# MEMORIA DE DESARROLLO

TRABAJO FINAL DE CARRERA
Ingeniería Técnica Informática de Sistemas

DE: Julián Botaya Villanúa

TEMA A DESARROLLAR: **CHAT SEGURO**

AÑO: 2004-05 / 1er Semestre

Universitat Oberta de Catalunya

# INDICE

# 1. Objetivos del proyecto

El objetivo de este proyecto ha sido la elaboración de una aplicación, un sistema Chat ó mensajería instantánea, que implementara técnicas criptográficas para el intercambio seguro de información punto a punto.

Cada uno de los usuarios dispone de un programa, que engloba las funcionalidades cliente/servidor que permite realizar conexiones a una determinada dirección IP. Una vez se ha realizado la conexión, el programa al que se ha llamado contesta y se realiza el primer paso para poder realizar el cifrado de mensajes, se produce el intercambio de una clave de sesión. Esta clave será la que se usará para realizar el posterior cifrado de los mensajes.

El intercambio de claves entre el cliente y el servidor se ha implementado a partir del protocolo de intercambio de claves Diffie-Hellman. Este protocolo evita el inconveniente de la distribución de claves y permite que dos usuarios puedan pactar una clave secreta compartida sobre un canal inseguro.

Para realizar el cifrado de los mensajes se ha utilizado el algoritmo DES que utilizará la clave intercambiada entre los dos usuarios para realizar el cifrado de los mensajes.

Cada comunicación punto a punto produce claves de sesión diferentes, es decir, cada comunicación que establezca un usuario generará claves diferentes, no se reutilizan las existentes en otras conexiones que puedan estar ya establecidas.

# 2. Estado del arte

En este apartado se van a exponer los conceptos teóricos en que se basan los algoritmos y procedimientos criptográficos que se han utilizado y se explicarán detalladamente las herramientas utilizadas para su implementación informática. Seguidamente se presentará la técnica utilizada para la comunicación de procesos a través de internet a nivel de la capa de trasporte OSI, los sockets. Una vez presentados los conceptos teóricos utilizados para llevar a cabo este proyecto se hará una exposición detallada del software utilizado y cómo se ha implementado.

Por las características de este proyecto se han utilizado dos técnicas criptográficas diferentes. Por un lado tenemos el intercambio de claves entre dos usuarios cualesquiera que se engloba en el apartado de cifrado de clave pública y que se lleva a cabo mediante el algoritmo Diffie-Hellman y por otro tenemos el cifrado de los mensajes con estas claves que pertenece al cifrado de clave compartida, cifrado de bloque, utilizando el algoritmo DES (Data Encryption Standard).

## 2.1 Conceptos Preliminares de cifrado de clave pública

En este apartado se resumirán los conceptos básicos de teoría de los números, que son necesarios para entender el proceso de generación de claves para este tipo de cifrado.

### 2.1.1 Aritmética modular

Dados los enteros a, b, y n $\neq 0$ se dice que a es congruente con b módulo n, que se escribe:

$$a \equiv_n b,$$

si y solo si $a - b = kn$ para algún entero $k$.

Una definición alternativa sería: $a \equiv_n b$ si y sólo si $n$ divide $(a-b)$ que se escribe $n \mid (a-b)$.

Si $a \equiv_n b$ entonces $b$ se llama residuo de $a$ **módulo** $n$. Recíprocamente, $a$ es un residuo de $b$ módulo $n$. Escribiremos $a \bmod n$ para representar el residuo de $a$ módulo $n$ en el rango $[0, n-1]$.

Un conjunto de enteros $r_1,...,r_n$ es un conjunto de residuos módulo $n$ si, para cada entero $a$, hay exactamente un $r_i$ en el conjunto tal que $a \equiv_n r_i$.

Para cualquier módulo, el conjunto de enteros $[0,1,...,n-1]$ es un conjunto completo de residuos módulo $n$.

Como para con los números enteros, los enteros $\bmod n$ con las operaciones de suma y de multiplicación forman un **anillo conmutativo.** Esto significa que es cumplen las propiedades asociativa, conmutativa y distributiva.

Además, calcular con aritmética modular (es decir, reducir cada resultado intermedio módulo $n$ da el mismo resultado que calcular con aritmética entera ordinaria y reducir el resultado final módulo $n$. De una manera más formal, se puede expresar este hecho con el teorema de la aritmética modular.

**Teorema 1: principio de la aritmética modular :** Sean $a_i$ y $b_i$ enteros, y sea OP uno de los operadores binarios +, - ó ·, entonces la reducción módulo $n$ es un homomorfismo de los enteros a los enteros módulo $n$, es decir:

$$(a_i OP a_2) \bmod n = [(a_1 o \bmod n) OP (a_2 \bmod n)] \bmod n$$

Calcular con aritmética modular tiene la ventaja que reduce el rango de los valores intermedios. Para un módulo $n$ de $k$ bits, el valor de cualquier suma, resta ó multiplicación cabrá en $2k$ bits, como mucho. Esto, por ejemplo, nos permitirá hacer exponenciaciones modulares del estilo $a^z \bmod n$ sin generar resultados intermedios monstruosos.

Presentamos un algoritmo de exponenciación rápida para calcular $a^z \bmod n$:

```
exp_rapid(a,z,n)

begin "Retorna x=a^z mod n"
   a1:=a; z1:=z;
   x:=1;
   while z1 <> 0 do "(x(a1^z1 mod n) = a^z mod n"
   begin
      while z1 mod 2 = 0 do
      begin "elevar al cuadrado a1 mientras z1 es par"
         z1:=[z1/2];
         a1:=(a1*a1)mod n
      end;
      z1:=z1-1;
      x:=(x*a1) mod n "Multiplicación"
   end;
   exp_rapid := x
end
```

Si $(z_{k-1},...,z_1,z_0)$ es la representación binaria del exponente z, el algoritmo anterior procesa los bits en orden $z_0, z_1,..., z_{k-1}$ de menos a más significativo).

Si $z_i = 0$, calcula un cuadrado; si $z_i = 1$, multiplica y calcula un cuadrado. Por cada bit de z se hacen, por tanto, hasta dos multiplicaciones (excepto para el más significativo, que sólo da lugar a una multiplicación). Como la longitud en bits de z es $\lfloor \log_2 z \rfloor$, tenemos que el número de multiplicaciones es, como mucho:

$$2 \cdot (\lfloor \log_2 z \rfloor - 1) + 1 = 2 \cdot \lfloor \log_2 z \rfloor + 1$$

Hay que tener presente que exponenciar por el método de las multiplicaciones sucesivas requeriría z multiplicaciones.

Para ilustrar el algoritmo anteriormente presentado, tomamos, por ejemplo, $a^{37}$. La siguiente tabla refleja los valores intermedios de las variables (únicamente se presentan los cambios de valor):

| $a_1$ | $z_1$ | $x$ |
|---|---|---|
| $a$ | 37 | 1 |
| - | 36 | $a$ |
| $a^2$ | 18 | - |
| $a^4$ | 9 | - |
| - | 8 | $a^5$ |
| $a^8$ | 4 | - |
| $a^{16}$ | 2 | - |
| $a^{32}$ | 1 | - |
| - | 0 | $a^{37}$ |

En total hay ocho multiplicaciones, que es inferior a la cota $2\lfloor \log_2 37 \rfloor + 1 = 11$. De hecho, el algoritmo anterior se basa en la descomposición siguiente:

$$a^{37} = a^{32+4+1} = \left( \left( \left( a^2 \right)^2 \right)^2 \right)^2 \left( a^2 \right)^2 a$$

## 2.1.2 Cálculos de inversos en aritmética modular

Si trabajamos con aritmética entera ordinaria, no hay inversos multiplicativos. Es decir, para un entero $a \neq 1$ no hay ningún entero x tal que se verifique $ax = 1$.

En cambio, si trabajamos con aritmética modular, para un entero $a$ en el rango de valores $[0, n-1]$ en ocasiones se puede encontrar un único entero en el mismo rango tal que se verifique $ax \bmod n = 1$. Por ejemplo, los enteros 11 y 3 son inversos multiplicativos mod 16 porque:

$$11 \cdot 3 \bmod 16 = 33 \bmod 16 = 1$$

En los algoritmos de cifrado Diffie-Hellman se utiliza esta propiedad, el inverso multiplicativo. El siguiente teorema da la condición que se ha de cumplir para que haya un inverso multiplicativo.

**Teorema 2:** Dado $a \in [0, n-1]$, $a$ tiene un único inverso multiplicativo módulo $n$ si $a$ es coprimo con $n$, es decir, $mcd(a, n) = 1$

Ahora bien, no sirve de mucho saber que existe el inverso si no disponemos de algoritmos para poder calcularlo. Para avanzar hay que presentar algunos conceptos adicionales.

**La función totient de Euler** $\phi(n)$ es el número de elementos del conjunto $[0, \ldots, n-1]$ que son corrimos con $n$.

**Teorema 3:** La función totient de Euler se puede calcular con las fórmulas siguientes:

1) Para p primo $\phi(p) = p - 1$
2) Para $n = pq$ con $p, q$ primos: $\phi(p)\phi(q) = (p-1)(q-1)$
3) Para $n$ arbitrario tenemos:

$$\phi(n) = \prod_{i=1}^{t} p_i^{e_i - 1}(p_i - 1)$$

donde que $n = p_1^{e_1} p_2^{e_2} \ldots p_t^{e_t}$ es la descomposición de $n$ en factores primos $p_1, \ldots p_t$ donde $p_i$ tiene multiplicidad $e_i$.

Los siguientes teoremas relacionados con la función de Euler son importantes en la teoría de los números.

**Teorema 4: teorema pequeño de Fermat.** Sea $p$ primo, entonces para todo $a$ tal que $mcd(a, p) = 1$ se cumple:

$$a^{p-1} \bmod p = 1$$

**Teorema 5: teorema de Euler.** Para todo $a$ y $n$ tales que $mcd(a, n) = 1$ se cumple que:

$$a^{\phi(n)} \bmod n = 1$$

El teorema de Euler da una primera manera de calcular el inverso multiplicativo de $a \bmod n$ resolviendo la ecuación:

$$ax \bmod n = 1$$

Cuando $mcd(a, n) = 1$, la solución se obtiene de la expresión siguiente:

$$x = a^{\phi(n)-1} \bmod n$$

**Cálculo del inverso multiplicativo de 7 mod 15.**

Supongamos que $a = 7$ y $n = 15$; entonces la función totient de Euler de 15 es:

$$\phi(15) = \phi(3)\phi(5) = 2 \cdot 4 = 8$$

Entonces si aplicamos el teorema de Euler para encontrar el inverso multiplicativo obtenemos:

$$x = 7^{8-1} \bmod 15 = 13$$

En efecto, 13 es el inverso de 7, ya que $7 \cdot 13 \bmod 15 = 1$

Si se conoce $\phi(n)$, el inverso de $a$ módulo $n$ se puede calcular con la ecuación anterior y el algoritmo exp_rapid(a,z,n). El problema está en que si el módulo $n$ es muy grande (como es el caso de los algoritmos criptográficos), encontrar $\phi(n)$ es difícil. El algoritmo de Euclides extendido permite calcular inversos sin conocer $\phi(n)$.

El algoritmo de Euclides básico sirve para calcular el máximo común divisor de dos números, $a$ y $n$:

```
Algoritmo mcd(a,n)

begin
     g_0 := n;
     g_1 := a;
     i := 1;
     while g_i ≠ 0 do
     begin
          g_{i+1} := g_{i-1} mod g_i ;
          i := i + 1
     end;
     mcd := g_{i-1}
end
```

La versión extendida del algoritmo de Euclides permite encontrar el inverso de $a$ módulo $n$ cuando $mcd(a,n)=1$:

```
Algoritmo inv(a,n)

begin "Devuelve x tal que ax mod n = 1 dónde 0 < a < n"
     g_0 := n; g_1 := a;
     u_0 := 1; v_0 := 0;
     u_1 := 0; v_1 := 1;
     i := 1;
     while g_i ≠ 0 do " g_i = u_i n + v_i a "
     begin
          y := [g_{i-1} / g_i];
          g_{i+1} := g_{i-1} - y * g_i ;
          u_{i+1} := u_{i-1} - y * u_i ;
          v_{i+1} := v_{i-1} - y * v_i ;
          i := i + 1
     end;
     x := v_{i-1}
     if x ≥ 0 then inv:= x else inv:=x+n
end
```

Si el módulo respecto al cual se invierte es $n$, entonces el algoritmo inv(a,n) requiere del orden de $\ln(n)$ divisiones; es por lo tanto, bastante eficiente.

**Utilización del algoritmo de Euclides extendido**

Ilustraremos el funcionamiento del algoritmo de Euclides extendido encontrando el inverso de 3 módulo 7, es decir, resolviendo la ecuación $3x \bmod 7 = 1$.

La siguiente tabla refleja los valores intermedios de las variables (sólo se presentan los cambios de valor):

| $i$ | $g_i$ | $u_i$ | $v_i$ | $y$ |
|---|---|---|---|---|
| 0 | 7 | 1 | 0 | - |
| 1 | 3 | 0 | 1 | 2 |
| 2 | 1 | 1 | -2 | 3 |
| 3 | 0 | - | - | - |

Como $v_2 = -2$ es negativo, la solución es $x = -2 + 7 = 5$

Finalmente, comentar un último resultado que es útil en algunos protocolos criptográficos. La idea es resolver un determinado tipo de sistemas de ecuaciones modulares.

**Teorema 6: teorema chino del residuo.** Sean $d_1 ... d_t$ enteros coprimos dos a dos, y sea $n = d_1 d_2 ... d_t$, entonces el sistema de ecuaciones:

$$(x \bmod d_i) = x_i \text{ ; para } i = 1,...,t$$

tiene una solución común $x$ en el rango $[0, n-1]$.

La demostración de este teorema sería la siguiente: Para cada $i = 1,...,t$ tenemos $mcd(d_i n / d_i) = 1$. Por tanto, está el inverso de $n/d_i$ módulo $d_i$, es decir, $y_i$ tal que $(n/d_i) \bmod d_i = 1$. Por otro lado $(n/d_i) y_i \bmod d_j = 0$ para $j \neq i$, porque $d_j$ es un factor de $n/d_i$. Sea:

$$x = \left[ \sum_{i=1}^{t} \left( \frac{n}{d_i} \right) y_i x_i \right] \bmod n ,$$

entonces $x$ es la solución de la siguiente ecuación:

$$\left(x \bmod d_i\right) = x_i \text{ ; para } i = 1,...,t$$

teniendo en cuenta que:

$$x \bmod d_i = \left( \frac{n}{d_i} \right) y_i x_i \bmod d_i = x_i$$

**Utilización del teorema chino del residuo**

Lo utilizaremos para resolver la ecuación $3x \bmod 10 = 1$ a base de ir resolviendo ecuaciones con módulos más pequeños. Observemos que 10=2·5; por consiguiente, $d_1 = 2$ y $d_2 = 5$. Entonces tenemos que encontrar las soluciones $x_1$ y $x_2$ de las ecuaciones:

- $3x \bmod 2 = 1$
- $3x \bmod 5 = 1$

Aplicando el algoritmo in(a,n) obtenemos que $x_1 = 1$ y $x_2 = 2$. Entonces el teorema chino de los residuos permite obtener una solución común $x$ de las ecuaciones:

- $x \bmod 2 = x_1 = 1$
- $x \bmod 5 = x_2 = 2$

Encontramos $y_1$ y $y_2$ tales que:

- $\left(\frac{10}{2}\right) y_1 \bmod 2 = 1$
- $\left(\frac{10}{5}\right) y_2 \bmod 5 = 1$

Obtenemos $y_1 = 1$ y $y_2 = 3$ con lo cual:

$$x = \left[\left(\frac{10}{2}\right) y_1 x_1 + \left(\frac{10}{5}\right) y_2 x_2\right] \bmod 10 = [5 \cdot 1 \cdot 1 + 2 \cdot 3 \cdot 2] \bmod 10 = 7$$

Por tanto 7 es el inverso de 3 módulo 10.

## 2.1.3 Terminología básica de la teoría de la complejidad

La complejidad computacional permite fundamentar el análisis de los requisitos computacionales de las técnicas de criptoanálisis, es decir, la dificultad que comporta romper una cifra.

### *2.1.3.1 Complejidad de un algoritmo*

La fortaleza de una cifra segura computacionalmente queda determinada por la complejidad de cálculo de los algoritmos que son necesarios para romperla.

Esta complejidad de cálculo se mide por el tiempo T y el espacio E que se tarda, donde T y E se expresan como funciones de la medida n de la entrada del algoritmo. Más que hacer servir complejidades exactas $f(n)$, se suelen utilizar órdenes de magnitud $O(g(n))$, de tal manera que $f(n) = O(g(n))$ quiere decir que hay constantes $c$ y $n_0$ tales que:

$$f(n) \leq c\left|g(n)\right| \text{; para } n \geq n_0$$

El propósito que hay detrás del uso de órdenes de magnitud es que $g(n)$ sea más simple que $f(n)$.

**Cálculo del orden de magnitud de la complejidad de un algoritmo**

Si la complejidad exacta de un algoritmo es $f(n) = 24n + 16$, podemos decir que $f(n) = O(n)$, ya que:

$$24n + 16 \leq 25n, \text{ para } n \geq 16$$

De la misma manera, si $f(n)$ es un polinomio de grado t en n, podemos escribir $f(n) = O(n^t)$.

Un **algoritmo** es **polinómico** o más exactamente, **de tiempo polinómico**, si su tiempo de ejecución es $T = O(n^t)$ para alguna constante $t$. Un algoritmo polinómico es: **constante**, si $t = 0$; **lineal**, si $t = 1$; **cuadrático**, si $t = 2$; etc.

Así mismo, un **algoritmo** es **exponencial** o **de tiempo exponencial**, si $T = O(t^{h(n)})$ para una constante $t$ y un polinomio $h(n)$.

**Tiempos de ejecución de diferentes algoritmos**

Para n grande, las diferencias de complejidad temporal dan lugar a tiempos mut diferentes. Por ejemplo, supongamos que tenemos una máquina capaz de procesar $10^8$ instrucciones por segundo. La tabla siguiente muestra el tiempo de ejecución para las diferentes clases de algoritmos comentados anteriormente.

| Clase | Complejidad | Operaciones n = $10^8$ | Tiempo |
|---|---|---|---|
| Pol. consrtante | O(1) | 1 | $10^{-8}$ segundos |
| Pol. lineal | O(n) | $10^8$ | 1 segundo |
| Pol. cuadrático | $O(n^2)$ | $10^{16}$ | 1000 dias |
| Pol. cúbico | $O(n^3)$ | $10^{24}$ | $2.7 \cdot 10^8$ años |
| Exponencial | $O(2^n)$ | $10^{9030900}$ | $10^{9030886}$ años |

Las cifras seguras computacionalmente se pueden romper buscando exhaustivamente el espacio de las claves y probando cada clave posible para saber si descifra bien (con sentido) ó no. Si la medida del espacio de claves es $n = 2^{H(K)}$, entonces el tiempo de ejecución de esta estrategia es $T = O(n) = O(2^{H(K)})$. Por tanto, el tiempo es lineal en el número de claves n, pero es exponencial en la longitud de las claves H(K). Esta es la razón por lo cual doblar la longitud de la clave DES (Data Encryption Standard) de 56 a 112 bits puede tener un impacto enorme en la dificultad de romperlo (a pasar de que la distancia de unicidad sólo se dobla).

### *2.1.3.2 Complejidad de un problema*

La teoría de la complejidad clasifica un problema según el espacio y el tiempo mínimos que se requieren para resolver las instancias (problema para un valor concreto a la entrada) más difíciles con una máquina de Tunning. Una máquina de Tunning es una máquina de estados finitos con una cinta infinita de lectura/escritura. Los problemas que se pueden resolver polinómicamente en una máquina de Tunning se pueden resolver polinómicamente en un sistema real, y al revés.

Los **problemas tratables** son aquellos que se pueden resolver en tiempo polinómico en una máquina de Tunning. El resto son los problemas intratables ó difíciles. Hay problemas que son

tan difíciles que ni tan solo se puede escribir un algoritmo capaz de resolverlo; son los **problemas indecidibles**.

El conjunto de los problemas resolubles en tiempo polinómico se llama clase P. La clase polinómica no determinista, clase NP, engloba todos los problemas que se pueden resolver en tiempo polinómico en una máquina de Tunning no determinista, es decir, si la máquina encuentra la solución, puede comprobar la corrección en tiempo polinómico. Hay que destacar que esto no significa que el problema se pueda resolver, porque no hay garantía que la máquina acierte la respuesta correcta.

Es evidente que la clase NP incluye la clase P. Pero, hay problemas NP que parecen que pidan un tiempo exponencial. De todas formas, no se ha podido demostrar que P $\neq$ NP; así pues, de momento, no se puede excluir que algún día se encuentren algoritmos polinómicos para resolver todos los problemas de la clase NP.

El **problema NP-difícil** es aquel que no se puede resolver en tiempo polinómico, si no es que $P = NP$.

Un **problema NP-completo** es aquel a que cualquier otro problema de NP se puede reducir en un tiempo polinómico.

Si se encontrase un algoritmo polinómico para un problema NP-completo, entonces se habría demostrado que $P = NP$.

## 2.1.4 Problemas difíciles: logaritmo discreto y factorización

La seguridad de los criptosistemas de clave pública que funcionan en la práctica se basa en la dificultad computacional de algunos problemas NP de la teoría de los números. Vamos a comentar dos, el problema del logaritmo discreto y el problema de la factorización. Para poder entender la formulación de ambos problemas se comentan previamente unas nociones básicas de teoría de los grupos.

### *2.1.4.1 Grupos y elementos primitivos*

El grupo es un conjunto G junto con una operación, que representaremos por * . La operación hace corresponder a cada pareja de elementos a y b un tercer elemento, en concreto el resultado a * b de aplicarles la operación. Las cuatro propiedades de un grupo son las siguientes:

- La operación es interna: si $a$, $b \in G$, entonces $a * b \in G$.
- La operación asociativa: $(a * b) * c = a * (b * c)$ .
- Hay un elemento de identidad I tal que $I * a = a * I = a$ para todo $a \in G$.
- Todo $a \in G$ tiene un inverso, representado por $a^{-1}$, tal que $a * a^{-1} = a^{-1} * a = I$

Ejemplos de grupos pueden ser los enteros con la suma, los reales con la multiplicación.

En un grupo finito G, el **orden del grupo** es su cardinal $|G|$.

Si G es un grupo y $a \in G$, al coleccionar las potencias de a, es decir, $a^0, a^1, a^2, ...,$ podemos formar el conjunto:

$$< a >= \{a^i : i \geq 0\}$$

Si m = $|G|$ es el orden de G, se cumple que $a^m = 1$. Por consiguiente, la secuencia se repite después de m pasos, es decir, $a^{m+1} = a$, pero también se podría repetir antes.

Si G es un grupo y $a \in G$, el subgrupo $< a >= \{a^i : i \geq 0\}$ se llama **subgrupo generado** por $a$ y tiene un orden $t = |< a >|$ que divide el orden $m = |G|$.

**Subgrupos generados por elementos del grupo $Z_9^*$**

Si $Z_9^* = \{1,2,4,5,7,8\}$, el orden del grupo (medida del conjunto) es el número de enteros menores que 9 que son corrimos, es decir, $\phi(9) = 6$. En este caso, algunos de los subgrupos generados por sus elementos son los siguientes:

- $<1>= \{1\}$
- $<2>= \{1,2,4,8,7,5\}$
- $<4>= \{1,4,7\}$
- $<5>= \{1,5,7,8,4,2\}$

Así pues, según que elemento tomemos, podemos hacer la vuelta sin que lleguen a salir todos los elementos de $Z_9^*$.

Un elemento $g \in G$ se llama **primitivo o generador de G** si las potencias de $g$ generan G. Es decir, $< g >= G$ es todo el grupo G. Igualmente, un grupo se llama **cíclico** si tiene un elemento primitivo.

**El grupo $Z_p^*$**

Se sabe que si p es primo, el siguiente grupo:

$$Z_p^* = \{1,2,..., p-1\}$$

es cíclico. Notar que el orden del grupo es el número de enteros menores que p que son corrimos, es decir, $\phi(p) = p - 1.$

### *2.1.4.2 El problema del logaritmo discreto*

En un grupo cíclico de orden m y generador g, para cualquier $y \in G$ hay un único $i \in \{0,...,m-1\}$ tal que $g^i = y$. Este único i se representa por $\log_g y$ y se llama **logaritmo discreto de y en la base g**.

Es especialmente interesante en criptografía el caso en que $G = Z_p^*$ para p primo y g es un generador de $Z_p^*$. Entonces se pueden calcular potencias módulo p de manera rápida. Por ejemplo, con el algoritmo exp_rapid(a,z,n).
Dado $y = g^i \bmod p$, encontrar el logaritmo discreto $i$ es difícil (problema del logaritmo discreto). El mejor algoritmo actual tarda un tiempo $O(\exp\left[\sqrt{\ln p \ln \ln p}\right])$, es decir, un tiempo

exponencial. Hay una variante reciente de este algoritmo que parece requerir sólo un tiempo $O(\exp[\ln p \ln\ln p]^{1/3})$.

#### *2.1.4.3 El problema de la factorización*

Factorizar un entero n significa encontrar la descomposición en factores primos. Esta descomposición existe siempre y es única.

En criptografía es especialmente interesante la factorización de enteros de la forma $n = pq$, en que $p$ y $q$ son primos. En este caso alguien conoce n, pero no conoce ni p ni q y le gustaría encontrarlos (problema de la factorización).

El problema de la factorización también es difícil. Los mejores algoritmos, que tardan un tiempo $O(\exp[\sqrt{\ln n \ln\ln n}]$, son los de Dixon y del tamiz cuadrático. Si p es el divisor primo más pequeño de n, hay un algoritmo llamado de curva elíptica que tarda un tiempo $O(\exp[\sqrt{2\ln p \ln\ln p}])$. Este último algoritmo, por tanto, sólo se aconseja cuando se sospecha que uno de los factores de n es mucho más pequeño que el otro. Hay una propuesta reciente del algoritmo de tipo tamiz que parece tardar sólo un tiempo $O(\exp[\ln n \ln\ln n]^{1/3}$.

La complejidad del cálculo de la factorización de n=pq, por lo tanto, es exponencial con los mejores algoritmos que se conocen actualmente. De hecho, es sorprendente ver la semejanza que hay entre los mejores algoritmos conocidos para calcular logaritmos discretos y para factorizar, teniendo en cuenta que los algoritmos utilizados para resolver un problema no tienen demasiado en común con los usados en el otro problema.

## 2.2 Fundamentos de los criptosistemas de clave pública

La criptografía de clave compartida presenta tres inconvenientes:

1) **La distribución de las claves**: dos usuarios han de elegir una clave secreta antes de comenzar a comunicarse. En este caso, o bien se han de encontrar personalmente o bien han de confiar en un canal seguro para distribuirse las calves. También es perfectamente posible que no se puedan encontrar personalmente y que no dispongan de ningún canal seguro.

2) **La gestión de claves:** en una red de n usuarios, cada pareja de usuarios ha de tener su clave compartida particular, lo cual implica un total de n(n-1)/2 claves para toda la red.

3) **No hay firma digital**: la firma digital es el equivalente de las firmas manuales en el caso de la información electrónica; con un criptosistema de clave compartida, generalmente no hay posibilidad de firmar los mensajes, por el mismo hecho de que todas las claves son compartidas al menos por dos usuarios.

El concepto de criptosistema de clave pública, que permite superar los inconvenientes anteriores, fue propuesto por **W. Diffie y M.E. Hellman** en el artículo "New directions in cryptography" aparecido en el año 1976. La idea, que se puede tildar de revolucionaria, era permitir un intercambio segur de mensajes entre emisor y receptor sin tenerse que encontrar previamente para acordar una clave secreta común.

### 2.2.1 Concepto de clave pública

Un criptosistema de clave pública se dirige más a menudo a una red de usuarios que no a una sola pareja. En este criptosistema cada usuario, u, tiene asociada una pareja de claves $< P_u, S_u >$: La **clave pública**, $P_u$, que se publica con el nombre del usuario en un directorio público que todo el mundo puede leer, mientras que la clave privada, $S_u$, sólo la conoce u. Los pares de claves se generan mediante un algoritmo de generación de claves.

Para enviar un mensaje secreto m a u, todo el mundo hace servir el mismo método:

1) Buscar $P_u$.
2) Calcular $c = E(P_u, m)$, dónde E es un **algoritmo público de cifrado**.
3) Enviar c al usuario u.

Al recibir el texto cifrado c, el usuario u lo puede descifrar de la manera siguiente:

1) Buscar su clave privada $S_u$.
2) Calcular $D(S_u, c)$ en que D es un **algoritmo público de descifrado**.

Para que este procedimiento funcione es necesario que $D(S_u, E(P_u, m)) = m$. De esta manera la gestión de las claves queda simplificada porque ahora sólo hay n pares de claves para n usuarios, en cuentas de las n(n-1)/2 claves que eran necesarias con la criptografía de clave compartida.

### 2.2.2 Funciones unidireccional y unidireccional con trapa

Los criptosistemas de clave pública se basan en la existencia de algunos tipos de funciones que son difíciles de invertir.

Una **función unidireccional** $f : M \to C$ es una función invertible tal que es fácil de calcular $f(m) = c$, mientras que es difícil de calcular $f^{-1}(c) = m$.

Una función **unidireccional con trapa** es una función unidireccional que puede ser invertida fácilmente cuando se conoce cierta información adicional. Esta información se llama trapa (con el sentido figurado de 'puerta falsa').

En las definiciones anteriores, fácil quiere decir en tiempo polinómico y difícil en tiempo exponencial. Curiosamente, no se sabe a ciencia cierta si hay funciones unidireccionales, ni mucho menos, funciones unidireccionales con trapa. De toda manera, hay funciones susceptibles de ser consideradas así.

#### Posibles funciones unidireccionales

Basándonos en lo que se ha comentado al explicar los problemas difíciles, podrían ser unidireccionales las funciones siguientes:

- Exponencial discreta. Con el algoritmo exp_rapid(a,z,n) o uno parecido, es fácil de exponenciar. En cambio, volver atrás calculando logaritmos discretos es difícil.

- Producto de primos. Es fácil multiplicar números primos. En cambio, es difícil volver atrás factorizando el producto obtenido.

## 2.3 Intercambio de claves Diffie-Hellman

En un artículo de 1976, W. Diffie y M.E. Hellman hicieron una descripción de un protocolo de intercambio de claves que evita el inconveniente de la distribución de claves. El hecho innovador es que dos usuarios pueden pactar una clave secreta compartida sobre un canal inseguro.

**Protocolo 1: Intercambio Diffie-Hellman**

Para llevar a cabo correctamente el protocolo de intercambio Diffie-Hellman, se han de seguir los pasos siguientes:

Los dos usuarios, A y B, eligen públicamente un grupo multiplicativo finito G de orden n y un generador $\alpha \in G$.

1. A genera un número aleatorio $a$, calcula $\alpha^a$ dentro de G y transmite el resultado a B.

2. B genera un número aleatorio $b$, calcula $\alpha^b$ dentro de G y transmite el resultado a A.

3. A recibe $\alpha^b$ y calcula $\left(\alpha^b\right)^a$ dentro de G.

4. B recibe $\alpha^a$ y calcula $\left(\alpha^a\right)^b$ dentro de G.

Al final del protocolo 1, A y B han pactado un elemento $\alpha^{ab}$ del grupo G que es común entre ellos dos y secreto para el resto de usuarios. La seguridad del intercambio de Diffie-Hellman se basa en la dificultad del problema de Diffi-Hellman generalizado.

**Problema de Diffie-Hellman generalizado (PDHG):** El criptoanalista puede conocer toda la ejecución del protocolo 1, con lo cual obtiene G, n, $\alpha$, $\alpha^a$ y $\alpha^b$. Con esta información quiere calcular $\alpha^{ab}$.

Si $G = Z_p^*$, con p primo, el PDHG se llama simplemente problema de Diffie-Hellman (PDH). Hay que remarcar la semejanza con el problema del logaritmo discreto generalizado, que enunciamos a continuación.

**Problema del logaritmo discreto generalizado (PLDG):** un criptoanalista obtiene G, $n$, $\alpha$, $\alpha^a$ y quiere calcular $\alpha^a$.

En realidad no se ha demostrado que PHDG y PLDG sean equivalentes. De toda manera, no se ha encontrado como resolver PHDG sin resolver PLDG. Lo que está claro es que si el

criptoanalista supiera resolver PLDG, entonces podría encontrar $a$ (respectivamente $b$) a partir de $\alpha^a$ (respectivamente, $\alpha^b$) y por consiguiente, $\alpha^{ab}$.

**Funcionamiento del protocolo Diffie-Hellman**
Vamos a mostrar un ejemplo del funcionamiento de Diffie-Hellman:

1. A y B eligen públicamente $G = Z_{53}^*$ y el generador $\alpha = 2$ (se puede comprobar que las potencias de 2 generan G).

2. A selecciona $a = 29$, calcula $\alpha^a = 2^{29} \bmod 53 = 45$ y envía 45 a B.

3. B selecciona $a = 19$, calcula $\alpha^b = 2^{19} \bmod 53 = 12$ y envía 12 a A.

4. A recibe 12 y calcula $12^{29} \bmod 53 = 21$

5. B recibe 45 y calcula $45^{19} \bmod 53 = 21$

Sólo A y B saben que el número compartido es 21. Un criptoanalista enemigo ve $Z_{53}^*$, 2, 45 y 12.

## 2.4 Introducción al cifrado de clave compartida. Cifrado de bloque

Este tipo de cifrado se incluye dentro del grupo de criptosistemas de clave compartida, ya que la clave que se utiliza para cifrar y descifrar es la misma ó tienen una relación de clave pública y se comparte entre emisor y receptor. El cifrado de bloque actúa sin memoria y el texto cifrado sólo puede depender del texto en claro y de la clave. Así pues, en el cifrado de bloque, dos textos en claro iguales se cifran siempre de la mima manera cuando se utiliza la misma clave.
Este hecho se tiene que tener muy en cuenta porque los sistemas de cifrado que resultan son bastante vulnerables y si no se corrigen, se podrían insertar o borrar bloques de texto cifrado sin que se pudiera detectar. Además, el hecho de que dos bloques de texto en claro queden cifrados de la misma manera puede dar pistas en un posible criptoanálisis de tipo estadístico.

### 2.4.1 Los principios de confusión y de difusión de Shannon

Para que un criptosistema sea seguro, es necesario que la información que proporciona el texto cifrado sea la mínima posible. Esto comporta, en particular, conseguir que las propiedades estadísticas del texto en claro no se mantengan en el texto cifrado.

En un trabajo de formalización de la teoría de la información, Shannon propuso las dos técnicas que han de seguir los criptosistemas de cifrado de bloque si se quiere evitar eventuales ataques basados en métodos estadísticos.

Las dos técnicas propuestas son las siguientes:

- La **confusión**, que incluye substituciones para que la relación entre la clave y el texto cifrado sea tan complicada como se pueda.

- La **difusión**, que utiliza transformaciones que disipen las propiedades estadísticas del texto en claro entre el texto cifrado.

### 2.4.2 Características comunes

Desde un punto de vista general, todos los esquemas de cifrado de bloque tienen la misma estructura básica de funcionamiento. Todos estos, a partir de un bloque de texto en claro, B, de una longitud fija y de una clave, K, ejecutan determinadas operaciones más o menos complicadas hasta obtener el texto cifrado correspondiente, Y, de manera que:

$$Y = E_K(B)$$

Además. Las operaciones anteriores han de permitir que se pueda volver a obtener el mismo texto en claro a partir del texto cifrado Y y la clave K, si se lleva a término el proceso de descifrado siguiente:

$$B = D_K(Y)$$

Así mismo, las operaciones que se ejecutan sobre el bloque que hay que cifrar sólo combinan los mismos elementos que hay en el bloque y en la clave. Esto ocurre porque estos métodos de cifrado no incorporan memoria, de manera que en el caso de la clave K sea la misma, dos bloques que sean idénticos producirán textos cifrados idénticos.

Normalmente, la estructura básica de los criptosistemas de bloque modernos consta de dos trasformaciones, una de entrada y otra de salida, entre las cuales hay un número determinado de interacciones de una cierta función $f$, no lineal, que combina los elementos que forman parte del bloque de texto en claro con los elementos que forman parte de la clave. De hecho, generalmente la clave, K, se utiliza para generar una serie de subclaves, $K_i$, a partir de una cierta función $f_K$ suficientemente complicada y son estas subclaves las que actúan en cada interacción.

El modelo básico de funcionamiento de un criptosistema de bloque es el que se conoce con la sigla ECB (Electronic Code Block). Como la mayoría de las veces el texto que se cifra tiene una longitud superior a la longitud del bloque con el que trabaja el criptosistema, lo que se hace es partir el texto que hay que cifrar, M, en diversos bloques, $M_1$, $M_2$,…, cada uno de los cuales tiene una longitud correspondiente al bloque a cifrar. De esta manera se cifra cada uno de los bloques con una misma clave, K, para obtener el texto cifrado resultante: $Y=Y_1Y_2\ldots$

El nombre de ECB no es casual, sino que se refiere al hecho de que, una vez fijada una clave, K, a cada bloque de texto en claro corresponde un bloque concreto de texto cifrado, como si tuviéramos un diccionario para cada clave K y fuéramos buscando que palabra (bloque) de texto cifrado corresponde a cada palabra de texto en claro.

A pesar de que el modo ECB es el que parece más natural de utilizar, desde la perspectiva de la seguridad tiene bastantes inconvenientes: el hecho que el cifrado de dos bloques sea totalmente independiente hace que sea vulnerable a determinados ataques. Por ejemplo:

a) Podría ser que un atacante borrase bloques de texto cifrado, y como esto no afectaría el proceso de descifrado del resto, el receptor no lo podría detectar.

b) El atacante podría insertar bloques de texto cifrado.

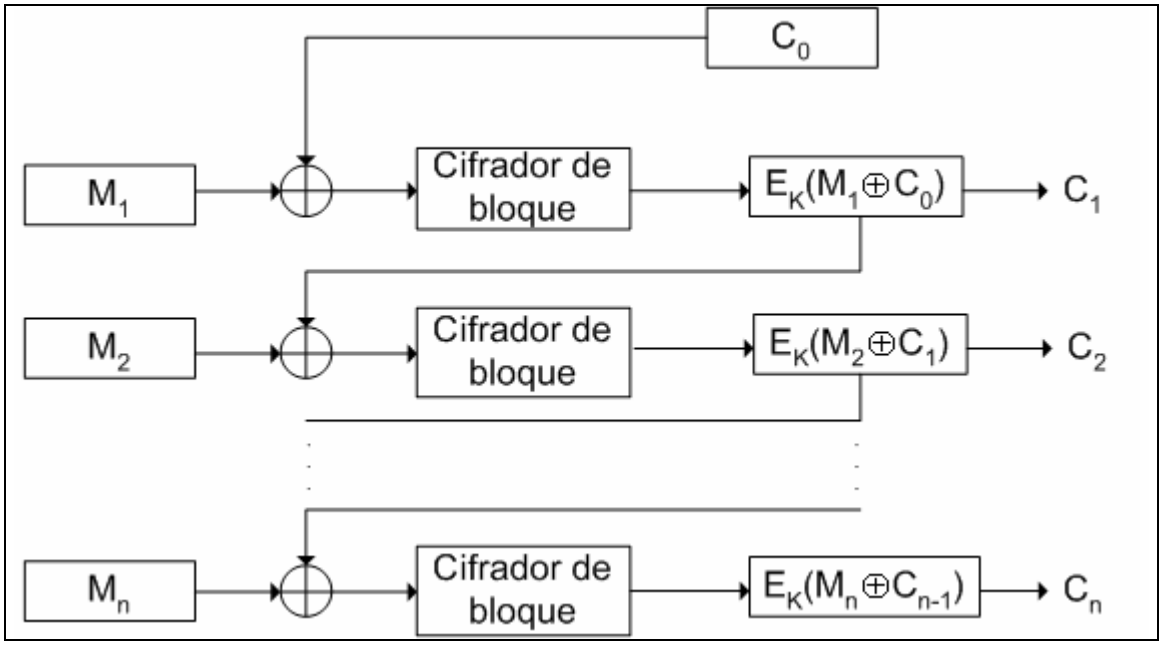
c) El hecho de que los mismos bloques queden cifrados siempre de la misma manera puede facilitar los ataques de tipo estadístico para obtener la clave K.

### 2.4.3 Modos de cifrado de bloque

Por lo que hemos visto hasta ahora, los cifrados de bloque pueden ser útiles para cifrar informaciones cortas, como identificadores, contraseñas, claves, etc.; en definitiva los mensajes en que la longitud del texto que se quiere cifrar no sobrepasa la longitud del bloque.

En cambio, la utilización del cifrado de bloque mediante el modo ECB sobre mensajes largos y, en especial, en textos que repiten determinados patrones, se desaconseja totalmente. Así pues, si queremos usar los cifrados de bloque en este tipo de textos, se tendrá que aplicar alguno de los modos de cifrado que se muestran a continuación.

**Modo CBC**
El CBC (Cipher Block Chaining) consiste en el encadenamiento de los bloques para el cifrado, de manera que se cree una dependencia del cifrado de cada bloque con el inmediatamente anterior, tal y como se muestra en la siguiente figura:




Suponiendo un cifrado de bloque con una clave K, una función de cifrado E y una de descifrado D. Si $M_1,\ldots,M_n$ son los bloques de texto en claro que hay que cifrar, mediante el sistema CBC el cifrado de bloque $M_i$ se lleva a cabo de la manera siguiente:

$$C_i = E_K(M_i \oplus C_{i-1})$$

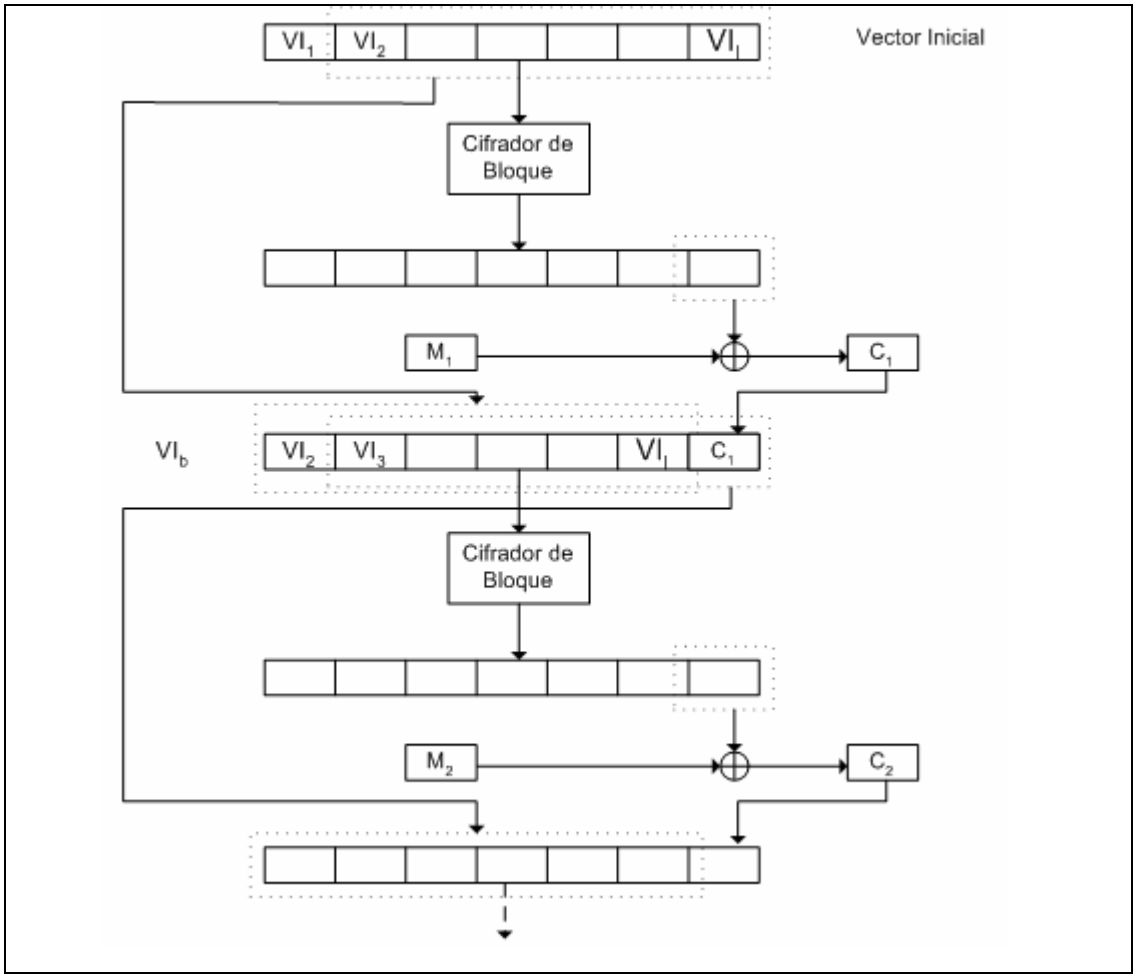
Para hacer el descifrado también nos hace falta partir el texto cifrado anterior y entonces se ha de ejecutar la operación siguiente:

$$D_K(C_i) \oplus C_{i-1} = D_K(E_K(M_i \oplus C_{i-1})) \oplus C_{i-1} = (M_i \oplus C_{i-1}) \oplus C_{i-1} = M_i$$

Para cifrar el primer bloque necesitaremos un bloque inicial aleatorio, $C_0$, que no es preciso que sea secreto. Entonces, incluyendo este nuevo vector inicial en el cifrado podremos obtener dos textos en claro iguales pero cifrados de manera diferente; así, aunque se tome la misma clave, K, sólo necesitaremos cambiar el vector inicial $C_0$, que, además, puede incorporar una marca temporal.

**Modo CFB**

El modo de cifrado CFB (Cipher Feedback) utiliza indirectamente el cifrador de bloque, como se verá a continuación. Por eso, la longitud de los bloques que se han de cifrar no es necesario que sea la misma que la de los bloques del criptosistema con que actúa, sino que puede ser más pequeña. El esquema general de funcionamiento de este método se muestra en la siguiente figura:




Dado $M = M_1 M_2 ...,$ en que $M$ es el mensaje en claro, y $M_1, M_2, ...$ representan los bloques de longitud $n$ que forman el mensaje, si consideramos el vector inicial $VI$ como una concatenación de $l$ bloques de longitud $n$, es decir, $VI = VI_1, VI_2 ... VI_l$ donde $VI_i$ tiene n bits de longitud, podremos calcular el cifrado del vector $VI$, $E(VI)$, mediante el criptosistema de bloque.

El resultado tendrá la misma longitud que $VI$ y, por tanto, lo podremos descomponer de la misma manera que aquel:

$$E(VI) = E(VI)_1 E(VI)_2 ... E(VI)_l$$

Finalmente, ya se podrá cifrar el primer bloque de texto en claro, $M_1$, haciendo la suma bit a bit con el último bloque, $E(VI)_l$:

$$C_1 = M_1 \oplus E(VI)_l$$

obtenemos así el primer bloque cifrado de longitud $n$, $C_1$.

Para cifrar el segundo bloque, $M_2$, se vuelve a hacer el mismo proceso, pero esta vez tomaremos como vector inicial el vector formado por los fragmentos siguientes:

$$VI_b = VI_2 VI_3 ... VI_l C_l$$

es decir, se han desplazado los bloques de $n$ bits hacia la izquierda para añadir el bloque $C_1$ y descartar el $VI_1$. De esta manera, el segundo bloque de texto cifrado se obtiene haciendo la operación siguiente:

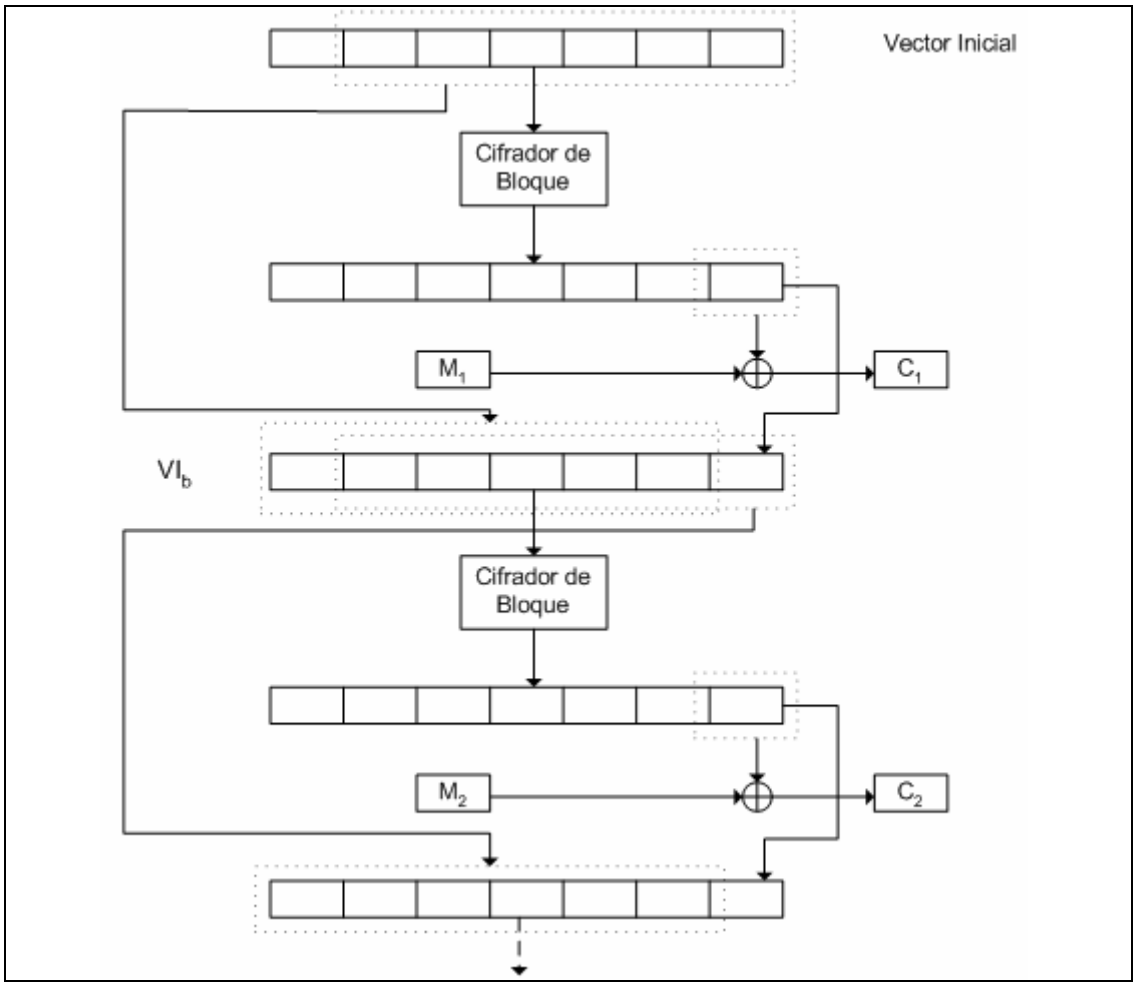
$$C_2 = E(VI_b)_1 \oplus M_2$$

El proceso se repite a lo largo de bloques de texto que se quiere cifrar: para el bloque siguiente se desplazan los bloques del vector inicial anterior, $VI_b$,... a la izquierda para añadir el último bloque de texto obtenido e ir aplicando el que ya hemos descrito anteriormente.

**Modo OFB**

El modo de cifrado OFB (Output Feedback) utiliza el criptosistema como generador pseudo aleatorio. Es un sistema muy parecido al anterior; la única diferencia que presenta es que el vector inicial se realimenta directamente con el resultado del cifrado de bloque antes de hacer la suma bit a bit con el bloque de texto en claro, como se puede ver en la siguiente figura.

Como el cifrador de bloque actúa como un generador pseudolaeatorio, es preciso que los criptosistemas de bloque que usan el modo OFB cumplan las características requeridas por los generadores pseudoaleatorios, tanto por lo que hace la impredecibilidad de la secuencia como a la complejidad lineal.




## 2.5 Criptosistemas de cifrado de bloque

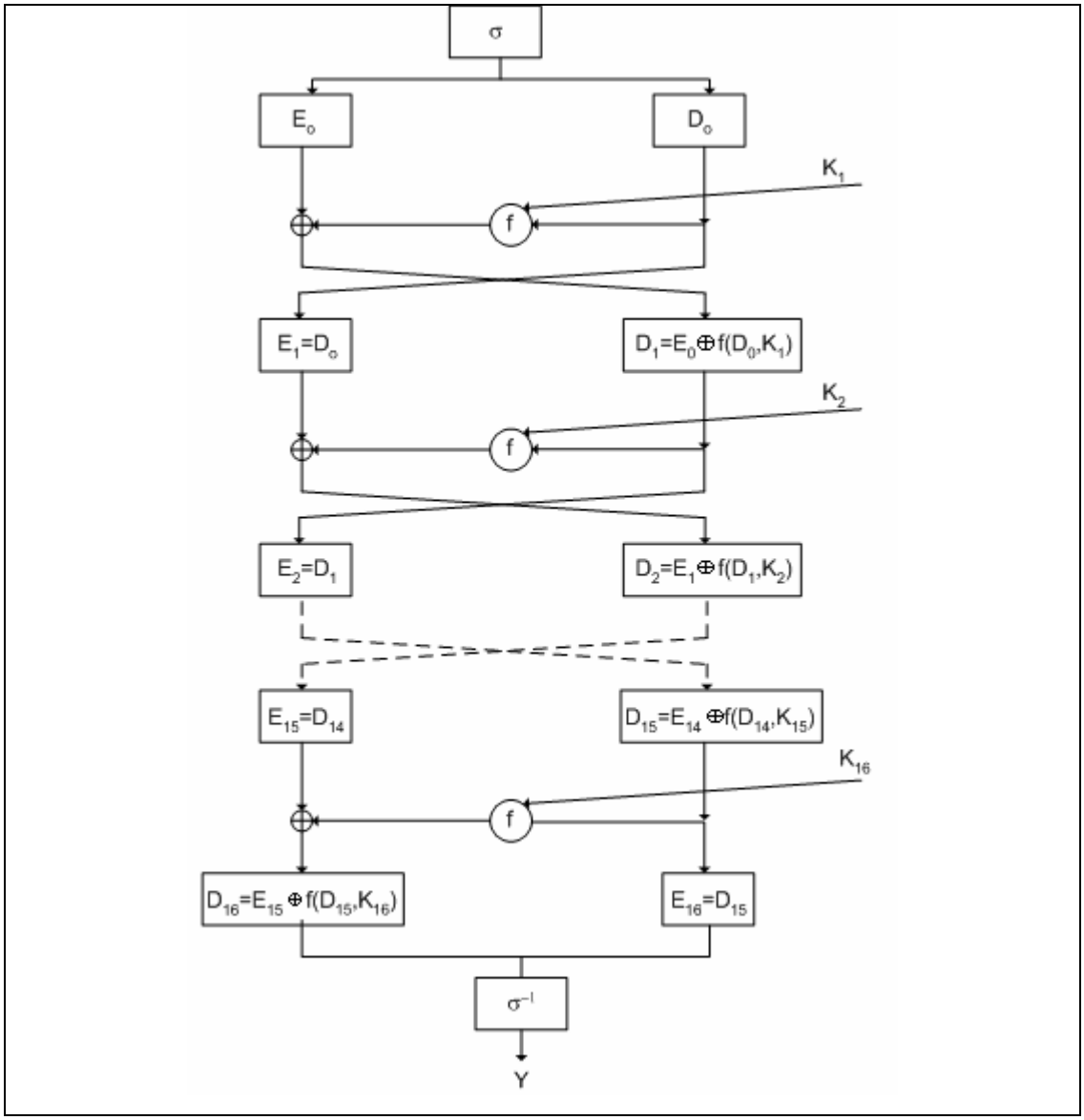
### 2.5.1 El estándar DES

EL año 1977, el Nacional Bureau of Standards (NBS) , una sección del Departamento de Defensa de los EUA, publicó un criptosistema estándar creado con la finalidad de proteger cualquier tipo de datos: el DES (Data Encryption Standard). En el desarrollo de este sistema participó la empresa IBM y la Nacional Security Agency (NSA).

El **criptosistema DES** es un criptosistema de cifrado de bloque que cifra bloques de datos de 64 bits de longitud por medio de una clave de 56 bits. El DES consigue cumplir tanto el principio de confusión como el de difusión, gracias a las acciones de las cajas S que contiene.

#### *2.5.1.1 Descripción del funcionamiento*

El funcionamiento del algoritmo de cifrado y descifrado DES queda determinado en la siguiente figura:




Como se puede observar, se suministra al algoritmo un bloque de entrada, M, sobre el cual se aplica una permutación inicial $\sigma$, de donde se obtiene $T_0 = \sigma(M)$. Cuando ya se han realizado las 16 iteracciones de la función $f$, que se explicará a continuación, el resultado se transpone por medio de la permutación de salida $\sigma^{-1}$. Las permutaciones $\sigma$ y $\sigma^{-1}$ se recogen en dos tablas:

Tabla de permutaciones $\sigma$

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 58 | 50 | 42 | 34 | 26 | 18 | 10 | 2 |
| 60 | 52 | 44 | 36 | 28 | 20 | 12 | 4 |
| 62 | 54 | 46 | 38 | 30 | 22 | 14 | 6 |
| 64 | 56 | 48 | 40 | 32 | 24 | 16 | 8 |
| 57 | 49 | 41 | 33 | 25 | 17 | 9 | 1 |
| 59 | 51 | 43 | 35 | 27 | 19 | 11 | 3 |
| 61 | 53 | 45 | 37 | 29 | 21 | 13 | 5 |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 63 | 55 | 47 | 39 | 31 | 23 | 15 | 7 |

Tabla de permutaciones $\sigma^{-1}$

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 40 | 8 | 48 | 16 | 56 | 24 | 64 | 32 |
| 39 | 7 | 47 | 15 | 55 | 23 | 63 | 31 |
| 38 | 6 | 46 | 14 | 54 | 22 | 62 | 30 |
| 37 | 5 | 45 | 13 | 53 | 21 | 61 | 29 |
| 36 | 4 | 44 | 12 | 52 | 20 | 60 | 28 |
| 35 | 3 | 43 | 11 | 51 | 19 | 59 | 27 |
| 34 | 2 | 42 | 10 | 50 | 18 | 58 | 26 |
| 33 | 1 | 41 | 9 | 49 | 17 | 57 | 25 |

La interpretación de estas tablas resulta muy sencilla. Por ejemplo, la tabla correspondiente a la permutación $\sigma$ nos indica que el bit de la posición 1 de entrada queda en la posición 58 de salida; el 2, en la 50; el 3 en la 42, etc.

### *2.5.1.2 Detalle de una interacción*

Entra las dos permutaciones, $\sigma$ y $\sigma^{-1}$, el algoritmo efectúa 16 interacciones de la función $f$ que combinan sustituciones y transposiciones.

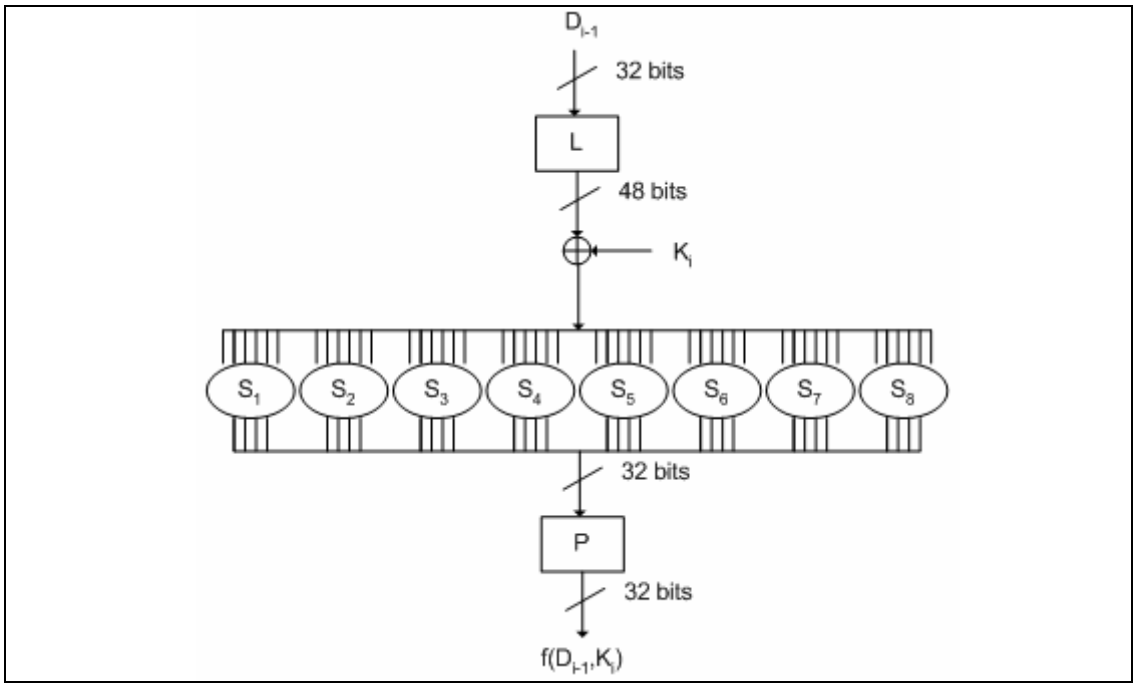
Si suponemos que $T_i$ es el resultado de la i-ésima interacción, $T_i$ tiene 64 bits de longitud, por tanto, lo podemos dividir en dos mitades iguales de 32 bits de manera que $T_i = E_i D_i$, en que la parte izquierda es $E_i = t_1 ... t_{32}$ y la derecha es $D_i = t_{33} ... t_{64}$; entonces, para cada interacción se verificarán las condiciones que se presentan a continuación:

- $E_i = D_{i-1}$
- $D_i = E_{i-1} \oplus f(D_{i-1}, K_i)$

en que $K_i$ es una clave de 48 bits de longitud, que se explica más adelante.

Hay que puntualizar que en el esquema general del DES no se intercambian los lados de las partes en la última interacción, sino que se hace la permutación inversa de la inicial, $\sigma^{-1}$, para obtener la salida final, $Y$. De esta manera, el mismo algoritmo sirve tanto para cifrar como para descifrar.

El valor que se obtiene de la función $f(D_{i-1}, K_i)$, se puede entender con la figura que se presenta a continuación:




En primer lugar se transforma $D_{i-1}$ de manera que tenga una longitud de 48 bits ya que hay que recordar que $D_{i-1}$ es la mitad derecha de $T_{i-1}$ y por tanto sólo tiene 32 bits. Para extender $D_{i-1}$ se han repetido algunos de sus bits por medio de la transformación de expansión $L$, que queda determinada en la siguiente tabla:

| | | | | | |
|---|---|---|---|---|---|
| 32 | 1 | 2 | 3 | 4 | 5 |
| 4 | 5 | 6 | 7 | 8 | 9 |
| 8 | 9 | 10 | 11 | 12 | 13 |
| 12 | 13 | 14 | 15 | 16 | 17 |
| 16 | 17 | 18 | 19 | 20 | 21 |
| 20 | 21 | 22 | 23 | 24 | 25 |
| 24 | 25 | 26 | 27 | 28 | 29 |
| 28 | 29 | 30 | 31 | 32 | 1 |

el primer bit de salida corresponde al bit 32 de entrada; el segundo bit de salida, al 1 de entrada, etc.

A continuación, siguiendo el esquema DES, hacemos la suma bit a bit de la mitad expandida, $L(D_{i-1})$, con la clave $K_i$.

El resultado de esta operación se descompone en ocho bloques de 6 bits cada uno, es decir:

$$L(D_{i-1}) \oplus K_i = B_1 B_2 ... B_8$$

Cada bloque $B_j$ se usa como entrada del se llama una caja S. Cada caja, $S_j$, recibe su bloque correspondiente de 6 bits $B_j = b_1 b_2 b_3 b_4 b_5 b_6$ y devuelve uno de 4 bits de longitud, según la siguiente tabla:

| | | Columna | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Caja S | Fila | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| $S_1$ | 0 | 14 | 4 | 13 | 1 | 2 | 15 | 11 | 8 | 3 | 10 | 6 | 12 | 5 | 9 | 0 | 7 |
| | 1 | 0 | 15 | 7 | 4 | 14 | 2 | 13 | 1 | 10 | 6 | 12 | 11 | 9 | 5 | 3 | 8 |
| | 2 | 4 | 1 | 14 | 8 | 13 | 6 | 2 | 11 | 15 | 12 | 9 | 7 | 3 | 10 | 5 | 0 |
| | 3 | 15 | 12 | 8 | 2 | 4 | 9 | 1 | 7 | 5 | 11 | 3 | 14 | 10 | 0 | 6 | 13 |
| $S_2$ | 0 | 15 | 1 | 8 | 14 | 6 | 11 | 3 | 4 | 9 | 7 | 2 | 13 | 12 | 0 | 5 | 10 |
| | 1 | 3 | 13 | 4 | 7 | 15 | 2 | 8 | 14 | 12 | 0 | 1 | 10 | 6 | 9 | 11 | 5 |
| | 2 | 0 | 14 | 7 | 11 | 10 | 4 | 12 | 1 | 5 | 8 | 12 | 6 | 9 | 3 | 2 | 15 |
| | 3 | 13 | 8 | 10 | 1 | 3 | 15 | 4 | 2 | 11 | 6 | 7 | 12 | 0 | 5 | 14 | 9 |
| $S_3$ | 0 | 10 | 0 | 9 | 14 | 6 | 3 | 15 | 5 | 1 | 13 | 12 | 7 | 11 | 4 | 2 | 8 |
| | 1 | 13 | 7 | 0 | 9 | 3 | 4 | 6 | 10 | 2 | 8 | 5 | 14 | 12 | 11 | 15 | 1 |
| | 2 | 13 | 6 | 4 | 9 | 8 | 15 | 3 | 0 | 11 | 1 | 2 | 12 | 5 | 10 | 14 | 7 |
| | 3 | 1 | 10 | 13 | 0 | 6 | 9 | 8 | 7 | 4 | 15 | 14 | 3 | 11 | 5 | 2 | 12 |
| $S_4$ | 0 | 7 | 13 | 14 | 3 | 0 | 6 | 9 | 10 | 1 | 2 | 8 | 5 | 11 | 12 | 4 | 15 |
| | 1 | 13 | 8 | 11 | 5 | 6 | 15 | 0 | 3 | 4 | 7 | 2 | 12 | 1 | 10 | 14 | 9 |
| | 2 | 10 | 6 | 9 | 0 | 1 | 11 | 7 | 13 | 15 | 1 | 3 | 14 | 5 | 2 | 8 | 4 |
| | 3 | 3 | 15 | 0 | 6 | 10 | 1 | 13 | 8 | 9 | 4 | 5 | 11 | 12 | 7 | 2 | 14 |
| $S_5$ | 0 | 2 | 12 | 4 | 1 | 7 | 10 | 11 | 6 | 8 | 5 | 3 | 15 | 13 | 0 | 14 | 9 |
| | 1 | 14 | 11 | 2 | 12 | 4 | 7 | 13 | 1 | 5 | 0 | 15 | 10 | 3 | 9 | 8 | 6 |
| | 2 | 4 | 2 | 1 | 11 | 10 | 13 | 7 | 8 | 15 | 9 | 12 | 5 | 6 | 3 | 0 | 14 |
| | 3 | 11 | 8 | 12 | 7 | 1 | 14 | 2 | 13 | 6 | 15 | 0 | 9 | 10 | 4 | 5 | 3 |
| $S_6$ | 0 | 12 | 1 | 10 | 15 | 9 | 2 | 6 | 8 | 0 | 13 | 3 | 4 | 14 | 7 | 5 | 11 |
| | 1 | 10 | 15 | 4 | 2 | 7 | 12 | 9 | 5 | 6 | 1 | 13 | 14 | 0 | 11 | 3 | 8 |
| | 2 | 9 | 14 | 15 | 5 | 2 | 8 | 12 | 3 | 7 | 0 | 4 | 10 | 1 | 13 | 11 | 6 |
| | 3 | 4 | 3 | 2 | 12 | 9 | 5 | 15 | 10 | 11 | 14 | 1 | 7 | 6 | 0 | 8 | 13 |
| $S_7$ | 0 | 4 | 11 | 2 | 14 | 15 | 0 | 8 | 13 | 3 | 12 | 9 | 7 | 5 | 10 | 6 | 1 |
| | 1 | 13 | 0 | 11 | 7 | 4 | 9 | 1 | 10 | 14 | 3 | 5 | 12 | 2 | 15 | 8 | 6 |
| | 2 | 1 | 4 | 11 | 13 | 12 | 3 | 7 | 14 | 10 | 15 | 6 | 8 | 0 | 5 | 9 | 2 |
| | 3 | 6 | 11 | 13 | 8 | 1 | 4 | 10 | 7 | 9 | 5 | 0 | 15 | 14 | 2 | 3 | 12 |
| $S_8$ | 0 | 13 | 2 | 8 | 4 | 6 | 15 | 11 | 1 | 10 | 9 | 3 | 14 | 5 | 0 | 12 | 7 |
| | 1 | 1 | 15 | 13 | 8 | 10 | 3 | 7 | 4 | 12 | 5 | 6 | 11 | 0 | 14 | 9 | 2 |
| | 2 | 7 | 11 | 4 | 1 | 9 | 12 | 14 | 2 | 0 | 6 | 10 | 13 | 15 | 3 | 5 | 8 |
| | 3 | 2 | 1 | 14 | 7 | 4 | 10 | 8 | 13 | 15 | 12 | 9 | 0 | 3 | 5 | 6 | 11 |

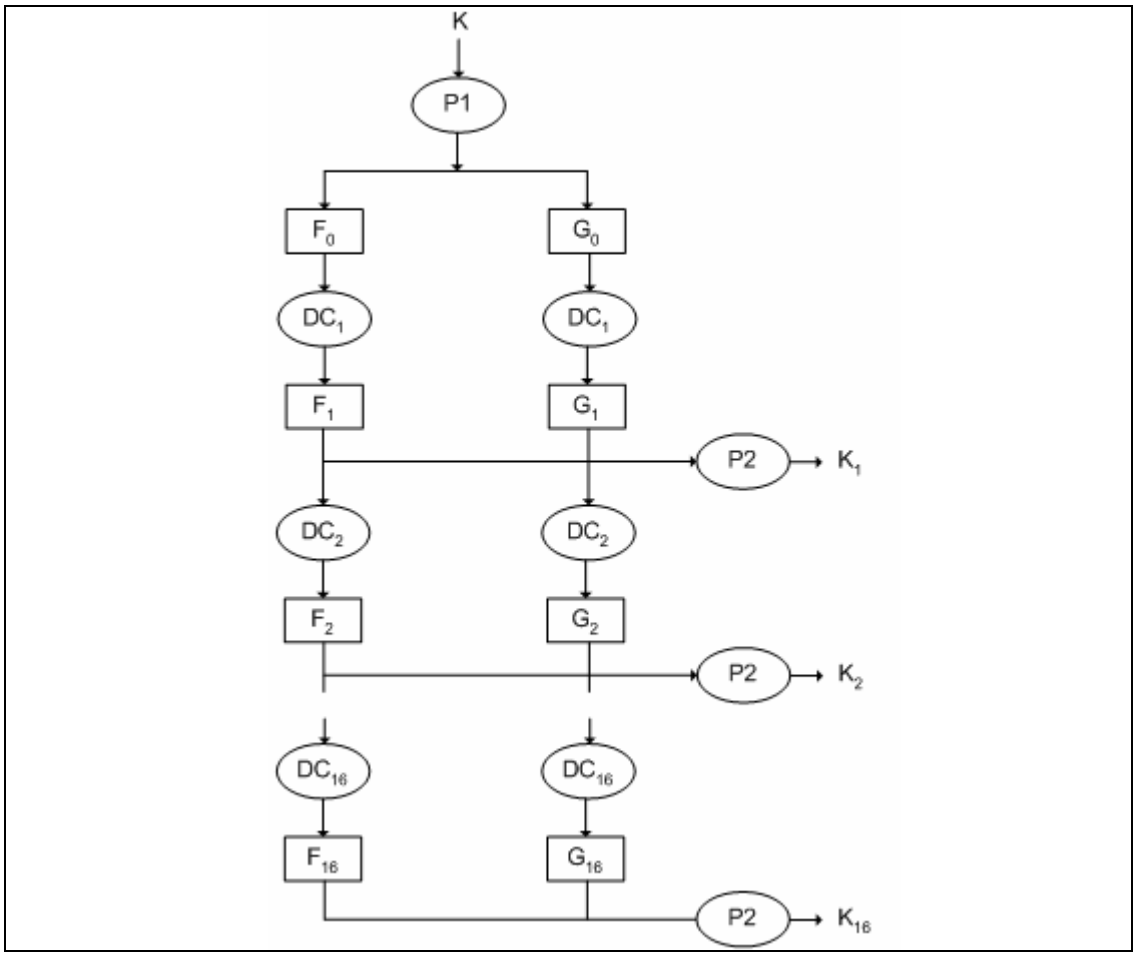
El criterio por el medio del cual se asigna una fila y una columna de una caja en $B_j$ es el siguiente: el índice $j$ fija la caja $S_j$, el entero correspondiente a $b_1b_6$ selecciona la fila y el entero que corresponde a $b_2b_3b_4b_5$ determina la columna.

A continuación se toman los bloques resultantes de las cajas $S$ y se concatenan hasta obtener un bloque de 32 bits de longitud. Finalmente se aplica sobre este último bloque la permutación $P$ definida en la siguiente tabla para obtener el valor $f(D_{i-1}, K_i)$.

| | | | |
|---|---|---|---|
| 16 | 7 | 20 | 21 |
| 29 | 12 | 28 | 17 |
| 1 | 15 | 23 | 26 |
| 5 | 18 | 31 | 10 |
| 2 | 8 | 24 | 14 |
| 32 | 27 | 3 | 9 |
| 19 | 13 | 30 | 6 |
| 22 | 11 | 4 | 25 |

### *2.5.1.3 Generación de subclaves*

Al comenzar la descripción del criptosistema DES se ha señalado que operaba con una clave, $K$, de 56 bits de longitud. En cambio, el algoritmo que hemos descrito utiliza 16 subclaves $K_i$ de 48 bits. Veamos, por tanto, como se pueden obtener estas subclaves a partir de la clave principal. El algoritmo de generación de subclaves se ilustra en la siguiente fgura:



La tabla de permutaciones es la siguiente:

| | | | | | | |
|---|---|---|---|---|---|---|
| 57 | 49 | 41 | 33 | 25 | 17 | 9 |
| 1 | 58 | 50 | 42 | 34 | 26 | 18 |
| 10 | 2 | 59 | 51 | 43 | 35 | 27 |
| 19 | 11 | 3 | 60 | 52 | 44 | 36 |
| 63 | 55 | 47 | 39 | 31 | 23 | 15 |
| 7 | 62 | 54 | 46 | 38 | 30 | 22 |
| 14 | 6 | 61 | 53 | 45 | 37 | 29 |
| 21 | 13 | 5 | 28 | 20 | 12 | 4 |

En esta tabla no aparecen las posiciones añadidas 8, 16,…,64

En la clave inicial, $K$ de 56 bits se han añadido 8 bits más en las posiciones 8, 16, …,64 que corresponden a las paridades. La permutación P1 descarta los bits de paridad y transporta los 56 restantes.
El resultado de hacer la permutación $P1(K)$ es dividir $K$ en dos mitades, $F_0$ y $G_0$, de 28 bits cada una. Los bloques $F_0$ y $G_0$ se desplazan a la izquierda para obtener cada subclave $K_i$. Si representamos por $F_i$ y $G_i$ los valores utilizados para obtener $K_i$, tenemos:

- $F_i = DC_i(F_{i-1})$
- $G_i = DC_i(G_{i-1})$

dónde $DC_i$ es un desplazamiento circular hacia la izquierda de tantas posiciones como determina la tabla de desplazamiento siguiente:

| **Número de Interacción i** | **Desplazamientos i a la izquierda** |
|---|---|
| 1 | 1 |
| 2 | 1 |
| 3 | 2 |
| 4 | 2 |
| 5 | 2 |
| 6 | 2 |
| 7 | 2 |
| 8 | 2 |
| 9 | 1 |
| 10 | 2 |
| 11 | 2 |
| 12 | 2 |
| 13 | 2 |
| 14 | 2 |
| 15 | 2 |
| 16 | 1 |

Finalmente, la subclave $K_i$ se obtiene por medio de la expresión:

$$K_i = P2(F_i G_i)$$

en que $P2$ es la permutación especificada en la tabla siguiente:

| | | | | | |
|---|---|---|---|---|---|
| 14 | 17 | 11 | 24 | 1 | 5 |
| 3 | 28 | 15 | 6 | 21 | 10 |
| 23 | 19 | 12 | 4 | 26 | 8 |
| 16 | 7 | 27 | 20 | 13 | 2 |
| 41 | 52 | 31 | 37 | 47 | 55 |
| 30 | 40 | 51 | 45 | 33 | 48 |
| 44 | 49 | 39 | 56 | 34 | 53 |
| 46 | 42 | 50 | 36 | 29 | 32 |

#### *2.5.1.4 Descifrado*

Hasta aquí se ha descrito el proceso de cifrado del criptosistema DES; ahora, para descifrar los datos de un bloque $Y$ cifrado con una clave $K$ se hace servir el mismo algoritmo. En este caso, sin embargo, una vez generadas las subclaves de la misma manera que antes, las haremos servir pero en orden inverso, en el interacciones que se muestran en la figura del esquema general del DES. Así en la primera interacción se utiliza $K_{16}$, en la segunda $K_{15}$ hasta llegar a $K_1$, utilizada en la última interacción.

#### *2.5.1.5 Particularidades del criptosistema*

El criptosistema DES se puede implementar tanto en software como en hardware. Naturalmente, la implementación en hardware le confiere una velocidad de cifrado y de descifrado muy superior a las implementaciones hechas en software.

Por lo que respecta a los modos de cifrado de bloques, hay que destacar que el DES puede trabajar con todos los modos (CBC, CFB, OFB), gracias a la complejidad de sus operaciones, que permiten que pueda actuar como un generador pseudo aleatorio.

Desde la aparición del DES el año 1976, hasta incluso su estandarización, ya surgieron diferentes voces críticas sobre este criptosistema. Las dos debilidades más graves que se le atribuyen son la poca longitud de la clave, de sólo 56 bits, y la arbitrariedad de las cajas S, que podría obedecer a la existencia de una clave maestra que permitiera descifrar cualquier mensaje sin tener la clave original.

Por lo que respecta a la longitud de la clave, Diffie y Hellman consideraron que se podría construir un ordenador con una arquitectura de procesadores paralelos que llegase a romper el DES en un día haciendo una búsqueda exhaustiva de claves. Estas estimaciones y otras realizadas a comienzos de los años ochenta, parece que no se cumplieron, por lo menos en medios amplios, ya que el coste que suponía construir un ordenador de estas características era muy elevado. No obstante, el tiempo ha pasado y los avances en materia de tecnología informática han sido muy importantes, tanto que han permitido romper sistemas, como el DES en modo ECB en 22 horas y 15 minutos. La clave del mensaje era 8558891AB0C851B6 y el mensaje que escondía era “Strong cryptography makes the world a saber place”.

Para resolver el problema de la longitud de la clave y continuar usando el DES como criptosistema seguro, a la espera que se homologue un nuevos Standard, se utiliza el cifrado triple, que en el caso del DES es conocido como triple DES.

## 2.6 Sockets

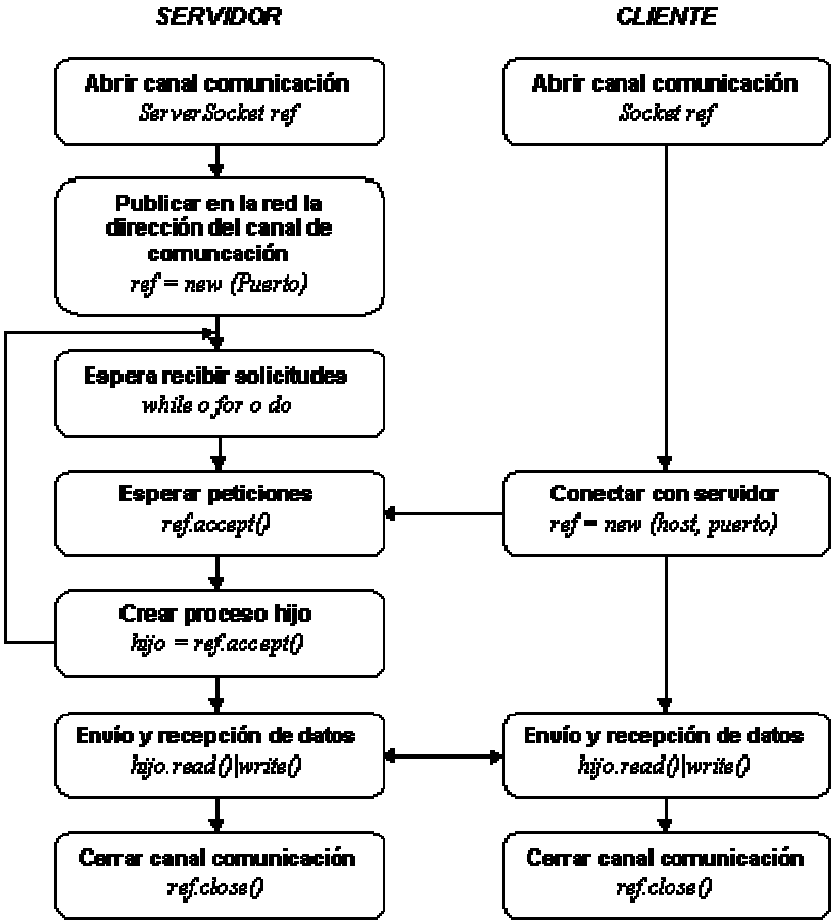
En este apartado se detalla de forma genérica el concepto de socket .

Los sockets son un sistema de comunicación entre procesos de diferentes máquinas de una red. Más exactamente, un socket es un punto de comunicación por el cual un proceso puede emitir o recibir información.

Fueron popularizados por *Berckley Software Distribution*, de la universidad norteamericana de Berkley. Los *sockets* han de ser capaces de utilizar el protocolo de streams TCP (Transfer Control Protocol) y el de datagramas UDP (User Datagram Protocol).

Utilizan una serie de primitivas para establecer el punto de comunicación, para conectarse a una máquina remota en un determinado puerto que esté disponible, para escuchar en él, para leer o escribir y publicar información en él, y finalmente para desconectarse.
Con todas primitivas se puede crear un sistema de diálogo muy completo.

En el siguiente se muestra el funcionamiento de una conexión socket.




El protocolo de comunicación de datagramas (UDP) es un protocolo no orientado a la conexión, lo que significa que cada vez que se envían datagramas también se necesita enviar el descriptor de socket local y la dirección del socket receptor. Esto se significa que se debe de enviar información adicional cada vez que se hace una comunicación

El protocolo de comunicación de flujo conocido como TCP (Transfer Control Protocol). A diferencia de UDP, TCP es un protocolo orientado a la conexión. Para realizar la conexión sobre el protocolo TCP, se debe de realizar en primer lugar una conexión entre ambos sockets. Mientras que uno de los sockets escucha una petición (servidor), el otro pregunta por un conexión (cliente). Una vez que los sockets se han conectado se pueden usar para transmitir datos en ambas direcciones.

En función del tipo de aplicación cliente-servidor se usa un protocolo u otro. En UDP, cada vez que se envía un datagrama se ha de enviar el descriptor y la dirección del socket receptor. En cambio TCP es un protocolo orientado a la conexión, se debe de establecer primero una conexión antes que comiencen las comunicaciones de los sockets. Por tanto se necesita un tiempo de preparación para la conexión TCP.

En UDP, hay un tamaño máximo de 64 kilobytes por datagrama en cada envío mientras que en TCP no hay límite. Cuando se establece una conexión, la pareja de sockets actúan como flujos: todos los datos disponibles son leídos inmediatamente en el mismo orden en que se reciben.

UDP es un protocolo poco fiable, no se garantiza que los datagramas enviados se reciban en el mismo orden por el socket receptor. El protocolo TCP es un protocolo fiable, se garantiza que los paquetes que se envían se reciban en el orden en que se han enviado.

En resumen, TCP es muy competente para implementar servicios de red, como login remoto y transferencia de archivos, que requieren una longitud de datos ilimitada para enviar. UDP es menos complejo e incurre en menos gastos. Es usado a menudo en implementar aplicaciones cliente/servidor en sistemas distribuidos construidos sobre redes de área local.

# 3. Software criptográfico de implementación

Este proyecto se ha desarrollado en su totalidad con el lenguaje de programación Java, se ha utilizado el SDK de Sun sin incorporar librerías extras. Este entorno de desarrollo, en su versión 1.4 ya tiene incorporado una amplia colección de clases para dar soporte a la implementación de técnicas criptográficas. Estas clases se recogen en la llamada JCE (Java Cryptography Extensión). La raíz del soporte de seguridad que ofrece Java es el API de seguridad. La primera versión del API Security en el JDK 1.1 introdujo la JCA (Java Cryptography Architecture), es un entorno de trabajo para acceder y desarrollar funcionalidades criptográficas en la plataforma Java. En la versión del JDK 1.1 la JCA incluyó APIs para firmar digitales y firmas de mensajes (Message Digest). En las siguientes versiones el JDK 1.2 amplió significativamente la JCA, también actualizó la infraestructura de manejo de certificados para soportar certificados X.509 v3 e introdujo una arquitectura de seguridad más manejable, altamente configurable, flexible y un control de acceso extendido.
La Java Crytography Extensión JCE amplió el API de la JCA para dar soporte a la encriptación, el intercambio de claves y el MCA (Message Authentication Code). Juntos, la JCE y los aspectos de la criptografía del SDK proporcionan una API criptográfica completa e independiente de la plataforma de trabajo. La JCE comenzó siendo una extensión, un paquete opcional, del SDK Java 2, a partir de la versión 1.4 ya ha sido incorporado.

## 3.1 JCA

### 3.1.1 Principios de diseño

La JCA fue diseñada siguiendo las siguientes premisas:

- Independencia de la implementación e interoperabilidad
- Independencia de algoritmo y extensibilidad.

La independencia

## 3.1.2 Arquitectura

### *3.1.2.1 Proveedores de servicio criptográficos*

La JCA introdujo la noción de un Cryptographyc Service Provider. Este término se refiere a un paquete o conjunto de paquetes que suministran una implementación concreta de un subconjunto de aspectos criptográficos del API de seguridad.

Por ejemplo, en el JDK 1.1, un proveedor podía contener una implementación de uno ó más algoritmos de firma digital, algoritmos de resumen de mensajes y algoritmos de generación de claves. El SDK Java 2 añade cinco tipos adicionales de servicios: Factorías de claves, creación y manejo de almacenes de claves, manejo de parámetros de algoritmos y factorías de claves. También proporciona un proveedor para suministrar un algoritmo de generación de números aleatorios (RNG).

Un programa puede únicamente solicitar un tipo particular de objeto (como un objeto Signatura) para un servicio particular (como un algoritmo de firma DSA) y tomar un implementación de uno de los proveedores instalados.

Alternativamente, el programa puede solicitar los objetos de un proveedor específico. (Cada proveedor tiene un nombre para ser referenciado).

La versión Sun de Java viene por defecto con el proveedor llamado SUN. Otros JRE de Java no tienen porqué llevar incorporado este proveedor.

El paquete proveedor SUN incluye:

- Una implementación del Digital Signatura Algorithm (DSA).
- Una implementación de los algoritmos de resumen de mensajes MD5 y SHA-1.
- Un generador de pares de claves DSA para la generación de parejas de de claves, clave privada y pública, adecuados para el algoritmo DSA.
- Un generador de parámetros del algoritmo DSA.
- Un manejador de parámetros del algoritmo DSA.
- Una factoría de claves DSA que proporciona conversiones en dos sentidos entre objetos de claves pública y privada DSA y su material de claves interno.
- Una implementación del algoritmo de generación de números pseudo-aleatorios “SHA1PRNG.

- Un constructor y validador de certificados para PKIX.

- Una implementación de almacén de claves de certificados para recuperar certificados y CRLs de colecciones y directorios LDAP.

- Una factoría de certificados para certificados X.509 y listas de revocación de certificados (CRLs).

- Una implementación de almacén de claves para el propietario de tipo de almacén de claves llamado JKS.

Cada instalación SDK tiene uno ó más paquetes de proveedor instalados. Proveedores nuevos pueden añadirse estática ó dinámicamente. La JCA ofrece un conjunto de APIs que permiten a los usuarios consultar qué proveedores están instalados y qué servicios soportan.

#### *3.1.2.2 Manejo de claves*

Una base de datos llamada "almacén de claves" se puede utilizar para manejar un repositorio de claves y certificados. Un almacén de claves está disponible para las aplicaciones que lo necesiten para propósitos de certificación o de firma.

Las aplicaciones pueden acceder a un almacén de calves a partir de una implmentación de la clase KeyStore, la cual, está en el paquete java.Security.

Sun Microsystems proporciona una implementación por defecto de KeyStore. Implementa el almacén de claves como un archivo utilizando un tipo ó formato de almacén propietario llamado "JKS".

Las aplicaciones pueden proporcionar diferentes tipos de implementación de almacenes de clave de proveedores diferentes.

### 3.1.3 Conceptos

En esta sección se cubre la mayor parte de los conceptos introducidos en la API.

#### *3.1.3.1 Clases motor y algoritmos*

Una clase motor define un servicio criptográfico de manera abstracta.

Un servicio criptográfico siempre se asocia a un algoritmo o tipo determinado, y, o bien proporciona operaciones criptográficas (como firmas digitales o resúmenes de mensajes), ó suministra el material criptográfico (claves ó parámetros) requeridos para operaciones criptográficas, ó genera objetos de datos (almacenes de claves o certificados) que encapsulan claves criptográficas (las cuales pueden ser usadas en una operación criptográfica) de una forma segura. Por ejemplo, dos de las clases motor son Signature y KeyFactory. La clase Signature genera y proporciona acceso a la funcionalidad de un algoritmo de firma digital. Un KeyFactory DSA suministra una clave pública o privada DSA en un formato que puede ser usado en los métodos initSign ó initVerify, respectivamente de un objeto Signature DSA.

La JCA incluye las clases del paquete de seguridad del SDK Java 2 relacionados con l criptografía incluyendo las clases motor. Los usuarios del API solicitan y usan instancias de las clases motor para llevar a cabo las operaciones correspondientes. Las siguientes clases motor son las definidas en el SDK Java 2.:

- MessageDigest: Se utiliza para calcular el resumen del mensaje (hash) de los datos especificados.

- Signature: Se utiliza para firmar datos y verificar firmas digitales.

- KeyPairGenerator: Se utiliza para generar una pareja de claves (pública y privada) utilizados para un algoritmo específico.

- KeyFactory: Se utiliza para convertir claves criptográficas opacas de tipo Key en especificaciones de claves (representación transparente del material clave tras de él) y viceversa.

- CertificateFactory: Se utiliza para crear certificados de clave pública y listas de revocación de claves.

- KeyStore: Se utiliza para crear y manejar un almacén de claves. Un almacén de claves es una base de datos de claves. Las claves privadas es un almacén de claves tienen asociado un certificado, el cual autentifica la correspondiente clave pública. Un almacén de claves contiene certificados de entidades certificadoras.

- AlgoritmParameters: Se utiliza para manejar lo parámetros para un algoritmo particular, incluyendo parámetros de codificación y decodificación.

- AlgorithmParameterGenerator: Se utiliza para generar un conjunto de parámetros utilizados por un algoritmo específico.

- SecureRandom: Se utiliza para generar números aleatorios o pseudo-aleatorios.

A partir de la versión 1.4 del SDK Java 2 se han incorporado nuevas clases motor:

- CertPathBuilder: Se utiliza para construir enlaces de certificados.

- CertPathValidator: Se utiliza para validar los enlaces de los certificados.

- CertStore: Se utiliza para recuperar certificados y CRLs de un repositorio.

Una clase motor proporciona el interface para la funcionalidad de un tipo específico de servicio criptográfico. Define métodos que permiten a las aplicaciones acceder al tipo específico de servicio criptográfico que proporciona. Las implementaciones específicas son para algoritmos específicos. La clase motor Signatura, por ejemplo, proporciona acceso a la funcionalidad de un algoritmo de firma digital. La implementación actual que está implementada en la subclase SignatureSpi, puede ser para un tipo específico de algoritmo de firmado, como SHA-1 con DSA, SHA1 con RSA ó MD5 con RSA.

Las interfaces de aplicación suministrados por una clase motor son implementados por un interfase de proveedor de servicios (SPI), es decir, para cada clase motor, hay una clase abstracta SPI que define los métodos SPI que los proveedores de servicios criptográficos deben implementar.

Una instancia de una clase motor, el objeto API, encapsula (como un campo privado) una instancia de la clase SPI correspondiente, el objeto SPI. Todos los métodos de un objeto API están declarados como final y sus implementaciones invocan los correspondientes métodos SPI del objeto SPI encapsulado. Una instancia de una clase motor (y de sus correspondientes clases SPI) se crea llamando al método de factoría getInstance de la clase motor.

El nombre de cada clase SPI es el mismo que el correspondiente a la clase motor seguido por Spi.

Cada clase SPI es abstracta. Para suministrar la implementación de un tipo particular de servicio, de un algoritmo específico, un proveedor debe hacer una subclase de la clase SPI correspondiente y proporcionar una implementación para todos los métodos abstractos.

Otro ejemplo de una clase motor es la clase MessageDigest que proporciona acceso a un algoritmo de resumen de mensaje. Sus implementaciones, en las subclases MessageDigestSpi, pueden ser para diversos algoritmos de resumen de mensajes tales como SHA-1, MD5 ó MD2.

Como último ejemplo, la clase motor KeyFactory soporta la conversión de claves opacas a especificaciones de claves transparentes y viceversa La subclase KeyFactorySpi suministra una implementación para un tipo específico de clases, por ejemplo, claves públicas y privadas DSA.

#### *3.1.3.2 Implementación y proveedores*

La JCA proporciona implementaciones para varios servicios criptográficos. Los servicios suministrados son esencialmente paquetes que suministran una ó más implementaciones del servicio criptográfico.

#### *3.1.3.3 Métodos factoría para obtener instancias de implementación*

Para cada clase motor en el API, una implementación particular se solicita e instancia llamando a un método factoría sobre la clase motor. Un método factoría es un método estático que devuelve una instancia de una clase.

El mecanismo básico, por ejemplo, para obtener un objeto Signature apropiado es el siguiente:

Cada usuario solicita un objeto llamando al método getInstance en la clase Signature, especificando el nombre del algoritmo de firma, por ejemplo, “SHA1withDSA”, y opcionalmente el nombre del proveedor o la clase Provider.

El método getInstance busca una implementación que satisfaga el algoritmo informado y los parámetros del proveedor. Si no se especifica proveedor, getInstance busca los proveedores registrados en orden de preferencia y con una implementación del algoritmo especificado.

### 3.1.4 Clases centrales e interfases

Esta sección presenta las clases centrales e interfases proporcionados en la JCA.

- Clases Provider y Security

- Clases motor MessageDigest, Signatura, KeyPairGenerator, KeyFactory, AlgorithmParameters, AlgorithmParameterGenerator, CertificateFactory, KeyStore, SecureRandom, CertificateBuilder, CertPathValidator y CertStore.

- Interfaces y clases Key

- Interfaces y clases de especificación de parámetros de algoritmo y las clases e interfases de especificaciones de claves.

#### *3.1.4.1 Clase Provider*

El término "Proveedores de Servicio Criptográfico" se refiere a un paquete o un conjunto de paquetes que suministran una implementación concreta de un subconjunto de las propiedades criptográficas del API del SDK Java 2. La clase Provider es la interfase que implementa este paquete o conjunto de paquetes. Tiene métodos para acceder al nombre del proveedor, la versión, el número y otra información. Esta clase también se puede utilizar para registrar implementaciones de otros servicios de seguridad que pueden estar definidas como parte del API de seguridad del SDK Java 2 ó una de sus extensiones.

Para suministrar implementaciones de servicios criptográficos, una entidad (por ejemplo, un grupo de desarrollo) escribe el código de implementación y crea una subclase de la clase Provider. El constructor de la subclase Provider inicializa los valores de varias propiedades; el API de seguridad del SDK Java 2 usa estos valores para buscar el servicio que el proveedor implementa. En otras palabras, la subclase especifica los nombres de las clases implementando los servicios.

Diferentes implementaciones pueden tener diferentes características. Algunas pueden estar basadas en software mientras que otras en hardware. Unas pueden ser independientes de la plataforma y otras dependientes.

#### *3.1.4.2 Clase Security*

La clase Security maneja proveedores instalados y proveedores de seguridad ampliada. Sólo contiene métodos static y nunca se instancia. Los métodos para añadir o eliminar proveedores y los métodos para poner a punto las propiedades de Secutity sólo pueden ser ejecutados por un programa de confianza.

#### *3.1.4.3 Clase MessageDigest*

La clase MessageDigest es una clase motor diseñada para proporcionar la funcionalidad de asegurar criptográficamente resúmenes de mensajes tales como SHA-1 ó MD5. Un resumen de mensaje asegurado criptográficamente toma entradas de tamaño arbitrario y genera una salida de tamaño fijo llamado resumen. Un resumen tiene dos propiedades:

- A de ser computacionalmente imposible encontrar dos mensajes que tengan el mismo mensaje.
- El resumen no tendría que revelar nada sobre los datos de entrada utilizados para crearlo.

Los resúmenes de mensajes son usados para producir identificadores de datos únicos y fiables. Son llamados en ocasiones "huellas digitales" de datos.

### *3.1.4.4 Clase Signature*

La clase Signatura es una clase motor diseñada para proporcionar la funcionalidad de un algoritmo de firma digital criptográfico tal como el DSA, DSA con MD5. Un algoritmo de firma seguro criptográficamente toma entradas de tamaño arbitrario y una clave privada, generando una pequeña cadena bytes llamada firma con las siguientes propiedades:

- Dada la clave pública correspondiente a la clave privada utilizada para generar la firma, tiene que ser posible verificar la autenticidad e integridad de la entrada.

- La firma y la clave pública no revelan nada sobre la clave privada.

Un objeto Signatura puede ser usado para firmar datos. También se puede usar para verificar si una firma es válida o no.

### *3.1.4.5 Clases de parámetros de algoritmo*

#### 3.1.4.5.1 Clases e interfases de especificación de parámetro de algoritmo

Una especificación de parámetro de algoritmo es una representación de los conjuntos de parámetros usados con un algoritmo.

Una representación transparente de un conjunto de parámetros significa que se puede acceder a cada valor de los parámetros individualmente. Se puede acceder a través de uno de los métodos get definidos en la correspondiente clase.

Por el contrario, la clase AlgoritmParameters suministra una representación opaca en que no se tiene acceso directo a los campos de los parámetros. Sólo se puede obtener el nombre del algoritmo asociado con el conjunto del parámetro y algún tipo de codificación para el conjunto parámetro.

**Interfase AlgorithmParameterSpec**

Es una interfase para una especificación transparente de los parámetros criptográficos. Esta interfase no contiene métodos o constantes. Su único propósito es agrupar todas las especificaciones de parámetros las cuales deben implementar esta interfase.

**Clase DSAParameterSpec**

Esta clase implementa la interfase AlgorithmParameterSpec, especifica el conjunto de parámetros usados con el algoritmo DSA.

#### 3.1.4.5.2 Clase AlgorithmParameters

Esta es una clase motor que proporciona una representación opaca de los parámetros criptográficos. Una representación opaca es una representación en que no se tiene acceso directo a los campos de los parámetros; sólo se puede tomar el nombre del algoritmo asociado con el conjunto del parámetro y algún tipo de codificación para el conjunto del parámetro. Esto es en contraste para una representación transparente de parámetros en que se puede acceder a cada

valor individualmente a través de alguno de los métodos get definidos en la correspondiente clase de especificación.

#### 3.1.4.5.3 Clase AlgorithmParameterGenerator

Esta es una clase motor que se usa para generar un conjunto de parámetros adecuados para un cierto algoritmo (el algoritmo especificado al crear la instancia de AlgorithmParameterGenerator).

#### 3.1.4.5.4 Interfase Key

La interfase Key es la interfase principal para todas las clases opacas. Define la funcionalidad compartida por todos los objetos claves.

Todas las claves opacas tienen tres características:

**Un algoritmo**
El algoritmo de clave para la clave. El algoritmo clave es normalmente un algoritmo de operación simétrico o asimétrico (como DSA o RSA), el cual trabajará con aquellos algoritmos y relacionados (como MD5 con RSA, SHA-1 con RSA, etc.).

**Una forma codificada**
La forma codificada externa para la clave utilizada cuando la clave se necesita fuera de la máquina virtual java una representación estándar.

**Un formato**
El nombre del formato de la clave codificada.

Las claves se obtiene generalmente a través de generadores de clave, certificados, especificaciones de clave, o una implementación de un KeyStore accediendo a una base de datos de claves usada para manejar claves.

Aquí se presenta una lista de interfases que extienden el interface Key.

- DHPrivateKey
- DHPublicKey
- DSAPrivateKey
- DSAPublicKey
- PBEKey
- PrivateKey
- PublicKey
- RSAMultiPrimePrivateCrtKey
- RSAPrivateCrtKey
- RSAPrivateey
- RSAPublicKey
- SecretKey

#### 3.1.4.5.5 Clase e interfases de especificación de claves

Las especificaciones de clave son representaciones transparentes del material de clave que constituye la clave.

Una clave puede ser especificada a partir de un algoritmo específico o en un formato independiente del algoritmo de codificación.

**Interfase KeySpec**
Esta interfase no contiene métodos o constantes. Su propósito sólo es agrupar y proporcionar una forma de contenedor de todas las especificaciones de la clave. Todas las especificaciones de clave deben de implementar esta interfase.

**Clase DSAPrivateKeySpec**
Esta clase (implementa la interfase KeySpec) especifica una clave privada con sus parámetros asociados.

**Clase DSAPublicKeySpec**
Esta clase especifica una clave pública con sus parámetros asociados.

**Clase RSAPrivateKeySpec**
Esta clase especifica una clave privada RSA.

**Clase RSAPrivateCrtKeySpec**
Esta clase especifica un clave privada RSA tal y como se defina en el estándar PKCS # usando el teorema chino del resto.

**Clase RSAMultiPrimePrivateCrtKeySpec**
Esta clase especifica una clave privada multi-primo RSA tal y como lo define el estándar PKCS# v2.1 usando el teorema chino del resto.

**Clase RSAPublicKeySpec**
Esta clase especifica una clave pública RSA.

**Clase EncodedKeySpec**
Esta clase abstracta representa una clave pública o privada en formato codificado.

### 3.1.4.5.6 Clase KeyFactory

La clase KeyFactory es una clase motor diseñada para proporcionar conversiones entre claves criptográficas opacas (ó tipo Key) y especificaciones de clave (representación transparente).

Las factorías de clave son bidireccionales. Permiten contruir objetos de claves opacos a partir de una especificación de clave o recuperar el material usado para construir el objeto de clave.

Pueden existir para una misma clave múltiples especificaciones de clave compatibles. Por ejemplo, una clave pública DSA se puede especificar por sus componentes o por su codificación DER.

Una factoría de clave puede ser usada para pasar entre especificaciones de clave compatibles.

### 3.1.4.5.7 Clase CertificateFactory

La clase CertificateFactory es una clase motor que define ka funcionalidad de una factoría de certificado, la cual puede ser usada para generar certificados y listas de revocación de certificados (CRLs) desde sus codificaciones.

#### 3.1.4.5.8 Clase KeyPair

La clase KeyPair es un simple receptáculo para una pareja de claves (una pública y la otra privada).

#### 3.1.4.5.9 Clase KeyPairGenerator

La clase KeyPairGenerator es una clase motor utilizada para generar pares de clave (pública + privada).

#### 3.1.4.5.10 Manejo de claves

Una base de datos llamada "almacén de claves" puede ser usada para manejar un repositorio de claves y certificados.

**Localización del almacén de claves**
El almacén de claves es por defecto almacenado en un archivo llamado .keystore en el directorio de inicio del usuario.

**Implementación del almacén de claves**
La clase KeyStore suministra interfases bien definidos para acceder y modificar le información en un almacén de claves. Es posible que existan múltiples implementaciones concretas diferentes dónde cada implementación sea para un tipo particular de almacén de claves.

**Clase KeyStore**
La clase KeyStore es una clase motor que suministra interfases para acceder y modificar la información de un almacén de claves.

Esta clase representa una colección en memoria de claves y certificados. KeyStore maneja dos tipos de entrada.

- Entrada Clave
- Entrada de certificado autentificado

#### 3.1.4.5.11 Clase SecureRandom

Esta clase es una clase motor que proporciona la funcionalidad de generación de números aleatorios.

## 3.2 JCE

La JCE proporciona un entorno de trabajo e implementaciones para realizar encriptación, generación y convenio de claves y código de autenticación de mensajes. El soporte para encriptación incluye cifradores de flujo y bloque, asimétricos y simétricos.

Utiliza la misma arquitectura de proveedor que la JCA.

El API JCE cubre lo siguiente:

- Encriptación por bloque simétrica como DES, RC2 e IDEA.
- Encriptación por flujo simétrica como RC4.
- Encriptación simétrica como RSA.
- Encriptación basada en password (PBE).
- Convenio de clave.
- Código de autenticación de mensaje (MAC).

El SDK Java 2 v.1.4 viene con el proveedor estándar llamado "SunJCE", el cual viene preinstalado y registrado y suministra los siguientes servicios criptográficos:

- Una implementación de los algoritmos de encriptación DES (FIPS PUB 46-1), Triple DES y Blowfish en el Electronic Code Book (ECB), modos Cipher Block Chiang (CBC), Cipher Feedback (CFB), Output Feedback (OFB) y Propagating Cipher Block Chiang (PCBC).

- Generadores de claves para la generación de claves adecuadas para los algoritmos DES, Triple-DES, Blowfish, HMAC-MD5 y HMAC-SHA1.

- Una implementación del algoritmo MD5 con encriptación basado en password (PCB) DES-CBC definido en PKCS #5.

- "Factorías de clave secreta" proporcionando conversiones bidireccionales entre objetos clave opacos DES, Triple DES y PBE y representación transparente de su material de implementación.

- Una implementación del algoritmo de convenio de clave Diffie-Hellman entre dos o más partes.

- Un generador de pares de clave Diffie-Hellman para generar una pareja de valores secreto y público adaptados para el algoritmo Diffie-Hellman.

- Un generador de parámetros del algoritmo Diffie-Hellman.

- Una "Factoría de Clave" Diffie-Hellman proporcionando conversiones bidireccionales entre objetos clave Diffie-Hellman opacos y su representación transparente.

- Manejadores de parámetros de algoritmo para Diffie-Hellman, DES, Triple DES, Blowfish y parámetros PBE.

- Una implementación del esquema de relleno descrito en PKCS #5

- Una implementación de almacén de claves para el tipo de almacén propietario "JCEKS"

El JCE incorporado en el SDK Java 2 v.1.4 incluye dos componentes de software:

- El entorno de trabajo que define y da soporte a servicios criptográficos y que los proveedores pueden suministrar implementaciones.

- Un proveedor llamado "SunJCE".

### 3.2.1 Conceptos

Esta sección proporciona una descripción a alto nivel de los conceptos implementados por la API y el significado de los términos técnicos usados en la especificación de la API.

**Encriptación y desencriptación**
La encriptación el proceso de tomar datos (texto en claro) y una pequeña cadena (clave) y producir datos (texto cifrado) sin sentido para terceras partes que no conocen la clave. La desencriptación es el proceso inverso, es decir, a partir del texto cifrado y con una pequeña cadena se llega al texto en claro.

**Encriptación basada en password**
Este tipo de encriptación deriva de una clave de encriptación de un password.

**Cifrador**
La encriptación y la desencriptación se hace a partir de una cifrador. Un cifrador es un objeto capaz de llevar a cabo la encriptación y la desencriptación de acuerdo a un esquema de encriptación (algoritmo).

**Convenio de clave**
El convenio de clave es un protocolo por el que dos ó mas partes pueden establecer las misma s claves criptográficas sin tener que intercambiar ninguna información secreta.

**Código de autentificación de mensaje**
Un MAC proporciona una forma de chequear la integridad de la información transmitida sobre o almacenada en un medio poco fiable, basado en una clave secreta. Normalmente, MAC son usados entre dos partes que comparten una clave secreta con el objetivo de validar información transmitida entre dos partes.

Un mecanismo MAC que se basa en funciones hash criptográficas está referenciado a HMAC. HMAC puede ser usado en cualquier función hash, por ejemplo MD5 ó SHA-1 en combinación con una clave secreta compartida.

### 3.2.2 Clases centrales

#### *3.2.2.1 Clase Cipher*

La clase Cipher proporciona la funcionalidad de un cifrador usado para encriptar y desencriptar. Forma la parte central del entorno de trabajo JCE.

#### *3.2.2.2 Clases de flujo y cifrado*

JCE introduce el concepto de flujos seguros, los cuales combinan un InputStream o OutputStream con un objeto Cipher. Los flujos seguros están implementados con las clases CipherInputStream y CipherOutputStream.

#### *3.2.2.3 Clase KeyGenerator*

Un generador de clave se usa para generar claves secretas para algoritmos simétricos.

#### *3.2.2.4 Clase SecretKeyFactory*

Esta clase representa una factoría para clases secretas. Las factorías de clave se usan para convertir claves (claves criptográficas opacas) en especificaciones de clave (representación transparente) y viceversa.

Un objeto de este tipo opera sólo sobre claves (simétricas) secretas si un objeto java.security.keyFactory procesa componentes de clave pública y privada de una pareja de claves.

Los objetos de tipo java.Security.Key de los que java.Secutity.PublicKey, PrivateKey y javax.crypto.SecretKey son subclases, son objetos clave opacos ya que no se puede decir cómo se han implementado. La implementación interna es dependiente del proveedor y puede estar basada en software o en hardware. Las factorías de claves permiten a los proveedores suministrar sus propias implementaciones de claves criptográficas.

Por ejemplo, si se tiene una especificación para una clave pública Diffie-Hellman consistente en un valor público, el módulo p y la base g, y se realiza la misma especificación a factorías de clave Diffie-Hellman de proveedores diferentes los objetos resultantes PublicKey tendrán implementaciones diferentes.

#### *3.2.2.5 Clase SealedObject*

Esta clase permite a un programador crear un objeto y protegerlo confidencialmente con un algoritmo criptográfico.

Dado un objeto que implementa la interfase java.io.Serializable, se puede crear un SealedObject que encapsule el objeto original en formato serializado y sellar (encriptar) su contenido serializado usando un algoritmo criptográfico como DES para protegerlo confidencialmente. El contenido puede ser desencriptado con el proceso inverso.

#### *3.2.2.6 Clase KeyAgreement*

Esta clase proporciona la funcionalidad de protocolo de convenio de clave. Las claves implicadas en establecer un secreto compartido se crean por uno de los generadores de claves (KeyPairGenerator ó KeyGenerator), un KeyFactory ó como resultado de una fase intermedia del protocolo de convenio de claves.

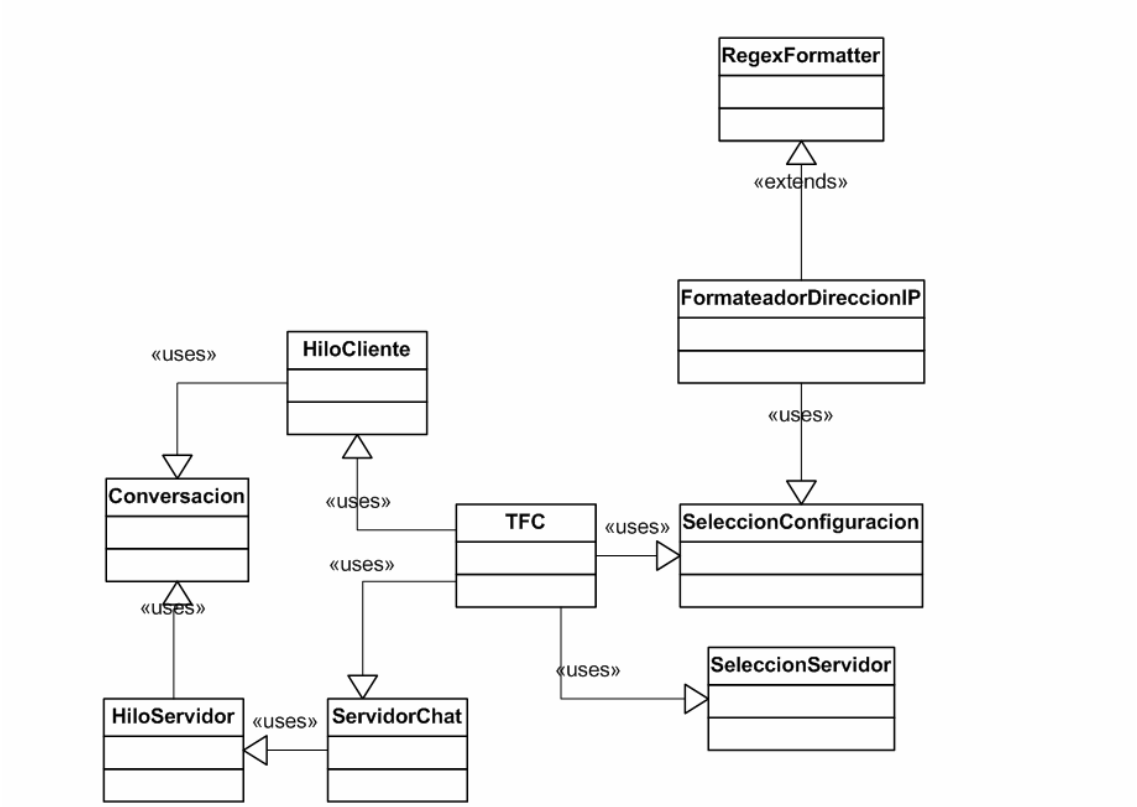
#### *3.2.2.7 Clase MAC*

Esta clase proporciona la funcionalidad de un código de autenticación de mensaje (MAC).

# 4. Diseño del programa

En este apartado se presenta el trabajo realizado con una explicación a alto nivel de su diseño y su funcionamiento.

El esquema general de la relación de clases que intervienen en este trabajo es el siguiente:




Como se puede observar se puede considerar que existe una clase central, la clase TFC. Esta clase representa el entorno gráfico o interfase de pantalla para permitir la entrada de datos del usuario y dónde irán llegando las peticiones externas de otros usuarios en forma de pantalla. Desde esta clase se usa de una forma u otra el resto de clases, las cuales tienen una función bien definida que se irá explicando en los siguientes apartados.

## 4.1 Descripción de clases

En este apartado se presenta la descripción y las funciones de cada una de las clases que intervienen en este trabajo:

### 4.1.1 Clase TFC

Tal y como hemos comentado anteriormente esta es la clase central y es ésta la clase de entrada a la aplicación.

Esta clase contiene el entorno gráfico para realizar la presentación y permitir el intercambio de datos con el usuario. Este entorno gráfico se compone de un componente gráfico Java llamado JDeskTopPane para agrupar en un mismo entorno todas las ventanas de conversación que se vayan creando. La idea es tener todos los componentes gráficos agrupados en un mismo entorno de trabajo y evitar tener ventanas dispersas.

Teniendo en cuenta que esta clase compone el entorno gráfico principal para realizar la entrada de datos por parte del usuario, posee las opciones necesarias para realizar tanto la

configuración del módulo servidor de la aplicación como la petición de conexión a otro host de la red que actúa como servidor.

Con esto se deduce que no necesariamente cuando se ejecuta la aplicación podamos recibir peticiones de conversación por parte de otros host de la red ya que el módulo servidor se arranca a petición del usuario. No es una opción que por defecto se arranque cuando se inicia la aplicación.

### 4.1.2 Clase SeleccionConfiguracion

Esta clase gestiona la configuración de la aplicación y es ejecutada desde la opción de menú "Configuración" del menú Configurar que gestiona la clase principal TFC.

Los valores que se han establecido para realizar una configuración de la aplicación son los siguientes:

- **Nick**: Este valor permite introducir un nombre o alias con el que el usuario quiere presentarse cuando establece una conexión con otro. Este valor se incorporará al título de la ventana de conversación del usuario con el que se esté chateando para que éste conozca en todo momento con quién esta conversando.

- **Puerto**: Este valor representa el puerto de escucha que la aplicación utilizará si se activa el módulo servidor, es decir, si se activa la aplicación para permitir recibir peticiones de charla, el socket servidor de la aplicación esperará conexiones escuchando el puerto que aquí se haya informado.

- **Activar Servidor**: Este valor indica si se desea activar el módulo servidor, por defecto está marcado como no. No se aceptan conexiones de otros usuarios.

- **Rastrear Comunicación**: Este valor indica si se desea rastrear el proceso de conexión con otro usuario, se utiliza para ver la información que se va intercambiando entre dos usuarios mediante la grabación en un archivo de la información que se va intercambiando.

### 4.1.3 Clase SeleccionServidor

Esta clase gestiona la petición de conexión a otro host que esté actuando como servidor.
Se solicitan los siguientes valores:

- Dirección: En este campo se informa la dirección IP del ordenador actuando como servidor con el que deseamos entablar una conversación.

- Puerto: En este campo se debe de informar el número de puerto por el cual el servidor que hemos seleccionado en el valor anterior está escuchando las peticiones que le van llegando.

Una vez informados y aceptados estos valores el proceso principal intentará establecer un socket de comunicación con el socket del servidor que debe de estar a la escucha. Si este proceso tiene éxito se crea un nuevo proceso para atender a esta conversación y aparece una nueva ventana de conversación en el escritorio principal de la aplicación para comenzar la conversación.

### 4.1.4 Clase HiloCliente

Esta clase gestiona las peticiones de conexión que se hace a otros host que actúan como servidor de conversaciones, es decir, en este caso se actúa como cliente y se solicita conversación con otro host actuando como servidor.

Cuando se va a establecer una conexión con el servidor se ejecuta esta clase, la cual, generará un nuevo hilo de ejecución independiente del resto de procesos para establecer la conversación establecida. Si la conexión tiene éxito aparecerá una nueva ventana en el escritorio de la aplicación para comenzar la conversación solicitada.

Hay que tener en cuenta que las conversaciones van encriptadas y que para ello se utiliza el protocolo Diffie-Hellman para realizar el intercambio de claves. Para que este protocolo funcione y se pueda realizar el intercambio de claves es necesario utilizar los mismos parámetros de generación entre los dos hosts.

En esta aplicación es el ordenador que solicita la conversación quien aporta estos valores especiales al algoritmo Diffie-Hellman, por tanto, es desde esta clase "HiloCliente" dónde se establecen estos valores especiales para su posterior envío al servidor.

### 4.1.5 Clase ServidorChat

Al comenzar esta descripción de las clases que componen esta aplicación se ha comentado que por defecto no se establece ningún servidor para escuchar las posibles peticiones de conversación que provengan del resto de la red. Cuando se activa el módulo servidor de la aplicación lo que se hace es generar un nuevo hilo de proceso a partir de esta clase y que únicamente se limita a estar a la escucha de forma indefinida por el puerto que se haya configurado.

De esta forma la aplicación está preparada para recibir peticiones de otros usuarios de la red para establecer conversaciones.

### 4.1.6 Clase HiloServidor

Cuando se tiene arrancado el módulo servidor de la aplicación, se está en disposición de recibir peticiones de conversación del resto de la red.

Cuando se recibe una petición y la conexión tiene éxito se lanza un nuevo proceso exclusivo para esta nueva conversación liberando al módulo servidor para que continúe a la escucha de nuevos peticiones externas. Este nuevo proceso lo lleva a cabo esta clase. Este nuevo proceso será quien realice todas las operaciones necesarias para comenzar una nueva conversación

### 4.1.7 Clase Conversacion

Esta es la clase principal para llevar a cabo una conversación. Dispone de dos constructores diferentes para diferenciar si la conversación se habilita como cliente o bien como servidor. La diferencia está en que actuando como cliente se reciben los parámetros comunes para la generación del intercambio de clave Diffie-Helman y actuando como servidor estos parámetros son recibidos por el cliente que ha solicitado la conversación.

En esta clase se realiza el intercambio de claves, se genera una nueva ventana de conversación que se incorpora al escritorio principal de nuestra aplicación y se encriptan y desencriptan los diferentes mensajes que se intercambian en la conversación.

### 4.1.8 Clase RegexFormatter y FormaetadorDireccionIP

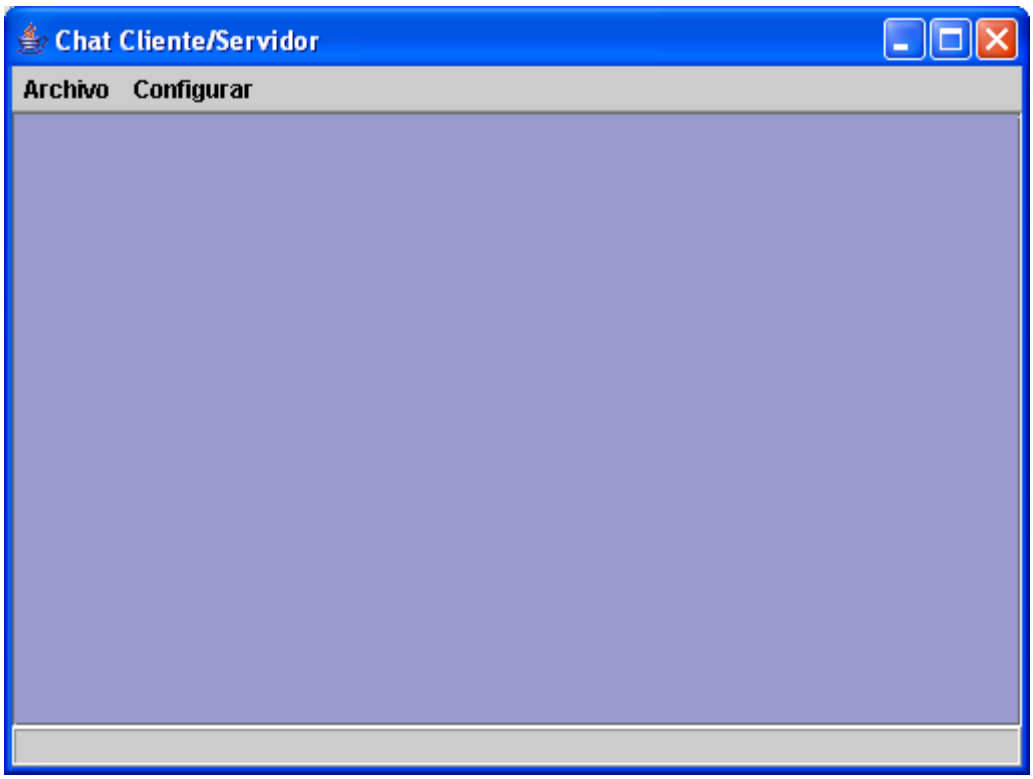
Estas clases son clases auxiliares que no intervienen directamente en la aplicación, es decir, no son necesarias para establecer conversaciones con otros usuarios.

Su objetivo es la de dar formato a los datos introducidos por pantalla y controlar su legalidad como valores correctos.

## 4.2 Funcionamiento

En este apartado se presenta el funcionamiento de la aplicación. La clase principal de entrada es la clase TFC. Esta clase no recibe parámetros para su ejecución.

La primera pantalla que aparece es la siguiente:




Como se puede ver aparece un escritorio vacío dónde se irán incorporando las diferentes ventanas que representan cada una de las conversaciones que se han establecido.

Existe un menú con dos opciones principales. En la opción "Archivo" existen dos opciones que son:

- Conectar
- Salir

La primera de ellas, **Conectar**, es la opción mediante la cual se solicita una conversación con otro host de la red que esté actuando como servidor. Nos aparece la siguiente pantalla:


Como se puede ver aparece una ventana solicitando los valores de la dirección IP y el puerto de escucha del host con el que se desea conectar, por defecto se cargan los valores del mismo ordenador, es decir, la dirección 127.0.0.1 (localhost) y el número de puerto 6666.

La siguiente opción, **Salir**, tal y como indica su nombre se limita a terminar la aplicación cerrando previamente todos los sockets activos en ese momento, ya que son recursos de la propia máquina y puede dar problemas en futuras conexiones si previamente no han sido finalizados normalmente.

En la segunda opción de menú "Configurar" existe una única opción:

- Configuración

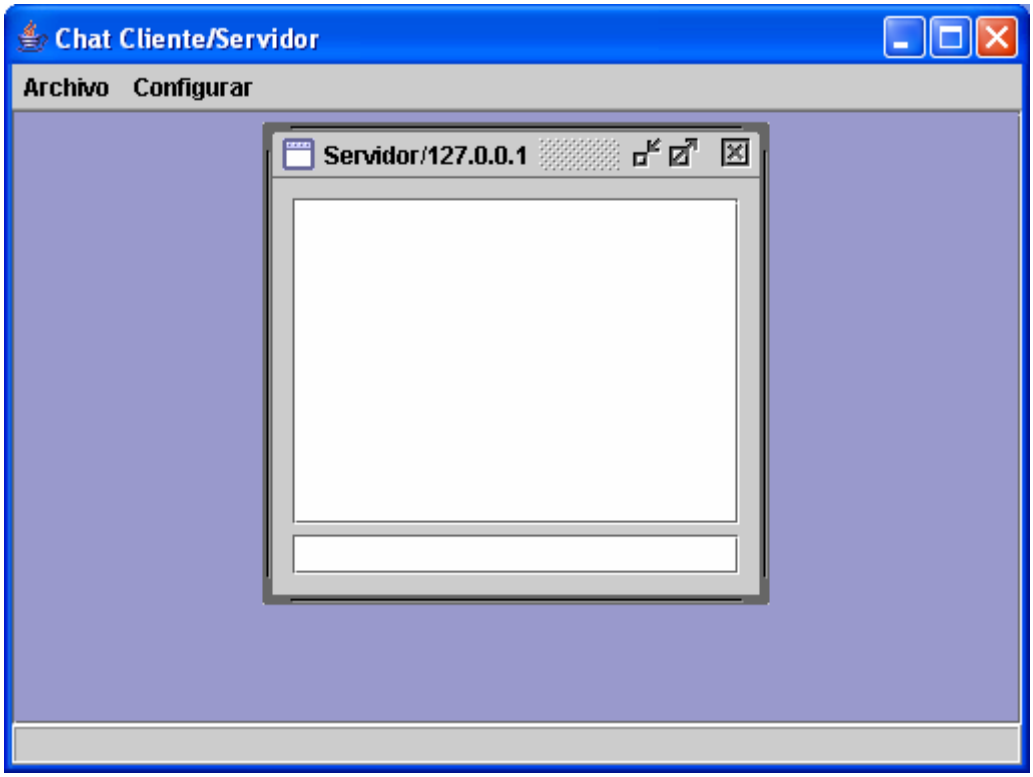
Esta opción permite realizar la configuración de nuestra aplicación. Al seleccionarla, aparece la siguiente pantalla:


En esta pantalla se informa el nick ó alias con el que el usuario desea darse a conocer en las conversaciones en las que participe. Se informa también el número de puerto que se usará para que el módulo servidor de la aplicación escuche las peticiones externas.

El primer ChekBox, **Activar Servidor**, se utiliza para activar o desactivar el módulo servidor de la aplicación, por defecto no está activo con lo que si se desea que otros usuarios soliciten conexiones se debe de activar este flag.

El segundo CheckBox, **Rastrear Comunicación**, se utiliza para generar un rastreo de la información que se intercambian dos usuarios desde que se establece el intercambio de claves. Este rastreo genera dos archivos llamados **Rastreo_Chat_Servidor.txt** y **Rastreo_Chat_Cliente.txt**, en el primero se guarda el rastreo de la información que se intercambia cuando la aplicación actúa como servidor de conversación y el segundo cuando actúa como cliente.

Una vez realizada la conexión se incorpora una nueva ventana en el escritorio principal de la aplicación para permitir la conversación, se presenta la siguiente pantalla:




Como se puede ver la ventana que aparece dispone de dos cuadros de texto, el superior se utiliza para ir grabando toda la conversación que se va produciendo y el inferior para escribir los mensajes que se van a enviar.

Podemos observar también que el título de la ventana de conversación está formado por un el literal "Servidor" más una dirección IP. El literal es el Nick ó alias del usuario con el que se ha establecido la comunicación, es el valor que ha informado en las opciones de configuración. La dirección IP es la dirección del host remoto, la dirección que se ha seleccionado desde la opción de menú **Conectar** para establecer una conversación.

Las siguientes conversaciones que se vayan estableciendo se irán distribuyendo por el escritorio de la misma manera que la que aparece en el ejemplo.

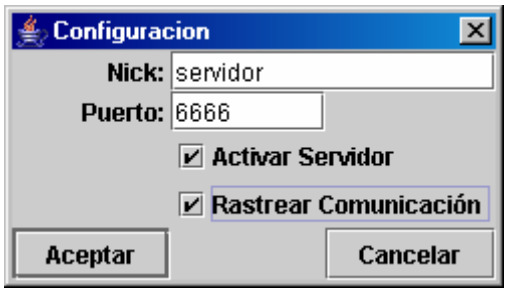
Para cerrar una conversación determinada se debe de usar el aspa que posee cada ventana.

# 5. Juegos de pruebas

Para realizar las pruebas de conexión e intercambio de mensajes se ha partido de la conexión entre dos ordenadores, uno de ellos, identificado como "cliente" será quién realizará la conexión al otro ordenador, identificado como "servidor" y que como su identificación indica estará configurado para actuar como servidor de conversación. En ambos ordenadores activaremos la opción de rastreo habilitada en la aplicación para obtener una salida a fichero del intercambio de claves y mensajes.

Comenzamos configurando el ordenador "servidor" para habilitarlo como servidor de conversación. Para ello accedemos a la opción de Configuración del menú principal de la aplicación e informamos como nick el nombre "servidor" y activamos las dos opciones,

"Activar Servidor" y "Rastrear Comunicación", como puerto de comunicaciones aceptamos el que se nos propone por defecto. La pantalla de configuración queda de la siguiente manera:




Una vez aceptamos los valores y se vuelve a la pantalla principal, la primera comprobación que se realiza es que la aplicación esté actuando como servidor tal y como se le ha indicado en la pantalla de configuración, es decir, esté a la escucha por el puerto seleccionado – en este caso el puerto 6666 – de la petición o peticiones de los ordenadores que actúen como clientes para entablar conversación. Para ello accedemos a una sesión MS-DOS a partir de la opción Ejecutar del menú de Inicio (Sistema operativo Windows) y tecleamos el mandato "netstat –a" para consultar el estado de los puertos. Obtenemos el siguiente resultado:

```
C:\>netstat -a

Conexiones activas

  Proto  Dirección local        Dirección remota       Estado
  TCP    portatil:epmap         portatil:0             LISTENING
  TCP    portatil:microsoft-ds  portatil:0             LISTENING
  TCP    portatil:1044          portatil:0             LISTENING
  TCP    portatil:3389          portatil:0             LISTENING
  TCP    portatil:6666          portatil:0             LISTENING
  TCP    portatil:1027          portatil:0             LISTENING
  TCP    portatil:4664          portatil:0             LISTENING
  TCP    portatil:netbios-ssn   portatil:0             LISTENING
  TCP    portatil:1048          216.127.33.119:http    CLOSE_WAIT
  TCP    portatil:1049          baym-cs335.msgr.hotmail.com:1863  ESTABLISHED
  UDP    portatil:microsoft-ds  *:*
  UDP    portatil:isakmp        *:*
  UDP    portatil:1025          *:*
  UDP    portatil:1045          *:*
  UDP    portatil:4500          *:*
  UDP    portatil:ntp           *:*
  UDP    portatil:1051          *:*
  UDP    portatil:1056          *:*
  UDP    portatil:1900          *:*
  UDP    portatil:ntp           *:*
  UDP    portatil:netbios-ns    *:*
  UDP    portatil:netbios-dgm   *:*
  UDP    portatil:1900          *:*
```

Se ha marcado en negrita la línea que indica que el ordenador está a la escucha por el puerto que se ha establecido como servidor de conversación.

A partir de este momento el ordenador "servidor" ya está disponible para recibir peticiones de conversación, por lo tanto, el siguiente paso es establecer una conexión desde el ordenador "cliente". Para ello, accedemos a la pantalla de configuración e informamos como nick el nombre "cliente" y activamos la casilla de "Rastrear comunicación".
Una vez que aceptamos los valores se empieza a realizar el intercambio de claves y mensajes, al tener habilitado en ambas sesiones la opción de rastreo de comunicación obtenemos dos archivos llamados "Rastreo_Chat_Servidor.txt" y "Rastreo_Chat_Cliente.txt" con el detalle de los mensajes intercambiados, claves Diffie Hellman, secreto compartido, clave DES y saludo inicial del cliente. En el primer archivo se guarda los mensajes de la aplicación actuando como

servidor generándose en el ordenador "servidor" y en el segundo los mensajes de la aplicación actuando como cliente generándose en el ordenador "cliente".

En esta sección se presentan cronológicamente los mensajes intercambiados entre ambos ordenadores. Para facilitar la lectura y el análisis de los mensajes, antes de realizar la grabación en el archivo de salida se realiza una traducción a códigos hexadecimales.

Envío de cliente a servidor, recepción del servidor

```
Envío clave pública a servidor:
30:81:DF:30:81:97:06:09:2A:86:48:86:F7:0D:01:03:01:30:81:89:02:41:00:A0:67:F8:B0:D2:
8C:04:5A:AC:78:01:12:1B:84:18:CF:11:E3:D7:34:33:9E:CD:01:82:8D:F5:BC:3D:B7:26:92:3B:
77:50:0F:0B:9A:CB:58:5B:B7:30:53:97:76:D4:80:73:1A:F8:49:9D:44:CB:CB:80:AB:EE:84:9E:
E0:A2:5D:02:40:27:C1:0D:1E:AE:E7:B2:E1:A3:FC:6C:F2:87:CB:16:DD:BB:72:5B:63:C1:96:D4:
BC:AB:D7:19:21:38:C6:55:65:87:12:D9:D0:32:BB:11:B8:8E:29:41:32:22:85:6E:41:2C:70:E5:
FC:42:9B:14:42:21:76:BE:97:9E:88:3D:E6:02:02:01:FF:03:43:00:02:40:2E:6A:F8:CC:A5:5D:
90:5F:8D:75:9C:BA:1E:E8:74:D9:0F:98:E6:BC:BB:FB:7F:07:D2:17:D8:B0:C2:5D:DA:58:4D:46:
EA:0F:23:77:88:96:6B:C6:08:3D:FD:2F:C2:42:E9:48:93:25:72:67:B3:2E:16:54:33:F4:78:D0:
75:0B

Recepción Clave Pública del cliente
30:81:DF:30:81:97:06:09:2A:86:48:86:F7:0D:01:03:01:30:81:89:02:41:00:A0:67:F8:B0:D2:
8C:04:5A:AC:78:01:12:1B:84:18:CF:11:E3:D7:34:33:9E:CD:01:82:8D:F5:BC:3D:B7:26:92:3B:
77:50:0F:0B:9A:CB:58:5B:B7:30:53:97:76:D4:80:73:1A:F8:49:9D:44:CB:CB:80:AB:EE:84:9E:
E0:A2:5D:02:40:27:C1:0D:1E:AE:E7:B2:E1:A3:FC:6C:F2:87:CB:16:DD:BB:72:5B:63:C1:96:D4:
BC:AB:D7:19:21:38:C6:55:65:87:12:D9:D0:32:BB:11:B8:8E:29:41:32:22:85:6E:41:2C:70:E5:
FC:42:9B:14:42:21:76:BE:97:9E:88:3D:E6:02:02:01:FF:03:43:00:02:40:2E:6A:F8:CC:A5:5D:
90:5F:8D:75:9C:BA:1E:E8:74:D9:0F:98:E6:BC:BB:FB:7F:07:D2:17:D8:B0:C2:5D:DA:58:4D:46:
EA:0F:23:77:88:96:6B:C6:08:3D:FD:2F:C2:42:E9:48:93:25:72:67:B3:2E:16:54:33:F4:78:D0:
75:0B
```

Envío de servidor a cliente, recepción del cliente

```
Envío Clave Pública a cliente
30:81:E0:30:81:97:06:09:2A:86:48:86:F7:0D:01:03:01:30:81:89:02:41:00:A0:67:F8:B0:D2:
8C:04:5A:AC:78:01:12:1B:84:18:CF:11:E3:D7:34:33:9E:CD:01:82:8D:F5:BC:3D:B7:26:92:3B:
77:50:0F:0B:9A:CB:58:5B:B7:30:53:97:76:D4:80:73:1A:F8:49:9D:44:CB:CB:80:AB:EE:84:9E:
E0:A2:5D:02:40:27:C1:0D:1E:AE:E7:B2:E1:A3:FC:6C:F2:87:CB:16:DD:BB:72:5B:63:C1:96:D4:
BC:AB:D7:19:21:38:C6:55:65:87:12:D9:D0:32:BB:11:B8:8E:29:41:32:22:85:6E:41:2C:70:E5:
FC:42:9B:14:42:21:76:BE:97:9E:88:3D:E6:02:02:01:FF:03:44:00:02:41:00:82:7E:D9:66:CC:
26:10:5D:F1:50:86:E9:71:A0:14:ED:FE:98:DE:9E:7F:2C:73:48:6D:AB:8A:FD:87:96:C4:B9:3A:
51:0C:92:A5:06:45:E9:CD:EA:40:79:F5:28:BF:44:9F:49:16:1B:CF:D7:E8:1D:5B:DB:14:1E:09:
FA:13:27

Recepción Clave pública del servidor:
30:81:E0:30:81:97:06:09:2A:86:48:86:F7:0D:01:03:01:30:81:89:02:41:00:A0:67:F8:B0:D2:
8C:04:5A:AC:78:01:12:1B:84:18:CF:11:E3:D7:34:33:9E:CD:01:82:8D:F5:BC:3D:B7:26:92:3B:
77:50:0F:0B:9A:CB:58:5B:B7:30:53:97:76:D4:80:73:1A:F8:49:9D:44:CB:CB:80:AB:EE:84:9E:
E0:A2:5D:02:40:27:C1:0D:1E:AE:E7:B2:E1:A3:FC:6C:F2:87:CB:16:DD:BB:72:5B:63:C1:96:D4:
BC:AB:D7:19:21:38:C6:55:65:87:12:D9:D0:32:BB:11:B8:8E:29:41:32:22:85:6E:41:2C:70:E5:
FC:42:9B:14:42:21:76:BE:97:9E:88:3D:E6:02:02:01:FF:03:44:00:02:41:00:82:7E:D9:66:CC:
26:10:5D:F1:50:86:E9:71:A0:14:ED:FE:98:DE:9E:7F:2C:73:48:6D:AB:8A:FD:87:96:C4:B9:3A:
51:0C:92:A5:06:45:E9:CD:EA:40:79:F5:28:BF:44:9F:49:16:1B:CF:D7:E8:1D:5B:DB:14:1E:09:
FA:13:27
```

Secreto compartido. En ambas ordenadores debe ser idéntico.

```
Secreto Compartido (Servidor):
5A:3B:37:F2:66:12:B8:C9:B8:EC:A1:A0:65:AC:77:13:7D:5C:BD:24:B9:3A:EB:D5:9E:44:72:9E:
D8:DB:C3:32:41:6E:A2:D7:43:89:F2:D6:39:D4:C5:EC:96:25:C3:05:43:67:52:D2:6C:D1:2A:8F:
61:3D:20:D5:8B:46:2E:48
Secreto Compartido (Cliente):
5A:3B:37:F2:66:12:B8:C9:B8:EC:A1:A0:65:AC:77:13:7D:5C:BD:24:B9:3A:EB:D5:9E:44:72:9E:
D8:DB:C3:32:41:6E:A2:D7:43:89:F2:D6:39:D4:C5:EC:96:25:C3:05:43:67:52:D2:6C:D1:2A:8F:
61:3D:20:D5:8B:46:2E:48
```

Clave de encriptación DES.

```
Clave DES (Servidor):
5B:3B:37:F2:67:13:B9:C8
Clave DES (Cliente):
5B:3B:37:F2:67:13:B9:C8
```

Intercambio de saludos

```
Saludo a Cliente (Plaintext):
40:6E:69:63:6B:73:65:72:76:69:64:6F:72
Saludo a Cliente (Cifrado):
49:62:AD:71:D7:9E:EA:D5:0E:01:D5:F9:49:EF:27:F0

Mensaje recibido (Cifrado): (Cliente)
49:62:AD:71:D7:9E:EA:D5:0E:01:D5:F9:49:EF:27:F0
Mensaje recibido (Plaintext): (Cliente)
@nickservidor

Saludo a Servidor (Plaintext):
40:6E:69:63:6B:63:6C:69:65:6E:74:65
Saludo a Servidor (Cifrado):
57:90:F8:C7:73:55:BF:2E:67:D1:05:AA:22:3B:01:14

Mensaje recibido (Cifrado): (Servidor)
57:90:F8:C7:73:55:BF:2E:67:D1:05:AA:22:3B:01:14
Mensaje recibido (Plaintext): (Servidor)
@nickcliente
```

Intercambio de mensaje de conversación de cliente a servidor

```
Mensaje enviado (Plaintext):
48:6F:6C:61:20:53:65:72:76:69:64:6F:72
Mensaje enviado (Cifrado):
05:79:62:57:BF:41:21:9C:0E:01:D5:F9:49:EF:27:F0

Mensaje recibido (Cifrado):
05:79:62:57:BF:41:21:9C:0E:01:D5:F9:49:EF:27:F0
Mensaje recibido (Plaintext):
Hola Servidor
```

Intercambio de mensaje de conversación de servidor a cliente

```
Mensaje enviado (Plaintext):
48:6F:6C:61:20:43:6C:69:65:6E:74:65
Mensaje enviado (Cifrado):
67:D6:7E:1F:2C:9F:B2:F9:67:D1:05:AA:22:3B:01:14

Mensaje recibido (Cifrado):
67:D6:7E:1F:2C:9F:B2:F9:67:D1:05:AA:22:3B:01:14
Mensaje recibido (Plaintext):
Hola Cliente
```

Indicador de final de conversación

Al finalizar una conversación cualquiera de las dos partes implicadas en ésta, se envía el mensaje especial @FIN para indicar el final de la conversación y de esta forma poder cerrar el canal de comunicación abierto (socket) y liberar el recurso ocupado en la máquina. En el ejemplo, el cliente avisa al servidor que termina la conversación

```
Mensaje recibido (Cifrado):
87:ED:B7:66:45:C5:FA:43
Mensaje recibido (Plaintext):
@FIN
```

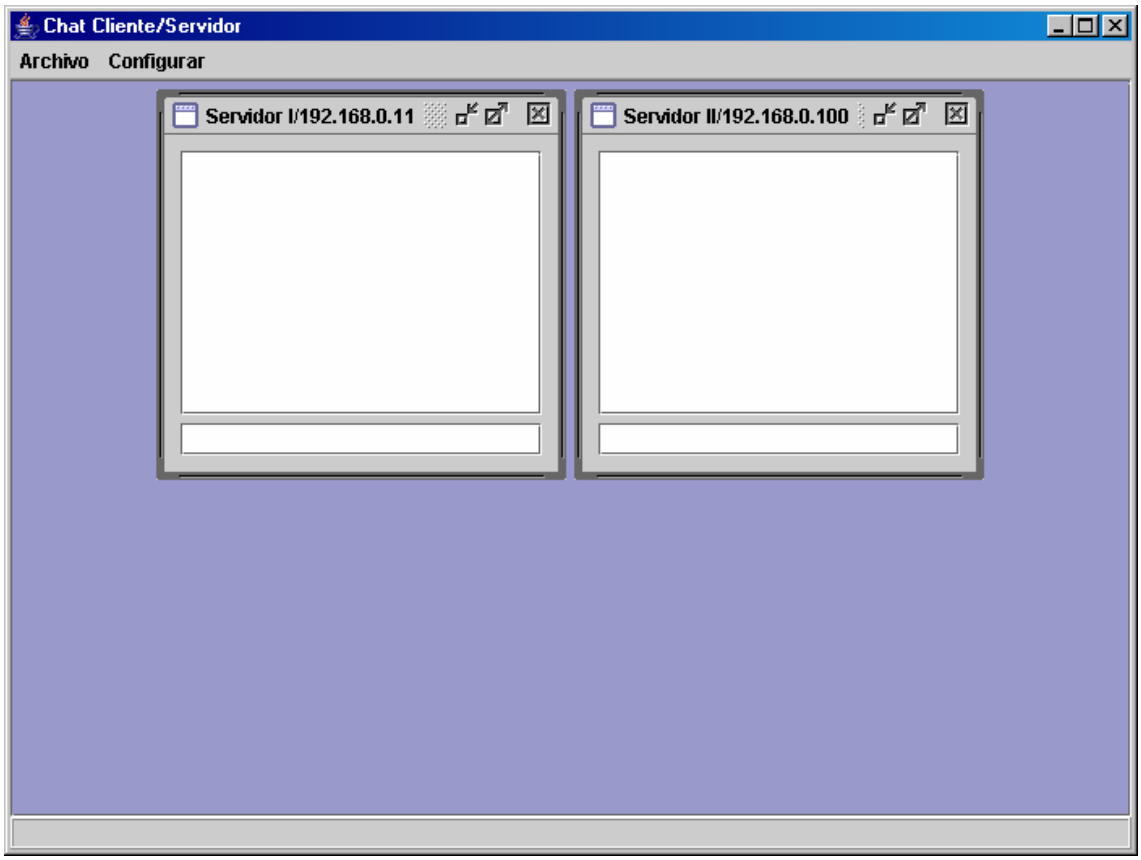
Una vez probada la conversación, el siguiente paso es cerrar la aplicación y asegurarse que los recursos de la máquina no quedan ocupados, es decir, el puerto ocupado queda liberado. Se realiza la misma prueba que se usó para comprobar que el servidor estaba activo. El resultado obtenido es el siguiente:

```
C:\>netstat -a

Conexiones activas

  Proto  Dirección local        Dirección remota       Estado
  TCP    portatil:epmap         portatil:0             LISTENING
  TCP    portatil:microsoft-ds  portatil:0             LISTENING
  TCP    portatil:1044          portatil:0             LISTENING
  TCP    portatil:3389          portatil:0             LISTENING
  TCP    portatil:1027          portatil:0             LISTENING
  TCP    portatil:4664          portatil:0             LISTENING
  TCP    portatil:netbios-ssn   portatil:0             LISTENING
  TCP    portatil:1048          216.127.33.119:http    CLOSE_WAIT
```

```
  TCP    portatil:1098          uvox1-nye-slot3l.stream.aol.com:http  ESTABLISHED
  TCP    portatil:1099          64.92.160.134:http     CLOSE_WAIT
  UDP    portatil:microsoft-ds  *:*
  UDP    portatil:isakmp        *:*
  UDP    portatil:1025          *:*
  UDP    portatil:1045          *:*
  UDP    portatil:4500          *:*
  UDP    portatil:ntp           *:*
  UDP    portatil:1051          *:*
  UDP    portatil:1056          *:*
  UDP    portatil:1900          *:*
  UDP    portatil:ntp           *:*
  UDP    portatil:netbios-ns    *:*
  UDP    portatil:netbios-dgm   *:*
  UDP    portatil:1900          *:*
```

Se puede observar que la asignación del puerto ha desaparecido.

La siguiente prueba que se realiza es la de entablar dos conversaciones con dos servidores diferentes. La pantalla principal del cliente toma este aspecto:




Como se puede ver se realiza la conexión con el servidor "Servidor I" (192.168.0.11) y con el servidor "Servidor II" (192.168.0.100).

El siguiente paso que comprobamos una vez establecida ambas conexiones es comprobar el estado de los puertos de comunicaciones. Para ello actuamos de la misma manera que anteriormente, es decir, utilizamos el mandato "netstat –a". Obtenemos el siguiente resultado:

```
C:\>netstat -a

Conexiones activas

  Proto  Dirección local        Dirección remota       Estado
  TCP    portatil:epmap         portatil:0             LISTENING
  TCP    portatil:microsoft-ds  portatil:0             LISTENING
```

```
TCP    portatil:1222          portatil:0             LISTENING
TCP    portatil:3389          portatil:0             LISTENING
TCP    portatil:1027          portatil:0             LISTENING
TCP    portatil:1266          localhost:1267         TIME_WAIT
TCP    portatil:4664          portatil:0             LISTENING
TCP    portatil:netbios-ssn   portatil:0             LISTENING
TCP    portatil:1048          216.127.33.119:http    CLOSE_WAIT
TCP    portatil:1098          uvox1-nye-slot3l.stream.aol.com:http  ESTABLISHED
TCP    portatil:1099          64.92.160.134:http     CLOSE_WAIT
TCP    portatil:1141          64.92.160.134:http     CLOSE_WAIT
TCP    portatil:1252          64.92.160.134:http     CLOSE_WAIT
TCP    portatil:1263          192.168.0.11:6666      ESTABLISHED
TCP    portatil:1264          192.168.0.100:6667     ESTABLISHED
TCP    portatil:1265          smtp.terra.es:pop3     TIME_WAIT
UDP    portatil:microsoft-ds  *:*
UDP    portatil:isakmp        *:*
UDP    portatil:1025          *:*
UDP    portatil:1045          *:*
UDP    portatil:4500          *:*
UDP    portatil:ntp           *:*
UDP    portatil:1051          *:*
UDP    portatil:1056          *:*
UDP    portatil:1900          *:*
UDP    portatil:ntp           *:*
UDP    portatil:netbios-ns    *:*
UDP    portatil:netbios-dgm   *:*
UDP    portatil:1900          *:*
```

Se han resaltado las dos líneas correspondientes a las conexiones como cliente a los dos servidores, también se puede ver que un servidor espera peticiones sobre el puerto 6666 y el otro sobre el puerto 6667.

Uno de los objetivos de este trabajo es que cada conversación utilizara sus propias claves. Se ha realizado el rastreo de ambas conexiones para observar como la información intercambiada entre el cliente y ambos servidores es diferente. Se ha incorporado la dirección IP del servidor al lado de cada literal descriptivo del tipo de información para tener una visión más clara de que conexión se trata. El archivo de salida "Rastreo_Chat_Cliente.txt" da el siguiente resultado:

Intercambio con servidor "Servidor I" (192.168.0.11)

```
Envío clave pública a servidor: /192.168.0.11
30:81:DF:30:81:97:06:09:2A:86:48:86:F7:0D:01:03:01:30:81:89:02:41:00:B4:66:88:A2:6B:
0A:A8:AC:16:BE:EE:83:C0:A8:D9:8B:FC:56:0B:7A:19:A1:30:1E:51:10:54:0D:6F:5E:49:A2:53:
08:95:C6:9F:65:39:22:A8:B3:0A:37:23:48:32:C9:80:C6:5B:12:4E:2B:FC:60:09:38:A9:5C:2A:
72:9B:3B:02:40:7A:3B:CB:B0:A4:40:81:12:DC:6E:ED:C8:2B:9C:8F:F7:34:93:DE:35:64:37:0D:
50:9C:FE:8D:C1:A1:FF:B2:1D:C6:2C:7E:E2:ED:37:43:F7:C7:80:BD:5A:4E:3D:84:38:BF:21:98:
50:C2:35:CB:E3:2B:AB:FC:95:30:BB:CC:60:02:02:01:FF:03:43:00:02:40:53:2B:28:AF:D0:84:
51:3E:EC:11:BB:B4:61:DA:41:DF:A9:04:74:06:69:B8:EE:43:C0:C2:07:D1:A6:CB:39:D7:19:28:
EC:34:C0:B1:CB:83:29:07:37:A5:F3:6F:9E:88:C6:6B:09:4A:CE:0A:2D:F4:C8:C5:14:09:7C:8C:
A6:1F

Recepción Clave pública del +servidor: /192.168.0.11
30:81:DF:30:81:97:06:09:2A:86:48:86:F7:0D:01:03:01:30:81:89:02:41:00:B4:66:88:A2:6B:
0A:A8:AC:16:BE:EE:83:C0:A8:D9:8B:FC:56:0B:7A:19:A1:30:1E:51:10:54:0D:6F:5E:49:A2:53:
08:95:C6:9F:65:39:22:A8:B3:0A:37:23:48:32:C9:80:C6:5B:12:4E:2B:FC:60:09:38:A9:5C:2A:
72:9B:3B:02:40:7A:3B:CB:B0:A4:40:81:12:DC:6E:ED:C8:2B:9C:8F:F7:34:93:DE:35:64:37:0D:
50:9C:FE:8D:C1:A1:FF:B2:1D:C6:2C:7E:E2:ED:37:43:F7:C7:80:BD:5A:4E:3D:84:38:BF:21:98:
50:C2:35:CB:E3:2B:AB:FC:95:30:BB:CC:60:02:02:01:FF:03:43:00:02:40:42:D8:29:4B:BB:84:
93:E1:C7:53:F5:75:77:3D:38:08:D2:BB:E6:D6:A2:5F:EC:E0:B2:E6:4E:62:F5:39:EA:9F:81:92:
A2:69:C7:4E:CD:41:DA:A2:1E:6C:9D:EF:89:F4:18:F3:F9:E1:85:D4:0F:6D:1C:32:35:6A:5A:1F:
DD:A8

Secreto Compartido: /192.168.0.11
AC:28:C2:1F:B2:70:2B:2E:C2:12:66:33:16:48:95:CD:FD:96:9F:D0:C6:AA:86:30:CE:92:19:85:
CE:8E:85:B2:F2:E8:E1:50:DA:53:87:7F:25:60:54:8E:60:B4:5B:0F:D5:41:C0:BA:65:1E:D6:DB:
3C:C1:D5:58:B9:A6:9D:AB

Clave DES: /192.168.0.11
AD:29:C2:1F:B3:70:2A:2F

Saludo a Servidor (Plaintext): /192.168.0.11
```

```
40:6E:69:63:6B:63:6C:69:65:6E:74:65

Saludo a Servidor (Cifrado): /192.168.0.11
B3:A2:DE:78:57:27:FA:5D:EA:C3:13:76:85:77:8E:6A

Mensaje recibido (Cifrado): /192.168.0.11
33:79:2C:2A:C4:4C:6D:0E:E0:5A:5D:92:D3:72:69:8F

Mensaje recibido (Plaintext): /192.168.0.11
@nickServidor I
```

Intercambio con servidor "Servidor II" (192.168.0.100)

```
Envío clave pública a servidor: /192.168.0.100
30:81:DF:30:81:97:06:09:2A:86:48:86:F7:0D:01:03:01:30:81:89:02:41:00:B4:66:88:A2:6B:
0A:A8:AC:16:BE:EE:83:C0:A8:D9:8B:FC:56:0B:7A:19:A1:30:1E:51:10:54:0D:6F:5E:49:A2:53:
08:95:C6:9F:65:39:22:A8:B3:0A:37:23:48:32:C9:80:C6:5B:12:4E:2B:FC:60:09:38:A9:5C:2A:
72:9B:3B:02:40:7A:3B:CB:B0:A4:40:81:12:DC:6E:ED:C8:2B:9C:8F:F7:34:93:DE:35:64:37:0D:
50:9C:FE:8D:C1:A1:FF:B2:1D:C6:2C:7E:E2:ED:37:43:F7:C7:80:BD:5A:4E:3D:84:38:BF:21:98:
50:C2:35:CB:E3:2B:AB:FC:95:30:BB:CC:60:02:02:01:FF:03:43:00:02:40:1A:28:AD:BC:CD:D6:
F8:C5:10:FF:98:63:95:50:EE:0C:03:9F:4C:1E:F4:D7:B2:8E:DB:2C:E7:2D:D9:A1:D5:50:C1:8E:
F0:FE:27:12:F1:B4:45:C7:02:B8:19:48:2B:4B:4E:A2:7F:70:09:59:B2:86:B6:6C:46:C7:A7:21:
A4:D3

Recepción Clave pública del +servidor: /192.168.0.100
30:81:DF:30:81:97:06:09:2A:86:48:86:F7:0D:01:03:01:30:81:89:02:41:00:B4:66:88:A2:6B:
0A:A8:AC:16:BE:EE:83:C0:A8:D9:8B:FC:56:0B:7A:19:A1:30:1E:51:10:54:0D:6F:5E:49:A2:53:
08:95:C6:9F:65:39:22:A8:B3:0A:37:23:48:32:C9:80:C6:5B:12:4E:2B:FC:60:09:38:A9:5C:2A:
72:9B:3B:02:40:7A:3B:CB:B0:A4:40:81:12:DC:6E:ED:C8:2B:9C:8F:F7:34:93:DE:35:64:37:0D:
50:9C:FE:8D:C1:A1:FF:B2:1D:C6:2C:7E:E2:ED:37:43:F7:C7:80:BD:5A:4E:3D:84:38:BF:21:98:
50:C2:35:CB:E3:2B:AB:FC:95:30:BB:CC:60:02:02:01:FF:03:43:00:02:40:1E:F2:72:07:47:AD:
6B:7B:EE:50:62:75:60:7A:19:04:80:D8:E1:C5:85:29:22:33:83:52:70:3F:8F:77:7A:2B:50:79:
D1:23:B7:41:33:A9:68:7D:46:AC:AD:F5:44:9C:9E:13:A8:89:0A:C8:B9:D6:04:73:17:80:24:BC:
A0:BE

Secreto Compartido: /192.168.0.100
1A:F5:B4:62:B5:72:22:3F:2B:0A:C0:A1:5B:34:D3:B2:C3:85:B6:25:C7:DC:BD:45:6C:94:5A:5E:
20:AD:8B:22:A1:E9:56:51:9C:88:67:9A:B1:9B:BF:62:1A:13:14:9E:B9:08:BA:5B:72:18:4C:67:
A5:73:8F:28:34:64:02:0B

Clave DES: /192.168.0.100
1A:F4:B5:62:B5:73:23:3E

Saludo a Servidor (Plaintext): /192.168.0.100
40:6E:69:63:6B:63:6C:69:65:6E:74:65

Saludo a Servidor (Cifrado): /192.168.0.100
37:48:1D:96:DE:20:2F:81:65:36:F4:34:54:A0:95:F9

Mensaje recibido (Cifrado): /192.168.0.100
36:6E:79:50:C3:AD:AC:E2:48:FF:7C:5A:85:B1:1E:AD:BF:CB:37:5F:59:17:E7:4B

Mensaje recibido (Plaintext): /192.168.0.100
@nickServidor II
```

El siguiente paso a consistido en cerrar la aplicación del ordenador cliente y observar como quedan los puertos del ordenador. En un primer término las conexiones con los servidores han quedado en el estado TIME_WAIT

```
TCP    portatil:1263          192.168.0.11:6666      TIME_WAIT
TCP    portatil:1264          192.168.0.100:6667     TIME_WAIT
```

Una vez transcurrido el tiempo establecido por el sistema operativo ambas líneas han desaparecido, lo que indica que los recursos se han liberado totalmente.

# 6. Recursos

- Bibliografía
  - Josep Domingo Ferrer, "Criptografía", UOC, 1999
  - Bruce Eckel, "Thinking in Java", Prentice Hall, 2002

- Cay Horstmann, Gary Cornell, “Java 2”, Prentice Hall, 2004
- David Geary, “Graphic Java 2 V.II”, Sun Microsystems, 1999

- Varios
  - www.java.sun.com

# 7. Anexo

## 7.1 Código fuente

### 7.1.1 Conversacion.java

Conversacion.java

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.event.*;
import java.net.*;
import java.io.*;
import java.security.*;
import java.security.spec.*;
import java.security.interfaces.*;
import javax.crypto.*;
import javax.crypto.spec.*;
import javax.crypto.interfaces.*;
import com.sun.crypto.provider.SunJCE;

/**
 * Clase que implementa cada una de las ventanas de conversación, tanto si el
 * programa actúa como servidor como si actúa como cliente. Además de actúar
 * como interfase gráfico para el usuario, también gestiona todo lo relativo a
 * la seguridad de la aplicación, es decir, realiza el intercambio de claves con
 * el otro host de la conversación y el cifrado y descifrado de los mensajes que
 * se van intercambiando. Esta clase extiende de la clase <code>JFrame<code> e
 * implementa el interface <code>InternalFrameListener<code> lo cual quiere
 * decir que cada una de las ventanas creadas a partir de esta clase forman
 * parte del conjunto de ventanas de tipo <code>JInernalFrame<code> que forman
 * el panel principal de la aplicción que es del tipo JDeskTopPane
 */
public class Conversacion extends JFrame implements InternalFrameListener {

    private TFCFrame tfcFrame = null;
    private JTextField entradaTexto;
    private JTextArea areaTexto;
    private JInternalFrame ventanaConversacion;
    private HiloServidor hiloServidor = null;
    private HiloCliente hiloCliente = null;
    private Socket socket = null;
    private DataOutputStream out = null;
    private DataInputStream in = null;
    private PublicKey conversacionPublicKey = null;
    private DHParameterSpec dhParamSpec = null;
    private KeyPair conversacionKeyPair = null;
    private KeyAgreement conversacionKeyAgree = null;
    private byte[] conversacionSharedSecret;
    static final String CLIENTE = "cliente";
    static final String SERVIDOR = "servidor";
    private SecretKey conversacionDesKey = null;
    private Cipher conversacionCipher = null;
    private FileOutputStream salidaRastreo = null;
    private  static final String INTRO = "\n";

    /**
     * Constructor cuando se ha solicitado desde el exterior una nueva
     * conversación al host que está actuando como servidor.
     * @param HiloServidor -> Hilo de proceso servidor.
```

```
     */
    public Conversacion(HiloServidor hiloServidor) throws Exception{

        this.tfcFrame = hiloServidor.getTFCFrame();
        this.hiloServidor = hiloServidor;
        this.socket = hiloServidor.getSocket();

        // Archivo de salida para guardar el rastreo de programa
        salidaRastreo = new FileOutputStream("Rastreo_Chat_Servidor.txt",true);

        // Flujos de entrada/salida para realizar comunicaciones con el cliente
        out = new DataOutputStream(socket.getOutputStream());
        in = new DataInputStream(socket.getInputStream());

        // Recibir la clave pública del cliente
        byte[] keyBytes = new byte[in.readInt()];
        in.readFully(keyBytes);

        if (rastreoActivo()){
            salidaRastreo.write(new String(INTRO +
            "Recepción Clave Pública del " +
            "cliente " + socket.getInetAddress()
            + INTRO + toHexString(keyBytes) + INTRO).getBytes());
            salidaRastreo.flush();
        }
        // Con clave pública recibida el server instancia una clave pública DH
        KeyFactory servidorKeyFac = KeyFactory.getInstance("DH");
        X509EncodedKeySpec x509KeySpec = new X509EncodedKeySpec(keyBytes);
        conversacionPublicKey = servidorKeyFac.generatePublic(x509KeySpec);

        // Obtener los parámetros asociados a la clave pública del cliente ya
        // que se deben de usar los mismos parámetros para generar las key pair
        // del servidor
        dhParamSpec = ((DHPublicKey)conversacionPublicKey).getParams();

        // Crear y enviar valores del servidor al cliente conectado
        KeyPairGenerator servidorKpairGen = KeyPairGenerator.getInstance("DH");
        servidorKpairGen.initialize(dhParamSpec);
        conversacionKeyPair = servidorKpairGen.generateKeyPair();

        // Servidor crea e inicializa su objeto DH KeyAgreement
        conversacionKeyAgree = KeyAgreement.getInstance("DH");
        conversacionKeyAgree.init(conversacionKeyPair.getPrivate());

        // Servidor codifica su clave pública y la envía al cliente
        byte[] servidorPubKeyEnc = conversacionKeyPair.getPublic().getEncoded();

        // Envía la clave pública al cliente
        out.writeInt(servidorPubKeyEnc.length);
        out.write(servidorPubKeyEnc);

        if (rastreoActivo()){
            salidaRastreo.write(new String(INTRO +
            "Envío Clave Pública a cliente " +
            socket.getInetAddress() + INTRO +
            toHexString(servidorPubKeyEnc)+INTRO).getBytes());
            salidaRastreo.flush();
        }

        // Generar el valor secreto
        conversacionKeyAgree.init(conversacionKeyPair.getPrivate());
        conversacionKeyAgree.doPhase(conversacionPublicKey, true);

        if (rastreoActivo()){
            conversacionKeyAgree.doPhase(conversacionPublicKey, true);
            conversacionSharedSecret = conversacionKeyAgree.generateSecret();
            salidaRastreo.write(new String(INTRO + "Secreto Compartido: " +
            socket.getInetAddress() +
            INTRO + toHexString(conversacionSharedSecret) + INTRO).getBytes());
            salidaRastreo.flush();
        }

        // Generar clave secreta del servidor
        conversacionKeyAgree.doPhase(conversacionPublicKey, true);
        conversacionDesKey = conversacionKeyAgree.generateSecret("DES");

        if (rastreoActivo()){
```

```
            byte [] desKey = conversacionDesKey.getEncoded();
            salidaRastreo.write(new String(INTRO + "Clave DES: " +
            socket.getInetAddress() +
            INTRO + toHexString(desKey)+INTRO).getBytes());
            salidaRastreo.flush();
        }

        // Tipo de encriptación usado
        conversacionCipher = Cipher.getInstance("DES/ECB/PKCS5Padding");
        conversacionCipher.init(Cipher.ENCRYPT_MODE, conversacionDesKey);

        // Crear ventana de conversación
        crearVentana();

        // Añadir la nueva ventana al DeskTop principal
        tfcFrame.getTFCPanel().getJDeskTopPane().add(ventanaConversacion);

        // Enviamos como primer mensaje nuestro nick, para ello identificamos
        // la cadena inicial de texto como @nick y a continuación nuestro nick.
        DatosConfiguracion dc = tfcFrame.getDatosConfiguracion();

        byte [] textoClaro = new String("@nick"+ dc.getNick()).getBytes();
        byte [] textoCipher = conversacionCipher.doFinal(textoClaro);
        out.writeInt(textoCipher.length);
        out.write(textoCipher);

        if (rastreoActivo()){
            salidaRastreo.write(new String(INTRO +
            "Saludo a Cliente (Plaintext): " +
            socket.getInetAddress() +
            INTRO + toHexString(textoClaro) + INTRO).getBytes());
            salidaRastreo.flush();
            salidaRastreo.write(new String(INTRO +
            "Saludo a Cliente (Cifrado): " +
            socket.getInetAddress() +
            INTRO + toHexString(textoCipher) + INTRO).getBytes());
            salidaRastreo.flush();
        }

        // Comenzar la conversación
        conversar();
    }

    /**
     * Constructor cuando se solicita conversación con un host que actúa como
     * servidor
     * @param HiloCliente -> Hilo del proceso cliente.
     */
    public Conversacion(HiloCliente hiloCliente, DHParameterSpec parameterSpec)
    throws Exception {

        this.tfcFrame = hiloCliente.getTFCFrame();
        this.hiloCliente = hiloCliente;
        this.socket = hiloCliente.getSocket();


        salidaRastreo = new FileOutputStream("Rastreo_Chat_Cliente.txt", true);


        // Crear key pair Diffie Helmman
        KeyPairGenerator clienteKeyPairGen = KeyPairGenerator.getInstance("DH");
        clienteKeyPairGen.initialize(parameterSpec);
        conversacionKeyPair=clienteKeyPairGen.generateKeyPair();

        // Crear e inicializar el objeto DH KeyAgreement
        conversacionKeyAgree = KeyAgreement.getInstance("DH");
        conversacionKeyAgree.init(conversacionKeyPair.getPrivate());

        // Flujos de entrada salida
        in = new DataInputStream(socket.getInputStream());
        out = new DataOutputStream(socket.getOutputStream());

        // Codificación de la clave pública y envío al servidor
        byte [] clientePubKeyEnc = conversacionKeyPair.getPublic().getEncoded();
        out.writeInt(clientePubKeyEnc.length);
        out.write(clientePubKeyEnc);
```

```
if (rastreoActivo()){
    salidaRastreo.write(new String(INTRO +
    "Envío clave pública a servidor: " +
    socket.getInetAddress() +
    INTRO + toHexString(clientePubKeyEnc)+INTRO).getBytes());
    salidaRastreo.flush();
}

// Recepción de la clave pública del servidor
byte [] keyBytes = new byte[in.readInt()];
in.readFully(keyBytes);

if (rastreoActivo()){
    salidaRastreo.write(new String(INTRO +
    "Recepción Clave pública del +" +
    "servidor: " + socket.getInetAddress() +
    INTRO + toHexString(keyBytes)+INTRO).getBytes());
    salidaRastreo.flush();
}

KeyFactory clienteKeyFac = KeyFactory.getInstance("DH");
X509EncodedKeySpec x509Spec = new X509EncodedKeySpec(keyBytes);
conversacionPublicKey = clienteKeyFac.generatePublic(x509Spec);

// Generar el valor secreto
conversacionKeyAgree = KeyAgreement.getInstance("DH");
conversacionKeyAgree.init(conversacionKeyPair.getPrivate());

if (rastreoActivo()){
    conversacionKeyAgree.doPhase(conversacionPublicKey, true);
    conversacionSharedSecret = conversacionKeyAgree.generateSecret();
    salidaRastreo.write(new String(INTRO + "Secreto Compartido: " +
    socket.getInetAddress() +
    INTRO + toHexString(conversacionSharedSecret)+INTRO).getBytes());
    salidaRastreo.flush();
}

// Convertir secreto compartido en instancia DES
conversacionKeyAgree.doPhase(conversacionPublicKey, true);
conversacionDesKey = conversacionKeyAgree.generateSecret("DES");

if (rastreoActivo()){
    byte [] desKey = conversacionDesKey.getEncoded();
    salidaRastreo.write(new String(INTRO + "Clave DES: " +
    socket.getInetAddress() +
    INTRO + toHexString(desKey)+INTRO).getBytes());
    salidaRastreo.flush();
}

// Tipo de encriptación usado
conversacionCipher = Cipher.getInstance("DES/ECB/PKCS5Padding");
conversacionCipher.init(Cipher.ENCRYPT_MODE, conversacionDesKey);

// Crear ventana de conversación
crearVentana();

// Añadir la nueva ventana al DeskTop principal
tfcFrame.getTFCPanel().getJDeskTopPane().add(ventanaConversacion);

// Enviamos como primer mensaje nuestro nick, para ello identificamos
// la cadena inicial de texto como @nick y a continuación nuestro nick.
DatosConfiguracion dc = tfcFrame.getDatosConfiguracion();
byte [] textoClaro = new String("@nick"+ dc.getNick()).getBytes();
byte [] textoCipher = conversacionCipher.doFinal(textoClaro);
out.writeInt(textoCipher.length);
out.write(textoCipher);

if (rastreoActivo()){
    salidaRastreo.write(new String(INTRO +
    "Saludo a Servidor (Plaintext): " +
    socket.getInetAddress() +
    INTRO + toHexString(textoClaro) + INTRO).getBytes());
    salidaRastreo.flush();
    salidaRastreo.write(new String(INTRO +
    "Saludo a Servidor (Cifrado): " +
    socket.getInetAddress() +
    INTRO + toHexString(textoCipher) + INTRO).getBytes());
```

```
            salidaRastreo.flush();
        }

        // Comenzar la conversación
        conversar();
    }

    /**
     * Método para crear nueva ventana de convesación que se
     * incorporará a nuestro JDesktopPane
     */
    private void crearVentana(){

        entradaTexto = new JTextField();
        areaTexto = new JTextArea(10, 20);

        JScrollPane scrollPane = new JScrollPane(areaTexto);

        JPanel top = new JPanel();
        JPanel botton = new JPanel();

        areaTexto.setEditable(false);

        entradaTexto.addActionListener(new listenerEntradaTexto());

        ventanaConversacion = new JInternalFrame("Conversacion",    // título
        true,  // Cambiar tamaño
        true,  // Cerrar
        true,  // Maximizar
        true); // iconificar

        top.setLayout(new GridLayout(1,1));
        botton.setLayout(new GridLayout(1,1));

        JPanel contentPane = new JPanel();
        contentPane.setBorder(BorderFactory.createEmptyBorder(10,10,10,10));
        contentPane.setLayout(new BoxLayout(contentPane, BoxLayout.PAGE_AXIS));
        areaTexto.setAlignmentX(CENTER_ALIGNMENT);

        contentPane.add(top);

        top.add(scrollPane);
        contentPane.add(Box.createRigidArea(new Dimension(0, 5)));
        contentPane.add(botton);
        botton.add(entradaTexto);

        ventanaConversacion.setContentPane(contentPane);
        ventanaConversacion.setTitle(socket.getInetAddress().toString());
        ventanaConversacion.pack();
        ventanaConversacion.setVisible(true);
        ventanaConversacion.addInternalFrameListener(this);
        tfcFrame.setCursor(Cursor.getDefaultCursor());

    }

    /**
     * Listener usado para leer las cadenas de texto informadas por el
     * usuario y enviarlas al destino previo cifrado con la clave compartida
     * que al inicio de la convesación se ha intercambiado.
     */
    public class listenerEntradaTexto implements ActionListener {
        public void actionPerformed(ActionEvent event){
            Object source = event.getSource();
            if (source==entradaTexto){
                String mensaje = event.getActionCommand();
                entradaTexto.setText("");
                try {
                    // Encifer el mensaje
                    byte [] textoClaro = mensaje.getBytes();
                    byte [] textoCipher=conversacionCipher.doFinal(textoClaro);
                    out.writeInt(textoCipher.length);
                    out.write(textoCipher);
                    String s = new String(
                    tfcFrame.getDatosConfiguracion().getNick() +
                    ": " +
                    new String(textoClaro));
                    mostrarMensaje(s + INTRO);
```

```
                if (rastreoActivo()) {
                    salidaRastreo.write(new String(INTRO +
                    "Mensaje enviado (Plaintext): " +
                    socket.getInetAddress() +
                    INTRO + toHexString(textoClaro) + INTRO).getBytes());
                    salidaRastreo.flush();
                    salidaRastreo.write(new String(INTRO +
                    "Mensaje enviado (Cifrado): " +
                    socket.getInetAddress() +
                    INTRO + toHexString(textoCipher) + INTRO).getBytes());
                    salidaRastreo.flush();
                }
            }catch(Exception ex){
                JOptionPane.showMessageDialog(tfcFrame, ex.getMessage(),
                "Error", 0);
            }
        }
    }
}

/**
 * Método del interface <code>InternalFrameListener<code> utilizado para
 * avisar al destino de la conversación del final de la conversación.
 * Se envío la cadena "@FIN" y se cierra el socket de comunicaciones abierto
 */
public void internalFrameClosing(InternalFrameEvent e) {
    try{
        if (socket.isConnected()){
            // Avisar del final de la conversación
            byte [] textoClaro = new String("@FIN").getBytes();
            byte [] textoCipher = conversacionCipher.doFinal(textoClaro);
            out.writeInt(textoCipher.length);
            out.write(textoCipher);
            socket.close();
        }
    }catch (IOException error){
    }
    catch (Exception error){
        error.printStackTrace();
    }
}

/*
 * Resto de métodos del interface InternalFrameListener.
 * En este caso no es necesario realizar ninguna operación con ellos.
 */
public void internalFrameClosed(InternalFrameEvent e) {}
public void internalFrameOpened(InternalFrameEvent e) {}
public void internalFrameIconified(InternalFrameEvent e) {}
public void internalFrameDeiconified(InternalFrameEvent e) {}
public void internalFrameActivated(InternalFrameEvent e) {}
public void internalFrameDeactivated(InternalFrameEvent e) {}


/**
 * Método para convertir un byte a un dígito hexadecimal, el resultado se
 * escribe en el buffer suministrado.
 * @param byte b byte para convertir en dígito hexadecmal
 * @paramStringBuffer buffer suministrado para retornar los bytes
 * convertidos en dígitos hexadecimales.
 */
private void byte2hex(byte b, StringBuffer buf) {
    char[] hexChars = { "0", "1", "2", "3", "4", "5", "6", "7", "8",
    "9", "A", "B", "C", "D", "E", "F" };
    int high = ((b & 0xf0) >> 4);
    int low = (b & 0x0f);
    buf.append(hexChars[high]);
    buf.append(hexChars[low]);
}

/**
 * Método para covertir un array tipo byte a un String hexadecimal
 * @param byte[] block array tipo byte con la información a convertir a
 * cadena hexadecimal
 * @return String con los valores en hexadecimal del array tipo byte recibido
 * por parámetro.
 */
```

```
    private String toHexString(byte[] block) {
        StringBuffer buf = new StringBuffer();

        int len = block.length;

        for (int i = 0; i < len; i++) {
            byte2hex(block[i], buf);
            if (i < len-1) {
                buf.append(":");
            }
        }
        return buf.toString();
    }

    /**
     * Método para saber si está activo el rastreo de la conversación
     * @return boolean indicando si lo está ó no (On ó Off)
     */
    private boolean rastreoActivo() {
        return tfcFrame.getDatosConfiguracion().getRastrear();
    }

    /**
     * Método conversar para realizar la recepción de los mensajes del otro host
     * que interviene en la conversación. Mientras que no se solicite finalizar
     * la conversación este método está en constante ejecución.
     */
    private void conversar() {

        String marca=null;
        String nickRemoto = null;

        try{

            boolean desconectar=false;
            byte[] textoCipher=null;

            while(!desconectar){

                textoCipher = new byte[in.readInt()];
                in.readFully(textoCipher);

                // Desencriptado
                Cipher servidorCipher=Cipher.getInstance("DES/ECB/PKCS5Padding");
                servidorCipher.init(Cipher.DECRYPT_MODE, conversacionDesKey);
                byte[] textoRecuperado = servidorCipher.doFinal(textoCipher);
                String s = new String(textoRecuperado);
                if (s.length() >= 5) {
                    marca = new String(textoRecuperado, 0, 5);
                }else{
                    marca = "";
                }

                if (rastreoActivo()) {
                    salidaRastreo.write(new String(INTRO +
                    "Mensaje recibido (Cifrado): " +
                    socket.getInetAddress() +
                    INTRO + toHexString(textoCipher) + INTRO).getBytes());
                    salidaRastreo.flush();
                    salidaRastreo.write(new String(INTRO +
                    "Mensaje recibido (Plaintext): " +
                    socket.getInetAddress() +
                    INTRO + new String(textoRecuperado) + INTRO).getBytes());
                    salidaRastreo.flush();
                }

                if (s.equals("@FIN")){
                    mostrarMensaje(new String("Usuario abandona la conversación"
                    + INTRO));
                    desconectar = true;
                    try{
                        socket.close();
                    }catch(Exception exc){
                        JOptionPane.showMessageDialog(tfcFrame, exc.getMessage(),
                        "Error", 0);
                    }
                }else if (marca.equals("@nick")){
```

```
                    ventanaConversacion.setTitle(s.substring(5) +
                    socket.getInetAddress().toString());
                    nickRemoto = new String(s.substring(5));
                }
                else{
                    mostrarMensaje(new String(nickRemoto + ": " + s + INTRO));
                }
            }
        }catch(Exception e){
            try{
                socket.close();
            }catch(Exception exc){
                JOptionPane.showMessageDialog(tfcFrame, exc.getMessage(),
                "Error", 0);
            }
        }
    }

    /**
     * Método para añadir las líneas de mensajes en el cuadro de texto de la
     * ventana de historial de conversació.
     * Se controla el scroll automático para visualizar el último mensaje
     * escrito
     * @param texto -> Cadena de texto a añadir en la ventana de conversación
     */
    protected void mostrarMensaje(String texto) {
        areaTexto.append(texto);
        areaTexto.setCaretPosition(areaTexto.getDocument().getLength());
    }
}
```

### 7.1.2 HiloCliente.java

HiloCliente.java

```
import java.net.*;
import java.io.*;
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import com.sun.crypto.provider.SunJCE;
import java.security.*;
import javax.crypto.spec.*;
import java.util.*;

/**
 * Clase que genera una nueva conversación en un nuevo hilo de ejecuación
 * cuando la apliación actúa como cliente, siendo el destino de la conversación
 * el host que actuará como servidor.
 */
public class HiloCliente extends Thread {

    private TFCFrame tfcFrame = null;
    private Socket socket = null;
    private static int conexiones = 0;
    private static DHParameterSpec dhParameterSpec = null;
    private SunJCE jce = null;
    private TFC tfc = null;


    /**
     * Constructor de un nuevo hilo de conversación cuando el programa actúa
     * como cliente. Se solicita conversación a un host que actúa como servdor
     * el cual en vez de lanzar una ejecución como Cliente lanzará un nuevo
     * proceso como servidor.
     * @param TFCFrame ventana pricncipal de la aplicación TFC, este parámetro
     * se usa para comunicar el proceso principal con los diferentes hilos ó
     * cnversaciones activas.
     * @param Socket socket de comunicación entre este hilo de ejecución y el
     * host servidor con el que se ha establecido una conexión
     */
    public HiloCliente(TFCFrame tfcFrame, Socket socket) throws Exception{
        this.tfcFrame = tfcFrame;
        this.socket = socket;
        start();
    }

    /**
     * Método sobrecargado que se ejecuta al generarse un nuevo hilo de
     * ejecuación desde la llamada al método start() de la clase Thread.
     * Se genera un nuevo objeto de la clase Conversacion para realizar la
     * charla con el servidor.
     */
    public void run(){
        try{

            if (conexiones++ == 0){
                jce = new SunJCE();
                Security.addProvider(jce);
                AlgorithmParameterGenerator paramGen =
                               AlgorithmParameterGenerator.getInstance("DH");
                paramGen.init(512);
                AlgorithmParameters params = paramGen.generateParameters();
                dhParameterSpec =
                (DHParameterSpec)params.getParameterSpec(DHParameterSpec.class);
            }
            Conversacion conv = new Conversacion(this, dhParameterSpec);
        }catch(Exception e){
            try{
                JOptionPane.showMessageDialog(tfcFrame, e.getMessage(),
                "Error", 0);
                socket.close();
            }catch(Exception exc){
                JOptionPane.showMessageDialog(tfcFrame, exc.getMessage(),
                "Error", 0);
```

```
            }
        }
    }

    /**
     * Método que retorna la variable objeto de tipo <code>TFCFrame<code>
     * @return varaible objeto de tipo TFCFrame que representa el marco
     * principal de la aplicación.
     */
    public TFCFrame getTFCFrame(){
        return tfcFrame;
    }

    /**
     * Método que retorna la variable objeto de tipo <code>Socket<code>
     * @return varaible objeto de tipo Socket que representa el socket de
     * comunicación entre dos hosts.
     */
    public Socket getSocket(){
        return socket;
    }
}
```

### 7.1.3 HiloServidor.java

HiloServidor.java

```
import java.net.*;
import java.io.*;
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

/**
 * Clase que representa una nueva conversación, la cual se ejecuta en un hilo
 * de proceso generado por la aplicación principal desde el módulo que actúa
 * como servidor. En este caso un cliente externo solicita una conexión y el
 * proceso local que está a la escuha actuando como servidor genera esta nueva
 * conversación
 */
public class HiloServidor extends Thread {

    private TFCFrame tfcFrame = null;
    private Socket socket = null;

    /**
     * Constructor de un nuevo hilo de conversación cuando el programa actúa
     * como servidor. Las peticiones las realizan los clientes externos.
     * @param TFCFrame ventana pricncipal de la aplicación TFC, este parámetro
     * se usa para comunicar el proceso principal con los diferentes hilos ó
     * cnversaciones activas.
     * @param Socket socket de comunicación entre este hilo de ejecución y el
     * host cliente que ha solicitado la conexión
     */
    public HiloServidor(TFCFrame tfcFrame, Socket pSocket) throws Exception{
        this.tfcFrame = tfcFrame;
        socket = pSocket;
        start();
    }

    /**
     * Método sobrecargado que se ejecuta al generarse un hilo de ejecuación
     * desde la llamada al método start() de la clase Thread. Se genera un nuevo
     * objeto de la clase Conversacion para realizar la charla con el cliente.
     */
    public void run(){

        try{
            Conversacion conv = new Conversacion(this);

        }catch(Exception e){
            try{
                socket.close();
            }catch(Exception exc){
                exc.printStackTrace();
            }
        }
    }

    /**
     * Método que retorna la variable objeto de tipo <code>TFCFrame<code>
     * @return varaible objeto de tipo TFCFrame que representa el marco
     * principal de la aplicación.
     */
    public TFCFrame getTFCFrame(){
        return tfcFrame;
    }

    /**
     * Método que retorna la variable objeto de tipo <code>Socket<code>
     * @return varaible objeto de tipo Socket que representa el socket de
     * comunicación entre dos hosts.
     */
    public Socket getSocket(){
        return socket;
    }
}
```

### 7.1.4 SeleccionConfiguracion.java

SeleccionConfiguracion.java

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import java.util.*;
import java.lang.reflect.*;
import javax.swing.text.*;
import java.text.*;

/**
 * Se informa el nick con el cual se presenta al resto de conversaciones, el
 * puerto en escucha si se desea que la aplicación actúe como servidor,
 * la activación del módulo servidor y si se desea depurar las conexiones, estos
 * valores están contenidos en un objeto de la clase interna DatosConfiguracion
 */
class SeleccionConfiguracion extends JPanel {

    /**
     * Constructor de la clase vacío.
     */
    public SeleccionConfiguracion() {

        setLayout(new GridBagLayout());

        // Construcción de componentes
        JLabel nickLabel = new JLabel("Nick: ");
        nickname = new JTextField();

        JLabel puertoLabel = new JLabel("Puerto: ");
        puerto = new JFormattedTextField();
        RegexFormatter formatoPuerto = new RegexFormatter("\\d{1,4}");
        formatoPuerto.setAllowsInvalid(false);
        formatoPuerto.setOverwriteMode(true);
        puerto.setFormatterFactory(new DefaultFormatterFactory(formatoPuerto));
        puerto.setValue("6666");
        puerto.setEnabled(false);
        puerto.setEditable(false);

        servidoractivo = new JCheckBox("Activar Servidor ");
        servidoractivo.addActionListener(new
        ActionListener() {
            public void actionPerformed(ActionEvent event) {
                if (servidoractivo.isSelected()){
                    if (anteriorConfiguracion != null &&
                    !anteriorConfiguracion.getServidorActivo()){
                        puerto.setEditable(true);
                        puerto.setEnabled(true);
                    }else {
                        puerto.setEditable(false);
                        puerto.setEnabled(true);
                    }
                }else{
                    puerto.setEditable(false);
                    puerto.setEnabled(false);
                }
            }
        });

        rastrear = new JCheckBox("Rastrear Comunicación ");

        //Añadir componentes al GridBag
        GridBagConstraints constraints = new GridBagConstraints();

        constraints.fill = GridBagConstraints.NONE;
        constraints.anchor = GridBagConstraints.EAST;
        constraints.weightx = 0;
        constraints.weighty = 50;

        add(nickLabel, constraints, 0,0,1,1);
        add(puertoLabel, constraints, 0,1,1,1);

        constraints.fill = GridBagConstraints.HORIZONTAL;
```

```
        constraints.weightx = 100;
        add(nickname, constraints, 1, 0, 2, 1);
        add(puerto, constraints, 1, 1, 1, 1);

        constraints.weightx = 0;
        constraints.weighty = 50;
        constraints.fill = GridBagConstraints.NONE;
        constraints.anchor = GridBagConstraints.WEST;
        add(servidoractivo, constraints, 1, 2, 2, 1);

        constraints.weightx = 0;
        constraints.weighty = 50;
        constraints.fill = GridBagConstraints.NONE;
        constraints.anchor = GridBagConstraints.WEST;
        add(rastrear, constraints, 1, 3, 2,1);

        // Crear botones Aceptar y Cancelar que terminan el dialogo
        okButton = new JButton("Aceptar");
        okButton.addActionListener(new
        ActionListener() {
            public void actionPerformed(ActionEvent event) {
                ok = true;
                guardarConfiguracion();
                dialog.setVisible(false);
            }
        });

        cancelButton = new JButton("Cancelar");
        cancelButton.addActionListener(new
        ActionListener() {
            public void actionPerformed(ActionEvent event) {
                dialog.setVisible(false);
            }
        });

        constraints.weightx = 0;
        constraints.weighty = 50;
        constraints.fill = GridBagConstraints.NONE;
        constraints.anchor = GridBagConstraints.SOUTH;
        add(okButton, constraints, 0, 4, 1, 1);
        constraints.anchor = GridBagConstraints.SOUTHWEST;
        add(cancelButton, constraints, 2, 4, 1, 1);

    }
    /**
     * Informa los valores por defecto del cuadro de dialogo.
     * @param dc los valores por defecto de los datos de configuración
     */
    public void setDatosConfiguracion(DatosConfiguracion dc) {
        nickname.setText(dc.getNick());
        puerto.setText(new Integer(dc.getPuerto()).toString());
        servidoractivo.setSelected(dc.getServidorActivo());
        rastrear.setSelected(dc.getRastrear());
    }

    /**
     * Obtener los datos del cuadro de dialogo.
     * @return un objeto DatosConfiguracion cuyo estado representa las entradas
     * de cuadro de dialogo
     */
    public DatosConfiguracion getDatosConfiguracion() {
        String sPuerto = puerto.getText();
        int puerto = Integer.parseInt(sPuerto);

        return new DatosConfiguracion(nickname.getText(),
        puerto, servidoractivo.isSelected(), rastrear.isSelected());
    }

    /**
     * Informa las entradas del cuadro de diálogo con los últimos valores
     * que se informaron.
     */
    public void recuperaConfiguracion() {
        nickname.setText(anteriorConfiguracion.getNick());
        puerto.setText(new Integer(anteriorConfiguracion.getPuerto()).toString());
        servidoractivo.setSelected(anteriorConfiguracion.getServidorActivo());
        if (servidoractivo.isSelected()){
```

```
            puerto.setEditable(false);
        }
        rastrear.setSelected(anteriorConfiguracion.getRastrear());
    }

    /**
     * Guardar la última configuración establecida. Se toman los valores actuales
     * del cuadro de diálogo.
     */
    public void guardarConfiguracion() {
        anteriorConfiguracion=new DatosConfiguracion(nickname.getText(),
        Integer.parseInt(new String(puerto.getText().toString())),
        servidoractivo.isSelected(),
        rastrear.isSelected());
    }

    /**
     * Mostar el cuadro de diálogo
     * @param propietario un componente en la frame propia o null
     * @param titulo título de la ventana de diálogo
     */
    public boolean showDialog(Component propietario, String titulo) {

        ok = false;

        // Se localiza la frame propietaria
        Frame owner = null;
        if (propietario instanceof Frame)
            owner = (Frame) propietario;
        else
            owner = (Frame)SwingUtilities.getAncestorOfClass(
            Frame.class, propietario);

        // Recperamos el tamaño de la pantalla disponible
        Toolkit kit = Toolkit.getDefaultToolkit();
        Dimension screenSize = kit.getScreenSize();
        int screenHeight = screenSize.height;
        int screenWidth = screenSize.width;

        // Si es la primera vez o el propietario ha cambiado,
        // generar nuevo diálogo
        if (dialog == null || dialog.getOwner() != owner) {
            dialog = new JDialog(owner, true);
            dialog.getContentPane().add(this);
            dialog.getRootPane().setDefaultButton(okButton);
            dialog.pack();
        }

        // Establecer título y mostrar diálogo
        dialog.setTitle(titulo);
        //dialog.setSize(screenWidth/6, screenHeight/8);
        dialog.setLocation((screenWidth-dialog.getWidth())/2,
        (screenHeight-dialog.getHeight())/2);
        dialog.show();

        return ok;
    }

    /**
     * Método para añadir componentes en un GridBagLayout
     * @param c el componente a añadir
     * @param constraints los constraints del gridBag a utilizar
     * @param x la posición x del grid
     * @param y la posición y del grid
     * @param w la anchura del grid
     * @param h la altura del grid
     */
    public void add(Component c, GridBagConstraints constraints,
    int x, int y, int w, int h) {
        constraints.gridx = x;
        constraints.gridy = y;
        constraints.gridwidth = w;
        constraints.gridheight = h;
        this.add(c, constraints);
    }

    private JTextField nickname;
```

```
    private JFormattedTextField puerto;
    private JCheckBox servidoractivo;
    private JCheckBox rastrear;
    private JButton okButton;
    private JButton cancelButton;
    private boolean ok;
    private JDialog dialog;

    private String antnickname;
    private int antpuerto;
    private boolean antservidoractivo;
    private boolean antrastrear;

    private DatosConfiguracion anteriorConfiguracion = null;

    /**
     * Método principal para realizar la llamada a esta clase. Sólo se establece
     * este método para realizar pruebas aisladas de esta clase.
     */
    public static void main(String args[]){
        SeleccionConfiguracion ss = new SeleccionConfiguracion();
        ss.showDialog(null, "Configuracion");
    }
}


/**
 * Clase que engloba los parámetros de configuración de la aplicación.
 */
class DatosConfiguracion {

    /**
     * Constructor de la clase que recibe los parámetros para establecer la
     * configuración.
     * @param aNick nombre o alias para presentarse en las conversaciones
     * @param aPuerto es el puerto que se escuchará cuando la aplicación actúe
     * como servidor.
     * @param aServidorActivo se utiliza para activar y desctivar el módulo
     * servidor de la aplicación.
     * @param aRastrear se utiliza para activar un rastreo del inicio de la
     * conversción entre dos host, es decir, el inercambio de claves y el
     * cifrado de mensajes.
     */
    public DatosConfiguracion(String aNick, int aPuerto,
    boolean aServidorActivo, boolean aRastrear) {
        nick = aNick;
        puerto = aPuerto;
        servidorActivo = aServidorActivo;
        rastrear = aRastrear;
    }

    public String getNick() { return nick; }
    public int getPuerto() { return puerto; }
    public boolean getServidorActivo() { return servidorActivo; }
    public boolean getRastrear() { return rastrear; }

    public void setNick(String aNick) { nick = aNick; }
    public void setPuerto(int aPuerto) { puerto = aPuerto;}
    public void setServidorActivo(boolean aServidorActivo){
        servidorActivo = aServidorActivo;
    }
    public void setRastrear(boolean aRastrear){
        rastrear = aRastrear;
    }

    private String nick;
    private int puerto;
    private boolean servidorActivo;
    private boolean rastrear;

}
```

### 7.1.5 SeleccionServidor.java

SeleccionServidor.java

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.text.*;
import java.text.*;
import java.util.*;
import java.lang.reflect.*;


/**
 * Se selecciona la dirección y el puerto remoto del host (servidor) con el que
 * se desea establecer una conexión.
 */
class SeleccionServidor extends JPanel {

    /**
     * Constructor de la clase vacío
     */
    public SeleccionServidor() {

        setLayout(new BorderLayout());

        JPanel panel = new JPanel();
        panel.setLayout(new BorderLayout());

        JPanel top = new JPanel();
        JPanel bottom = new JPanel();

        top.setLayout(new GridLayout(2,2));

        top.add(new JLabel("Dirección"));

        direccion = new JFormattedTextField();

        FormateadorDireccionIP formatoIP =
        new FormateadorDireccionIP("\\d{1,3}\\.\\d{1,3}\\.\\d{1,3}\\.\\d{1,3}");

        formatoIP.setAllowsInvalid(false);
        formatoIP.setOverwriteMode(false);
        formatoIP.setCommitsOnValidEdit(true);

        direccion.setFormatterFactory(new DefaultFormatterFactory(formatoIP));
        direccion.setValue("127.0.0.1");

        top.add(direccion);

        puerto = new JFormattedTextField();
        RegexFormatter formatoPuerto = new RegexFormatter("\\d{1,4}");
        formatoPuerto.setAllowsInvalid(false);
        formatoPuerto.setOverwriteMode(true);
        puerto.setFormatterFactory(new DefaultFormatterFactory(formatoPuerto));
        puerto.setValue("6666");
        top.add(new JLabel("Puerto:"));
        top.add(puerto);

        add(top, BorderLayout.NORTH);

        // Crear botones Ok y Cancelar que terminan el cuadro de diálogo
        botonOK = new JButton("Ok");
        botonOK.addActionListener(new
        ActionListener() {
            public void actionPerformed(ActionEvent event) {
                Object value = direccion.getValue();
                if (value.getClass().isArray()){
                    StringBuffer buffer = new StringBuffer();
                    for (int i = 0; i < Array.getLength(value); i++){
                        if (i > 0) buffer.append(".");
                        buffer.append(Array.get(value, i).toString());
                    }
```

```
                }
                ok = true;
                dialog.setVisible(false);
            }
        });

        JButton botonCancelar = new JButton("Cancelar");
        botonCancelar.addActionListener(new
        ActionListener() {
            public void actionPerformed(ActionEvent event) {
                dialog.setVisible(false);
            }
        });

        // Añadir botones de confirmación y cancelación
        JPanel panelBotones = new JPanel();
        panelBotones.add(botonOK);
        panelBotones.add(botonCancelar);
        add(panelBotones, BorderLayout.SOUTH);
    }

    /**
     * Se muestra el panel de selección en un diálogo
     * @param propietario un componente en la frame propia o null
     * @param titulo título de la ventana de diálogo
     */
    public boolean showDialog(Component parent, String title) {

        ok = false;

        // Localizar el propietario del frame
        Frame owner = null;
        if (parent instanceof Frame)
            owner = (Frame) parent;
        else
            owner = (Frame)SwingUtilities.getAncestorOfClass(
            Frame.class, parent);

        // Recperamos el tamaño de la pantalla disponible
        Toolkit kit = Toolkit.getDefaultToolkit();
        Dimension screenSize = kit.getScreenSize();
        int screenHeight = screenSize.height;
        int screenWidth = screenSize.width;

        // Si es la primera vez ó el propietario ha cambiado,
        // hacer nuevo diálogo
        if (dialog == null || dialog.getOwner() != owner) {
            dialog = new JDialog(owner, true);
            dialog.getContentPane().add(this);
            dialog.getRootPane().setDefaultButton(botonOK);
            dialog.pack();
        }
        dialog.setTitle(title);
        //dialog.setSize(screenWidth/7, screenHeight/7);
        dialog.setLocation((screenWidth-dialog.getWidth())/2,
        (screenHeight-dialog.getHeight())/2);
        dialog.show();
        return ok;
    }

    /**
     * Retorna la variable direccion
     * @return retorna la varaible direccion
     */
    public JTextField getDireccion(){
        return direccion;
    }

    /**
     * Retorna la variable puerto
     * @return retorna la varaible puerto
     */
    public JTextField getPuerto(){
        return puerto;
    }

    private JFormattedTextField direccion;
```

```
    private JFormattedTextField puerto;
    private JButton botonOK;
    private boolean ok;
    private JDialog dialog;

    /**
     * Método main para ejecutar la clase. Su objetivo es dar un
     * mecanimo de ejecuación únicamente para probar su funcionamiento.
     */
    public static void main(String args[]){
        SeleccionServidor ss = new SeleccionServidor();
        ss.showDialog(null, "Dirección del servidor");
    }
}
```

### 7.1.6 ServidorChat.java

ServidorChat.java

```
import java.io.*;
import java.net.*;
import javax.swing.*;

/**
 * Esta clase genera el proceso servidor de la aplicación. Se crea un nuevo
 * proceso que está a la escucha por el puerto establecido de las peticiones de
 * clientes externos. Con cada petición que llega, se crea una nueva conversacion
 * en un nuevo proceso y se continúa a la escucha atender a las nuevas posibles
 * peticiones.
 */
public class ServidorChat extends Thread {

    private int puerto;
    private ServerSocket serversocket= null;
    private Socket socket = null;
    private TFCFrame tfcFrame;

    /**
     * Constructor de la clase. Se reciben dos parámetros, por un lado un
     * objeto tipo TFCFrame con el cual se mantendrá la conexión entre la nueva
     * conversación creada y la aplicación principal y el número de puerto que
     * se escuchará.
     */
    public ServidorChat(TFCFrame tfcFrame, int puerto) throws Exception {
        this.tfcFrame = tfcFrame;
        this.puerto = puerto;
        serversocket = new ServerSocket(this.puerto);
        start();
    }

    /**
     * Método para finalizar el proceso de escucha ó el módulo servidor de la
     * aplicación.
     */
    public void finServidor(){
        try {
            serversocket.close();
        }catch(Exception e){
        }
    }

    /**
     * Método sobrecargado que se ejecuta al reallizar la llamada al método
     * start() de la clase Thread. Se realiza la escucha del puerto establecido
     * y se genera un nuevo proceso por cada petición recibida.
     */
    public void run(){
        try{
            while(true){
                socket = serversocket.accept();
                try{
                    HiloServidor hs = new HiloServidor(tfcFrame, socket);
                } catch(Exception e){
                    socket.close();
                }
            }
        } catch (IOException e){
        }
    }
}
```

### 7.1.7 TFC.java

TFC.java

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.border.*;
import java.io.*;
import java.net.*;
import com.sun.crypto.provider.SunJCE;
import java.security.*;
import javax.crypto.spec.*;
import java.util.*;

/**
 * Clase principal de la aplicación de chat seguro. Esta clase contiene el
 * método main de acceso a la aplicación. Contiene un menú con las diversas
 * opciones necesarias para establecer una conversación con otra dirección IP
 * o bien con el mismo orenador a partir de la dirección "localhost".
 */
public class TFC {

    // Ventana principal de la aplicación Chat
    private TFCFrame tfcFrame = null;


    /**
     * Constructor del objeto instaciado de la clase TFC.
     */
    public TFC() {
        tfcFrame = new TFCFrame();
    }


    /**
     * Método para recuperar el marco principal de la ventana
     * @return la variable objeto que representa el <code>frame<code> de la
     * ventana principal
     */
    public TFCFrame getTFCFrame(){
        return tfcFrame;
    }

    /**
     * Método <code>main<code> de la clase para ejecutar la clase.
     * @parms args[] array de parámetros para iniciar la aplicación
     * @trows se lanzan excepciones del tipo IOExcepion, InterruptedExcption,
     * Exception
     */
    public static void main(String args[]) throws IOException,
    InterruptedException, Exception {
        TFC tfc = new TFC();
        tfc.getTFCFrame().show();
    }
}


/**
 * Clase interna de la clase principal <code>TFC<code> que representa el
 * marco principal de la ventana mostrada.
 */
class TFCFrame extends JFrame {

    private TFCPanel tfcpanel;
    private JMenuItem configuracionItem=null;
    private SeleccionConfiguracion dialogSeleccionConfiguracion = null;
    private SeleccionServidor dialogServidor = null;
    private JMenuItem connectItem = null;
    private ServidorChat servidor = null;
    private DatosConfiguracion dc = null;

    /**
```

```
     * Constructor de la clase <code>TFCFrame<code> que representa la ventana
     * principal de la clase <code>TFC<code>
     */
    public TFCFrame() {

        // Recperamos el tamaño de la pantalla disponible
        Toolkit kit = Toolkit.getDefaultToolkit();
        Dimension screenSize = kit.getScreenSize();
        int screenHeight = screenSize.height;
        int screenWidth = screenSize.width;

        // Titulamos la ventana principal de la aplicación
        setTitle("Chat Cliente/Servidor");

        // Ajustamos el tamaño de la ventana principal en base al tamaño
        // disponible
        setSize(screenWidth/2, screenHeight/2);

        // Posicionamos la ventana en el centro de la pantalla
        setLocation(screenWidth/4, screenHeight/4);

        // Construir un menú
        JMenuBar mbar = new JMenuBar();
        setJMenuBar(mbar);
        JMenu menuArchivo = new JMenu("Archivo");
        mbar.add(menuArchivo);
        JMenu menuServidor = new JMenu("Configurar");
        mbar.add(menuServidor);

        // Añadir los elementos de conexión y salir
        connectItem = new JMenuItem("Conectar");
        connectItem.addActionListener(new accionSolicitarConexion());
        menuArchivo.add(connectItem);

        menuArchivo.addSeparator();

        // Inicializar datos de configuración del servidor
        dc = new DatosConfiguracion("tunick", 6666, false, false);

        // Con el elemento salir fin de la aplicación
        JMenuItem exitItem = new JMenuItem("Salir");

        // escuchador del elemento de menú "salir" para finalizar la aplicación
        exitItem.addActionListener(new
        ActionListener() {
            public void actionPerformed(ActionEvent event) {
                // Cerrar todos los sockets abiertos, cada uno pertenece a cada
                // una de las ventanas de conversación abiertas hasta ese momento.
                JInternalFrame[] frames=(tfcpanel.getJDeskTopPane()).getAllFrames();
                for (int i = 0; i < frames.length; i++){
                    frames[i].doDefaultCloseAction();
                }
                // Final del servidor.
                try {
                    servidor.finServidor();
                }catch (Exception ex){ }
                System.exit(0);
            }
        });

        menuArchivo.add(exitItem);

        configuracionItem = new JMenuItem("Configuracion");
        configuracionItem.addActionListener(new accionConfiguracion());
        menuServidor.add(configuracionItem);

        // Añadir un panel al frame
        tfcpanel = new TFCPanel();
        Container contentPane = getContentPane();
        contentPane.add(tfcpanel);

        // escuchador de la ventana. Se implementa el método windowClosing para
        // cerrar todos los sockets abiertos en ese momento.
        this.addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent e) {
                // Cerrar todos los sockets abiertos
                JInternalFrame[] frames=(tfcpanel.getJDeskTopPane()).getAllFrames();
```

```
                for (int i = 0; i < frames.length; i++){
                    frames[i].doDefaultCloseAction();
                }
                // Final del servidor.
                try {
                    servidor.finServidor();
                }catch (Exception ex){ }

                System.exit(0);
            }
        });
    }


    /**
     * Método para retornar el panel principal
     * @return panel principal de la ventana
     */
    public TFCPanel getTFCPanel(){
        return tfcpanel;
    }

    /**
     * Método para retornar la opción de menú de configuración
     * @return elemento de menú para llamar a la opción de configuración de la
     * aplicación.
     */
    public JMenuItem getconfiguracionItem(){
        return configuracionItem;
    }

    /**
     * Método para retornar la opción de menú de solicitud de conexión.
     * @return elemento de menú para realizar la solicitud de conexión a una IP.
     */
    public JMenuItem getconnectItem(){
        return connectItem;
    }

    /**
     * Método para retornar un objeto de la clase de configuración
     * @return devuelve un objeto del cuadro de diálogo que se presenta al
     * solicitar realizar la configuración de la aplicación.
     */
    public SeleccionConfiguracion getdialogSeleccionConfiguracion(){
        return dialogSeleccionConfiguracion;
    }

    /**
     * Método para indicar la dirección IP y el puerto de comunicaciones
     * del ordenador que hará las funciones de servidor. En este caso la
     * aplicación actúa como cliente.
     * @return se retorna el cuadro de diálogo que se presenta para informar
     * la dirección IP y el puerto de comunicaciones para solicitar una llamada.
     */
    public SeleccionServidor getdialogServidor(){
        return dialogServidor;
    }

    /**
     * Método para guardar el objeto que representa la configuración
     * de la aplicación.
     * @param objeto de la clase SeleccionConiguracion.
     */
    public void setdialogSeleccionConfiguracion(SeleccionConfiguracion
    dialogSeleccionConfiguracion){
        this.dialogSeleccionConfiguracion=dialogSeleccionConfiguracion;
    }

    /**
     * Método para guardar el objeto que representa los parámetros
     * de conexión del ordenador que actuará como servidor.
     * @param objeto de la clase SeleccionServidor.
     */
    public void setdialogServidor(SeleccionServidor dialogServidor){
        this.dialogServidor=dialogServidor;
    }
```

```
/**
 * Método para retornar los valores de la configuración del
 * aplicativo
 * @return se retorna un objeto de la clase DatosConfiguracion.
 */
public DatosConfiguracion getDatosConfiguracion(){
    return dc;
}

/**
 * Clase escuchadora para realizar la configuración del programa
 * Alias ó nombre del usuario del Chat (Nick)
 * Puerto de escucha
 * Activar servidor
 * Activar depuración
 */
private class accionConfiguracion implements ActionListener {

    public void actionPerformed(ActionEvent event) {

        // La primera vez construimos la ventana de dialogo de la
        // configuracion e iniciallizamos las variables por defecto.
        if (getdialogSeleccionConfiguracion() == null){
            setdialogSeleccionConfiguracion(new SeleccionConfiguracion());
            dialogSeleccionConfiguracion.setDatosConfiguracion(
            new DatosConfiguracion("tunick", 6666, false, false));
            dialogSeleccionConfiguracion.guardarConfiguracion();
        }
        else{
            dialogSeleccionConfiguracion.recuperaConfiguracion();
        }
        // Mostrar cuadro de dialogo de Configuración del Chat
        if (getdialogSeleccionConfiguracion().showDialog(TFCFrame.this,
        "Configuracion")) {
            dc = dialogSeleccionConfiguracion.getDatosConfiguracion();
            try{
                if (dc.getServidorActivo() && (servidor == null)){
                    servidor = new ServidorChat(TFCFrame.this, dc.getPuerto());
                }
                else {
                    try {
                        servidor.finServidor();
                        servidor = null;
                    }catch (Exception e){ }
                }
            }catch (Exception e){
                JOptionPane.showMessageDialog(TFCFrame.this, e.getMessage(),
                "Error", 0);
            }
        }
    }
}

/*
 * Clase escuchadora para realizar la solicitud de conexión. En este caso
 * la aplicación actuaría como cliente. Se solicitan los valores de:
 * Dirección IP y puerto de comunicaciones del host que hará las funciones
 * de servidor.
 */
private class accionSolicitarConexion implements ActionListener {
    public void actionPerformed(ActionEvent event) {
        // Si es la primera vez que solicitamos la conexión
        if (getdialogServidor() == null)
            setdialogServidor(new SeleccionServidor());

        // Mostar pantalla de diálogo y recuperar los valores informados
        if (getdialogServidor().showDialog(TFCFrame.this, "Conectar")) {
            try{
                setCursor(Cursor.getPredefinedCursor(Cursor.WAIT_CURSOR));
                int s = Integer.parseInt(
                new String(getdialogServidor().getPuerto().getText()));

                String a = getdialogServidor().getDireccion().getText();

                Socket ss =
                new Socket(getdialogServidor().getDireccion().getText(), s);
```

```
                    HiloCliente hilo = new HiloCliente(TFCFrame.this, ss);

                }catch (Exception e){
                    JOptionPane.showMessageDialog(TFCFrame.this, e.getMessage(),
                    "Error", 0);
                    setCursor(Cursor.getPredefinedCursor(Cursor.DEFAULT_CURSOR));
                }
            }
        }
    }
}

/**
 * Clase interna de la clase principal <code>TFC<code> que representa el
 * panel principal dentro del marco de la clase TFCFrame
 */
class TFCPanel extends JPanel {

    private JDesktopPane desktopPane;
    private JLabel etiquetaComentarios;

    /**
     * Constructor de la clase <code>TFCPanel<code> que representa el panel
     * principal de la clase <code>TFCFrame<code>
     */
    public TFCPanel() {

        setLayout(new BorderLayout());
        desktopPane = new JDesktopPane();
        JScrollPane scrollPane = new JScrollPane(desktopPane);
        desktopPane.putClientProperty("JDesktopPane.dragMode", "outline");
        desktopPane.setLayout(new FlowLayout());
        add(scrollPane, BorderLayout.CENTER);
        etiquetaComentarios = new JLabel(" ");
        Border raisedetched = BorderFactory.createEtchedBorder(EtchedBorder.RAISED);
        etiquetaComentarios.setBorder(raisedetched);
        add(etiquetaComentarios, BorderLayout.SOUTH);

    }

    /**
     * Método para retornar la variable objeto de tipo DeskTopPane
     * @return objeto de la clase JDeskTopPane
     */
    public JDesktopPane getJDeskTopPane(){
        return desktopPane;
    }

    /**
     * Método para informar el valor de la etiqueta <code>EtiquetaComentarios<code>
     * @param String con el nuevo valor de la etiqueta
     */
    public void setEtiquetaComentarios(String comentarios){
        etiquetaComentarios.setText(comentarios);
    }

    /**
     * Método para recuperar el objeto de tipo JLabel EtiquetaComentarios
     * @return variable objeto etiquetaComentarios
     */
    public JLabel getEtiquetaComentarios(){
        return etiquetaComentarios;
    }

}
```

### 7.1.8 RegexFormatter.java

RegexFormatter.java

```
import java.text.*;
import java.util.*;
import java.util.regex.*;
import javax.swing.*;
import javax.swing.text.*;

/**
 * Una expresión regular basada en la implementación de
 * <code>AbstractFormatter</code>.
 */
public class RegexFormatter extends DefaultFormatter {

    private Pattern pattern;
    private Matcher matcher;

    /**
     * Crea un nuevo objeto de formateado vacío
     */
    public RegexFormatter() {
        super();
    }

    /**
     * Crea una expresión regular basada en <code>AbstractFormatter</code>.
     * <code>pattern</code> especifica la expresión regular que se utilizará
     * para determinar si un valor es legal.
     */
    public RegexFormatter(String pattern) throws PatternSyntaxException {
        this();
        setPattern(Pattern.compile(pattern));
    }

    /**
     * Crea una expresión regular basada en <code>AbstractFormatter</code>.
     * <code>pattern</code> especifica la expresión regular que se utilizará
     * para determinar si un valor es legal.
     */
    public RegexFormatter(Pattern pattern) {
        this();
        setPattern(pattern);
    }

    /**
     * Se informa el patrón que se usará para determinar si un valor es legal.
     * @param Pattern patrón para deerminar si el valor es legal
     */
    public void setPattern(Pattern pattern) {
        this.pattern = pattern;
    }

    /**
     * Se retorna el <code>Pattern</code> utilizado para determinar si un valor
     * es legal.
     * @return Pattern patrón ustilizado para determinar si un valor es legal.
     */
    public Pattern getPattern() {
        return pattern;
    }

    /**
     * Se informa el <code>Matcher</code> usado en la prueba más reciente de
     * si un valor es legal.
     * @param Matcher
     */
    protected void setMatcher(Matcher matcher) {
        this.matcher = matcher;
    }

    /**
     * Retornar el <code>Matcher</code> de la mayoría de las pruebas.
     */
    protected Matcher getMatcher() {
```

```
        return matcher;
    }

    /**
     * Convierte <code>text</code> retornando un objeto arbitrario. Algunos
     * formateadres pueden retornar null.
     * <p>
     * Si se ha especificado un <code>Pattern</code> y el texto coincide
     * totalmente la expresión regular invocará <code>setMatcher</code>.
     *
     * @throws ParseException si hay un error en la conversión
     * @param text cadena a convertir
     * @return Object representación del texto
     */
    public Object stringToValue(String text) throws ParseException {
        Pattern pattern = getPattern();

        if (pattern != null) {
            Matcher matcher = pattern.matcher(text);

            if (matcher.matches()) {
                setMatcher(matcher);
                return super.stringToValue(text);
            }
            throw new ParseException("Patrón no coincidente", 0);
        }
        return text;
    }
}
```

### 7.1.9 FormateadorDireccionIP.java

FormateadorDireccionIP.java

```
import java.text.*;
import java.util.*;
import javax.swing.text.*;
import java.util.regex.*;

/**
 * Formateador para direcciones IP de 4 bytes de la forma a.b.c.d
 */
public class FormateadorDireccionIP extends RegexFormatter{

    /**
     * Constructor para crear un formateador de direcciones IP vacío y
     * llamando al constructor de su superclase RegexFormatter.
     */
    public FormateadorDireccionIP(){
        super();
    }

    /**
     * Crea una expresión regular basada en <code>AbstractFormatter</code>.
     * <code>pattern</code> especifica la expresión regularque se utilizará
     * para determinar si una dirección IP es correcta
     */
    public FormateadorDireccionIP(String pattern)throws PatternSyntaxException{
        this();
        setPattern(Pattern.compile(pattern));
    }

    /**
     * Crea una expresión regular basada en <code>AbstractFormatter</code>.
     * <code>pattern</code> especifica la expresión regularque se utilizará
     * para determinar si una dirección IP es legal
     */
    public FormateadorDireccionIP(Pattern pattern) {
        this();
        setPattern(pattern);
    }

    /**
     * Formatea <code>text</code> devolviendo el texto formateado que
     * representa una dirección IP.
     * <p>
     * Si un <code>Pattern</code> ha sido especificado y el texto
     * coincide completamente con la expresión regular se invocará a
     * <code>setMatcher</code>.
     *
     * @throws ParseException si hay algún error en la conversión
     * @param text String a convertir
     * @return Object representación del texto
     */

    public Object stringToValue(String text) throws ParseException {

        Pattern pattern = getPattern();

        if (pattern != null) {
            Matcher matcher = pattern.matcher(text);
            if (matcher.matches()) {
                setMatcher(matcher);
                StringTokenizer tokenizer = new StringTokenizer(text, ".");
                byte[] a = new byte[4];
                for (int i = 0; i < 4; i++) {
                    int b = 0;
                    String s = null;
                    try {
                        s = tokenizer.nextToken();
                        b = Integer.parseInt(s);
                        a[i] = (byte)b;
                    }
                    catch (NumberFormatException e) {
```

```
                    throw new ParseException("Valor digitado no es un integer", 0);
                }
                catch (Exception ex) {
                    throw new ParseException("IP incompleta", 0);
                }
                if (b < 0 || b >= 256) {
                    throw new ParseException("Valor fuera de rango", 0);
                }
                a[i] = (byte)b;
            }
            return text;
        }
        throw new ParseException("Error en patrón de datos", 0);
    }
    return text;
  }
}
```