Luego genera al azar un entero $a \in [2, p-2]$ y calcula $X = g^a \bmod p$.
2. Alicia envía p, g, X a Bartolo.
3. Bartolo recibe p, g, X. Genera al azar un entero $b \in [2, p-2]$ y calcula $Y = g^b \bmod p$. Obtiene $k_1 = X^b \bmod p$.
4. Bartolo envía Y a Alicia.
5. Alicia recibe Y, con esto calcula $k_2 = Y^a \bmod p$.

Al final ambos obtienen el mismo valor $k = k_1 = k_2$. La seguridad radica en que, mientras los valores a y b se mantengan privados, no se conoce un algoritmo eficiente para calcular k a partir de los demás datos.

Usando las funciones de los primeros ejercicios, crea las funciones necesarias para ejecutar el intercambio de Diffie-Hellman. Basta el cálculo de los números, omite lo de "enviar" y "recibir". En un archivo de texto describe cómo usar estas funciones para poder probarlas.

6. Entrega

- La práctica puede hacerse entre dos personas o de forma individual.
- Organiza tus archivos en un directorio y comprímelo en un zip. Tanto el directorio como el archivo zip tendrán un nombre de la forma `Practica3_Lopez` o `Practica3_Lopez_Juarez`, que indican el apellido paterno de una o dos personas, respectivamente.
- Sube tu archivo zip en el siguiente formulario: `https://goo.gl/forms/jtVKPMUv2YkN6Ncx1`
- Fecha límite: 20 de noviembre a cualquier hora.
- No se aceptará por correo.