Universidad de Buenos Aires

Facultades de Ciencias Económicas, Ciencias Exactas y Naturales e Ingeniería

Maestría en Seguridad Informática

Tesis

Título

Implementación de un Protocolo de Intercambio de Claves Diffie-Hellman empleando Anillos No Conmutativos

Autor: Byron Ron H.

Director de Tesis: Dr. Pedro Hecht

Año

2013

RESUMEN

Los protocolos usados en criptografía asimétrica (AC) y criptografía de clave pública (PKC), en su mayoría, están estrechamente relacionados con funciones trampa de una vía, principalmente con los problemas de logaritmo discreto y de factorización de enteros. Estos problemas usan operaciones aritméticas en estructuras algebraicas conmutativas, además necesitan del uso de bibliotecas de precisión extendida y requieren de costosas operaciones de cálculo.

Sin embargo, es interesante ver como en los últimos años ha surgido un gran interés por el uso de estructuras algebraicas no conmutativas. Es así que Hecht presenta un interesante modelo, que lo denomina “compacto”, de criptografía asimétrica usando anillos no conmutativos de matrices.

En el presente trabajo se desarrolló una implementación del protocolo Diffie-Hellman propuesto por Hecht. La cual no hace uso de bibliotecas de precisión extendida y que permite establecer una clave compartida entre dos entes, la misma que es usada para el intercambio seguro de mensajes de texto. Esta implementación es utilizada en dispositivos con poder computacional limitado, como es el caso de la telefonía celular.

Palabras clave: criptografía asimétrica, álgebra no conmutativa, Intercambio de claves Diffie-Hellman.

TABLA DE CONTENIDOS

RESUMEN ..... **iii**

AGRADECIMIENTOS ..... **vii**

ABREVIATURAS ..... **ix**

CAPITULO 1 Introducción **10**

1.1 Estado Actual del Tema 11

1.2 Objetivos 11

1.3 Organización de la Tesis 12

CAPITULO 2 Antecedentes **13**

2.1 Criptografía de Clave Pública 13

2.2 Protocolo Diffie-Hellman 13

2.2.1 Algoritmo 14

2.3 Modelo Compacto de NCAS [6] 15

2.3.1 Fundamentos Algebraicos y Computacionales 15

2.3.2 Problemas complejos en grupos (y semigrupos) no conmutativos 16

2.3.3 Problemas complejos en anillos no conmutativos 17

2.4 Protocolo Compacto Diffie-Hellman (DH-C) 19

2.4.1 Algoritmo 19

2.4.2 Generalización de Exponentes 20

CAPITULO 3 Implementación de DH-C **23**

3.1 Software y Hardware utilizado 23

3.2 Diffie-Hellman Compacto (DH-C) 24
3.2.1 Clase DHC 25
3.2.2 Clase Matrix 26
3.2.3 Clase DHCTest 27
3.2.4 Clase ExecTimer 27
3.2.5 DHC-Demo 27
CAPITULO 4 Seguridad de las Técnicas de Intercambio de Claves.. 28
4.1 Principales Métodos de Ataque a DL en Campos Numéricos [5] 28
4.1.1 Búsqueda Exhaustiva o Fuerza Bruta 29
4.1.2 Baby-Step Giant-Step 29
4.1.3 Rho de Pollard 30
4.1.4 Pohlig-Hellman 32
4.1.5 Index-Calculus 33
4.2 Seguridad Computacional del Protocolo DH-C [6] 35
CAPITULO 5 Evaluación y Pruebas de DH-C 37
5.1 Pruebas de la aplicación DHC-Demo 37
5.2 Evaluación del protocolo DH-C 47
5.2.1 Tiempo de Ejecución 47
5.2.1 Uso de Memoria 48
CAPITULO 6 Conclusiones y Trabajos Futuros 51
6.1 Conclusiones 51
6.2 Trabajos Futuros 52
ANEXOS 53

A.1. Estructura de la Interfaz de Usuario 53

A.2. Ejecución de la clase DHCTestClass 60

BIBLIOGRAFIA 62

AGRADECIMIENTOS

A mis queridos padres, Martha y Homero, por todo el apoyo que me han brindado incondicionalmente y en todo sentido, sin ustedes nada de esto hubiera sido posible.

A mi hermana, Nadia, por ser mi amiga, mi consejera y por estar junto a mí cuando más te necesito, eres mi ejemplo a seguir.

A mi Director de Tesis Dr. Pedro Hecht, por aceptar ser mi tutor y confiar en mi trabajo aún sin conocerme, gracias por su orientación, paciencia y motivación.

ABREVIATURAS

Abreviatura	Significado
IFP	Problema de Factorización Entera
DLP	Problema del Logaritmo Discreto
DL	Logaritmo Discreto
AC	Criptografía Asimétrica
PKC	Criptografía de Clave Pública
SDP	Problema de la Descomposición Simétrica
NCAS	Estructura Algebraica No Conmutativa
DH-C	Diffie Hellman Compacto
GDLP	Problema Generalizado del Logaritmo Discreto
CSP	Problema de la Búsqueda del Conjugador
DP	Problema de la Descomposición
GSDP	Problema de la Descomposición Simétrica Generalizada
PDH	Problema Diffie-Hellman
DSA	Algoritmo de Firma Digital
JDK	Java Development Kit
IDE	Integrated Development Environment
SDK	Software Development Kit
ADT	Android Development Tools
API	Application Programming Interface
ADB	Android Debug Bridge
PID	Process Identifier
USB	Universal Serial Bus

CAPITULO 1 Introducción

Antes que Whitfield Diffie y Martin Hellman introdujeran la criptografía de clave pública al mundo en 1976, los algoritmos de cifrado utilizados eran algoritmos de cifrado simétricos donde tanto el remitente como el destinatario deben compartir la misma clave, la cual es intercambiada a través de un canal seguro.

Diffie y Hellman propusieron un método por el cual un individuo A puede cifrar un mensaje utilizando la clave pública de un individuo B, pero solamente B sería capaz de descifrar este mensaje con su clave privada. Para lograr esto se utiliza lo que se conoce comúnmente como “funciones trampa de una vía”. De hecho, la seguridad en la mayoría de los sistemas de clave pública se basa en el problema de la factorización de enteros positivos (IFP) y el problema del logaritmo discreto (DLP) [5]. El intercambio de claves Diffie-Hellman se basa en el problema DLP. Este protocolo permite que dos usuarios A y B puedan acordar una clave secreta, mediante una secuencia de transmisiones a través de un canal público.

Una desventaja de los problemas IFP y DLP reside en que se conocen ataques de complejidad sub-exponencial y ataques eficientes a través de algoritmos cuánticos, de modo que hay serios motivos para desconfiar de ellos. Para reemplazar de raíz a la criptografía asimétrica (AC) convencional (y su aplicación en criptografía de clave pública), ha surgido en años recientes un interés creciente en la aplicación de estructuras algebraicas no conmutativas [6].

Es así que en 2007, Cao, Dong y Wang [7] presentan un interesante esquema general de criptografía de clave pública (PKC), basándose en la dificultad de resolver el problema de la descomposición simétrica (SDP) en un anillo no conmutativo de polinomios matriciales (PSDP).

1.1 Estado Actual del Tema

En la actualidad existe una fuerte tendencia al reemplazo de AC y PKC basada en sistemas algebraicos numéricos y conmutativos por otros de naturaleza no conmutativa [1] [2] [3] [4].

Este trabajo toma como base el desarrollo de Hecht [6] en el cual se aplica el problema DLP en el marco de una estructura algebraica no conmutativa (NCAS) empleando el anillo no conmutativo de matrices $M_4[\mathbb{Z}_{256}]$ que no requiere el empleo de bibliotecas numéricas de precisión extendida.

1.2 Objetivos

El principal objetivo que se pretende alcanzar con el presente trabajo es el de desarrollar en Java una implementación práctica del protocolo Hecht [6], que pueda ser utilizado en telefonía móvil de forma rápida, segura y eficiente, para esto se ha planteado los siguientes objetivos específicos:

- Analizar el modelo compacto de criptografía asimétrica usando anillos no conmutativos desarrollado por Hecht [6].
- Definir la herramienta de desarrollo que se utilizará para esta implementación.
- Identificar los principales componentes que formarán parte de la aplicación final.
- Implementar dicho modelo en dispositivos con poder computacional limitado.
- Efectuar pruebas de eficiencia computacional (benchmarking) para poder comparar lo desarrollado con soluciones Diffie-Hellman convencionales.

1.3 Organización de la Tesis

La presente tesis se divide en 6 capítulos. El primero contiene una introducción a la temática a tratar, se muestra el estado actual del tema y se definen los objetivos a alcanzar con el desarrollo de este trabajo. El capítulo 2 contiene el marco teórico con respecto a la criptografía de clave pública, protocolo Diffie-Hellman y protocolo compacto Diffie-Hellman (DH-C). En el capítulo 3 se detalla información referente al lenguaje de programación escogido, especificaciones de software y hardware utilizados y se explica más detalladamente cómo se llevó a cabo el diseño e implementación de DH-C. En el capítulo 4 se hace un análisis de la seguridad relativa de las técnicas de intercambio de claves, en el cual se detalla los principales métodos de ataque a DL en campos numéricos. Los resultados de las pruebas realizadas durante el desarrollo de la tesis en lo referente al funcionamiento de la aplicación así como los tiempos de ejecución y consumo de memoria del protocolo DH-C están expuestos en el capítulo 5. Finalmente el capítulo 6 contiene las conclusiones a las que se llegaron y el trabajo futuro que se puede realizar siguiendo esta línea, así como las recomendaciones para mejorarlo.

CAPITULO 2 Antecedentes

2.1 Criptografía de Clave Pública

Antes que Whitfield Diffie y Martin Hellman introdujeran la criptografía de clave pública, se requería que tanto el emisor como el receptor compartan una clave secreta común y esta clave no podía ser simplemente enviada por un canal de comunicación inseguro, ya que podría comprometer dicha clave.

Diffie y Hellman, en 1976 [8], proponen un nuevo tipo de criptografía llamada “criptografía de clave pública” (PKC). PKC utiliza dos claves: la primera, la “clave pública”, es utilizada por el emisor para cifrar el mensaje, y la segunda, la “clave privada”, es usada por el receptor para descifrar el mensaje.

PKC utiliza lo que se conoce como “funciones trampa de una vía”. Un ejemplo de este tipo de funciones es la factorización del producto de dos números primos grandes. La multiplicación de estos números primos es fácil de realizar, sin embargo, factorizar el producto resultante es muy difícil. De hecho, la seguridad en la mayoría de los sistemas de clave pública se basa en el problema de la factorización de enteros positivos (IFP) y el problema del logaritmo discreto (DLP) [5]. El intercambio de claves Diffie-Hellman se basa en el problema DLP, a continuación se describen los aspectos más importantes del Protocolo de Intercambio de claves Diffie-Hellman.

2.2 Protocolo Diffie-Hellman

El protocolo de intercambio de claves Diffie-Hellman es una solución práctica al problema de distribución de claves, permitiendo que dos entidades puedan establecer un secreto compartido, mediante un intercambio de mensajes a través de un canal de comunicación público

(inseguro), todo esto sin previamente haber compartido información de las claves secretas [5]. Este intercambio o acuerdo se lleva a cabo sin que una tercera entidad, la cual tiene acceso completo a los datos que son transmitidos por este canal, pueda conocerlo o calcularlo en un tiempo razonable.

En la Figura 1 se muestra el protocolo Diffie-Hellman, en donde Alice y Bob son las entidades que quieren realizar el acuerdo.

2.2.1 Algoritmo

El siguiente intercambio entre Alice y Bob muestra el protocolo de intercambio de claves Diffie-Hellman:

1. Elementos Públicos:
- Se establece un número primo p y un generador $\alpha \in \mathbb{Z}_{\mathrm{p}}^{*}$, los cuales son públicos y conocidos no solo por Alice y Bob.

2. Generación de Claves de Usuario:
- Alice escoge $\mathrm{X_A} \in \mathbb{Z}_{\mathrm{p}-1}$ al azar, calcula $\mathrm{Y_A} = \alpha^{\mathrm{X_A}} \bmod p$, y envía $\mathrm{Y_A}$ a Bob.
- Bob escoge $\mathrm{X_B} \in \mathbb{Z}_{\mathrm{p}-1}$ al azar, calcula $\mathrm{Y_B} = \alpha^{\mathrm{X_B}} \bmod p$, y envía $\mathrm{Y_B}$ a Alice.

3. Envío de Claves:
- Alice envía $\mathrm{Y_A}$ a Bob.
- Bob envía $\mathrm{Y_B}$ a Alice.

4. Cálculo de Clave Compartida
- Bob recibe Y_A y calcula la clave compartida $K = (Y_A)^{X_B} \bmod p$.
- Alice recibe Y_B y calcula la clave compartida $K = (Y_B)^{X_A} \bmod p$.

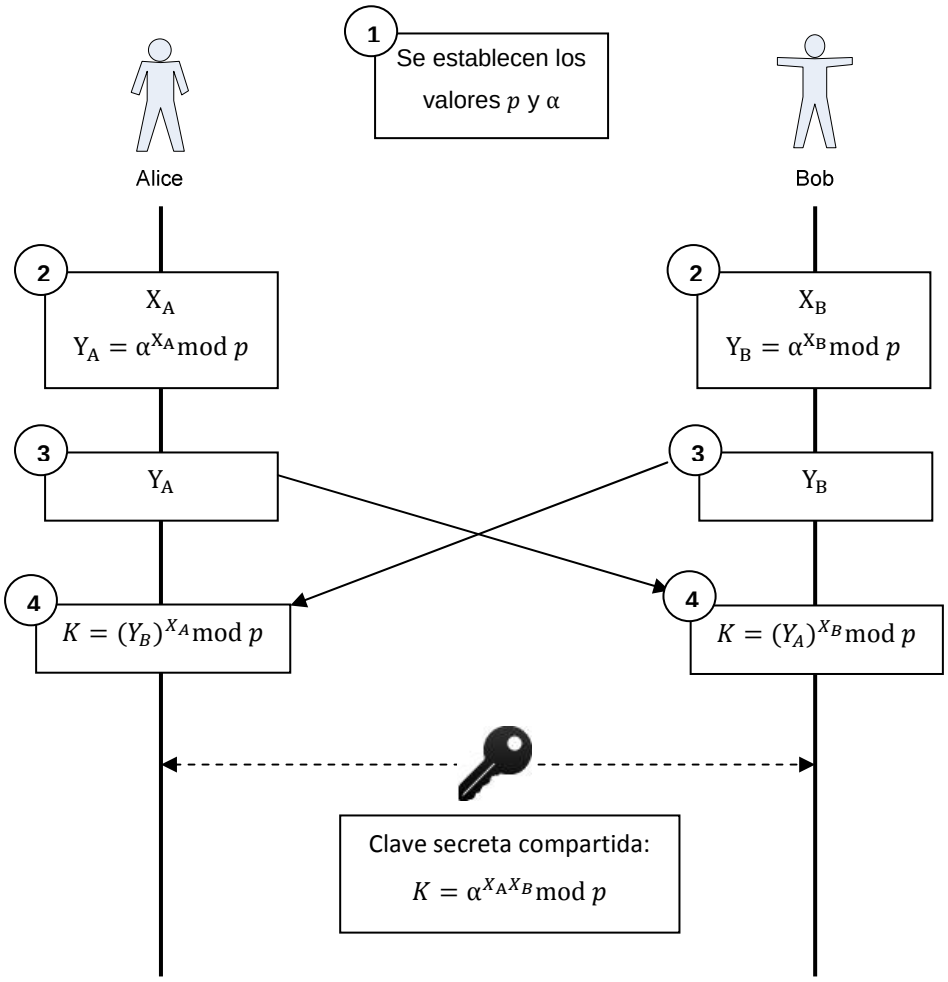


Figura 1: Pasos del Protocolo Diffie-Hellman

En este momento Alice y Bob tienen ya un secreto compartido, que un tercero no puede conocer, pese a haber tenido acceso a todo el intercambio de información.

2.3 Modelo Compacto de NCAS [6]

2.3.1 Fundamentos Algebraicos y Computacionales

Los fundamentos mencionados a continuación se basan en el trabajo de Hecht.

.

2.3.2 Problemas complejos en grupos (y semigrupos) no conmutativos

En un grupo no conmutativo G, dos elementos (x, y) son *conjugados,* representado como $x \sim y$, si $y = z^{-1}x\,z$ para algún $z \in G$. Aquí a z o a z^{-1} se lo denomina el elemento *conjugador*. Así se definen problemas que a su manera, representan instancias algebraicas del problema generalizado del *logaritmo discreto* (GDLP).

(CSP) Problema de la Búsqueda del Conjugador: dados $(x, y) \in G \times G$, hallar $z \in G$ tal que $y = z^{-1}\,x\,z$.

(DP) Problema de la Descomposición: dados $(x, y) \in G \times G$ y $S \subseteq G$, hallar $(z_1, z_2) \in S$ tal que $y = z_1\,x\,z_2$.

(SDP) Problema de la Descomposición Simétrica: dados $(x, y) \in G \times G$, y $(m, n) \in \mathbb{Z}$, hallar $z \in G$ tal que $y = z^m\,x\,z^n$.

(GSDP) Problema de la Descomposición Simétrica Generalizada: dados $(x, y) \in G \times G$ y $S \subseteq G$ y $(m, n) \in \mathbb{Z}$, hallar $z \in S$ tal que $y = z^m\,x\,z^n$.

Acorde a los conocimientos presentes, todos los problemas aquí aplicados sobre grupos no conmutativos generales son lo suficientemente difíciles como para ser empleados con seguridad criptográfica [7]. Dichos problemas se encuentran ordenados por grado de complejidad creciente. Es así que GSDP es al menos tan complejo como SDP. Se concluye que no se conoce (y casi seguramente no exista) algoritmo probabilístico de tiempo P capaz de resolver GSDP si la escala dimensional es la adecuada. Dado que GSDP puede ser considerado una instancia del GDLP, se puede agregar:

(CDH) Problema computacional Diffie-Hellman en el grupo no conmutativo G (con respecto a algún subgrupo abeliano S): Computar $x^{z_1 z_2}$ $(= x^{z_2 z_1})$ para un dado (x, x^{z_1}, x^{z_2}) tal que $x \in G, (z_1, z_2) \in S$ y $S \subseteq G$.

Hoy día no hay ninguna idea acerca de cómo extraer z_1 (o z_2) de los datos provistos [9] y eso es alentador si se lo compara con el empleo

tradicional de grupos numéricos conmutativos. Las mismas consideraciones son válidas si G fuese un semigrupo no conmutativo, basta con restringir las condiciones de los exponentes a valores enteros positivos $(m,n) \in \mathbb{Z}_{>0}$ o $(z_1, z_2) \in \mathbb{Z}_{>0}$.

2.3.3 Problemas complejos en anillos no conmutativos

Para ejemplificar los siguientes problemas computacionalmente complejos trabajaremos sobre anillos no conmutativos de polinomios con coeficientes enteros positivos. Sea R un anillo $(R, +, 0)$ y $(R, \cdot, 1)$, donde la suma forma un *grupo abeliano* y el producto forma un *semigrupo no abeliano.*

Definición 1. El producto escalar se define para $k \in \mathbb{Z}_{>0}$ y $r \in R$ y vale

$$(k)r \triangleq \langle r + \cdots + r \rangle_{k\ veces} \quad (1)$$

Cuando $k \in \mathbb{Z}_{<0}$ vale

$$(k)r \triangleq (-k)(-r) = \langle (-r) + \cdots + (-r) \rangle_{k\ veces} \quad (2)$$

Para $k = 0$ es natural que $(k)r = 0$. Similarmente, vale $r^k = \langle r \cdot \ldots \cdot r \rangle_{k\ veces}$.

Teorema 1. $(a)r^m.(b)r^n = (ab)r^{m+n} = (b)r^n.(a)r^m,\ \forall a, b, m, n \in \mathbb{Z}$ y $\forall r \in R$, lo que se demuestra elementalmente aplicando la propiedad asociativa del anillo y la propiedad conmutativa de la suma.

Lema 1. Nótese que en general $(a)r.(b)s \neq (b)s.(a)r$ si $r \neq s$ dada la *no conmutatividad* del semigrupo producto. Esta distinción es crucial para el modelo desarrollado.

Definición 2. Se define el anillo de polinomios de coeficientes enteros positivos para todo elemento $r \in R$.

$$f(r) = \sum_{i=0}^{n} (a_i) r^i = (a_0).1 + (a_1).r + \cdots + (a_n).r^n \ \ donde\ \forall i | a_i \in \mathbb{Z}_{>0} \quad (3)$$

Aquí r puede considerarse como una simple variable (x) y el conjunto de los polinomios $f(r)$ puede considerarse como la extensión polinómica del anillo $\mathbb{Z}_{>0}[r]$ para todo polinomio con coeficientes enteros positivos $f(r)$ y todo $r \in R$.

Teorema 2. Dados dos polinomios cualesquiera de la extensión del anillo $f(r), h(r) \in \mathbb{Z}_{>0}[r]$, se demuestra

$$f(r).h(r) = h(r).f(r) \tag{4}$$

Es elemental la demostración usando el Teorema 1 y la *propiedad distributiva* del producto respecto de la suma.

Lema 2. Nótese que en general $f(r).h(s) \neq h(s).f(r)$ si $r \neq s$ dada la no conmutatividad del semigrupo producto. También esta distinción resulta importante para fundamentar al modelo desarrollado.

Definición 3. Dado un anillo no conmutativo $(R, +, \cdot)$ y cualquier elemento $a \in R$ elegido al azar, definimos un subanillo $P_a \subseteq R$ por:

$$P_a \triangleq \{f(a): f(x) \in \mathbb{Z}_{>0}[x]\} \tag{5}$$

Es decir P_a abarca al conjunto de todos los polinomios de la extensión aplicados sobre la misma variable a.

Ahora podemos definir nuevas versiones de los problemas SDP y CDH definidos en el apartado 2.3.2.

(SDP) Problema de la descomposición simétrica sobre el anillo no conmutativo de polinomios con coeficientes enteros positivos R: dados $(a, x, y) \in R^3$, y $(m, n) \in \mathbb{Z}$, hallar $z \in P_a$ tal que $y = z^m \, x \, z^n$.

(PDH) Problema computacional Diffie-Hellman en el anillo no conmutativo de polinomios con coeficientes enteros positivos R: Computar $x^{z_1 z_2}$ $(= x^{z_2 z_1})$ para un dado (a, x, x^{z_1}, x^{z_2}) tal que $a \neq x, (a, x) \in R^2$ y $(z_1, z_2) \in P_a$.

Nótese que P_a es precisamente el subanillo conmutativo necesario para poder implementar el protocolo de intercambio de claves Diffie-Hellman. No importa que instancias polinómicas $f(a),\ h(a)$ se elijan, su producto será independiente del orden. Pero esto deja de ser válido si el producto de esos mismos polinomios se aplica a distintos elementos $f(a), h(b)$, ver el Teorema 2 y su Lema. Resulta evidente que si valen todos los antecedentes previamente expuestos, ambos problemas (SDP y PDH) son computacionalmente intratables, o sea no existe (o al menos no se conoce) un algoritmo probabilístico de tiempo P que pueda resolverlos con precisión no despreciable con respecto a la escala dimensional del problema [10] (lo que se consigue con cardinales $|R|$ y $|P_a|$ suficientemente grandes como para no ser resuelto con los recursos computacionales vigentes).

2.4 Protocolo Compacto Diffie-Hellman (DH-C)

Una vez analizados los fundamentos teóricos necesarios, estamos en condiciones de presentar el protocolo definido como *compacto* de intercambio de claves Diffie-Hellman empleando anillos no conmutativos de polinomios matriciales con coeficientes enteros positivos.

En la Figura 2 se muestra el protocolo de DH-C, en donde Alice y Bob son las entidades que quieren realizar el acuerdo.

2.4.1 Algoritmo

El siguiente intercambio entre Alice y Bob muestra el protocolo de intercambio de claves Diffie-Hellman empleando anillos no conmutativos de polinomios matriciales con coeficientes enteros positivos:

1. Alice escoge al azar los siguientes valores: exponentes y matrices, y se los envía a Bob a través de un canal público:

   - $(m, n) \in \mathbb{Z}_{16} > 0$, exponentes.

- $(a, b) \in R$, cada uno de estos elementos es una matriz $M_4[\mathbb{Z}_{256}]$.

2. Alice define su clave privada:

   - Un polinomio entero al azar no nulo $f(x) \in \mathbb{Z}_{16}[x] \mid f(a) \neq 0$, donde sus coeficientes y exponentes sean enteros positivos $(\in \mathbb{Z}_{16} > 0)$.

3. Bob define su clave privada:

   - Un polinomio entero al azar no nulo $h(x) \in \mathbb{Z}_{16}[x] \mid h(a) \neq 0$, donde sus coeficientes y exponentes sean enteros positivos $(\in \mathbb{Z}_{16} > 0)$.

4. Alice computa $r_A = f(a)^m.\ b.\ f(a)^n$ y se lo envía a Bob.

5. Bob computa $r_B = h(a)^m.\ b.\ h(a)^n$ y se lo envía a Alice.

6. Alice recibe r_B y calcula la clave compartida $K_A = f(a)^m.\ r_B.\ f(a)^n$.

7. Bob recibe r_A y calcula la clave compartida $K_B = h(a)^m.\ r_A.\ h(a)^n$.

   Observar que $K_A = K_B$ dado que se verifica:

$$K_A = f(a)^m.h(a)^m.b.h(a)^n.f(a)^n = h(a)^m.f(a)^m.b.f(a)^n.h(a)^n = K_B$$

2.4.2 Generalización de Exponentes

Debido a la conmutatividad de los polinomios de igual base, Alice y Bob están en la capacidad de seleccionar sus propios exponentes. Esto incrementa el nivel de seguridad del protocolo DH-C, ya que se retira 2 componentes de la clave pública (exponentes (m, n)) y se generan 4 números privados como se muestra a continuación:

1. Alice escoge al azar los siguientes valores y se los envía a Bob a través de un canal público:

   - $(a, b) \in R$, cada uno de estos elementos es una matriz $M_4[\mathbb{Z}_{256}]$.

2. Alice define su clave privada:

   - $(m, n) \in \mathbb{Z}_{16} > 0$, exponentes.
   - Un polinomio entero al azar no nulo $f(x) \in \mathbb{Z}_{16}[x] \mid f(a) \neq 0$, donde sus coeficientes y exponentes sean enteros positivos $(\in \mathbb{Z}_{16} > 0)$.

3. Bob define su clave privada:

   - $(r, s) \in \mathbb{Z}_{16} > 0$, exponentes.
   - Un polinomio entero al azar no nulo $h(x) \in \mathbb{Z}_{16}[x] \mid h(a) \neq 0$, donde sus coeficientes y exponentes sean enteros positivos $(\in \mathbb{Z}_{16} > 0)$.

4. Alice computa $r_A = f(a)^m.\ b.\ f(a)^n$ y se lo envía a Bob.

5. Bob computa $r_B = h(a)^r.\ b.\ h(a)^s$ y se lo envía a Alice.

6. Alice recibe r_B y calcula la clave compartida $K_A = f(a)^m.\ r_B.\ f(a)^n$.

7. Bob recibe r_A y calcula la clave compartida $K_B = h(a)^r.\ r_A.\ h(a)^s$.

   Observar que $K_A = K_B$ dado que se verifica:

$$K_A = f(a)^m.h(a)^r.b.h(a)^s.f(a)^n = h(a)^r.f(a)^m.b.f(a)^n.h(a)^s = K_B$$

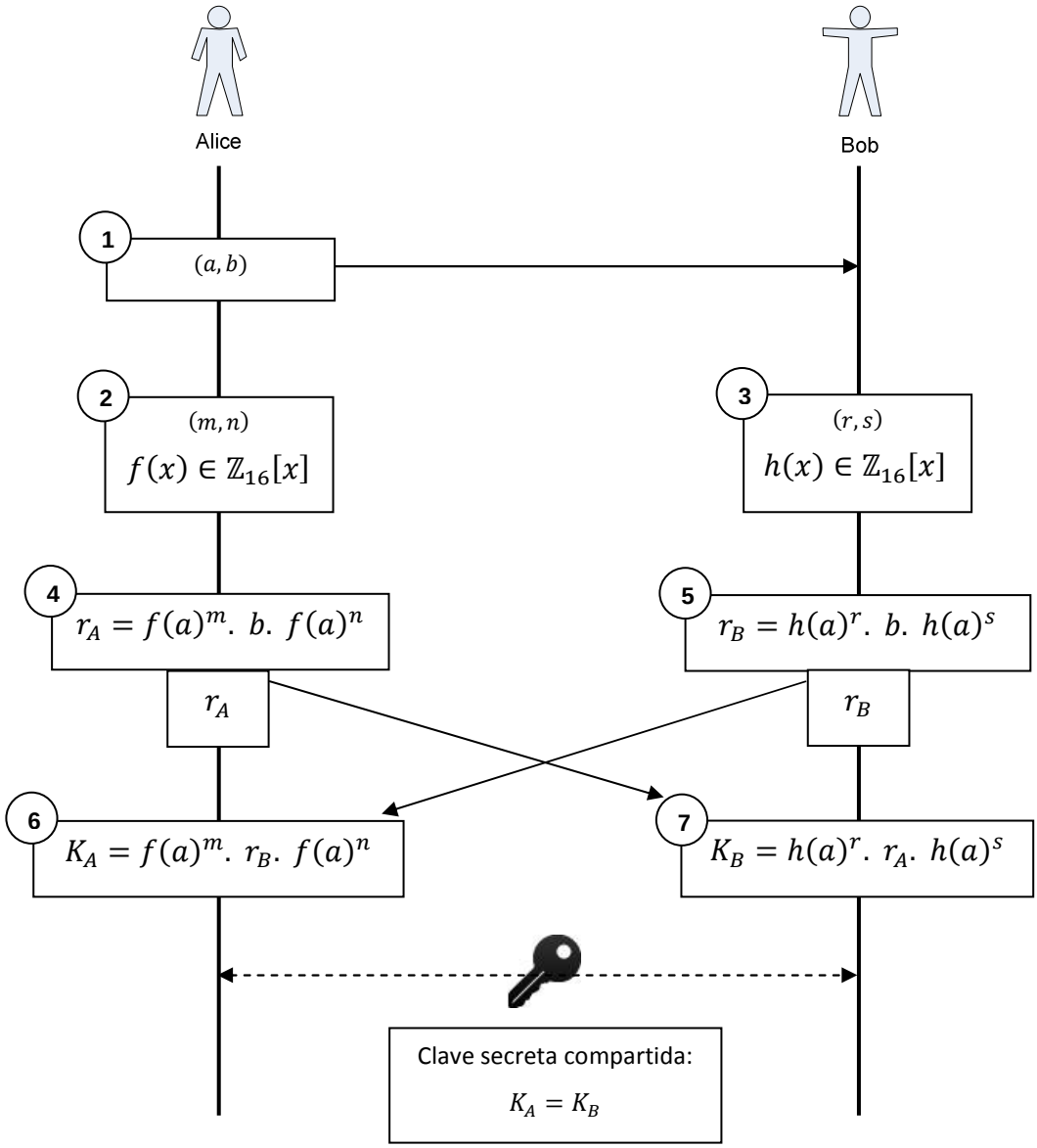


Figura 2: Pasos del Protocolo Compacto Diffie-Hellman

En este momento Alice y Bob tienen ya un secreto compartido, que un tercero no puede conocer, pese a haber tenido acceso a todo el intercambio de información.

CAPITULO 3 Implementación de DH-C

En este capítulo se detalla el software y hardware utilizado para la implementación de DH-C, así como la implementación del mismo. Para esto se ha elegido el lenguaje de programación JAVA, debido a que puede ser utilizado en un gran número de plataformas y dispositivos. Como se planteo en un inicio, uno de los principales problemas de otras implementaciones del protocolo de intercambio de claves es que requieren de costosas operaciones de cálculo y por lo tanto del uso de bibliotecas de precisión extendida, aquí se presenta una implementación que no requiere ninguna biblioteca de este tipo.

3.1 Software y Hardware utilizado

El siguiente software fue usado para la realización de este proyecto:

- Java JDK (Java Development Kit), versión: 1.6.0_24 para Windows x64[1].
- Eclipse IDE (Integrated Development Environment), versión: Helios SR2 - Eclipse Classic 3.6.2 (Windows 64-bits)[2].
- Android SDK (Software Development Kit), versión: Android SDK revisión 20.0.3[3].
- ADT (Android Development Tools) Plug-in para Eclipse, versión: 20.0.3[4].
- Android 2.3.3 Platform, API nivel 10, revisión 2.
- Servidor Openfire 3.7.1[5].

Para las pruebas se ha utilizado el siguiente hardware:

[1] http://www.oracle.com/technetwork/java/javase/downloads/index.html

[2] http://www.eclipse.org/downloads/packages/eclipse-classic-362/heliossr2

[3] http://developer.android.com/sdk/index.html

[4] http://dl-ssl.google.com/android/eclipse/

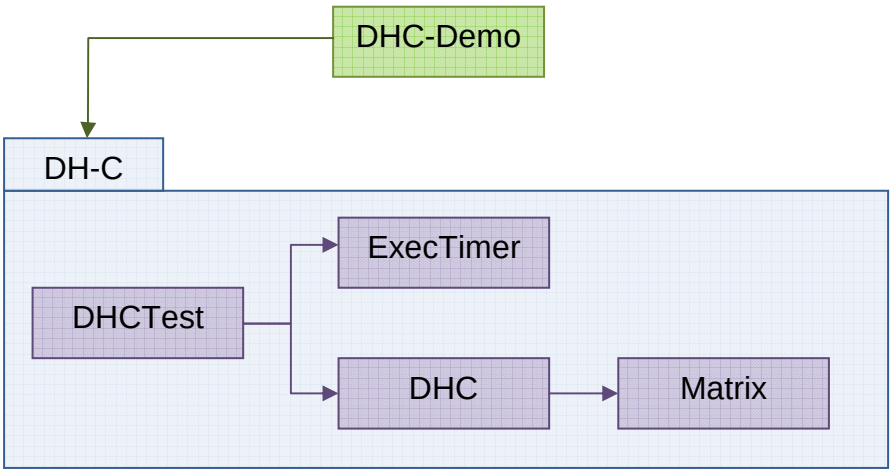
[5] http://www.igniterealtime.org/downloads/index.jsp

- Portátil Samsung (modelo NP300v4A-A0BVE):
  - Memoria RAM: 4GB
  - Procesador: Intel Core i5-2450M a 2,50GHz.
  - Disco Duro: 500GB
  - SO: Windows 7 Home Premium de 64bits.

- Smartphone Samsung Galaxy Y (modelo GT-S5360B):
  - Memoria Interna: 180MB
  - Almacenamiento Externo: MicroSD 2GB
  - Procesador: Broadcom 832 MHz
  - SO: Android versión 2.3.6 (Gingerbread)

3.2 Diffie-Hellman Compacto (DH-C)

La biblioteca desarrollada se llama DH-C, la cual, como se puede observar en la Figura 3 posee cuatro clases las cuales son detalladas más adelante. Esta biblioteca será la encargada de realizar todos los cálculos y de generar las claves secretas compartidas. DHC-Demo es una aplicación para el intercambio de mensajes de texto a través de internet para teléfonos móviles con sistema operativo Android en su versión 2.3.6. Esta aplicación utilizará la biblioteca DH-C para realizar las pruebas prácticas de esta implementación.



j

Figura 3: Biblioteca de Clases DH-C

3.2.1 Clase DHC

La clase DHC contiene la funcionalidad necesaria del protocolo Diffie-Hellman. A continuación se presenta una descripción de los principales métodos que componen esta clase.

- *Constructor DHC*, es el encargado de inicializar los valores aleatorios m, n, r, s, a, b. También crea el polinomio aleatorio que corresponde a la clave privada mediante la llamada al método *genPrivateKey.*

- *genPrivateKey*, este método es el encargado de generar un polinomio aleatorio, para esto utiliza dos métodos adicionales *randArray* y *sortPol.*

- *randArray*, este método genera un arreglo bidimensional con elementos aleatorios, donde cada par dentro de este arreglo corresponden a los coeficientes y exponentes de cada elemento del polinomio.

- *sortPol*, este método es el encargado de ordenar de mayor a menor los elementos del polinomio de acuerdo a sus exponentes.

- *genKey*, este método es el encargado de calcular $f(a)$ y r_A, para el cálculo de r_A se utiliza el método *genrA.*

- *genrA,* este método se encarga de realizar las operaciones necesarias para calcular: $f(a)^m.\ b.\ f(a)^n$ y $f(a)^m.\ r_B.\ f(a)^n$.

- *genSharedKey*, este método es el encargado de calcular la clave de sesión compartida K, para ello utiliza el método *genrA*.

3.2.2 Clase Matrix

- *Constructor Matrix*, es el encargado de inicializar los valores de los objetos *Matrix* creados, esta clase posee tres constructores, el primero, inicializa una matriz $n \times n$ con valores en cero, el segundo, inicializa una matriz $n \times n$ con los mismos valores de la matriz enviada como parámetro y el tercero inicializa una matriz $n \times n$ con valores aleatorios positivos menores a un valor enviado como parámetro.

- *identity,* este método devuelve un objeto tipo Matrix $n \times n$ que corresponde a la *Matriz Identidad*

- add, este método devuelve la suma de dos matrices $n \times n.$

- *multiply,* este método sobrecargado realiza dos operaciones de acuerdo a los parámetros enviados, el primero, devuelve el producto de dos matrices, y el segundo devuelve el producto de un número entero por una matriz.

- *copy,* este método devuelve una matriz con los mismos valores de una matriz enviada como parámetro

- *pow,* este método devuelve el resultado de elevar una matriz a una potencia enviada como parámetro.

- *eq*, este método determina si dos matrices son exactamente iguales.

- *det*, este método calcula el determinante de una matriz, el cual es usado para verificar si una matriz es nilpotente.

3.2.3 Clase DHCTest

La clase DHCTest tiene la función principal (main) que puede ser utilizada para probar y observar los resultados de los cálculos realizados durante el proceso de generación de la clave secreta compartida.

3.2.4 Clase ExecTimer

La clase ExecTimer es utilizada para calcular los tiempos utilizados para generar las claves compartidas secretas

3.2.5 DHC-Demo

DHC-Demo es una aplicación desarrollada en Android para el intercambio de mensajes de textos a través de internet para dispositivos celulares, esta aplicación utilizará la biblioteca DH-C para el intercambio de claves secretas, esto nos permitirá realizar pruebas prácticas de esta implementación.

CAPITULO 4 Seguridad de las Técnicas de Intercambio de Claves

La seguridad de muchas técnicas criptográficas depende de la intratabilidad de DLP. Entre estas técnicas podemos mencionar el Intercambio de Claves Diffie-Hellman, ElGamal y el Algoritmo de Firma Digital (DSA)

EL logaritmo discreto (DL) se lo define de la siguiente forma:

Definición 4. Sea G un grupo cíclico finito de orden n. Sea α un generador de G y sea $\beta \in G$. El logaritmo discreto de β en base α, definido por $log_{\alpha}\beta$, es el único entero x $(0 \leq x \leq n-1)$, tal que $\beta = \alpha^{x}$.

(DLP) Problema del Logaritmo Discreto: Dado un grupo cíclico finito G de orden n, un generador α de G y un elemento $\beta \in G$. Hallar el único entero x $(0 \leq x \leq n-1)$ tal que:

$$\alpha^{x} = \beta, \text{ es decir } x = \log_{\alpha}\beta$$

Nótese que conocidos α y x, se puede calcular $\alpha^{x} = \beta$ de manera eficiente, pero en cambio, conocidos α y β, encontrar x es un problema computacionalmente difícil.

En este capítulo se resumen los principales algoritmos conocidos para resolver este problema.

4.1 Principales Métodos de Ataque a DL en Campos Numéricos [5]

Los algoritmos existentes para el cálculo de logaritmos discretos, se los puede clasificar de la siguiente forma:

- Algoritmos que trabajan en grupos arbitrarios, como por ejemplo: Búsqueda Exhaustiva, Baby-Step Giant-Step o Algoritmo Rho de Pollard.

- Algoritmos que trabajan en grupos arbitrarios, pero que son especialmente eficientes si el orden del grupo tiene factores o divisores pequeños, como por ejemplo: Algoritmo Pohlig-Hellman

- Algoritmos que son eficientes solo en ciertos grupos, como por ejemplo: Algoritmo Index- Calculus.

4.1.1 Búsqueda Exhaustiva o Fuerza Bruta

La forma más elemental para el cálculo de DL consiste en hallar las sucesivas potencias del generador $\alpha^0, \alpha^1, \alpha^2, \ldots$ hasta encontrar el valor de β requerido. Sin embargo, el tiempo de cómputo de este algoritmo es excesivamente largo, por lo que resulta ineficiente si el tamaño del conjunto es grande, como lo son los casos de interés criptográfico.

4.1.2 Baby-Step Giant-Step

Este algoritmo es una mejora al método de búsqueda exhaustiva, mencionado anteriormente, y se basa en la siguiente observación:

Sea $m = \lceil \sqrt{n} \rceil$, donde n es el orden del grupo con el cual se está trabajando.

Si $\beta = \alpha^x$, entonces se puede escribir $x = im + j$, donde $0 \leq i, j < m$. Por lo tanto $\alpha^x = \alpha^{im}\alpha^j$, lo cual implica que $\beta(\alpha^{-m})^i = \alpha^j$. Esto sugiere el siguiente algoritmo para hallar x.

Algoritmo Baby-Step Giant-Step

Entrada: generador α de un grupo cíclico G de orden n, y un elemento $\beta \in G$.

Salida: logaritmo discreto $x = \log_{\alpha}\beta$.

1. Establecer $m \leftarrow \lceil \sqrt{n} \rceil$.
2. Construir una tabla con las entradas (j, α^j) para $0 \leq j < m$. Ordenar esta tabla por el segundo componente.
3. Calcular α^{-m} y establecemos $\gamma \leftarrow \beta$.
4. Para i desde 0 hasta $m-1$, hacemos lo siguiente:
   - **4.1.** Verificamos si γ coincide con el segundo componente de alguna de las entradas de la tabla inicial.
   - **4.2.** Si $\gamma = \alpha^j$, entonces devolver $(x = im + j)$.
   - **4.3.** $\gamma \leftarrow \gamma \cdot \alpha^{-m}$

Este algoritmo tampoco resulta ser eficiente ya que el cálculo de las entradas de la tabla, ordenar dicha tabla y hacer la comparación también requerirá de un tiempo exponencial.

4.1.3 Rho de Pollard

El algoritmo Rho de Pollard basa su ataque en el llamado "problema del cumpleaños[6]" y con el mismo tiempo de ejecución esperado del algoritmo Baby-Step Giant-Step, sin embargo es más eficiente por que necesita menos recursos de memoria. Por facilidad, se asumirá en esta sección que G es un grupo cíclico cuyo orden n es primo.

[6] Este problema establece cuantas personas necesitan estar reunidas para tener un 50%de posibilidades de una colisión, es decir, de que al menos dos de ellas cumplan años el mismo día. Si las personas son elegidas al azar de una distribución uniforme, la respuesta es 23, que es aproximadamente $\sqrt{366}$.

El grupo G es dividido en tres conjuntos S_1, S_2 y S_3 de aproximadamente el mismo tamaño. Se define una secuencia de elementos de los grupos $x_0, x_1, x_2, \ldots$ para $x_0 = 1$ y

$$x_{i+1} = \begin{cases} \beta \cdot x_i, & if\ x_i \in S_1, \\ x_i^2, & if\ x_i \in S_2, \\ \alpha \cdot x_i, & if\ x_i \in S_3, \end{cases} \tag{4.1}$$

para $i \geq 0$, esta secuencia de elementos del grupo a su vez define dos secuencias de enteros $a_0, a_1, a_2, \ldots$ y $b_0, b_1, b_2, \ldots$ tal que $x_i = \alpha^{a_i}\beta^{b_i}$ para $i \geq 0$, con $a_0 = 0,\ b_0 = 0$ y

$$a_{i+1} = \begin{cases} a_i, & if\ x_i \in S_1, \\ 2a_i \bmod n, & if\ x_i \in S_2, \\ a_i + 1 \bmod n, & if\ x_i \in S_3, \end{cases} \tag{4.2}$$

y

$$b_{i+1} = \begin{cases} b_i + 1 \bmod n, & if\ x_i \in S_1, \\ 2b_i \bmod n, & if\ x_i \in S_2, \\ b_i, & if\ x_i \in S_3, \end{cases} \tag{4.3}$$

para $i \geq 0$. Por lo tanto $\alpha^{a_i}\beta^{b_i} = \alpha^{a_{2i}}\beta^{b_{2i}}$, entonces $\beta^{b_i - b_{2i}} = \alpha^{a_{2i} - a_i}$. Aplicando el logaritmo base α a ambos lados de esta última ecuación tenemos:

$$(b_i - b_{2i}) \cdot \log_\alpha \beta \equiv (a_{2i} - a_i) (\bmod n)$$

Algoritmo Rho de Pollard

Entrada: generador α de un grupo cíclico G de orden primo n, y un elemento $\beta \in G$.

Salida: logaritmo discreto $\mathrm{x} = \log_\alpha \beta$.

1. Establecer $x_0 \leftarrow 1,\ a_0 \leftarrow 0,\ b_0 \leftarrow 0$.

2. Para $i = 1, 2, \ldots$ hacemos lo siguiente:

2.1. Con los valores x_{i-1}, a_{i-1}, b_{i-1} y x_{2i-2}, a_{2i-2}, b_{2i-2} hallados anteriormente, calculamos x_i, a_i, b_i y x_{2i}, a_{2i}, b_{2i} usando las ecuaciones (4.1), (4.2) y (4.3).

2.2. Si $x_i = x_{2i}$, hacer los siguiente:

Establecer $r \leftarrow b_i - b_{2i} \bmod n$.

Si $r = 0$, terminar el algoritmo con error, caso contrario calcular $x = r^{-1}(a_{2i} - a_i) \bmod n$, y devolver (x).

4.1.4 Pohlig-Hellman

Este algoritmo es especialmente eficiente en grupos para los que se conoce la factorización de su orden y ésta tiene factores o divisores pequeños. Sea $n = p_1^{e_1} p_2^{e_2} \ldots p_r^{e_r}$ la descomposición en factores primos de n, donde n es el orden del grupo G. Si $x = \log_\alpha \beta$, entonces el objetivo es determinar $x_i = x \bmod p_i^{e_i}$ para $1 \leq i \leq r$, y luego usar el algoritmo de Gauss para obtener $x \bmod n$. Cada entero x_i es determinado mediante el cálculo de los dígitos $l_0, l_1, \ldots, l_{e_i-1}$ en términos de su representación p_i:

$$x_i = l_0 + l_1 p_i + \cdots + l_{e_i-1} p_i^{e_i-1}, \text{ donde } 0 \leq l_j \leq p_i - 1$$

Algoritmo Pohlig-Hellman

Entrada: generador α de un grupo cíclico G de orden n, y un elemento $\beta \in G$.

Salida: logaritmo discreto $x = \log_\alpha \beta$.

1. Encontrar los factores primos de n: $n = p_1^{e_1} p_2^{e_2} \ldots p_r^{e_r}$, donde $e_i \geq 1$.

2. Para i desde 1 hasta r, hacemos lo siguiente:

(*Calculamos* $x_i = l_0 + l_1 p_i + \cdots + l_{e_i-1} p_i^{e_i-1}$, *donde* $x_i = x \bmod p_i^{e_i}$)

2.1. (*Simplificamos la* notación) Establecemos $q \leftarrow p_i$ y $e \leftarrow e_i$.

2.2. Establecemos $\gamma \leftarrow 1$ y $l_{-1} \leftarrow 0$.

2.3. Calculamos $\bar{\alpha} \leftarrow \alpha^{n/q}$.

2.4. (*Calculamos* l_j) Para j desde 0 hasta $e-1$, hacemos lo siguiente:

Calculamos $\gamma \leftarrow \gamma \alpha^{l_{j-1} q^{j-1}}$ y $\bar{\beta} \leftarrow (\beta \gamma^{-1})^{n/q^{j+1}}$

Calculamos $l_j \leftarrow \log_{\bar{\alpha}} \bar{\beta}$

2.5. Establecemos $x_i \leftarrow l_0 + l_1 q + \cdots + l_{e_i-1} q^{e-1}$

3. Usar el algoritmo de Gauss para calcular el entero $x, 0 \leq x \leq n-1$, tal que $x = x_i (\bmod p_i^{e_i})$ para $1 \leq i \leq r$.

4. Devolvemos (x).

4.1.5 Index-Calculus

Este algoritmo es el método más poderoso conocido para el cálculo de logaritmos discretos. Sin embargo, esta técnica no es aplicable para todos los grupos, pero cuando lo es (como es caso de los grupos $\mathbb{Z}_p^*$ y $\mathbb{F}_{2^m}^*$), es a menudo un algoritmo de tiempo subexponencial.

Este algoritmo requiere la selección de un subconjunto S relativamente pequeño de elementos de G, llamado *base de factores*, de tal forma que una fracción significativa de los elementos de G pueda ser expresada eficientemente como productos de los elementos de S.

El algoritmo presentado a continuación está incompleto por dos razones: la primera, no se indica una técnica para la selección de la base de

factores S, la segunda, no se especifica un método para generar de manera eficiente las relaciones de la forma (4.4) y (4.6).

Algoritmo Index-Calculus

Entrada: generador α de un grupo cíclico G de orden n, y un elemento $\beta \in G$.

Salida: logaritmo discreto $y = \log_{\alpha}\beta$.

1. (*Seleccionar la base de factores* S) Escogemos un subconjunto $S = \{p_1, p_2, \ldots, p_t\}$ de G, de tal manera que una "proporción significativa" de todos los elementos en G se puedan expresar de manera eficiente como un producto de elementos de S.

2. (*Hallar las relaciones lineales que involucran logaritmos de los elementos de* S).

2.1. Seleccionamos un entero aleatorio k, $0 \leq k \leq n-1$, y calculamos α^k.

2.2. Tratamos de escribir α^k como un producto de elementos en S:

$$\alpha^k = \prod_{i=1}^{t} {p_i}^{c_i}, \quad c_i \geq 0. \tag{4.4}$$

Si tenemos éxito, aplicamos el logaritmo en ambos lados de la ecuación (4.4) para obtener una relación lineal:

$$k = \sum_{i=1}^{t} c_i \log_{\alpha} p_i \pmod{n}. \tag{4.5}$$

2.3. Repetimos los pasos 2.1 y 2.2 hasta que se obtengan $t + c$ relaciones de la forma (4.5) (c es un número entero positivo pequeño, por ejemplo, $c = 10$, de tal manera que el sistema de ecuaciones dadas por las relaciones $t + c$ tiene una solución única con una alta probabilidad).

3. (*Encontrar los logaritmos de los elementos de* S) Trabajando modulo n, resolvemos el sistema lineal de $t + c$ ecuaciones (con t incógnitas) de la forma (4.5) halladas en el paso 2 para obtener los valores de $\log_\alpha p_i$, $1 \leq i \leq t$.

4. (*Calculamos* y)

4.1. Seleccionamos un entero aleatorio k, $0 \leq k \leq n-1$, y calculamos $\beta \cdot \alpha^k$.

4.2. Tratamos de escribir $\beta \cdot \alpha^k$ como el producto de elementos de S:

$$\beta \cdot \alpha^k = \prod_{i=1}^{t} {p_i}^{d_i}, \quad d_i \geq 0. \tag{4.6}$$

Si el intento no tiene éxito repetimos el paso 4.1. Caso contrario, aplicamos el logaritmo en ambos lados de la ecuación (4.6) obteniendo $\log_\alpha \beta = (\sum_{i=1}^{t} d_i \log_\alpha p_i - k) \bmod n$; de esta forma calculamos $y = (\sum_{i=1}^{t} d_i \log_\alpha p_i - k) \bmod n$ y devolvemos (y).

4.2 Seguridad Computacional del Protocolo DH-C [6]

Como señala Hecht [6] en su trabajo, basados en la dificultad computacional de los problemas mostrados (en la sección 2.3.3) SDP y PDH y en una adecuada selección del tamaño de las claves privadas, podemos decir que este protocolo DH-C será potencialmente inmune a ataques.

Es importante tomar en cuenta que la seguridad de este protocolo se basa exclusivamente en el grado y coeficientes de los polinomios y en el orden de los parámetros (m, n) y (r, s) que elijan cada una de las partes, es por esto que no tiene sentido incrementar el orden de las matrices (a, b) para aportar un mayor nivel de seguridad al protocolo ya que estos elementos son públicos.

En cuanto al grado y coeficientes de los polinomios, tenemos que considerar que hay $O(m^2)$ polinomios enteros $\mathbb{Z}_m[x]$ de grado $\leq m$ y coeficientes en $\mathbb{Z}_m$, por lo que para poder utilizar este protocolo en aplicaciones prácticas, como por ejemplo transferencias bancarias electrónicas que son transacciones de vida útil breve, sería necesario trabajar por lo menos con polinomios $\mathbb{Z}_{256}[x]$, en este caso el número de pares de claves privadas será de $O(2^{32})$. En aplicaciones en las que se requiera que la información sea protegida por un lapso de tiempo mucho mayor, se podría operar con claves privadas del orden $\mathbb{Z}_{4096}[x]$, en cuyo caso la complejidad del ataque será $O(2^{48})$ con lo cual se lograría un nivel de seguridad computacional aceptable.

A la fecha de realización del presente trabajo no se ha podido encontrar datos publicados que nos permitan construir una tabla de tamaños de claves de seguridad equivalentes entre el protocolo expuesto en este trabajo DH-C y otros protocolos convencionales como son: DH y ECDH.

CAPITULO 5 Evaluación y Pruebas de DH-C

Este capítulo está organizado en dos secciones, en la primera 5.1 se describen las pruebas realizadas sobre la aplicación DHC-Demo y en la segunda 5.2 se muestran los resultados obtenidos sobre el protocolo DH-C en lo referente a tiempos de ejecución y consumo de memoria.

5.1 Pruebas de la aplicación DHC-Demo

En esta sección se mostrará paso a paso la ejecución de la aplicación DHC-Demo, para lo cual empezaremos levantando dos instancias de la aplicación (para fines explicativos las denominaremos Alice y Bob). Una vez levantadas las instancias podremos ver en cada una de ellas la pantalla de inicio de sesión:

Alice: Bob:

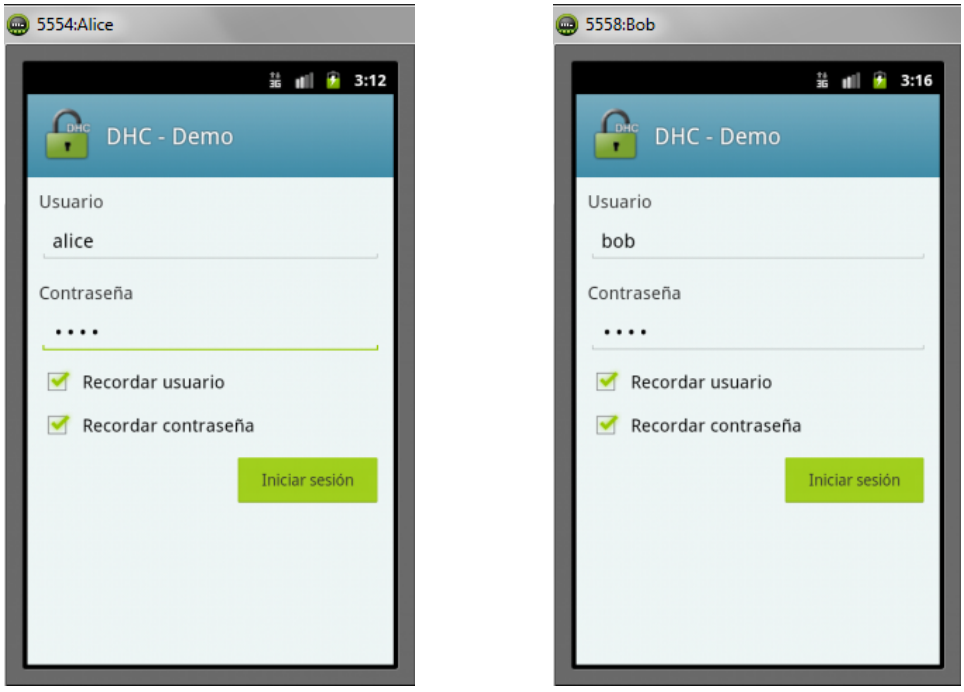


Figura 4: Instancias de la aplicación. Alice y Bob.

Después de haber ingresado los datos requeridos (Nombre de Usuario y Contraseña, adicionalmente se puede activar las opciones para recordar el usuario y contraseña) procedemos a iniciar sesión (para mayor información sobre la configuración de Servidor o Creación de cuentas ver Anexo A.1). Si no hubo problemas en el inicio de sesión se mostrará la lista de contactos correspondiente a cada usuario.

Alice: Bob:

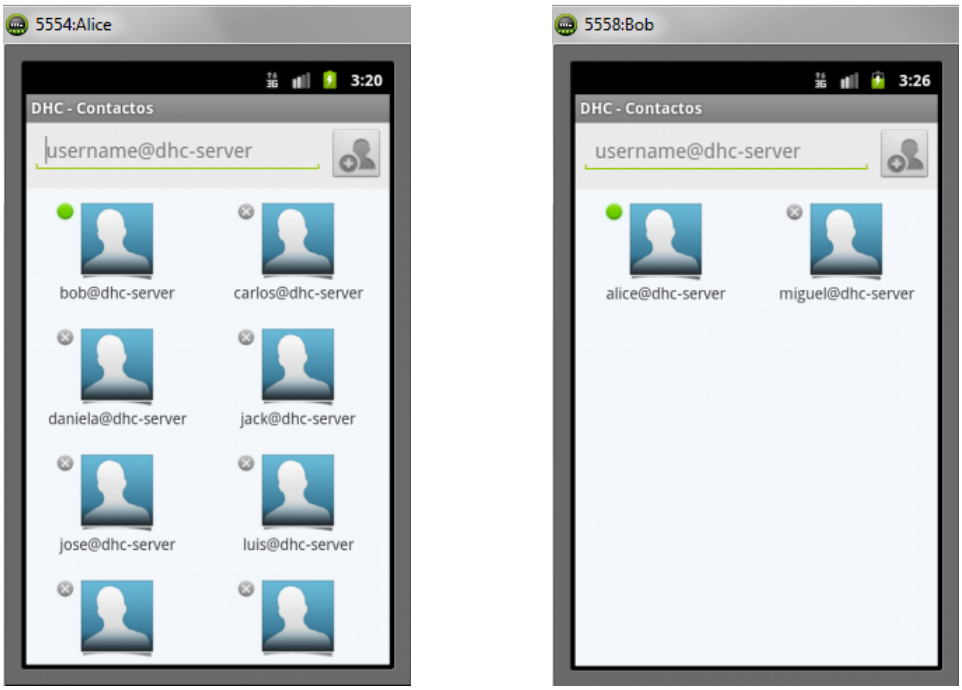


Figura 5: Lista de Contactos Alice y Bob

En este instante Alice enviará un mensaje a Bob, para esto seleccionamos a bob@dhc-server de la lista de contactos de Alice y enviamos el mensaje deseado, en el lado de Bob se abrirá la pantalla de chat con el mensaje de Alice:

Alice: Bob:

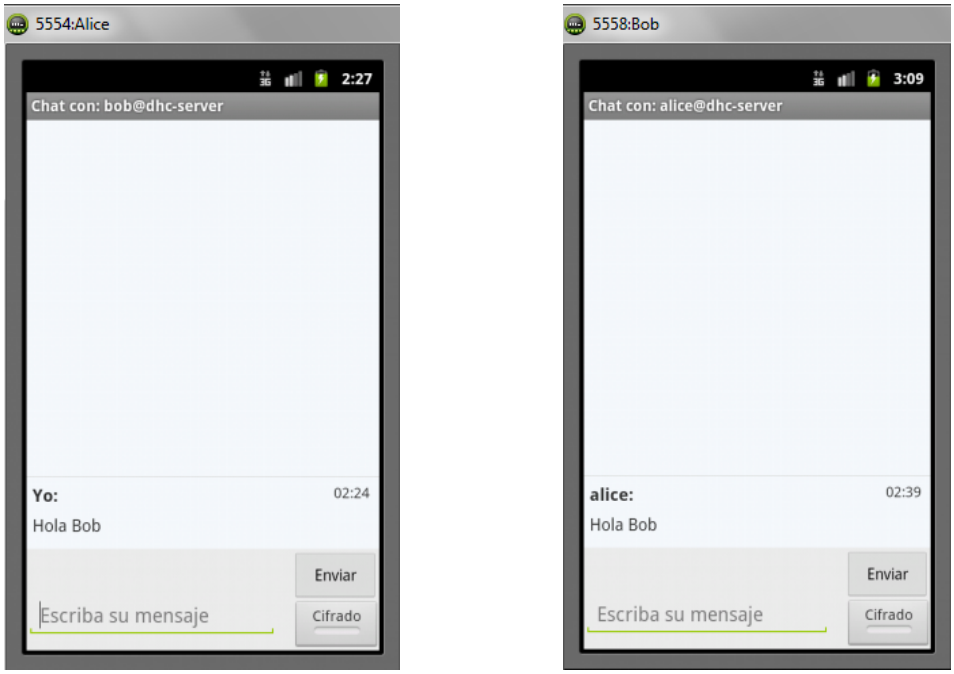


Figura 6: Alice envía mensaje de texto a Bob.

A continuación Bob responde a Alice con el siguiente mensaje:

Alice: Bob:

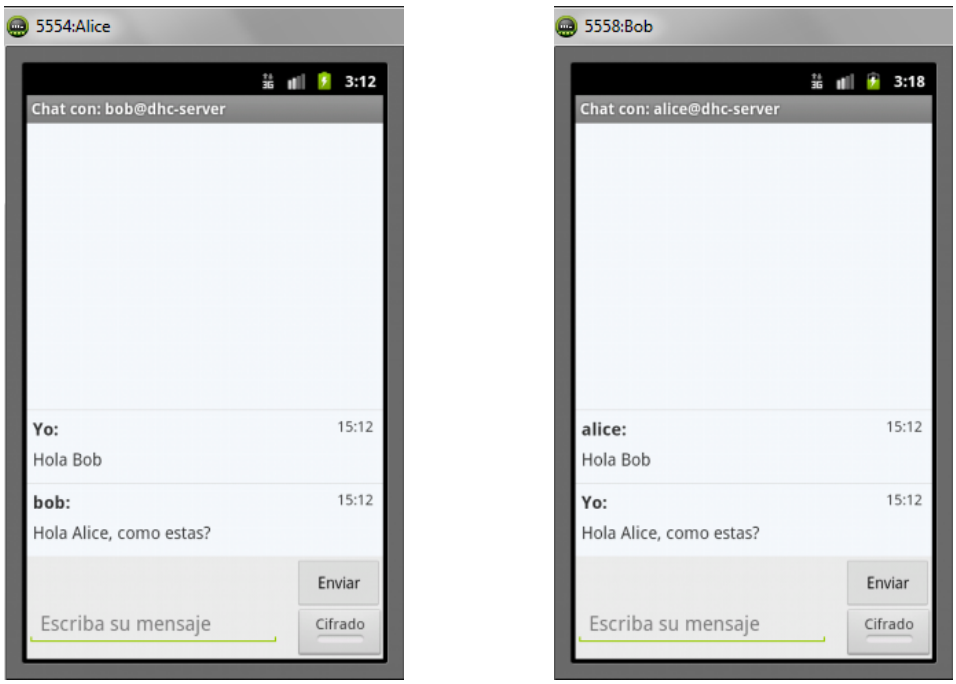


Figura 7: Bob responde mensaje de texto a Alice

Ahora, Alice desea establecer una comunicación segura con Bob, para ello presiona el botón “Cifrado” para activarlo, esto hará que se inicie el intercambio de claves, como paso inicial se envía una “Solicitud de cifrado” a Bob, la cual podrá ser aceptada o rechazada por él:

Alice: Bob:

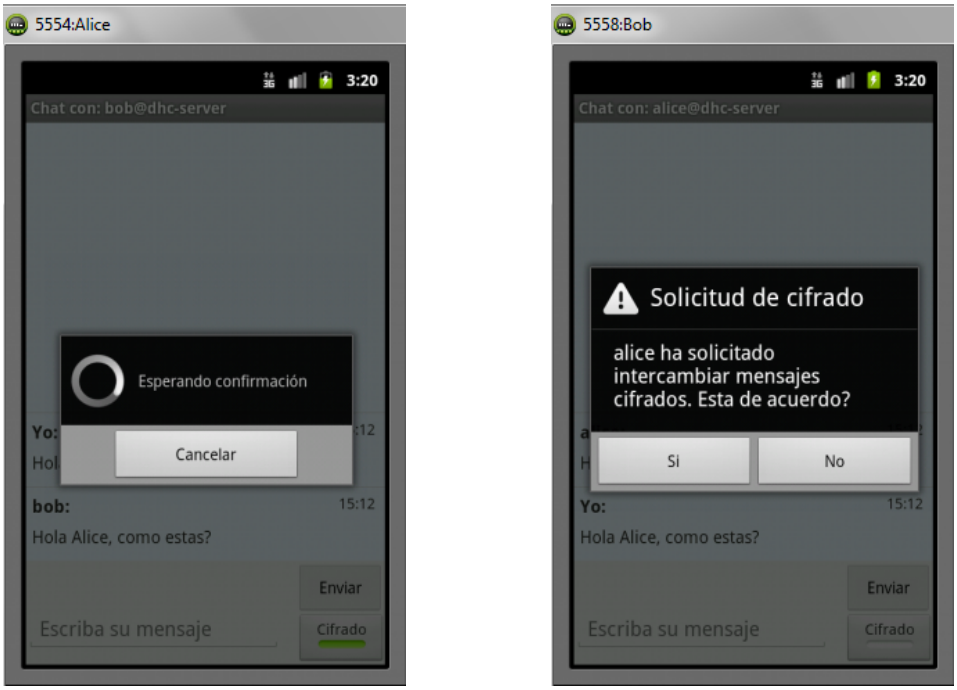


Figura 8: Solicitud de cifrado

Una vez que Bob acepte la solicitud se procederá con el intercambio de claves entre Alice y Bob para poder cifrar los mensajes. Este proceso de intercambio de claves realiza los siguientes pasos:

1. Alice: Si recibe una respuesta positiva a la Solicitud de Cifrado de Bob, genera sus valores aleatorios: m, n, a y b, y a continuación envía a Bob las matrices a y b:

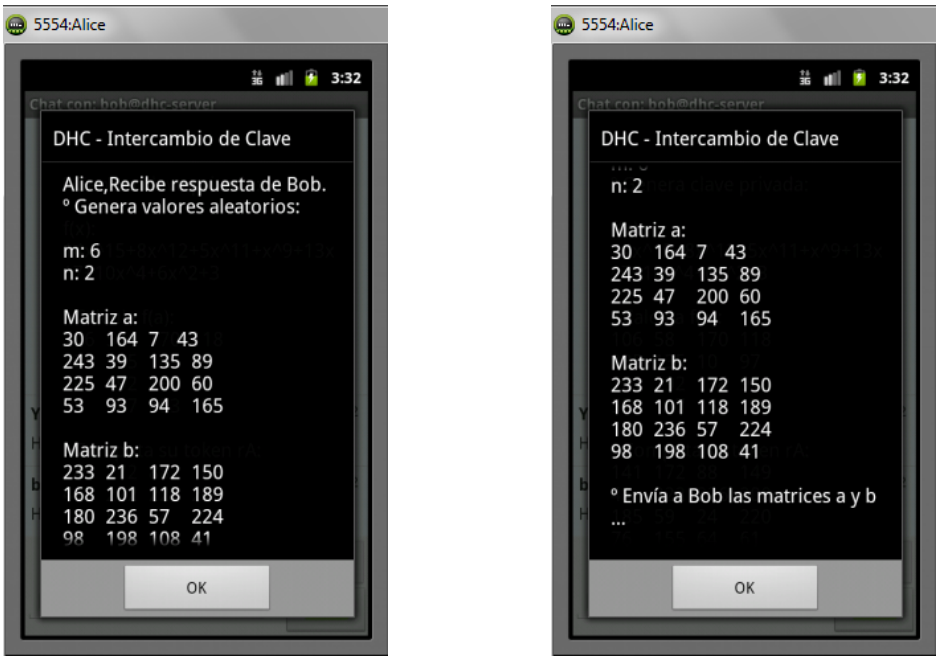


Figura 9: DH-C Paso 1

2. Alice: Una vez que envía las matrices a y b a Bob, procede a generar su clave privada, calcula $f(a)$ y computa su token r_A:

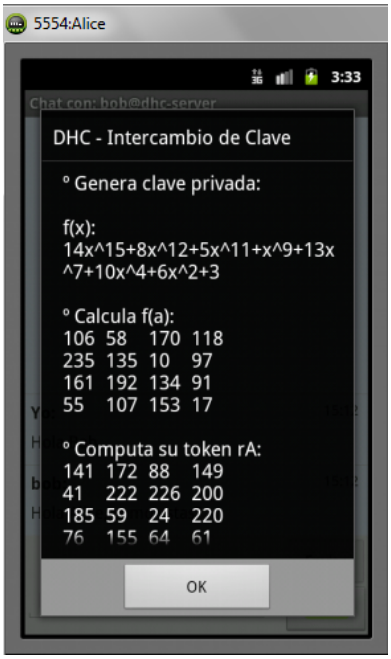


Figura 10: DH-C Paso 2 y 4

3. Bob: Recibe las matrices a y b enviadas por Alice, procede a generar sus valores aleatorios r y s, genera su clave privada, calcula $h(a)$,

computa su token r_B y lo envía a Alice:

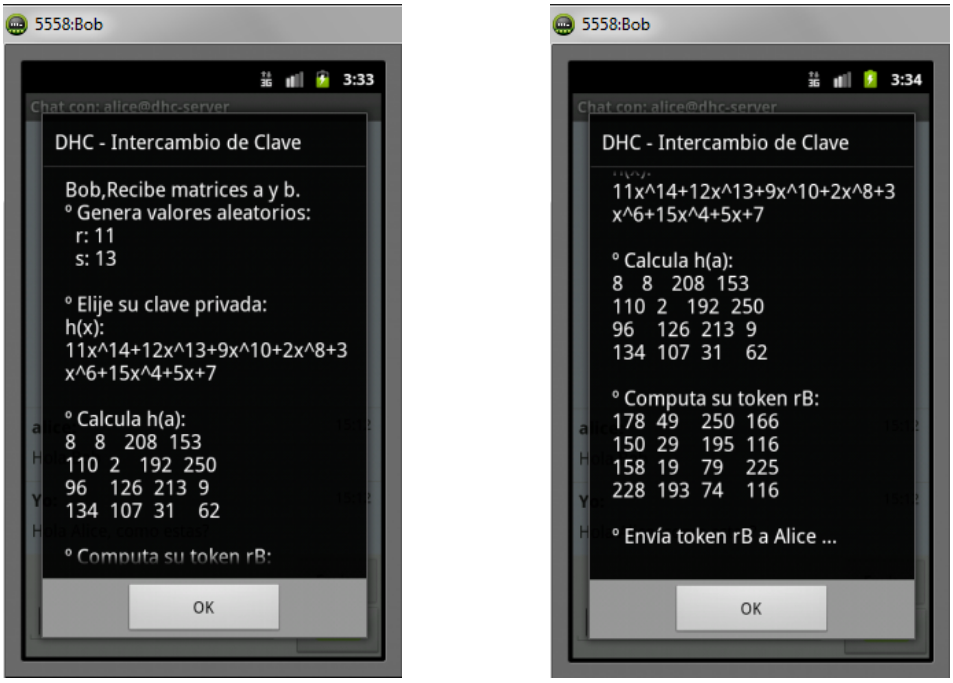


Figura 11: DH-C Paso 3 y 5

4. Alice y Bob: Intercambian sus tokens r_A y r_B respectivamente, y calculan la clave secreta compartida $K = K_A = K_B$:

Figura 12: DH-C Paso 6 y 7

Una vez intercambiada la clave se utiliza un cifrado simétrico AES 256 para cifrar los mensajes a enviar, para esto se usa la clave previamente calculada (para almacenar la clave compartida en el dispositivo se usa una función hash SHA2). Una vez activado el cifrado, Alice y Bob pueden intercambiar mensajes de forma segura. Visualmente ambos usuarios ven el texto en claro de los mensajes, pero como se verá más adelante los mensajes viajan cifrados por el canal público:

Alice: Bob:

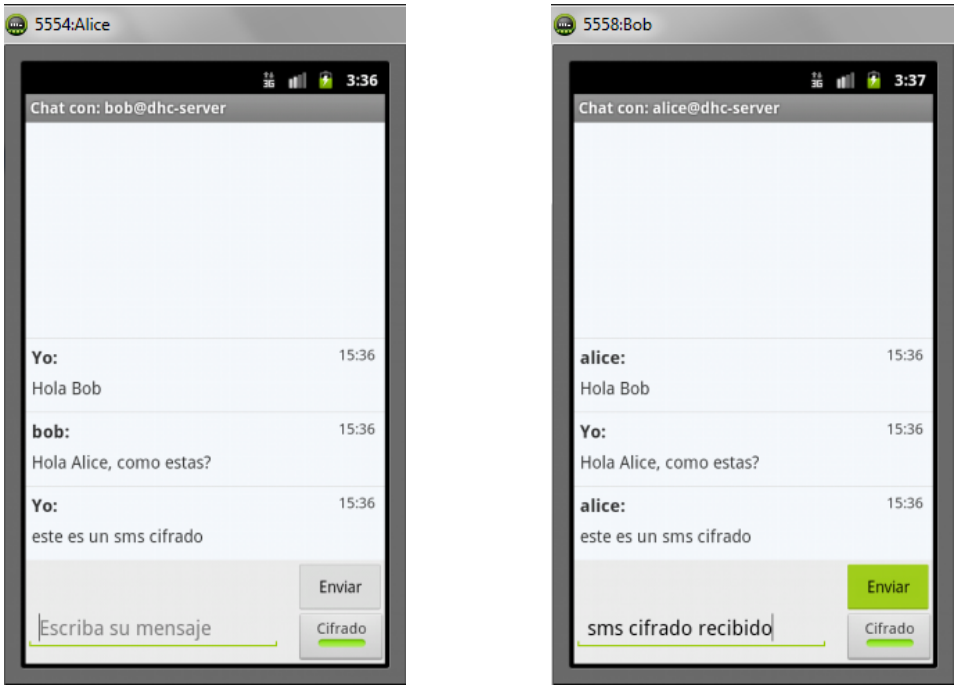


Figura 13: Intercambio de mensajes cifrados

Para desactivar o terminar con el cifrado de mensajes, será necesario que cualquiera de las partes: Alice o Bob presione el botón “Cifrado” para desactivarlo. A continuación Alice desactiva el cifrado y envía el siguiente mensaje a Bob: este es un sms no cifrado.

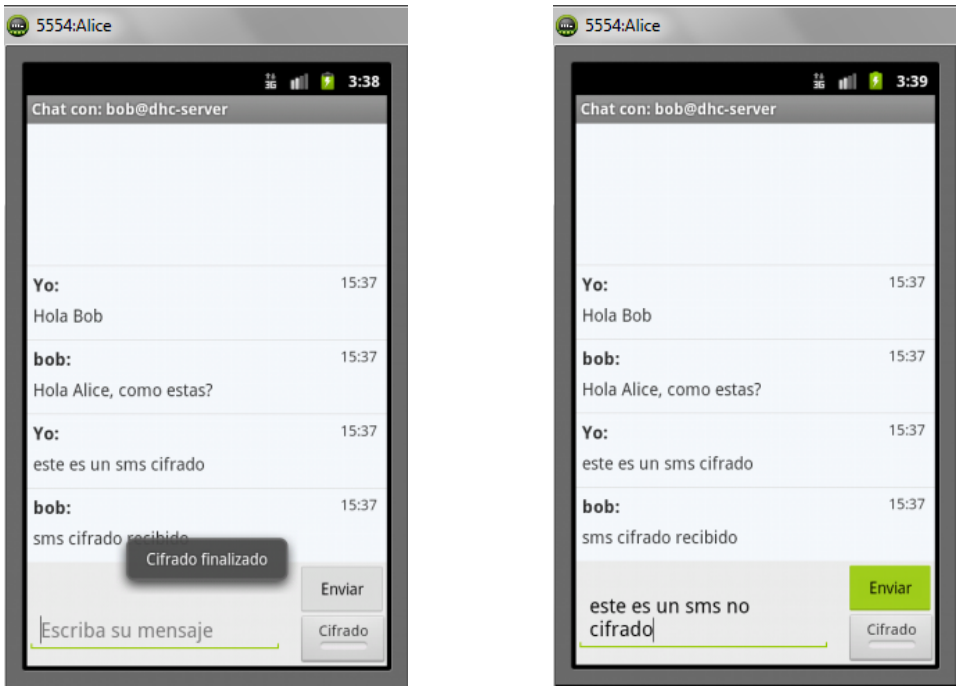


Figura 14: Desactivación de cifrado

Una vez desactivado el cifrado Bob recibe el mensaje y lo responde con el mensaje: sms no cifrado recibido.

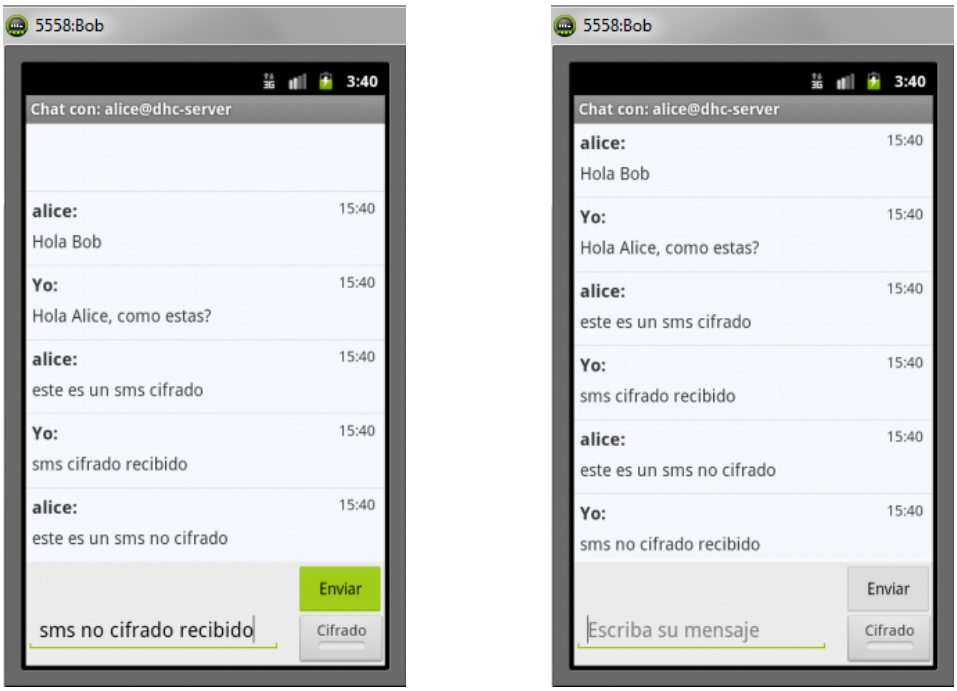


Figura 15: Intercambio de mensajes no cifrados

Todos los mensajes intercambiados a través del servidor Openfire pueden ser registrados, para nuestras pruebas esto es de gran utilidad ya que nos permite identificar como se está realizando el intercambio de claves y poder observar el cifrado de mensajes entre las partes:

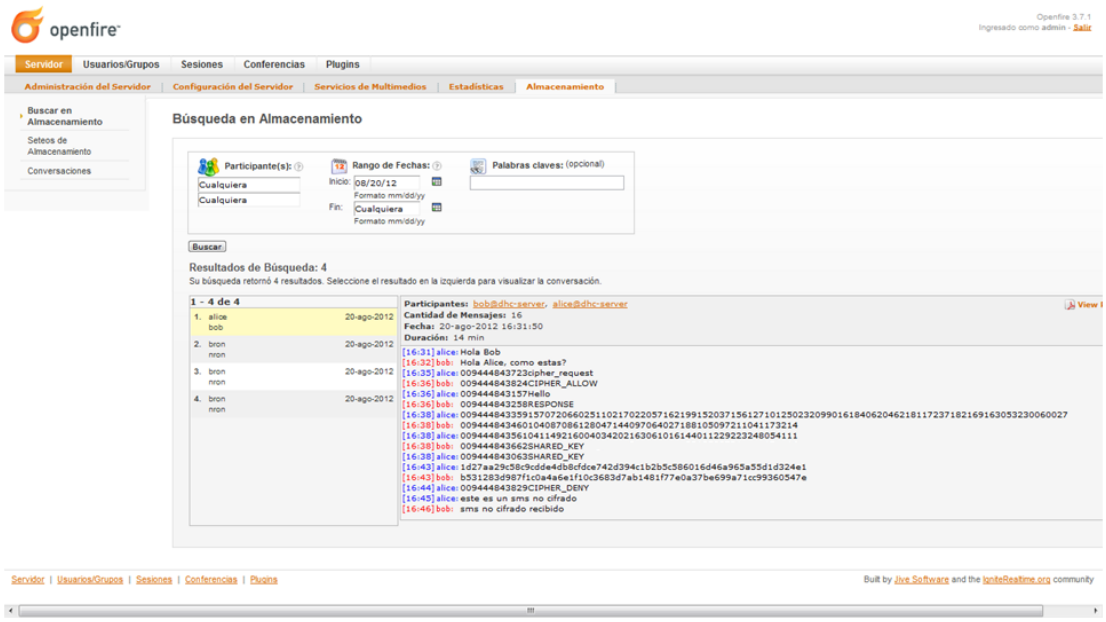


Figura 16: Mensajes en servidor Openfire

A continuación se detallarán cada uno de los mensajes intercambiados entre Alice y Bob, los cuales se muestran en la Figura 16:

- Alice y Bob intercambian mensajes sin cifrar.

```
[16:31]  alice:  Hola Bob
[16:32]  bob:    Hola Alice, como estas?
```

- Alice envía solicitud de cifrado.

```
[16:35]  alice:  009444843723CIPHER_REQUEST
```

- Bob acepta la solicitud de cifrado.

```
[16:36]  bob:    009444843824CIPHER_ALLOW
```

- Una vez que Bob acepta la solitud, Alice envía mensaje inicial.

```
[16:36]  alice:  009444843157HELLO
```

- Bob responde al mensaje inicial de Alice.

```
[16:36]  bob:   009444843258RESPONSE
```

- Una vez que Alice recibe respuesta de Bob, le envía matrices A y B.

```
[16:38]  alice: 009444843359157072066025110217022057162199152037
                156127101250232099016184062046218117237182169163
                053230060027
```

- Bob computa r_B y se lo envía a Alice.

```
[16:38]  bob:   009444843460104087086128047144097064027188105097
                211041173214
```

- Alice computa r_A y se lo envía a Bob.

```
[16:38]  alice: 009444843561041149216004034202163061016144011229
                223248054111
```

- Bob calcula la clave compartida y envía mensaje de confirmación a Alice.

```
[16:38]  bob:   009444843662SHARED_KEY
```

- Alice calcula la clave compartida y envía mensaje de confirmación a Bob.

```
[16:38]  alice: 009444843063SHARED_KEY
```

- Alice envía a Bob el mensaje cifrado: este es un sms cifrado.

```
[16:43]  alice: 1d27aa29c58c9cdde4db8cfdce742d394c1b2b5c586016d4
                6a965a55d1d324e1
```

- Bob envía a Alice el mensaje cifrado: sms cifrado recibido.

```
[16:43]  bob:   b531283d987f1c0a4a6e1f10c3683d7ab1481f77e0a37be6
                99a71cc99360547e
```

- Alice termina cifrado de mensajes.

```
[16:44]  alice: 009444843829CIPHER_DENY
```

- Alice y Bob intercambian mensajes sin cifrar.

```
[16:45]  alice: este es un sms no cifrado
[16:46]  bob:   sms no cifrado recibido
```

5.2 Evaluación del protocolo DH-C

A continuación se muestran los resultados obtenidos de la evaluación del protocolo de intercambio de claves DH-C, para ello se ha tomado como factores a evaluar el consumo de memoria y el tiempo de ejecución utilizados en el intercambio de claves entre dos entes.

Se utilizó el dispositivo móvil indicado en la sección 3.1 para obtener los valores que se muestran en esta sección, realizando cientos de intercambios de claves para de esta forma obtener la media aritmética y calcular un tiempo promedio.

5.2.1 Tiempo de Ejecución

Es importante mencionar que para el tiempo de ejecución solo se tomó en cuenta las operaciones criptográficas en cada una de las partes involucradas. Para llevar a cabo estas mediciones se utilizó la clase *TimingLogger* [7] del API de Android. Se utilizó esta clase debido a que nos permite obtener tiempos de ejecución de secciones específicas dentro de nuestro código. El tiempo es medido en milisegundos.

Protocolo de Negociación	Tiempo (ms.)
DH-C 256	258
DH-C 512	539
DH-C 1024	1291
DH-C 2048	3056
DH-C 3072	5180
DH-C 4096	6997

Tabla 1: Tiempos estimados del protocolo DH-C.

[7] Una mayor información sobre el uso de esta clase se encuentra en: http://developer.android.com/reference/android/util/TimingLogger.html

La Tabla 1 muestra los tiempos de ejecución obtenidos para DH-C de acuerdo a su longitud de clave. A continuación se muestra una gráfica de los tiempos de la Tabla 1.

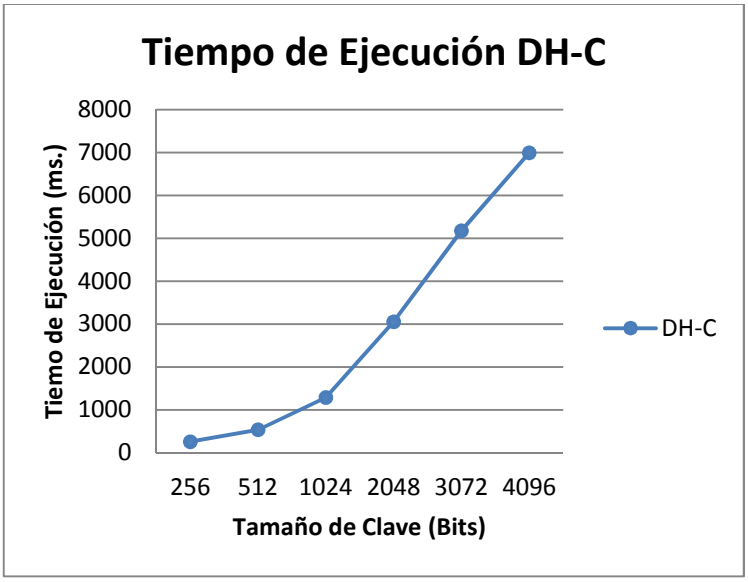


Figura 17: Tiempos de ejecución DH-C.

5.2.1 Uso de Memoria

Para estas mediciones solo se tomó en cuenta las operaciones criptográficas en cada una de las partes involucradas, sin considerar el uso de memoria requerida por la aplicación misma.

La gestión de memoria en Android es un tema complejo, debido a que la memoria en este tipo de sistemas es compartida entre varios procesos, por lo que resulta un reto medir el uso de memoria de un proceso individual, tomando en cuenta que adicionalmente se requiere realizar estas mediciones en tiempos específicos dentro del programa.

La herramienta en línea de comandos utilizada para este propósito fue “dumpsys meminfo”, la cual está basada en Linux y nos muestra el uso de memoria de una determinada aplicación. Para el uso de este comando es necesario ejecutar la aplicación sobre el dispositivo móvil, para esto se debe

conectar este dispositivo al computador mediante un cable USB y hacer uso de la herramienta ADB[8]:

adb –s <dispositivo> shell dumpsys meminfo <proceso>

Donde, <dispositivo> corresponde al código del dispositivo móvil conectado (este código se lo puede obtener ejecutando el comando: adb devices) y <proceso> corresponde al PID del proceso.

Protocolo de Negociación	Uso de Memoria (Mb.)
DH-C 256	0,28
DH-C 512	0,4
DH-C 1024	0,48
DH-C 2048	0,61
DH-C 3072	0,78
DH-C 4096	0,88

Tabla 2: Uso de memoria estimada del protocolo DH-C.

La Tabla 2 muestra el uso de memoria para DH-C de acuerdo a su longitud de clave. A continuación se muestra una gráfica de los tiempos de la Tabla 2.

[8] ADB (Android Debug Bridge). Herramienta que permite comunicarnos y ejecutar aplicaciones sobre el dispositivo Android por medio de línea de comandos.

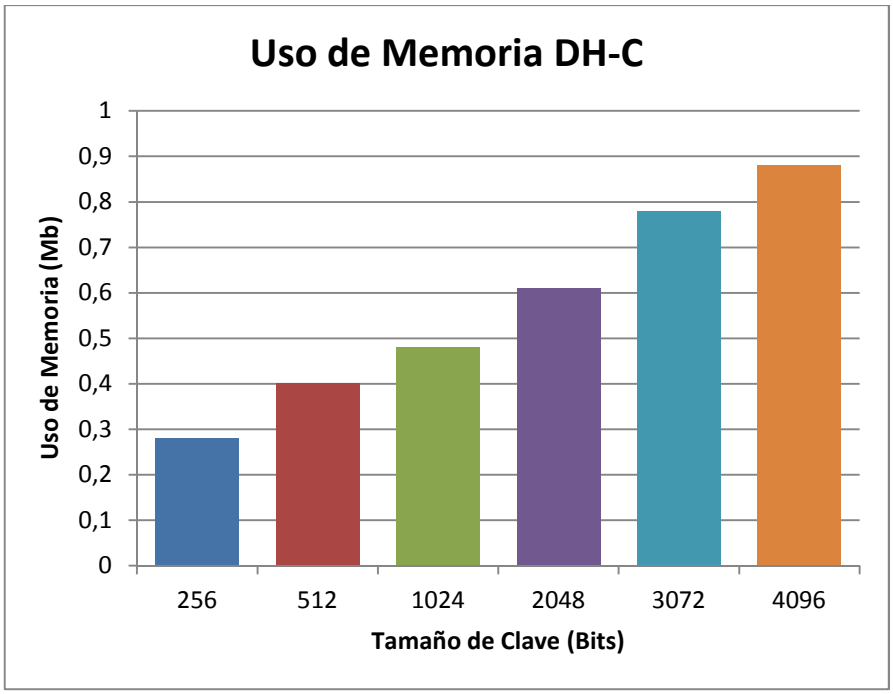


Figura 18: Uso de memoria DH-C.

CAPITULO 6 Conclusiones y Trabajos Futuros

Este capítulo está dividido en dos secciones. La sección 6.1 resume las contribuciones principales de esta tesis y la evaluación de los objetivos. La sección 6.2 muestra los trabajos futuros para nuevas líneas de investigación.

6.1 Conclusiones

- Se desarrolló una implementación del protocolo propuesto por Hecht, que puede ser utilizada en dispositivos con poder computacional limitado y que no hace uso de bibliotecas de precisión extendida, las cuales son utilizadas en otras implementaciones del protocolo de intercambio de claves y que requieren de costosas operaciones de cálculo.

- Se realizó una mejora de seguridad al algoritmo original del protocolo DH-C, la Generalización de Exponentes (sección 2.4.2), gracias a la cual los 2 valores (m, n) que eran componentes de la clave pública se reemplazaron por 4 valores privados: (m, n) y (r, s). Como consecuencia de esto se incrementa el nivel de seguridad del protocolo, ya que un atacante tendrá que trabajar ya no solo sobre los polinomios sino también sobre los exponentes secretos.

- Los criptosistemas tradicionales se basan en el problema del logaritmo discreto (DLP) o en el problema de la factorización de enteros (IFP) y operan en estructuras conmutativas, para los cuales se conocen ataques de complejidad subexponencial en el tiempo. Por esto es interesante ver cómo en los últimos tiempos la criptografía no conmutativa se está usando exitosamente en pruebas de conocimiento cero, firma digital y como en este caso para intercambio de claves.

- No fue posible construir una tabla de tamaños de claves de seguridad equivalentes entre el protocolo expuesto en este trabajo DH-C y otros protocolos convencionales como son: DH y ECDH, debido a que a la fecha no se ha podido encontrar datos publicados que nos permitan realizar esta tarea.

6.2 Trabajos Futuros

Existen ciertos aspectos dentro de este trabajo que pueden ser ampliados o implementados en trabajos futuros, a continuación se detallan estos aspectos:

- La aplicación mostrada en este trabajo fue desarrollada para un dispositivo celular basado en el sistema operativo Android, se podría extender el uso del protocolo DH-C para otras plataformas como por ejemplo: iOS, Symbian, Windows Phone, etc.

- La aplicación desarrollada hace uso del protocolo DH-C para establecer una clave compartida entre dos entes, la cual es usada para el intercambio seguro de mensajes de texto. Una característica que puede ser añadida a esta aplicación podría ser el uso de este protocolo para establecer comunicaciones de voz seguras.

- En el presente trabajo se empleó anillos no conmutativos de polinomios matriciales, como un trabajo futuro se podría utilizar otras estructuras no conmutativas en las cuales la conjugación o la descomposición son problemas complejos como por ejemplo: anillos matriciales no polinómicos, quasigrupos NC, grupos de trenzas, grupos de palabras, grupos nilpotentes, etc.

ANEXOS

A.1. Estructura de la Interfaz de Usuario

A continuación se detallan cada una de las pantallas y opciones que están disponibles en esta aplicación.

Una vez que el usuario haya instalado la aplicación DHC-Demo, podrá ingresar a ella a través del acceso que se encuentra en el menú principal del teléfono móvil, tal como se muestra en la Figura 19

Figura 19: Acceso a la Aplicación DHC-Demo

Pantalla de Inicio de Sesión

Cuando el usuario ingresa a la aplicación se encontrará con la pantalla de inicio de sesión, en la que deberá ingresar su usuario y contraseña en los campos correspondientes para poder ver la lista de sus contactos y empezar a comunicarse con ellos a través de mensajes de texto.

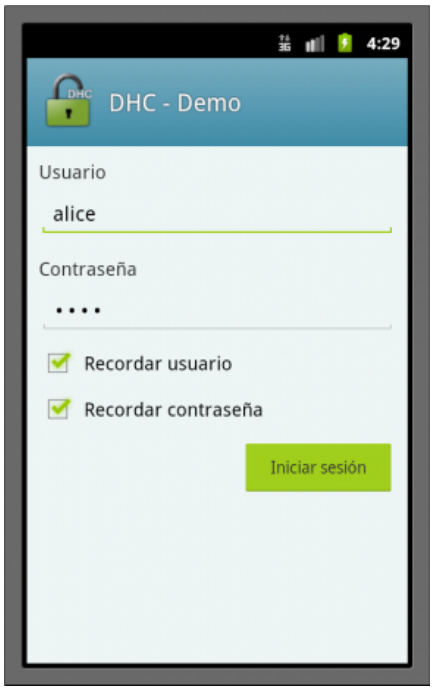


Figura 20: Pantalla de Inicio de Sesión

En esta pantalla se puede apreciar dos casillas de validación las cuales permitirán al usuario recordar su usuario y/o contraseña de inicio de sesión, esto con el fin de que el usuario no tenga que volver a ingresar esta información cada vez que desee ingresar a la aplicación.

Si el usuario no posee su cuenta para poder conectarse al servidor deberá crear una Cuenta y también será necesario realizar la Configuración del Servidor al cual se va a conectar. Para poder acceder a estas opciones será necesario desplegar el menú (Figura 21) desde la pantalla de Inicio de Sesión.

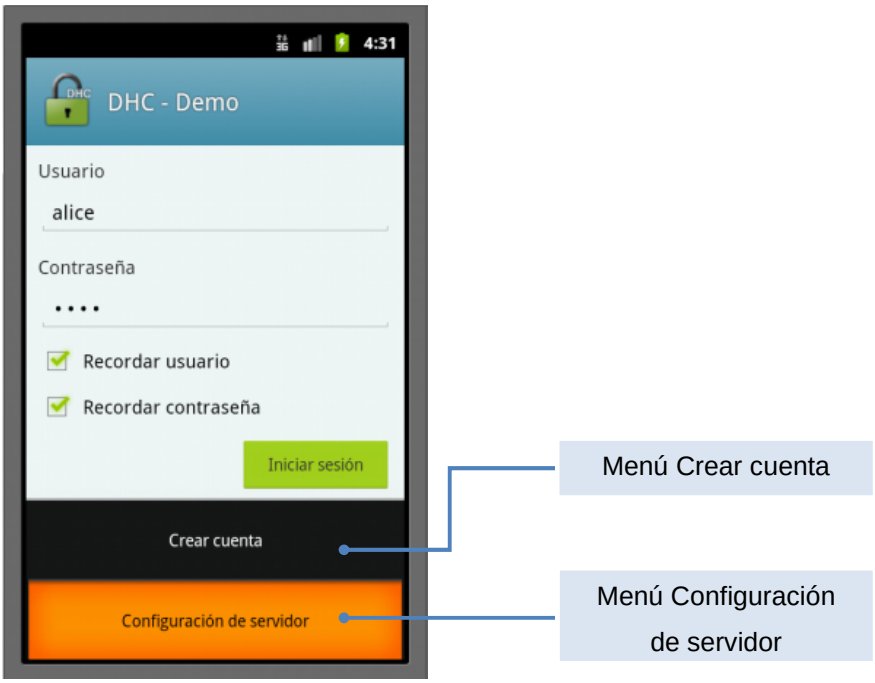


Figura 21: Menú de Pantalla de Inicio de Sesión

Pantalla Crear Cuenta

Si el usuario no posee una cuenta o si requiere crear una nueva lo podrá hacer desde esta pantalla, la cual presentará el siguiente formulario:

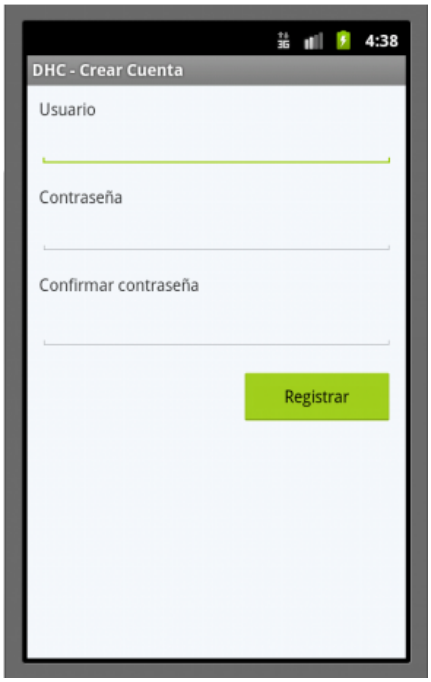


Figura 22: Pantalla de Creación de cuenta.

Donde necesitará ingresar su nombre de usuario y una contraseña.

Pantalla de Configuración de Servidor

Una vez que el usuario ha creado su cuenta, es necesario que configure la información del servidor al cual se va a conectar. Los servidores que se puede utilizar para esta aplicación son los siguientes: GTalk, Facebook o DHC- Demo, este último lo utilizamos para configurar el acceso a nuestro propio servidor Openfire (Este servidor utiliza el protocolo abierto ampliamente adoptado para la mensajería instantánea XMPP, también llamado Jabber), como se muestra a continuación:

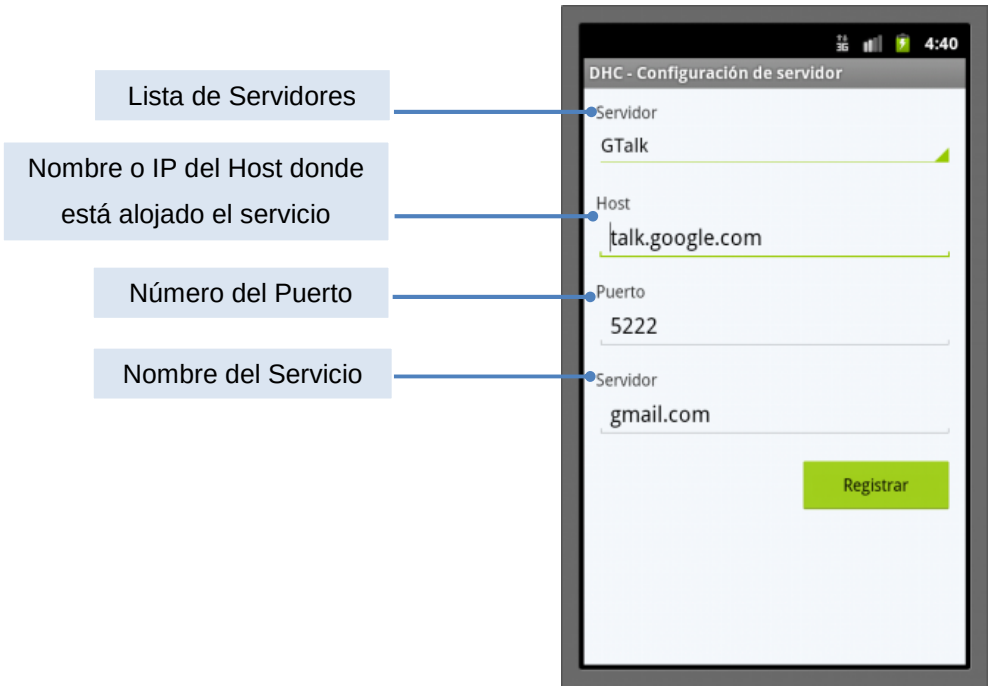


Figura 23: Pantalla de Configuración de servidor.

Pantalla de Contactos

Una vez que el usuario ha creado su cuenta y ha configurado el servidor al cual se va a conectar correctamente, ya puede ingresar a la aplicación desde la pantalla de Inicio de Sesión (Figura 20). Una vez que ha iniciado sesión se mostrará la pantalla de Contactos (Figura 24).

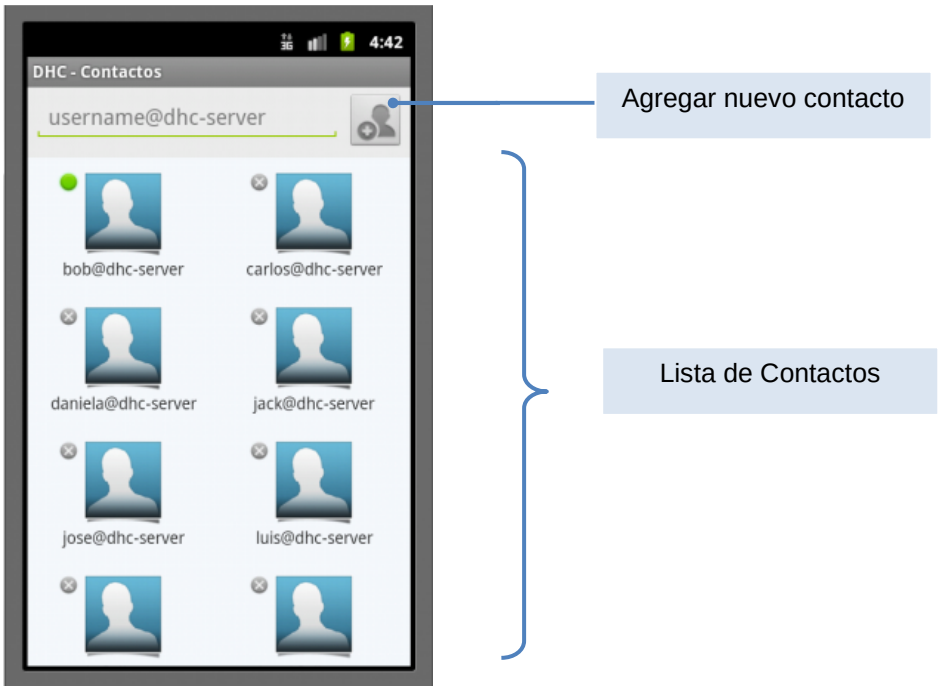


Figura 24: Pantalla de Contactos

Esta pantalla consta de dos secciones, la primera nos servirá para agregar un nuevo contacto a nuestra lista y la segunda es donde se mostrará el listado de todos nuestros contactos.

Pantalla de Chat

En la lista de contactos podemos ver el estado de cada uno de ellos (podemos ver si están en línea o no). Al Seleccionar alguno de estos contactos se mostrará la pantalla de Chat, en la cual podremos intercambiar mensajes con el contacto seleccionado (solo se podrá intercambiar mensajes con usuarios que estén en línea).

En esta pantalla se encuentra el botón Cifrado, el cual, al ser activado nos permitirá iniciar el protocolo de intercambio de claves DH-C, para poder cifrar los mensajes que se envíen y reciban con el contacto seleccionado anteriormente. Si se desea terminar con el cifrado de mensajes bastará con volver a presionar el botón para desactivarlo.

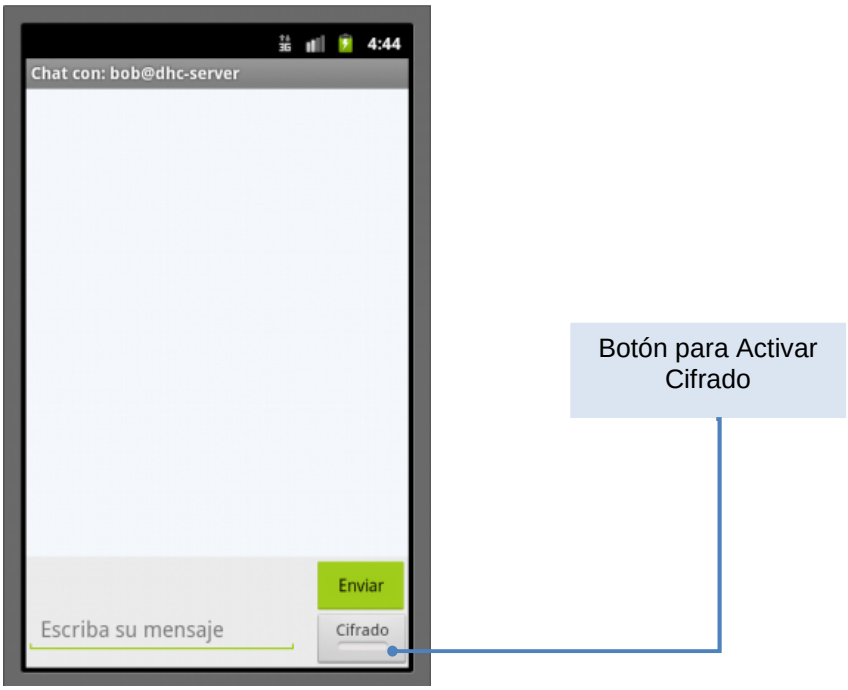


Figura 25: Pantalla de Chat.

Pantalla de Configuración de DHC

Para el intercambio de claves es necesario definir tres parámetros: Módulo, Orden de las Matrices y Grado del Polinomio. Para realizar esta configuración se deberá acceder a esta pantalla por medio del menú DHC – Configuración de la pantalla de Chat (Figura 26).

En esta pantalla de configuración se puede observar una casilla de validación, si la misma esta activada, el usuario podrá observar los mensajes que son enviados y recibidos durante el intercambio de claves Diffie-Hellman Compacto. Si se desea utilizar los valores por defecto (módulo = 251, orden de matrices = 4, grado de polinomio = 16) será necesario presionar el botón Default y a continuación presionar el botón Guardar para almacenar esta información.

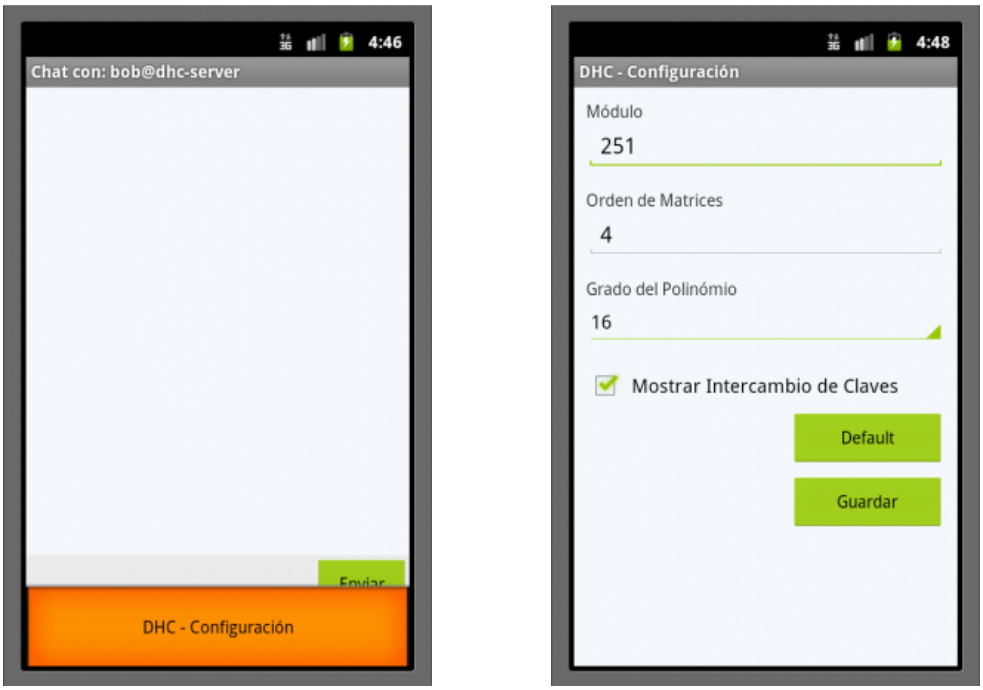


Figura 26: Pantalla de Configuración DHC.

A.2. Ejecución de la clase DHCTestClass

A continuación se muestra la salida de la ejecución de este programa:

```
1. Alice escoge al azar los siguientes valores y se los envía a Bob
a través de un canal público:

Matriz A:
93    73    112   71
35    83    135   45
245   19    90    62
162   43    197   145

Matriz B:
145   19    121   10
70    117   204   159
189   113   135   67
45    162   40    22

================================================

2. Alice define su clave privada:

Exponentes:
m: 12
n: 8

Polinomio A:
x^14 + 3x^13 + 5x^10 + 11x^9 + 4x^8 + 6x^7 + 12x^2

Calcula f(a):
134   61    177   22
87    200   106   238
104   35    95    91
183   127   214   47

================================================

3.Bob define su clave privada:

Exponentes
r: 9
s: 7

Polinomio B:
3x^14 + 11x^9 + 12x^8 + 10x^7 + 5x^2 + 4x + 6

Calcula h(a):
94    3     8     39
173   199   145   234
56    138   104   126
117   32    228   134

================================================
```

```
4. Alice computa rA = f(a)^m.b.f(a)^n y se lo envía a Bob:
4     100   91    116
243   182   108   216
236   18    194   63
180   238   203   4

================================================

5. Bob computa rB = h(a)^r.b.h(a)^s y se lo envía a Alice:
204   55    191   30
204   4     146   96
186   160   235   102
205   57    44    4

================================================

6. Alice recibe rB y calcula la clave compartida
kA = f(a)^m.rB.f(a)^n
119   97    211   113
39    82    104   209
64    55    179   231
155   5     66    16

================================================

7. Bob recibe rA y calcula la clave compartida
kB = h(a)^r.rA.h(a)^s
119   97    211   113
39    82    104   209
64    55    179   231
155   5     66    16

================================================

 "OK"

================================================
```

BIBLIOGRAFIA

1. Paeng S., Ha K., Kim J., Chee S. and Park C., New Public Key Cryptosystem Using Finite Non Abelian Group, Advances in Cryptology - Crypto 2001, (ed). Kilian J., Springer Berlin Heidelberg, (2001), pp.470-485.

2. Paeng S., Kwon D., Ha K. and Kim J., Improved public key cryptosystem using finite non abelian groups, Cryptology ePrint Archive, Report 2001/066, (2001).

3. Birget J., Magliveras S. and Sramka M., On public-key cryptosystems based on combinatorial group theory, Cryptology ePrint Archive, Report 2005/070, (2005).

4. González Vasco M., Martinez C. and Steinwandt R., Towards a Uniform Description of Several Group Based Cryptographic Primitives, Designs, Codes and Cryptography, Kluwer Academic Publishers, 33, (2004), pp.215-226.

5. Menezes A., Van Oorschot P. and Vanstone S., Handbook of Applied Cryptography, CRC Press, (1997).

6. Hecht J. P., Un modelo compacto de criptografía asimétrica empleando anillos no conmutativos, Actas del V Congreso Iberoamericano de Seguridad Informática CIBSI'09, Montevideo, Uruguay, (2009), pp.188-201.
   http://www.fing.edu.uy/inco/eventos/cibsi09/docs/Papers/CIBSI-Dia1-Sesion2%281%29.pdf

7. Cao Z., Dong X. and Wang L., New Public Key Cryptosystems Using Polynomials over Non-commutative Rings, Cryptology ePrint Archive, Report 2007/009, (2007).
eprint.iacr.org/2007/009.pdf

8. Diffie W. and Hellman M., New directions in cryptography, IEEE Transactions on Information Theory, (1976).

9. Anjaneyulu G., Vasudeva Reddy P. and Reddy U., Secured Digital Signature Scheme using Polynomials over Non-Commutative Division Semirings, IJCSNS International Journal of computer science and network security, 8:8, (2008), pp.278-284.
http://paper.ijcsns.org/07_book/200808/20080839.pdf

10. Koblitz N., Algebraic aspects of cryptography, Springer, Alemania, (1998).

11. Gálvez S., Ortega L., Java a Tope, J2ME, Edición Electrónica.
http://www.lcc.uma.es/~galvez/J2ME.html

12. Knudsen J., Wireless Java, Developing with J2ME, Second Edition, Apress, New York, (2003).

13. Sing L. and Knudsen J., Beginning J2ME, From Novice to Professional, Third Edition, Apress, New York, (2005).

14. MacLean D., Komatineni S. and Hashimi S., Pro Android 2, Apress, New York, (2010).

15. Meier R., Android 2 Application Development, Wiley Publishing, Indiana, (2010).