

# Modelo de clave pública

- ▶ Diseñado para evitar el problema de la distribución de las claves entre emisor y receptor.
- ▶ Dos claves diferentes:
  - **Clave privada**  $K_c$ : que solo el usuario conoce.
  - **Clave pública**  $K_p$ : publicada para todo el mundo que lo desee.
  - Relacionadas entre sí por una función de sentido único. Debe ser muy difícil calcular una clave a partir de la otra. **Relación Par de claves**
- ▶ **Sistema asimétrico**, pues se emplean claves diferentes para encriptar y desencriptar la información.
  - Si alguien desea enviar un mensaje, buscará la clave pública de aquel al que desea enviárselo, y lo cifrará con dicha clave.
  - La única forma de desencriptarlo es utilizando la clave privada del receptor



# Algoritmo RSA

- ▶ Debe su nombre a sus tres inventores: Ronald Rivest, Adi Shamir y Leonard Adleman, del MIT.
- ▶ Algoritmo asimétrico de cifrado por bloques, que utiliza:
  - Una clave pública, conocida por todos.
  - Una clave otra privada, guardada en secreto por su propietario.
  - La relación de claves se basa en el producto de dos números primos muy grandes elegidos al azar. No hay maneras rápidas conocidas de factorizar un número grande en sus factores primos.



# Algoritmo RSA

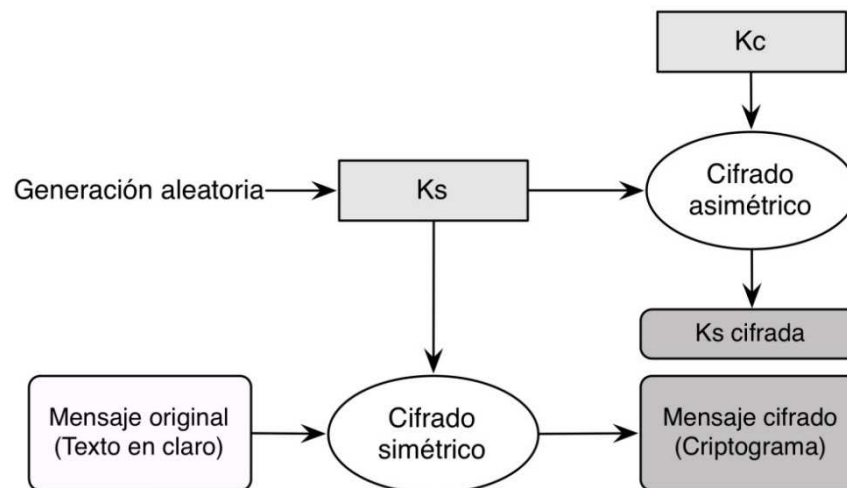
## ► Proceso:

- Se eligen al azar y en secreto dos números primos  $p$  y  $q$  lo suficientemente grandes y se calcula:
  - $n = p \cdot q$
  - $\Phi(n) = (p-1) \cdot (q-1)$
- Se eligen dos exponentes  $e$  y  $d$  tales que:
  - $e$  y  $\Phi(n)$  no tengan números en común que los dividan, es decir  $\text{mcd}(e, \Phi(n)) = 1$
  - $e \cdot d = 1 \text{ mod } \Phi(n)$ , es decir,  $e$  y  $d$  son inversos multiplicativos en  $\Phi(n)$
- La clave pública es el par  $(e, n)$ .
- La clave privada  $(d, n)$  o  $(p, q)$ .
- Para cifrar  $M$  lo único que se debe hacer es  $M^e \text{ mod } n = M'$ , siendo el descifrado  $M'^d \text{ mod } n$ . Como se puede observar, conmuta.



# Modelo híbrido

- ▶ Los sistemas de clave pública son computacionalmente mucho más costosos que los sistemas de clave privada.
- ▶ Se emplea una combinación de ambos sistemas
  - En el intercambio de información por rapidez se codifican los mensajes mediante algoritmos simétricos y una única clave  $K_s$ .
  - Dicha clave se suele crear aleatoriamente por el emisor.
  - Se utilizan sistemas de clave pública para codificar y enviar la clave simétrica  $K_s$ .
  - El resto de mensajes intercambiados durante esa sesión cifrados con la clave  $K_s$ .



# Programación de cifrado asimétrico

- ▶ La interfaz **PublicKey** (*java.security.PublicKey*). Implementa a su vez la interfaz *Key*, y se emplea para representar claves públicas. (public interface `PublicKey` extends [Key](#))
- ▶ La interfaz **PrivateKey** (*java.security.PrivateKey*). Implementa a su vez la interfaz *Key*, y se emplea para representar claves privadas. (public interface `PrivateKey` extends [Key](#))
- ▶ La clase **KeyPair** (*java.security.KeyPair*). Los objetos de esta clase se utilizan para almacenar pares de claves.
- ▶ La clase **KeyPairGenerator** (*java.security.KeyPairGenerator*). Se utiliza para generar pares de claves.
- ▶ La clase **KeyFactory** (*java.security.KeyFactory*). Se usa como fábrica de claves para convertir claves opacas en transparentes y viceversa.



# Clase KeyPair

- ▶ public final class **KeyPair** extends Object implements Serializable
- ▶ Objetos que representan pares de claves.

Método	Retorno	Descripción
keyPair(PublicKey publicKey, PrivateKey privateKey)	KeyPair	Constructor de la clase
getPrivate()	PrivateKey	Método para obtener la clave privada almacenada
getPublic()	PublicKey	Método para obtener la clave pública almacenada



# Clase KeyPairGenerator

- ▶ Se usa para generar claves. A partir de esta clase se generan los pares de claves con el método **generateKeyPair()**

Método	Retorno	Descripción
getInstance(String algorithm)	Static KeyPairGenerator	Método estático para crear objetos de clase KeyPairGenerator. Recibe como parámetro el nombre del algoritmo de cifrado asimétrico que se desea emplear
Generatekeypair()	KeyPair	Genera aleatoriamente un par de claves (pública y privada) y las devuelve como un objeto de clase KeyPair





# Ejemplo de cifrado asimétrico

```
import javax.crypto.*; import java.security.*;

public class CifradoAsimetrico {
    public static void main(String[] args) {
        try {
            //Obteniendo generador de claves con cifrado RSA
            KeyPairGenerator keygen = KeyPairGenerator.getInstance("RSA");
            KeyPair keypair = keygen.generateKeyPair(); //Generando par de claves
            Cipher rsaCipher = Cipher.getInstance("RSA"); //Obteniendo objeto Cipher RSA
            //Configurar Cipher para encriptar con clave privada
            rsaCipher.init(Cipher.ENCRYPT_MODE, keypair.getPrivate());
            String mensaje = "Mensaje de prueba del cifrado asimétrico";
            String mensajeCifrado = new String(rsaCipher.doFinal(mensaje.getBytes()));
            //Descifrado usando la clave pública
            rsaCipher.init(Cipher.DECRYPT_MODE, keypair.getPublic());
            String mensajeDescifrado = new
            String(rsaCipher.doFinal(mensajeCifrado.getBytes()));
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

Cifrar con la publica y descifrar con la privada



# Clase KeyFactory

Es la fábrica de claves por defecto para claves del modelo de cifrado asimétrico. Funciona de forma similar al resto de factorías, permitiendo cambiar entre claves opacas y transparentes y viceversa.

Retorno	Método y Descripción
Static KeyFactory	<code>getInstance( String algorithm)</code> Método estático para crear objetos de clase KeyFactory. Recibe como parámetro el nombre del algoritmo de cifrado asimétrico que se desea emplear
PrivateKey	<code>generatePrivate( KeySpec keySpec)</code> Genera una clave privada opaca, a partir de su versión transparente
PublicKey	<code>generatePublic( KeySpec keySpec)</code> Genera una clave pública opaca, a partir de su versión transparente
<code>&lt;T extends KeySpec&gt;</code>	<code>getKeySpec( Key, key.class&lt;T&gt; keySpec)</code> Genera una clave transparente, a partir de su versión opaca. Se le debe especificar como parámetro además la clase derivada de KeySpec que se desea usar para crear la clave transparente. Esto es necesario porque la estructura interna de las claves depende mucho del algoritmo de cifrado, por lo que sus propiedades fundamentales pueden cambiar

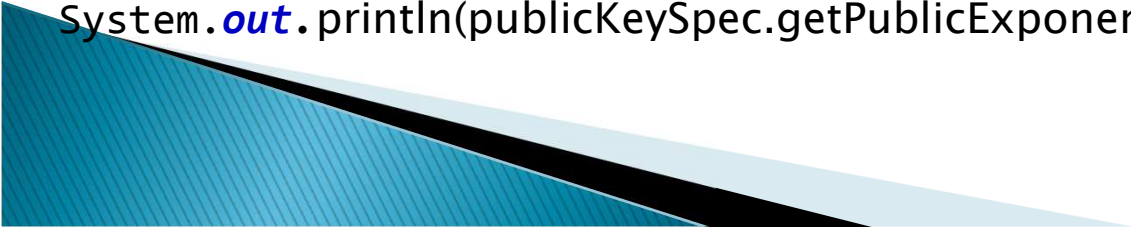
# Ejemplo KeyFactory

```
System.out.println("Generando par de claves");
KeyPairGenerator keygen = KeyPairGenerator.getInstance("RSA");
KeyPair keypair = keygen.generateKeyPair();

System.out.println("Obteniendo objeto Cipher con cifrado RSA");
Cipher desCipher = Cipher.getInstance("RSA");
System.out.println("Configurando Cipher para encriptar");
desCipher.init(Cipher.ENCRYPT_MODE, keypair.getPrivate());

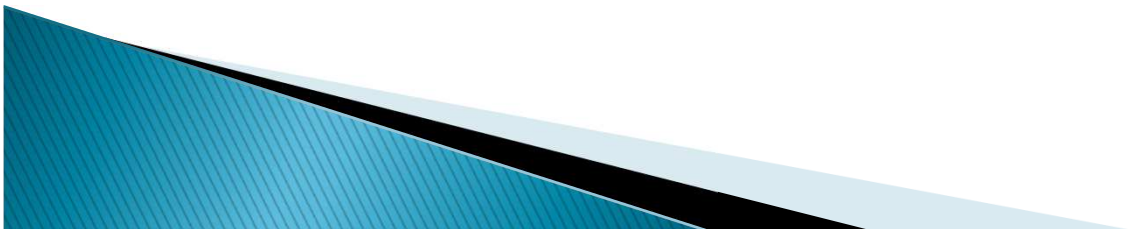
. . . . .
System.out.println("Obteniendo factoría de claves con cifrado RSA");
KeyFactory keyfac = KeyFactory.getInstance("RSA");

System.out.println("Generando keyspec");
RSAPublicKeySpec publicKeySpec=keyfac.getKeySpec(keypair.getPublic(),
RSAPublicKeySpec.class);
//Imprimo la clave o la puedo guardar en un archivo
System.out.println(publicKeySpec.getModulus());
System.out.println(publicKeySpec.getPublicExponent());
```



# Ejercicio Cifrado asimétrico

- ▶ Escribir un programa que utilice cifrado asimétrico para cifrar el contenido de un fichero, usando la clave privada.
- ▶ El resultado del cifrado y la clave pública se deben volcar en dos ficheros adicionales de salida.
- ▶ Posteriormente, escribe un segundo programa que cargue dichos ficheros y descifre el archivo cifrado. utiliza cifrado RSA.



# Solución

- ▶ Ver código en
- ▶ 02 – Ejercicio – Cifrado fichero asimétrico



# Ejercicio Práctico

- ▶ Implementar los ejercicios anteriores de cifrado simétrico y asimétrico para que funcionen en Red mediante Sockets.
- ▶ Un Servidor Cifra y el cliente descifra.
- ▶ Y Viceversa.



# Ejercicio Práctico. Final

- ▶ Implementar la solución del programa del Ferry con las siguientes modificaciones:
- ▶ En cada orilla existen una cola y tres taquillas para obtener el pase al ferry.
- ▶ Cada orilla se ejecuta en un pc diferente.
- ▶ Cada vez que sale un ferry hacia la otra orilla se envían a un servidor los datos de facturación, número de personas y edad media de las personas que viajan.
- ▶ La comunicación se realiza mediante sockets y con cifrado de clave pública
- ▶ Al final del día (10+10 viajes) el servidor obtiene las estadísticas por cada orilla que guarda en un fichero de texto con la siguiente nomenclatura:  
**año\_mes\_dia\_estadísticas.txt**

