DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN

IMPLEMENTACIÓN Y EVALUACIÓN DE UN SISTEMA DE VOTACIÓN ELECTRÓNICA, BASADO EN TÉCNICAS CRIPTOGRÁFICAS, PARA UNA VOTACIÓN DE PEQUEÑA ESCALA

MEMORIA PARA OPTAR AL TÍTULO DE INGENIERO CIVIL EN COMPUTACIÓN

SERGIO EDUARDO MIRANDA MANRIQUEZ

PROFESOR GUÍA:
ALEJANDRO HEVIA A.

MIEMBROS DE LA COMISIÓN:
MARCOS KIWI
PATRICIO INOSTROZA

SANTIAGO DE CHILE
MARZO 2008

RESUMEN DE LA MEMORIA
PARA OPTAR AL TITULO DE
INGENIERO CIVIL EN COMPUTACIÓN
POR: SERGIO MIRANDA MANRIQUEZ
FECHA: 28/03/2008
PROF. GUÍA: SR ALEJANDRO HEVIA

"IMPLEMENTACIÓN Y EVALUACIÓN DE UN SISTEMA DE VOTACIÓN ELECTRÓNICA, BASADO EN TÉCNICAS CRIPTOGRÁFICAS, PARA UNA VOTACIÓN DE PEQUEÑA ESCALA"

El objetivo general del presente trabajo de título es la implementación de un sistema de votación electrónica apropiado para una votación local a pequeña escala. Además, este trabajo analiza las condiciones y supuestos razonables bajo los cuales tal elección puede ser llevada a cabo exitosamente.

En este trabajo se estudian los mecanismos criptográficos adecuados para implementar un sistema de votación correcto, verificable y cuyos votos sean privados.

El sistema implementado cumple con las características de seguridad de un sistema de votación electrónica, es práctico y puede ser usado en votaciones de pequeña escala. Este sistema considera la existencia de varias autoridades electorales, las cuales necesitan cooperar para que la elección sea exitosa, y solamente la colusión de la mayoría de ellas puede poner en riesgo la privacidad de los votantes.

Se realizaron diversas votaciones de prueba, las que ilustran como el sistema implementado es capaz de detectar tanto alteraciones de los votos, como autoridades que traten de alterar el proceso eleccionario. El sistema es eficiente y transparente y entrega resultados precisos en forma rápida y oportuna. El proceso completo es universalmente verificable, desde la generación de las claves de encriptación que se usaran durante la elección, hasta el cómputo final, pasando por la verificación de la correctitud de los votos emitidos.

Se concluye que es factible implementar un sistema de votación electrónica práctico, seguro y eficiente con requisitos simples: conectividad de red entre los participantes, mecanismos de comunicación consistente de mensajes (broadcast) entre las autoridades y observadores externos, e infraestructura para manejar claves públicas y privadas de los votantes.

Finalmente se entregan recomendaciones para extender el sistema a elecciones de gran escala.

ÍNDICE

1 Introducción

Este trabajo de título pretende implementar un sistema de votación electrónica a pequeña escala con el fin de comprender sus alcances y restricciones en la práctica. En esta introducción se revisarán algunos antecedentes históricos de votación electrónica y los conceptos básicos involucrados.

1.1 Antecedentes históricos

A lo largo de la historia, los procesos de votación han evolucionado en parte con el fin de reducir fraudes, como por ejemplo agregar votos extras para favorecer a un candidato, o evitar la coerción a los votantes [1]. Las primeras máquinas de votación mecánicas datan de 1892, éstas fueron incorporadas con el fin de eliminar la papeleta de votación y permitir conteos automáticos. En los años 70s se introdujeron los primeros sistemas de votación completamente automatizados, son los llamados DRE ("Direct recording electronic systems"). Estos sistemas permiten que el votante escoja los candidatos desde una papeleta que puede ser impresa o bien desplegada en una pantalla de computador. Los votantes pueden marcar sus preferencias ya sea presionando botones, tocando la pantalla o usando algún otro dispositivo. El votante envía su preferencia, por ejemplo presionando un botón "votar", y los votos son guardados electrónicamente. Cada máquina puede ser programada fácilmente para desplegar información en diversos lenguajes y desplegar diversos votos de acuerdo a las necesidades de la elección. Los DREs pueden permitir accesibilidad a discapacitados, la utilización de pantallas touchscreen y además previenen el marcaje de votos ambiguos, el "overvote" (que se escojan más candidatos de los permitidos) y el "undervote" (escoger menos candidatos de los permitidos). En EEUU una de cada nueve personas utilizó DREs el año 2000 para votar [2].

Existen sin embargo diversos problemas de seguridad que afectan los DREs y variados problemas documentados de errores en elecciones pasadas. Entre los diversos problemas de seguridad se pueden mencionar:

- Frecuentemente no existe registro independiente de los votos que puedan utilizarse para un recuento en caso de fallas en una máquina.
- La existencia de código malicioso en la máquina podría alterar el resultado.
- La complejidad del software lleva a que modificaciones no autorizadas sean difíciles de detectar.
- La mayor parte de las máquinas DRE ejecutan código propietario, por lo que no está disponible para el análisis público de posibles vulnerabilidades.

Varias propuestas de solución a estos problemas han estado en discusión, algunas de ellas son:

- Certificación del software: el código de máquina de las DRE debe ser certificado de modo de establecer que sigue procedimientos autorizados y en particular que no altera el resultado.

- Uso de software Open Source, entendido como programas de computación cuyo código fuente se encuentra disponible al público general de modo de ser mejorado o modificado de acuerdo a los deseos del usuario.

- Los programas no debieran correr sobre plataformas muy complejas. En caso de ser así, separarlas en componentes más simples, de modo de minimizar las posibilidades de falla y facilitar las auditorías [2].

- Mecanismos de verificación y transparencia, como son la generación de papeletas impresas con la preferencia del votante, de modo que éste pueda comprobar que se marcó la opción correcta y ésta papeleta (que se deposita en una urna) pueda ser usada en un eventual recuento de votos. Este método se denomina "voter verified paper ballot" y fue propuesto por Rebecca Mercuri [3].

Un aspecto crucial de un sistema de votación electrónica es que tenga mecanismos de verificación del resultado, tanto por los votantes como por observadores externos al proceso. Los sistemas de votación electrónica que utilizan técnicas criptográficas, como el que se implementará en el presente trabajo de título, proveen esta funcionalidad**.**

1.2 Conceptos Básicos

El diseño de un sistema de votación electrónica debe idealmente poseer al menos las siguientes características:

- Seguro, definido bajo los siguientes conceptos:

    - Democrático: Solamente los votantes válidos pueden sufragar, esto es, manifestar su preferencia por medio de un voto, y ningún votante puede sufragar más de una vez.

    - Correcto: El valor de cada voto no puede ser alterado, duplicado ni eliminado sin ser detectado, por lo tanto el resultado final debe corresponder a los votos correctamente emitidos.

    - Privado: El valor de cada voto debe mantenerse en secreto mientras se desarrolla el proceso. Al concluir la votación ninguna información acerca del voto debe revelarse, excepto aquella deducible del conteo total (y posiblemente conteos parciales). No debe ser posible asociar un voto con el votante que lo emitió.

    - Verificable: Un observador externo puede convencerse que el proceso fue correcto.

    - Robusto: El sistema debe soportar un nivel razonable de ataques intencionados por parte de alguna colusión pequeña de entidades, ya sean externas o partícipes del proceso.

    - No coercibilidad: El valor del voto debe reflejar la voluntad del usuario.

- Práctico: Debe ser un sistema que sea conveniente de usar y compatible con plataformas y tecnologías estándares.

- Escalable: Puede ser usado tanto por un grupo reducido de votantes como por millones de ellos a nivel nacional. También puede permitir múltiples preguntas.

La votación electrónica permite variados beneficios, entre ellos se pueden destacar:

- La entrega de garantías de seguridad aún mayores que sistemas manuales, como por ejemplo, el poder garantizar que solo la colusión de la mayoría de las entidades encargadas de realizar el recuento de votos puede comprometer la privacidad de los votantes.

- Obtención de resultados en forma rápida y eficiente, con menor probabilidad de errores humanos.

- Reducción de costos (al menos en el largo plazo).

- Mejores posibilidades de accesibilidad para personas con discapacidad.

- Presentar el voto en diferentes lenguajes.

- Introducir preguntas complejas o múltiples.

Los beneficios descritos hacen atractiva la tecnología de votación electrónica, fomentando en diversas partes del mundo iniciativas para migrar sistemas de votación tradicionales a sistemas de votación electrónicos. El agregar la propiedad de verificabilidad hace que el sistema sea además difícil de abusar (por ejemplo en regímenes dictatoriales).

1.3 Motivación

Una medida para incentivar el interés en participar en los procesos de votación es la introducción de votación electrónica, lo cual permitiría aprovechar la extendida infraestructura computacional existente e introducir un proceso eleccionario que sea práctico, eficiente y seguro. Este último punto, seguridad, es de crucial importancia, dado que la confianza en los resultados existirá solo en función de la percepción popular de que el sistema electrónico entrega una seguridad igual o mejor que la entregada por el sistema actual basado en papel.

La tendencia es que la votación electrónica reemplace a la votación tradicional en el largo plazo en todos los niveles de elecciones (locales o nacionales), pero para que ello ocurra deben resolverse adecuadamente los problemas de privacidad, identificación de los votantes, verificación de los resultados, auditoría del proceso y sincerar los costos económicos asociados.

El auditar una votación electrónica no es un problema simple, ya que el voto emitido debe permanecer anónimo durante todo el proceso. En la práctica la propiedad usualmente buscada es denominada “verificación universal”. En un sistema universalmente verificable tanto los participantes como los observadores externos pueden verificar la correctitud de la elección.

En este trabajo se investiga cuál de los diversos protocolos criptográficos que proveen mecanismos para auditar el proceso, es el más adecuado para ser implementado en una elección de tipo local, a nivel de una organización pequeña.

Adicionalmente este trabajo discute los supuestos razonables para que la implementación de una elección electrónica sea factible.

1.4 Objetivo General

El principal objetivo de esta memoria consiste en la implementación de un sistema de votación electrónica apropiado para una votación local a pequeña escala. Además se analizan bajo qué condiciones esta implementación puede ser extendida a elecciones de mayor tamaño.

El objetivo es obtener una implementación de un sistema de votación capaz de cumplir con la mayor cantidad de los requisitos mínimos necesarios (idealmente todos) de un sistema de votación electrónica seguro.

1.5 Objetivos Específicos

Los objetivos específicos para este trabajo de titulo son:

- Estudiar los diversos modelos criptográficos y protocolos de votación electrónica.
- Definir las restricciones que se aplicarán a la implementación del sistema.
- Escoger el sistema adecuado para ser implementado de acuerdo a las restricciones y tamaño de la elección.
- Implementar el sistema de votación electrónica adecuado para una elección a escala pequeña.
- Realizar votaciones de prueba.
- Establecer las conclusiones sobre la aplicabilidad del sistema a nivel local y en elecciones mayores.

2 Revisión Bibliográfica

2.1 Criptografía para votación electrónica

La criptografía se define como el "estudio y construcción de sistemas robustos ante intentos maliciosos para desviarlos de una funcionalidad predefinida" [4] y trata habitualmente aspectos de seguridad de información, tales como confidencialidad, integridad de datos, autenticación de entidades y de origen de datos [5]. En la implementación de los sistemas de votación electrónica se utilizan diversas herramientas criptográficas, algunas de las cuales son ampliamente conocidas en la literatura, y otras más específicas. En esta sección se revisarán conceptos matemáticos subyacentes en la implementación de dichas herramientas.

2.1.1 Teoría de números

En primer lugar se revisaran conceptos básicos de la teoría de números. Esta sección se basa en la descripción de Bellare y Rogaway [6].

Conceptos básicos

Si *x,y* son enteros no ambos iguales a cero, entonces su máximo común divisor, denominado gcd (*x,y*) es el mayor entero *d*, tal que *d* divide a *x* y *d* divide a *y*. Si gcd(x,y)=1 entonces se dice que *x* e *y* son primos relativos (por ejemplo, 4 y 15 son primos relativos entre sí).

Si *a,*N son enteros con N > 0 entonces existen los enteros *r,q* tales que

$$a = Nq + r \text{ con } 0 \leq r < N$$

donde *r* se denomina el resto de la división de *a* por N y se denota por *a mod N*. Si *a,b* son enteros y N un entero positivo, se denota: *a* ≡ *b (mod N)* si *a mod N = b mod N*.

Además se definen los siguientes conjuntos:

$$Z_N = \{0,1,\ldots,N-1\}$$

$$Z^*_N = \{\ i \in Z : 1 \leq i \leq N-1 \text{ y gdc}(1,N)=1\ \}$$

El primer conjunto Z_N se denomina los *enteros módulo N*. El conjunto Z^*_N se define como el conjunto de enteros entre 1 y N-1 cuyos elementos son primos relativos con N. Por ejemplo, si N=8, entonces $Z^*_8 = \{1,3,5,7\}$.

Grupos

Si G es un conjunto no vacío y ● una operación binaria sobre G, es decir, una función de GxG en G, entonces se dice que G es *grupo* si cumple las siguientes propiedades:

1. Asociatividad: Para todo $a,b,c \in G$ se cumple que $a \bullet (b \bullet c) = (a \bullet b) \bullet c$

2. Identidad: Existe un elemento $1 \in G$ tal que $a \bullet 1 = 1 \bullet a = a$ para todo $a \in G$

3. Invertibilidad: Para todo $a \in G$ existe un único $b \in G$ tal que $a \bullet b = b \bullet a = 1$. El elemento *b* se conoce como el inverso de *a* y se denota a^{-1}

Si además $\bullet$ satisface:

4. Conmutatividad (el orden de los factores no altera el producto): Para todo $a, b \in G, a \bullet b = b \bullet a$

entonces se dice que el grupo es G *abeliano*.

El subconjunto H de G es un subgrupo de G si H forma un grupo bajo la operación $\bullet$.

Propiedades:

1. Z_N es grupo bajo la suma módulo N y Z^*_N es grupo bajo la multiplicación módulo N.

2. Sea G un grupo y *m* = |G| su tamaño (también denominado orden), entonces $a^m = 1$ para todo $a \in G$.

Grupos cíclicos y generadores

Dado un grupo G, con |G |=m, un elemento *g* de un grupo es denominado un *generador* de G si para todo $a \in G$ existe un único entero $i \in Z_m$ tal que $g^i = a$. El elemento *i* se conoce como el *logaritmo discreto* de *a* en base *g* y lo denotaremos por $DLog_{G,g}(a)$.

Si existe un generador para un grupo, entonces se dice que el grupo es cíclico. Por ejemplo: Z^*_{11}= {1,2,3,4,5,6,7,8,9,10} es un grupo de orden 10. Los subgrupos generados por los elementos 2 y 5 son:

a	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
2^a mod 11	2	4	8	5	10	9	7	3	6	1
5^a mod 11	5	3	4	9	1	5	3	4	9	1

En este ejemplo se ve que el elemento 2 genera los siguientes elementos: {1,2,3,4,5,6,7,8,9,10}, en cambio el elemento 5 solo genera los elementos {1,3,4,5,9}.

El elemento 2 es un generador del grupo dado que sus potencias conforman $Z^*_{11.}$ Por otra parte el elemento 5 no es generador. Dado que existe un generador para Z^*_{11} (elemento 2) entonces dicho grupo es cíclico.

Invertir la función i= $DLog_{G,g}(a)$ es considerada infactible de resolver, o también denominada función unidireccional [4, sección 2.4.3.4]. Es por ello que se utiliza a menudo en criptografía..

Dos propiedades importantes son las siguientes:

1. Si *p* es un número primo, entonces el grupo Z^*_p es cíclico.

2. Sea G un grupo y *m*=|G| su orden. Si *m* es un número primo entonces G es cíclico.

Los grupos de orden primo son muy utilizados en criptografía, particularmente en algoritmos de

encriptación asimétricos como ElGamal [5]. Este punto se discute en el siguiente capítulo.

2.1.2 Criptografía asimétrica de clave pública

Las herramientas de criptografía de clave pública permiten la comunicación segura entre las partes sin tener acceso a una clave secreta compartida. Para ello se utiliza un par de claves criptográficas denominadas clave pública y clave privada, las cuales satisfacen una relación matemática. La clave privada es mantenida en secreto, mientras que la pública es distribuida a todas las partes.

Un esquema de encriptación asimétrico (o de clave pública) consiste en tres algoritmos: el algoritmo de generación de claves (*G*), el algoritmo de encriptación (*E*) y el algoritmo de desencriptación (*D*).

La Figura 1 muestra un esquema simplificado del funcionamiento de la encriptación con clave pública.

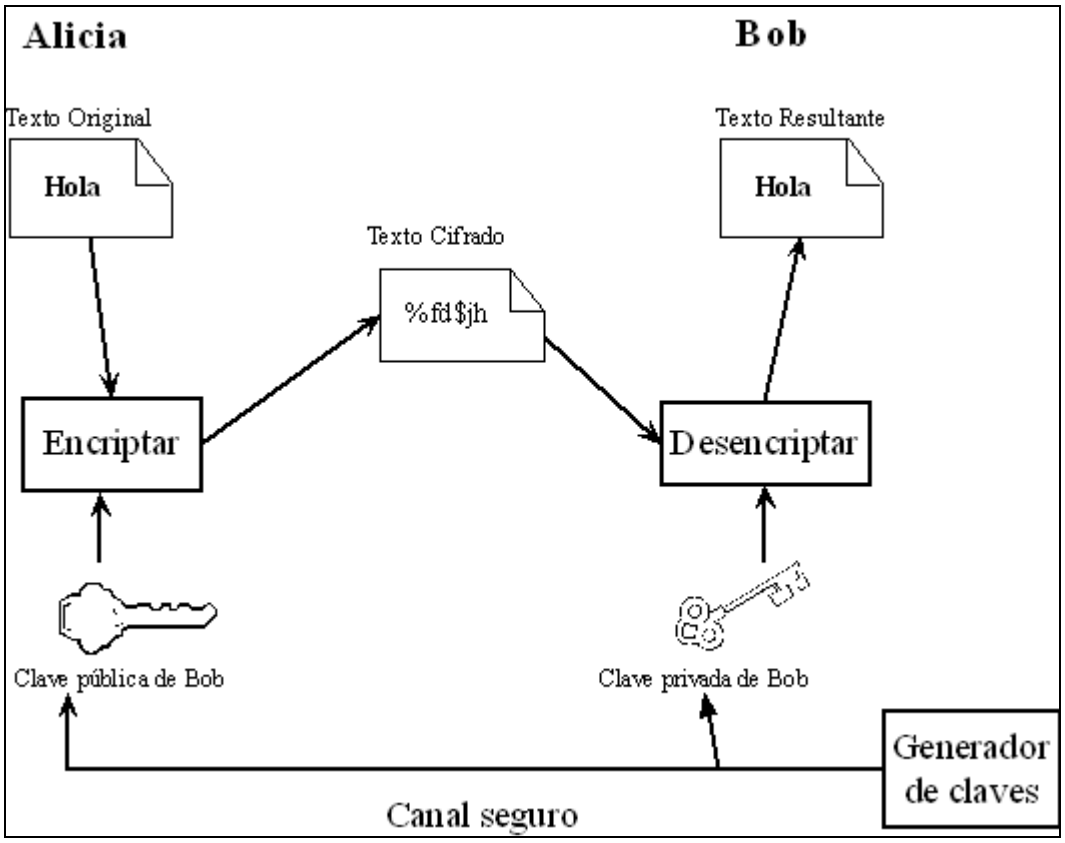

Figura 1: Esquema de criptografía asimétrica

En el proceso de encriptación de un mensaje se ocupa la clave pública del destinatario (Bob) para generar el texto cifrado. El texto cifrado puede ser desencriptado por el destinatario usando su clave privada. Ambas claves son generadas por el destinatario. La clave pública necesita ser comunicada al enviador a través de un canal seguro (que garantice integridad) solo una vez antes de encriptar los mensajes.

2.1.2.1 Esquema de encriptación ElGamal

El esquema criptográfico de encriptación ElGamal [5] se basa en el problema computacional del logaritmo discreto. En la siguiente descripción supondremos que existen dos partes Alicia y Bob (personas, computadoras, etc.) que desean comunicarse. Este esquema consiste en tres componentes: algoritmo de generación de claves, algoritmo de encriptación y algoritmo de desencriptación.

a) Algoritmo de generación de claves *G*

Sea k un entero fijo (por ejemplo k=1024).

Bob crea una clave pública y su correspondiente clave privada:

- Bob genera un primo *p* aleatorio de largo k-bits y un generador *g* del grupo Z^*_p.
- Bob elige un entero aleatorio *x* en el conjunto {1,…,p-2}.
- Bob calcula $y=g^x \bmod p$.
- La clave pública de Bob es pk=(*p,g,y*).
- La clave secreta de Bob es sk=(*p,g,x*).

b) Algoritmo de encriptación *E*

Dada la clave pública pk=(*p,g,y*) de Bob, si Alicia desea encriptar un mensaje *m* para Bob, debe realizar los siguientes pasos:

- Alicia representa *m* como un entero en {0,1,…,*p*-1}.
- Alicia selecciona aleatoriamente un entero *r* en {1,…,*p*-2}.
- Se calcula $a = g^r \bmod p$ y $b = m * y^r \bmod p$.
- El texto cifrado es $c=E_{pk}(m)=(\alpha,\beta)$.

c) Algoritmo de desencriptación *D*

Dada la clave privada sk de Bob, para desencriptar el mensaje de Alicia, Bob realiza lo siguiente:

- Calcula $m' = D_{sk}(a,b) = b * a^{-x} \bmod p$.

Notar que el algoritmo de desencriptación produce el mensaje *m* puesto que:

$$m' \equiv b * a^{-x} \equiv m * (g^x)^r * (g)^{-xr} \equiv m$$

Propiedad homomórfica

Una propiedad útil de ElGamal es la denominada propiedad homomórfica. Informalmente esta

propiedad consiste en que la multiplicación de dos textos cifrados de mensajes m_1 y m_2 produce el texto cifrado de la multiplicación de m_1 y m_2 [7,8].

Formalmente, sean $C_1=(\alpha,\beta)$, $C_2=(\gamma,\delta)$, $\alpha,\beta,\gamma,\delta$ en Z^*_p ,donde $C_1=E_{pk}(m_1)$ y $C_2= E_{pk}(m_2)$. Se define la multiplicación de C_1 y C_2 como

$$(a,b)*(c,d)=(a\cdot c, b\cdot d)$$

Donde la operación $\cdot$ es la multiplicación mod *p* y se tiene que $C=C_1*C_2= E_{pk}(m_1*m_2)$.

Es posible modificar ElGamal para obtener una encriptación que contenga $m_1 + m_2$ como sigue:

Usando un nuevo generador, h en Z^*_p, se reemplaza m por h^m en el protocolo.

Entonces usando la propiedad homórfica antes vista, se obtiene:

$C=C_1*C_2=E_{pk}(h^{m1}*h^{m2})=E_{pk}(h^{m1+m2})$.

El proceso de desencriptación ElGamal no require ser modificado.

2.1.3 Algoritmo de firma digital (DSA)

La firma digital permite asociar el contenido de un mensaje con el autor del mismo, además de garantizar la integridad de los datos y la no repudiación de los mismos. El algoritmo DSA es un esquema de firma digital aceptado y reconocido como estándar por el gobierno de EEUU (FIPS 186)[5].

El protocolo requiere de los siguientes parámetros:

- q: número primo tal que $2^{159} < q < 2^{160}$.
- p: número primo donde $2^{251+64t} < p < 2^{512+64t}$, con $0 \le t \le 8$, tal que q divide a (p-1).
- g: generador de un grupo cíclico de orden q en Z^*_p.

Para generar las claves de usuario se siguen los siguientes pasos:

a) Se selecciona un entero aleatorio x en el conjunto {1,...,q-1}

b) Se calcula $y=g^x \bmod p$

La clave pública del usuario es (p,q,g,y) y la clave privada es x.

Alicia puede firmar un mensaje m y Bob verificarlo, usando la clave pública de Alicia, con el siguiente algoritmo:

1. Firma: Alicia realiza los siguientes pasos:
    a. Escoge un entero al azar k, $0<k<q$, donde el número de bits de q es el mismo de H(m), con H función de hash SHA-1.
    b. Calcula $r=(g^k \bmod p) \bmod q$.

c. Calcula $s=k^{-1}(H(m)+xr) \bmod q$, donde H es nuevamente la función de hash SHA-1.

d. La firma para m es el par (r,s).

2. Verificación: Para verificar la firma de (r,s) sobre m, Bob debe realizar los siguientes pasos:

   a. Calcula $w=s^{-1} \bmod q$.

   b. Calcula $u1=w*H(m) \bmod q$.

   c. Calcula $u2=rw \bmod q$.

   d. Calcula $v=(g^{u1}y^{u2} \bmod p) \bmod q$.

   e. La firma es válida sólo si v=r.

2.1.4 Protocolo para compartir secretos (secret sharing)

El objetivo de un protocolo de secreto compartido-(n,t) es que dado un mensaje *S* (el secreto) y un parámetro t (denominado umbral), permitir distribuir "trozos" $S_1,..,S_n$ del secreto entre *n* participantes, de modo que[1]:

1. Propiedad de confidencialidad: ningún subconjunto $B \subseteq \{P_1,..,P_n\}$ de *t-1* participantes o menos posee alguna información sobre el secreto.

2. Cualquier subconjunto $C \subseteq \{P_1,..,P_n\}$ de al menos t participantes, puede reconstruir el secreto.

Los parámetros son $n \in N$ y $0 \leq t \leq n$.

Esquema de Shamir

El esquema de Shamir [9] es un protocolo de secreto compartido-(n,t) y se basa en interpolación polinomial para permitir la distribución de trozos de un secreto *S* entre *n* usuarios, de modo que se requieran *t* participantes para su reconstrucción. El esquema consta de dos etapas: la fase de distribución y la fase de reconstrucción.

a) Fase de distribución

Se denomina dealer (*D*) a la entidad que produce o conoce originalmente el secreto, la cual está encargada de iniciar esta fase del protocolo.

Los parámetros son: un entero $S \geq 0$ (secreto que solo conoce *D*), valores *n* y *t.*

El dealer ejecuta las siguientes acciones:

1. Se define $a_0=S$ y se escoge un primo $p > \max(S,n)$.

[1] Esta sección se basa en [7]

2. Se escogen $a_1,...,a_{t-1} \in Z_p$ al azar y se define el polinomio aleatorio $f(x) = \sum_{j=0}^{t-1} a_j x^j$.

3. *D* calcula $S_i = f(i) \bmod p$, 1 ≤ i ≤ n y envía en forma privada S_i a cada participante P_i, junto con el índice público *i.*

b) Fase de reconstrucción

Cualquier grupo de *t* participantes junta sus trozos S_i y se calcula el secreto S como sigue**:**

$$S = \sum_{i=1}^{t} S_i C_i \text{ , donde } C_i = \prod_{1 \le j \le t, j \ne i} \frac{x_j}{x_j - x_i}$$

Ecuación 1: reconstrucción de Shamir

2.1.5 Protocolos de secreto compartido verificables (verifiable secret sharing)

El esquema anterior asume que el dealer es confiable, sin embargo si éste no es honesto, podría enviar trozos incorrectos del secreto a los participantes, de modo que el secreto no podría ser reconstruido (o permitiría reconstruir un secreto diferente dependiendo del subconjunto de participantes que reconstruye el secreto). Para prevenir este comportamiento se requiere implementar un protocolo a través del cual los participantes puedan verificar la consistencia del proceso de distribución del secreto, de modo que cualquier subconjunto de *t* participantes reconstruya el mismo secreto.

Protocolo de Feldman

En el protocolo VSS de Feldman [7,8] solo el dealer puede enviar mensajes a los participantes, y éstos no pueden comunicarse entre ellos ni con el dealer para verificar los trozos.

El protocolo requiere los siguientes parámetros:

p: primo al azar
g: generador de Z^*_p
n: número de participantes
t: umbral

a) Fase de distribución

Dado un secreto S, el dealer realiza las siguientes acciones:

1. Dado a_0=S, utiliza el esquema de Shamir para generar $f(x) = \sum_{j=0}^{t-1} a_j x^j$ y $S_i = f(i) \bmod p, 1 \le i \le n$ y distribuye dichos secretos a través de un canal seguro (encriptados) a los participantes.

2. Calcula y publica $b_i = g^{a_i} \bmod p$ para todo $0 \le i \le n$

3. Cada participante puede verificar la siguiente "ecuación de verificación":

$$g^{S_j} = \prod_{k=0..t-1} b_k{}^{j^k}$$

4. Si la verificación es exitosa entonces los secretos son válidos, de lo contrario el dealer revela S_j y todos verifican. Si S_j no es válido entonces el dealer es corrupto. Si existen *t* reclamos se concluye que el dealer es deshonesto y se aborta el proceso.

b) Fase de reconstrucción

Cada participante P_j revela S_j y chequea la ecuación de verificación para todos los S_j revelados. Esta verificación se realiza debido a que se asume que puede haber participantes deshonestos. Para *t* valores S_j que pasan la verificación anterior se reconstruye el secreto S.

Protocolo de Pedersen

El protocolo de Feldman revela algo de información al compartir el secreto *S*. Si *S* es aleatoriamente escogido de un conjunto suficientemente grande, esto no es problema, sin embargo si *S* pertenece a un rango restringido (por ejemplo {0,1}), el adversario podría verificar, usando la información revelada, si un valor dado es el secreto compartido [7].

El protocolo de Pedersen [10] es una modificación del protocolo de Feldman para evitar el problema descrito. Para ello, se definen los parámetros extras: g,h generadores de Z^*_p escogidos al azar y se modifica la fase de distribución como sigue:

1. Se generan $a_0,a_1,..,a_{t-1}$, $b_0,b_1,..,b_{t-1}$ números aleatorios en Z^*_p y se calcula:

   $u_x=l(x)= b_0+b_1x+\ldots+b_{t-1}x^{t-1}$
   $S_x=f(x)=a_0+a_1x+\ldots+a_{t-1}x^{t-1}$ (Polinomio de Shamir)
   $\beta_i=g^{ai}h^{bi} \bmod p$

2. El dealer envía s_i, u_i a cada participante P_i. La ecuación de verificación es entonces:

$$g^{S_j} h^{u_j} = \prod_{k=0..t-1} b_k{}^{j^k}$$

El protocolo continúa igual que Feldman, revelando s_i, u_i durante la fase de reconstrucción.

2.1.6 Protocolo de distribución de claves

El protocolo de generación de claves distribuidas (DKG por sus siglas en inglés) es indispensable en el diseño de un sistema de encriptación distribuido, como el usado en los sistemas de votación electrónica homomórfica y mix-nets (Capítulo 2.2.1). El protocolo DKG es utilizado para generar y distribuir las claves pública/privada para la encriptación o desencriptación de mensajes de una forma segura. El modelo simplificado consiste en *n* servidores (participantes) los cuales cooperan para generar dos claves, una pública y la otra privada. La clave pública se comparte y la clave privada se mantiene secreta en forma distribuida, aún si menos de *n*/2 participantes están comprometidos (corruptos).

Protocolo DKG

Esta solución fue propuesta por Pedersen [10] y se basa en el protocolo para secreto compartido del mismo autor. Los parámetros son los siguientes:

p: primo grande
q: primo, el mayor divisor de p-1
g: generador de subgrupo en Z^*_p de tamaño q
h: potencia de *g* en Z^*_p
n: número de participantes
t: umbral (al menos *n*/2)

Fase I: Generación y verificación del secreto *x*

1. Cada participante P_i escoje en forma aleatoria los siguientes polinomios:

   $f_i(x)=a_{i0}+a_{i1}x+a_{i2}x^2+\ldots+a_{it}x^t$
   $f'_i(x)=b_{i0}+b_{i1}x+b_{i2}x^2+\ldots+b_{it}x^t$

2. Cada partcipante P_i calcula:

   $z_i=a_{i0}$
   $C_{ik}= g^{a_{ik}} h^{b_{ik}}$ con k=0..t
   $S_{ij}=f_i(j) \bmod q$
   $S'_{ij}=f'_i(j) \bmod q$,con j=1..n

3. Cada participante envía S_{ij} y S'_{ij} al participante P_j

4. Cada participante P_j verifica:

   $$g^{S_{ij}} h^{S'_{ij}} = \prod_{k=0}^{t} (C_{ik})^{j^k} \bmod p$$, para todo i=1..n

   Se define el conjunto QUAL como aquel formado por aquellos participantes cuya verificación en el paso anterior ha sido exitosa.

5. El secreto se puede construir como $x = \sum_{j \in QUAL} z_j \bmod q$, el cual no es conocido por ningún participante en particular ya que se encuentra distribuido. Cada participante P_i define su secreto como $x_i = z_i$.

Fase II: Generación de la clave pública

La clave pública es $y=g^x \bmod p$. Para obtenerla se sigue el siguiente procedimiento:

a) Cada participante P_i en QUAL distribuye $y_i = g^{z_i} \bmod p$ via Feldman VSS

b) P_i emite $A_{ik} = g^{a_{ik}} \bmod p$, para k=0..t

c) P_j verifica para cada i en QUAL $g^{S_{ij}} = \prod_{k=0}^{t} (A_{ik})^{j^k} \bmod p$

d) $y_i=A_{i0}= g^{z_i} \bmod p$. Con ello se calcula finalmente $y = \prod_{i \in QUAL} y_i \bmod p$

Nota: Este protocolo ha sido mejorado por Gennaro y otros [11] a fin de quitar ciertas posibles desviaciones en la distribución del secreto. Por simplicidad en este trabajo consideramos el protocolo original de Pedersen.

2.1.7 Protocolos de nula divulgación (zero knowledge proof)

Los protocolos de nula divulgación (ZK por sus iniciales en inglés), están diseñados para permitir a una entidad (el "demostrador" o *prover*) convencer a otra entidad (el "verificador"o *verifier*) que una cierta relación matemática se satisface sin revelar ningún tipo de información, salvo aquella implicada por la validez de la afirmación [7,8].

Se definen:

1. Participantes: P ("prover"), V("verifier")
2. Un conjunto de pertenencia $L \subseteq \sum^*$, donde $\sum^*$ es el conjunto de todos los strings binarios posibles.
3. Elemento a demostrar: $x \in \sum^*$

El objetivo de P es demostrar a V que $x \in L$. En este trabajo por simplicidad se considera el lenguaje L definido por una relación polinomialmente calculable R de la siguiente manera: L={ $x \in \{0,1\}^*$ | $w \in \{0,1\}^*$ tal que R(x,w)}.

Propiedades de un protocolo ZK

1. Correctitud: Si $x \in L$ y ambos P y V son honestos, entonces la interacción entre P y V finaliza con V entregando un estado "ok". Se denota <P,V>(x)=ok

2. Soundness: Para todo P* deshonesto, V honesto y $x \notin L$ se tiene que la probabilidad del evento "< P*,V>(x)=*no*" es despreciable.

3. Blackbox Zero Knowledge: Si V* es deshonesto, P es honesto y $x \in L$ entonces existe un algoritmo S (conocido como el "simulador") que solo interactúa con V* y teniendo solo input x es capaz de generar la misma conversación que P y V* generarían con input x. Intuitivamente esto significa que lo que sea que V* "aprende" después de interactuar con P, puede generarlo por si solo usando el simulador S.

Además, para el presente trabajo se necesita que el protocolo ZK satisfaga una condición adicional:

4. Proof of knowledge: Supongamos que P necesita usar un secreto w para convencer a V que $x \in L$, esto es, que R(x,w) se cumpla. Entonces, para cualquier valor x y cualquier P* (posiblemente malicioso), existe un algoritmo eficiente E, tal que si P* convence a V que $x \in L$, entonces E usando x y a P* como subrutina puede calcular el secreto w' tal

que R(x,w) se cumpla. Intuitivamente esto significa que si P* es capaz de crear una demostración convincente, entonces P* debe conocer el secreto respectivo.

En este trabajo de título se utilizaran protocolos ZK para:

1. Demostrar la correctitud de la papeleta de votación (Capítulo 3.2.5.1)

2. Demostrar la correctitud de la desencriptación del resultado final (Capítulo 3.2.6.3)

Estos protocolos son los propuestos por Chaum y Pedersen [18].

2.2 Sistemas de votación electrónica

Para los propósitos del presente trabajo, se distinguen dos tipos de votación electrónica [13]:

1. Votación en ambiente controlado. En este caso tanto los dispositivos como el lugar físico de votación se encuentran supervisados por una autoridad electoral. En este tipo de votación se usa alguna clase de máquina de votación (DRE) y los votantes se identifican personalmente ante la autoridad electoral y/o bien digitalmente en las máquinas (con algún método criptográfico).

2. Votación en ambiente no controlado. En este caso la votación se realiza a través de Internet, en cualquier lugar físico (hogar, trabajo, equipos públicos, etc.) conectado a la red. Este tipo de votación es susceptible a ataques de coerción, venta de votos o ataques de malware. Sin embargo en elecciones de escala pequeña estas amenazas pueden no ser significativas.

2.2.1 Paradigmas de votación electrónica

Actualmente existen al menos cuatro paradigmas criptográficos de votación electrónica [7,13]:

1. **Paradigma de mixnets**. El paradigma criptográfico denominado “mixnet” [14] está compuesto de varios servidores enlazados denominados *mixes.* Cada uno de ellos toma un grupo de votos encriptados, los reordena y reencripta al azar y los entrega al siguiente servidor, de modo que no es posible asociar los mensajes de entrada con los de salida. La desencriptación de los votos puede hacerse en forma parcial en cada servidor (con su propia clave) o al final del proceso utilizando una clave distribuida en varios servidores. La principal crítica a este esquema es que las pruebas de correctitud son voluminosas. Existen algunas implementaciones de tipo comercial de sistemas de elección electrónica basados en este esquema.

2. **Paradigma de firma ciega**. Este paradigma utiliza las propiedades de los algoritmos criptográficos de firma ciega, los cuales autentifican digitalmente un mensaje sin saber el contenido de él [15]. Esto permite validar el contenido del voto encriptado (o papeleta de votación), sin sacrificar la privacidad del votante. En este paradigma el votante puede validar que su voto encriptado es parte del proceso y envía anónimamente su clave de desencriptación al final del proceso para la contabilización del voto. Este esquema ha sido utilizado en algunas votaciones piloto a escala pequeña pero posee serios problemas de robustez [13].

3. **Paradigma basado en secretos compartidos (Benaloh)**. Este paradigma [16] utiliza el

esquema criptográfico homomórfico de secreto compartido. En este esquema el votante comparte su voto entre n autoridades de votación. Al final del proceso cada autoridad suma los votos recibidos y los comparte con el resto de las autoridades para obtener el cómputo final. La implementación de este esquema posee altos costos en términos de comunicación, ya que cada voto debe enviarse por n canales diferentes.

4. **Paradigma basado en encriptación homomórfico**. Este paradigma, propuesto por Cramer et al [17] utiliza la propiedad homomórfica de los algoritmos de encriptación vistos en el Capítulo 2.1.2.1 de modo que el "producto" de la encriptación de dos votos es equivalente a la encriptación de la suma de ambos votos, esto puede generalizarse para n votos. Cada votante encripta su voto con la clave pública de las autoridades y publica esta encriptación en un canal público ("bulletin board"), junto con una prueba de correctitud (variante no interactiva de un protocolo ZK). Al final del proceso las autoridades "multiplican" todos los votos recibidos para obtener la encriptación del total de la votación. Finalmente, en conjunto desencriptan el resultado. Para obtener robustez se utilizan algoritmos de secreto compartido y generación de claves (DKG) entre las autoridades [18]. Existen sistemas pilotos de elecciones restringidas basados en este esquema [13]. Una característica de este paradigma es que solo permite dos opciones en el voto (si/no). Implementaciones conocidas que usan este esquema se discuten en el Capítulo 5.4.

3 Implementación de un modelo de votación electrónica

De los paradigmas de votación electrónica descritos en el Capítulo 2.2.1, el basado en encriptación homomórfica fue considerado el más adecuado para la implementación de un sistema a pequeña escala en el presente trabajo. Los principales motivos son los siguientes:

1. Es eficiente.

2. No requiere canales anónimos.

3. Es escalable (permite muchos votantes simultáneamente).

4. Es comparativamente más simple de implementar.

La implementación del sistema de votación electrónica propuesto requiere de la existencia de un “bulletin board”. En forma abstracta, un “bulletin board” es un canal de comunicaciones compartido "con memoria", esto es, todo lo escrito en ese canal es recibido en forma idéntica por todos los participantes, y además, el canal recuerda los valores enviados. En la práctica, un bulletin board es un servicio de memoria compartida consistente, que permite ser leído públicamente y ser escrito solo por usuarios autorizados, proveyendo un canal de comunicación entre los participantes. Debido a que es accesible en forma pública, el bulletin board es crucial para lograr verificación universal del sistema de votación, dado que las pruebas de correctitud son publicadas en el mismo. Una forma simple de implementarlo es a través de un servidor web, el cual recibe mensajes a publicar, y despliega públicamente el listado de los mensajes ya enviados.

Descripción del protocolo de votación electrónica a implementar

El protocolo en cuestión, es el propuesto por Cramer et al [17] y consiste en las siguientes etapas:

1. Generación de los parámetros públicos que se usaran en los procesos de encriptación y verificación de datos.

2. Proceso de registro. El votante debe generar su clave pública y privada, para ello se utiliza el algoritmo de firma digital DSA visto en el Capítulo 2.1.3. La clave pública es publicada en el bulletin board y la clave privada es conservada en forma protegida por el votante.

3. Generación de claves de las autoridades de votación. Las autoridades utilizan el protocolo de DKG (Capítulo 2.1.6) para generar una clave pública la cual es publicada en el bulletin board.

4. Proceso de emisión de los votos.

   4.1. El votante emite su preferencia en el voto.

   4.2. El voto es encriptado con la clave pública de las autoridades.

   4.3. Al voto se le agrega una prueba de correctitud utilizando un protocolo de ZK (Capítulo 2.1.7).

4.4. El voto encriptado es firmado por el votante. Al voto encriptado junto con la prueba ZK y la firma, se le denomina “papeleta de votación del votante”.

4.5. La papeleta de votación es enviada al bulletin board.

5. Verificación del proceso de emisión de votos. El votante puede verificar que su papeleta se encuentra en el bulletin board (por lo tanto que será contabilizado). Cualquier observador puede verificar la correctitud de cada papeleta de votación utilizando el algoritmo de verificación del protocolo ZK.

6. Cierre del proceso de emisión de votos. Utilizando el paradigma basado en encriptación homomórfica, se multiplican los votos encriptados y se obtiene la encriptación de un único voto con el resultado final de la votación.

7. Verificación del proceso de cierre. Un observador puede verificar la correctitud del proceso, ya que éste es público y repetible.

8. Emisión del resultado final. Las autoridades en conjunto, utilizando el protocolo DKG, desencriptan el voto final, demuestran la correctitud de la operación e informan del resultado.

9. Auditoría. Notar que el proceso completo, más específicamente las etapas 4 y 6, puede ser auditado en forma “online” por un observador externo utilizando los protocolos de verificación respectivos (etapas 5 y 7).

El esquema simplificado de interacciones se resume en la siguiente figura:

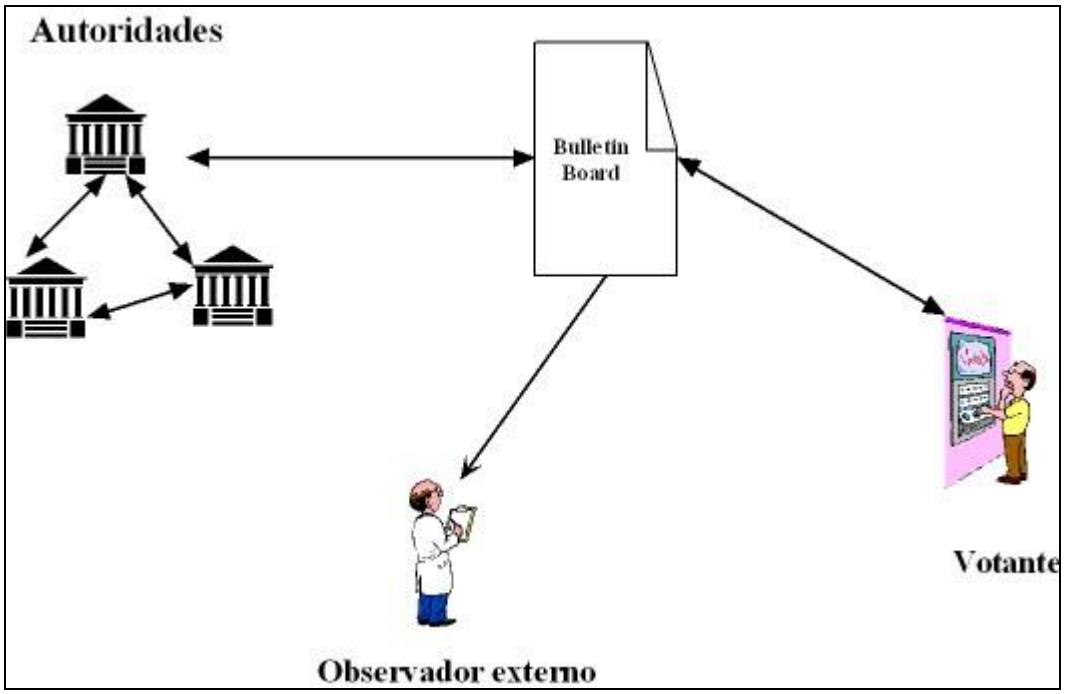

Figura 2: Esquema simplificado de interacción entre participantes

3.1 Herramientas de diseño

Para la implementación del sistema se utilizará el lenguaje de programación Java [19]. Además de las propiedades que hacen de Java un lenguaje de alto nivel y ser éste un lenguaje orientado a objetos, Java provee una plataforma que permite la programación de primitivas criptográficas (JCA [20]), y entrega diversas APIs para encriptación, generación y manejo de claves, generación de números aleatorios entre otros.

3.2 Construcción del sistema

3.2.1 Definición del bulletin board

En este trabajo, el "broadcast con memoria" mencionado en el artículo de Cramer et al [17], es implementado por un bulletin board centralizado.[2]

Se definirá el bulletin board como un archivo público (accesible vía web), en el cuál se publicarán los parámetros iniciales, claves públicas, los votos encriptados y cómputos, además de todos los mensajes públicos que se requieran para auditar el sistema. Para su implementación se utilizará el paquete RMI de Java [21], que permite invocar métodos remotos desde máquinas virtuales Java ejecutándose incluso en servidores distintos.

El esquema simplificado de comunicación es el siguiente:

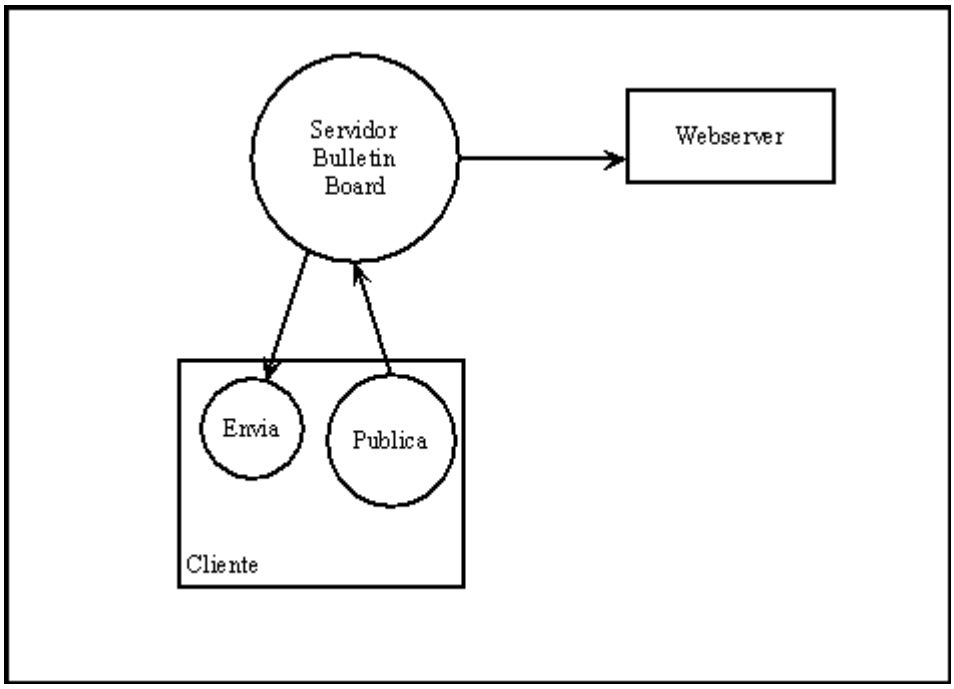

Figura 3: bulletin board

Los clientes que se comunican con el bulletin board pueden ser:

[2] Notar que la implementación del bulletin board puede ser fácilmente distribuida entre n participantes

1. Votantes: publican sus claves públicas y sus votos.

2. Autoridades: publican sus valores DKG, obtienen los votos emitidos y publican los cómputos parciales, el cómputo final y mensajes que permitan auditar y dar transparencia al sistema.

3. Observadores externos: obtienen los valores de votos y cómputos con el fin de verificar su correctitud.

Los valores publicados en el bulletin board son desplegados a través de un servidor web con el fin de entregar transparencia al proceso.

El servidor de bulletin board provee las siguientes interfaces:

```
public interface BulletinBoard extends Remote
{
        // Publica valor en el BB
        public boolean publica(Valor value)
                throws RemoteException;

        // Envia valor al cliente
        public ArrayList<Valor> envia (String pat,int id)
                throws RemoteException;
}
```

El servidor (llamado *bbserver*) implementa los métodos "*publica"* y "*envia"* que permiten escribir un valor y recuperarlos del bulletin board respectivamente. Los valores a escribir tienen el siguiente formato:

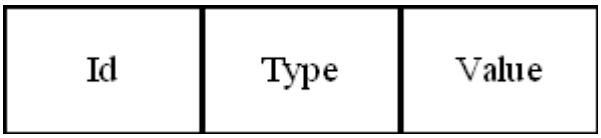

Figura 4: tipo de datos "Valor" usado en el bulletin board

El código que implementa el servidor se encuentra en el Apéndice A de este documento.

Los diversos paquetes que requieran publicar en el bulletin board deben incluir la clase *Public*a, cuyo código es el siguiente:

```
public class Publica {
    String Addr; // dirección IP del servidor

    public Publica(String add) {
       this.Addr=add;
     }

    public void publica(Valor v)  {

        if (System.getSecurityManager() == null) {
            System.setSecurityManager(new SecurityManager());
        }
        try {
            String name = "BulletinBoard";
            Registry registry = LocateRegistry.getRegistry(Addr);
            BulletinBoard  bb = (BulletinBoard) registry.lookup(name);
```

```
            bb.publica(v);
        } catch (Exception e) {
            System.err.println("Publica exception:");
            e.printStackTrace();
        }
    }
}
```

La clase *Publica* implementa el método llamado *publica*, el cual recibe como parámetro un dato de tipo Valor definido en la Figura 4. El método se comunica con el servidor bbserver, que implementa el objeto “BulletinBoard” definido anteriormente, con la siguiente instrucción:

```
BulletinBoard  bb = (BulletinBoard) registry.lookup(name);
```

El código entonces invoca al método *publica* que se encarga de escribir el valor en el bulletin board, con la instrucción:

```
bb.publica(v);
```

Los paquetes que requieran obtener datos desde el bulletin board deben incluir la clase *Recibe*, cuyo código es el siguiente:

```
public class Recibe {
    String Addr;

    public Recibe() {
      this.Addr="<direccion IP>";
     }

    public ArrayList<Valor> leetodo(String pat,int campo) {

        if (System.getSecurityManager() == null) {
            System.setSecurityManager(new SecurityManager());
        }

        ArrayList<Valor> d=new ArrayList<Valor>();

        try {
            String name = "BulletinBoard";
            Registry registry = LocateRegistry.getRegistry(Addr);
            BulletinBoard  bb = (BulletinBoard) registry.lookup(name);
            d=bb.envia(pat,campo);
        } catch (Exception e) {
            System.err.println("Recibe exception:");
            e.printStackTrace();
        }
     return d;
    }
}
```

La clase Recibe implementa una función llamada *leetodo()*. Esta función recibe como parámetros el patrón a buscar en el bulletin board y el campo en el cuál se busca dicho patrón, siendo los valores posibles para campo los siguientes:

- 0: Id
- 1: Type
- 2: Value

El código, al igual que el caso anterior, define el objeto al cuál se comunica (objeto "BulletinBoard") e invoca al método "*envia*" del servidor para recibir todas las líneas que cumplan con la restricción "patrón,campo" solicitadas. La instrucción que permite recibir los valores es:

```
d=bb.envia(pat,campo);
```

La función retorna un conjunto de líneas desde el bulletin board, con el formato de datos Valor de acuerdo a la Figura 4.

3.2.2 Generación de parámetros iniciales

Para el manejo de los valores numéricos se utilizará la clase java.math.BigInteger, la cual contiene métodos para realizar las operaciones requeridas tales como multiplicación, módulo y exponenciación en módulo (función modPow).

Durante todo el proceso se requieren algunos parámetros globales para operar. En una elección real estos parámetros se generan en forma distribuida, sin embargo por simplicidad en el presente trabajo se generan en forma centralizada por alguna autoridad. Los parámetros se utilizarán durante la generación de la clave pública de las autoridades (proceso DKG), para las pruebas de correctitud del voto y durante la generación del cómputo final. Estos parámetros son:

p: primo aleatorio.
q: primo grande, divisor de p-1.
g: generador de un subgrupo G_q de orden q en Z^*_p.
h: potencia de g en G_q.

Los valores utilizados tendrán largo de 1024 bits (bitlength=1024) y para la generación de los números primos se utiliza la constante CERTEZA, la cual es la probabilidad de que el número generado sea primo, escogiéndose una probabilidad del 99,999%. Se utiliza un objeto SecureRandom, el cual provee salidas aleatorias para la generación de los números aleatorios que se requieren a lo largo del proceso.

El valor q por ejemplo, se escoge como:

q=new BigInteger(bitlength-1,CERTEZA,srandom)

El valor de p es calculado como p=2q+1, siempre que p sea primo, de lo contrario se escoge un nuevo valor para q. Este procedimiento es estándar en criptografía. Matemáticamente se desconoce cuantos valores de p y q cumplen con estas propiedades.

El siguiente valor que se requiere es el generador g, para ello se utiliza el siguiente algoritmo [6]:

Se escoge un elemento aleatorio $g \in Z_p$, tal que $g \neq 1$ y $g^q=1 \bmod p$ vale decir que g

es de orden q. La implementación es la siguiente:

```
do {
    //aux es un número aleatorio de 1024 bits, 0 < aux < p-1
    do{
         aux=new BigInteger(bitlength,srandom);
        }while(!(aux.compareTo(BigInteger.ONE)>0 && aux.compareTo(p-1)<0));

} while (aux.modPow(q,p).compareTo(BigInteger.ONE)!=0 &&
aux.modPow(cuadrado,p).compareTo(BigInteger.ONE)!=0); // aux^q=1 || aux^2=1

g=aux.modPow(cuadrado,p); // g=aux^2 mod p
```

Finalmente el valor h se calcula como $h=g^r \bmod p$, con $1 < r < q-1$ aleatorio

Del mismo modo se deben calcular los parámetros para la firma digital DSA (Capítulo 2.1.3), los cuales se utilizarán en la generación de las claves de los votantes y los procesos de firma y verificación de los votos. Estos parámetros se denominan: *qdsa*, *pdsa* y *gdsa* que corresponden a los valores *q,p,g* definidos en el Capítulo 2.1.3.

El valor *qdsa* es un número primo de 160 bits y *pdsa* es un primo de 1024 bits, por lo tanto se calcula pdsa=n*qdsa+1, con n un entero de 864 bits.

Para el cálculo del generador *gdsa* se debe cumplir la propiedad de que $gdsa^{pdsa-1/qdsa} \neq 1$

Todos los valores generados son publicados en el bulletin board, utilizando el campo “value” para almacenar el valor generado y el campo “type” para definir su tipo. Un ejemplo de un valor qdsa en el bulletin board es el siguiente:

```
<id>q</id><type>qdsa</type><value>1387545245517890709630841443151373406887183534039</va
lue>
```

3.2.3 Generación de claves de los votantes

Los votantes poseen una clave privada y una clave pública que les permite firmar el voto usando el algoritmo DSA. Las claves se generan de acuerdo al procedimiento descrito en el protocolo y la clave pública es publicada en el bulletin board con un mensaje que contiene:

- Campo id: identificación del votante
- Campo type: “pubk”
- Campo value: valor de la clave pública del votante

La clave privada debe ser guardada en secreto por el votante y presentada al momento de votar para la firma del voto. En este trabajo las claves privadas se almacenan como un archivo de nombre id.priv en un directorio privado en el equipo que implementa el programa de votación, de modo de poder accederla cuando firma el voto. En una elección real las claves deben almacenarse en algún dispositivo personal del tipo “tamper resistant”, los cuales proveen protección física de las llaves almacenadas, asegurando que no puedan ser maliciosamente leídas o adulteradas [22] como por ejemplo una smart card.

El algoritmo implementado es el siguiente:

```
String keypath=DIR; // Directorio para almacenar las llaves
 String keyfile;

 System.out.println("Ingrese su id");
 id = userIn.readLine();

// Generación de clave privada DSA, valor entre 0 < x < q
 do{
    x=new BigInteger(160,srandom); //valor aleatorio de 160 bits
  }while(!(x.compareTo(BigInteger.ZERO)>0 && x.compareTo(q)<0));

 // Generación de clave pública
 y=g.modPow(x,p);

// publica en el bulletin board
 v=new Valor(id,"pubk",y.toString());
 publica (v);

 // Almacena la clave privada
 keyfile=keypath+id+".priv";
 fout = new FileOutputStream(keyfile);
 pr = new PrintStream(fout );
 pr.println (x.toString());
 pr.close();
```

El programa solicita la identificación del usuario en línea de comandos. En una elección real, la generación de claves puede ser realizada por el usuario y su validez verificada por una autoridad.

El id de la persona se obtiene de la línea de comandos con la siguiente instrucción:

```
System.out.println("Ingrese su id");
id = userIn.readLine();
```

Las claves pública y privada se generan de acuerdo al protocolo DSA con las siguientes instrucciones:

```
// Generación de clave privada DSA, valor entre 0 < x < q
do{
   x=new BigInteger(160,srandom); //valor aleatorio de 160 bits
 }while(!(x.compareTo(BigInteger.ZERO)>0 && x.compareTo(q)<0));

// Generación de clave pública
y=g.modPow(x,p);
```

La clave pública es enviada al bulletin board invocando a la clase *Publica* vista en el Capítulo 3.2.1. Las instrucciones son las siguientes:

```
// publica en el bulletin board
v=new Valor(id,"pubk",y.toString());
publica (v);
```

Finalmente la clave privada del usuario es almacenada en un archivo en el sistema.

3.2.4 Protocolo DKG

En la implementación de este protocolo (Capítulo 2.1.6), asumimos n participantes

(autoridades) ejecutando el protocolo en diferentes computadores. Cada autoridad debe ejecutar el mismo código e intercambiar información sin utilizar un servidor intermedio. El bulletin board sirve como mecanismo para enviar mensajes de broadcast a las demás autoridades y como mecanismo de transparencia del sistema. El modelo implementado sigue un esquema cliente/servidor utilizando RMI. En este esquema cada autoridad juega ambos roles, por lo que existe un proceso encargado de recibir datos de otras autoridades y otros encargados de enviar datos. El esquema se muestra en la siguiente figura:

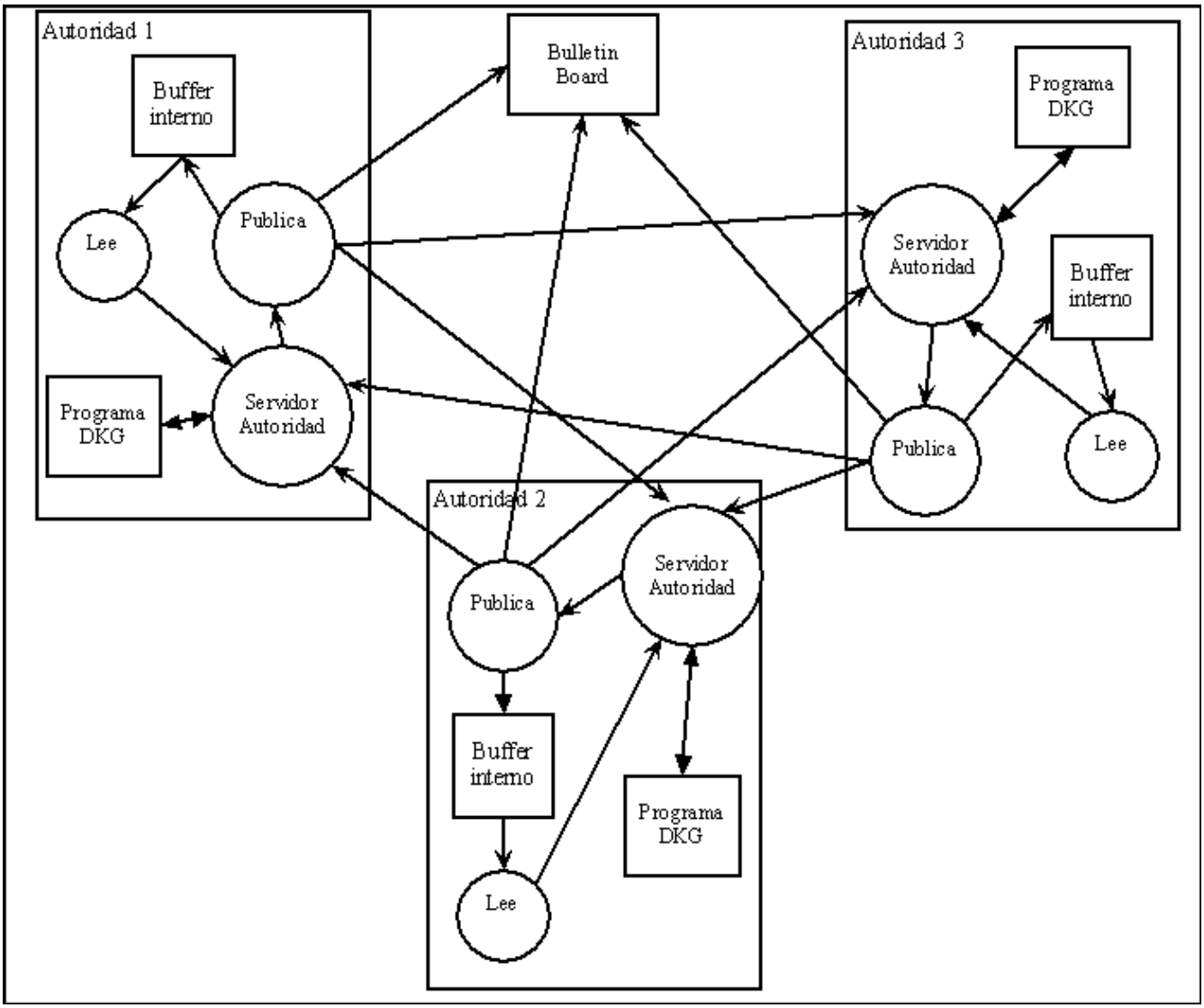

Figura 5: protocolo DKG

La descripción de procesos es la siguiente:

1) Servidor Autoridad: Este proceso se encarga de escuchar, en la dirección IP de la máquina correspondiente a la autoridad, los mensajes enviados tanto por las otras autoridades como por sí misma.

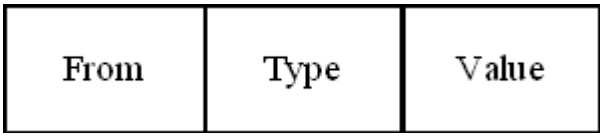

Figura 6: Tipo de dato "Dato" usado en los mensajes del protocolo DKG

Un mensaje DKG tiene el formato que muestra la Figura 6, donde los campos definidos son los siguientes:

From: La identificación (id) de la autoridad que envía el mensaje.

Type: El tipo de dato enviado. Los tipos se califican usando los caracteres:

- S: valor compartido S definido en DKG.
- S1: valor compartido S' definido en DKG.
- a,b: valores privados de los polinomios f () y f'().
- C: valores C definidos en DKG.
- A: valores A definidos en DKG.
- Q: Queja enviada por una autoridad a todos los participantes respecto a los valores recibidos en el primer paso del protocolo.
- QR: Respuesta de una autoridad a una queja Q.
- DISQ: Descalificación (cuando se recibe se usa para eliminar un participante del conjunto QUAL).
- Q5: Queja respecto a los valores de verificación en el paso 5 del protocolo.

Value: Valor del mensaje recibido.

El siguiente es un ejemplo de un mensaje de tipo S enviado por la autoridad A2:

```
<from>A2</from><type>S</type><value>5557596106933040550870675962602138700668847458059641091648060022059575002821771466264914259678841826722649532961380600815335620056206361073797131629497213279575418148187235667226056313230238465546162586224688553867326869666224479270391319727478476656881421535126720928745850508622994458853263374417405557053 9</value>
```

2) Buffer interno: Se implementa como un archivo privado de cada autoridad donde se almacenan los datos recibidos desde los diversos participantes, de modo de poder compartir información entre los procesos que escriben y que leen desde dicho archivo.

3) Publica: Este proceso se encarga de enviar los datos a cada autoridad o al bulletin board. Este proceso invoca el método "envia" de la autoridad para su publicación en el buffer interno.

4) Lee: Este proceso obtiene un dato local desde el buffer interno, invocando el método "recibe" del servidor autoridad.

5) Programa DKG: El protocolo DKG considera un extenso intercambio de mensajes entre las autoridades para finalmente obtener la generación de las claves de encriptación/desencriptación requeridas. El programa DKG implementado simplifica el problema de la sincronización de eventos de lectura/escritura. En este proceso una

autoridad debe verificar los datos recibidos por otra una vez que haya recibido la totalidad de ellos en un tiempo determinado. La implementación considera la división del protocolo en fases, las cuales se ejecutan secuencialmente. Existen 6 programas (pasos) que se ejecutan secuencialmente en cada autoridad, hasta finalizar el protocolo.

La definición de los programas es la siguiente:

a. Paso 1
    i. Genera los polinomios aleatorios.
    ii. Envía por broadcast los valores C generados.
    iii. Envía los valores S y S' a cada autoridad.
b. Paso 2
    i. Lee los datos recibidos por cada autoridad.
    ii. Verifica los datos y envía una queja por broadcast en caso de que la verificación falle para una autoridad.
c. Paso 3
    i. Chequea si se recibió una queja en su contra, si es así envía los valores S y S' correctos a todas las autoridades.
d. Paso 4
    i. Construye el conjunto de participantes QUAL.
    ii. Envía por broadcast los valores A generados.
e. Paso 5
    i. Lee los valores recibidos por cada autoridad.
    ii. Verifica los datos y envía una queja por broadcast en caso de que la verificación falle para una autoridad.
f. Paso 6
    i. Reconstruye valores en caso de quejas, usando la ecuación 1 de reconstrucción de Shamir.
    ii. Genera la clave pública de las autoridades.

Las autoridades pueden comunicarse con otra autoridad (o sí mismas) utilizando clases que se implementan en forma similar a las clases *Publica* y *Recibe* del bulletin board (Capítulo 3.2.1).

El servidor se encarga de implementar los métodos "envia" y "recibe" definidos por la clase *Autoridad* bajo RMI. La definición de autoridad es la siguiente:

```
package AUT;

import java.rmi.Remote;
import java.rmi.RemoteException;
import java.util.ArrayList;

public interface Autoridad extends Remote
{
        // envia a alguna autoridad
        public boolean envia (Dato dat)
                throws RemoteException;

        // busca valores recibidos de otras autoridades
        public ArrayList<Dato> recibe (String pat,int campo)
                throws RemoteException;
}
```

El servidor de las autoridades (llamado autserver) implementa estas interfaces ("envia" y

“recibe”), y su código es similar al de bulletin board del Apéndice A.

A modo de ejemplo se explica el código que implementa el paso 4 del protocolo:

El programa principal que implementa el paso 4, ejecuta las siguientes funciones:

```
leeinitparams();
chequea_quejas();
chequea_respuestas();
publica_qual();
genera_As();
broadcast_As();
```

La explicación de cada función es la siguiente:

a) leeinitparams(): recupera los parámetros iniciales desde el bulletin board (p,q,g y h).

b) chequea_quejas(): revisa las quejas recibidas por una autoridad, si son más de t quejas la autoridad queda descalificada. El código es el siguiente:

```
public void chequea_quejas() {
 ArrayList<Dato> d=new ArrayList<Dato>();
 Lee lee;
 String aux;
 int j;
 int disq[]=new int[n]; // arreglo para las autoridades descalificadas. Si el
valor es 1 está descalificada.

// inicializa arreglo de autoridades descalificadas, inicialmente todas son
válidas
 for (int i=0; i<n;i++)
  disq[i]=0;

 // lee los mensajes de tipo Q recibidos
 lee=new Lee();
 d=lee.leetodo("Q",1);

// obtiene el Id de la autoridad con mensaje de descalificación
for (int i=0; i<d.size();i++)   {
  aux=Character.toString(d.get(i).value.charAt(1));
  j=Integer.parseInt(aux);
  disq[j]++;
 }

// chequea si hubo más de t quejas
  for (int i=0; i<n;i++)
   if(disq[i]>t)  // más de t quejas?
     prequal[i]=0;   // marca como descalificado en el arreglo prequel, que
contendrá a las autoridades válidas para formar el conjunto QUAL
}
```

c) chequea_respuestas(): recupera los mensajes de respuesta a las quejas (mensajes de tipo QR) y valida la ecuación de verificación de los valores recibidos. El código es el siguiente:

```
public void chequea_respuestas() {
ArrayList<Dato> d=new ArrayList<Dato>();
Lee lee;
String aux,type,id;
int from;
```

```
   c=new BigInteger[t+1];

   lee=new Lee();
   d=lee.leetodo("QR",1);

   // Recorre todos los mensajes de respuesta recibidos
   for (int i=0;i<d.size();i++)
    {
      type=d.get(i).value;
      id=Character.toString(d.get(i).from.charAt(1));
      from=Integer.parseInt(id);

      get_s(type);  // recupera los valores S recbidos
      get_c(from);  // recupera los valores C recibidos

      // la función verifica implementa la ecuación del paso 4 DKG
      if (!verifica(s,s1,c))
        prequal[from]=0; //descalificado

    }
}
```

d) publica_qual(): publica en el bulletin board el conjunto QUAL con los participantes previamente validados por la autoridad. El siguiente es un mensaje de ejemplo, con el que la autoridad A0 publicaría su QUAL en el que las autoridades 0,1,2 son válidas:

```
<id>A0</id><type>QUAL</type><value><q>0</q><q>1</q><q>2</q></value>
```

El código es el siguiente:

```
public static void  publica_qual() {
      Publica pub;
      Valor v;
      String dat="";
      int j=0;

      pub=new Publica(BBAddr); //BBAddr dirección IP del bulletin board

      for (int i=0;i<n;i++) {
        if (prequal[i]==1) { // autoridad válida
           qual[j++]=i;
         }
        else { // autoridad descalificada
          v=new Valor("A"+MYID,"DISQ",String.valueOf(i)); //publica un mensaje
de tipo DISQ en el bulletin board
          pub.publica(v);
        }
       }

     // Construye conjunto QUAL y lo publica
      for (j=0;j<n;j++) {
        if(qual[j]!=-1)
          dat=dat+"<q>"+qual[j]+"</q>";
       }

      v=new Valor("A"+MYID,"QUAL",dat);
      pub.publica(v);
    }
```

e) genera_As(): Genera los valores A del protocolo. El código es el siguiente:

```
public void genera_As() {
  A=new BigInteger[t+1];

  recupera_as(); // recupera los valores privados a del polinomio f de la Autoridad

  for (int k=0;k<t+1;k++)
    A[k]=g.modPow(a[k],p);
}
```

f) broadcast_As(): envía los valores A generados a cada autoridad publicando en el bulletin board

```
public void broadcast_As() {
   Publica pub;
   Valor v;
      pub=new Publica (BBAddr);

      for(int k=0;k<t+1;k++) {
         v=new Valor("A"+MYID,"A"+MYID+Integer.toString(k),A[k].toString());
         pub.publica(v);
       }
     }
   }
```

3.2.5 Construcción de la papeleta de votación

Una papeleta de votación (ballot) debe contener los siguientes datos al ser publicado:

1. Voto encriptado: es el texto cifrado ElGamal con la preferencia del votante.
2. Proof: la prueba de correctitud (ZK) del voto (Capítulo 3.2.5.1)
3. Id: la identificación del usuario, por ejemplo el rut. Se requiere para verificar la firma y para la proof.
4. Firma: firma del usuario usando DSA (Capítulo 2.1.3)

El programa de votación solicita la preferencia del usuario, la cual es mapeada a los valores {1,-1}, encripta el voto con la clave pública de las autoridades, genera tanto la proof como la firma del usuario y construye el voto. La papeleta de votación se publica en el bulletin board con la siguiente estructura:

ID	Type	Value
IdVotante	B	c1,c2,a1,b1,a2,b2,d1,d2,r1,r2,r,s

Figura 7: Formato de la papeleta de votación en el bulletin board

El valor “B” en el campo “Type”, define el tipo de mensaje como una papeleta de votación.

Los valores que componen el campo “Value” son los siguientes:

- c1,c2: Preferencia del usuario cifrada con ElGamal usando la clave pública de las autoridades.

- a1,b1,a2,b2,d1,d2,r1,r2: Valores para la demostración de correctitud del voto (protocolo de Zero Knowledge) según algoritmo de Chaum-Pedersen (Capítulo 3.2.5.1).

- r,s: Valores de la firma del usuario usando DSA .

Por ejemplo, una papeleta de votación del votante con id 100 sería recibido en el bulletin board con la siguiente estructura:


```
<id>100</id><type>B</type><value><c1>32063242165479899470469885728028705506566564375972498552030156607304221378599243610645211610420193143836674186580104001988673012693075742324197997697763111016730945672609330273644631626033795836264729951673396773399044326145270784405680744945381695580909633820976017292436996998216179464727295274616189920171433</c1><c2>807326343890834552870640118677875974597956541505757226533229343068269916405064206283495176713091349914845367215928895030759907724648954369980898922649779441649664535066973152840995460948395753289445008276768841977323183129900175033175648811402758616486790045942492218781801747474144474021257421649672404419218</c2><a1>26104370487685245461250357846702755612160929290559713947830187958707796578433661447071525418012810109594360807383156310723042685395273778042157998005976037457216067009753006396283336629932635845905564287311697715562663594316543743802247057110283890716089466316733546643317285617419182542247150325508370299965876</a1><a2>61170536690531632664015223222148109921693439718665674451360672886186159737246288983519400681595961840357132277865861747707278331026434838242560854191346184726926044660396440769771811621704254439977131644839323911657485387027274312799931287496424076987762243231062504981289095329922494182585804249872396213064199</a2><b1>12822665603316227061472199671296467739756327631336844494853075022442907599243036332400869247748022110461734640017707244137892646277510982446385213252233663587473717124842037003139246154017845610230853473436715069629132029576232193797752490636581473491921875995033034095425783702484982375280389464808396110784400</b1><b2>31979372682331278217631235892177663570420757054461199806828613603504504173811654768023160851374140954575858268335501008368043109881479708612715502784065780243390688087182400090109757981298093980481967212475259471701057218941628819754364492538847199970703467838575568136455797448725377022932049203810530325482513</b2><d1>517351308983260796390397455959503727218335339111954077674381951857570830764788935335290452745925652301810957985602267118386684924984871941804803592693680702609816101694929912757585377545854820974920401248482927733438025389283134686677348217312146404662031859134448741328124116416895272962585520895396307451364</d1><d2>1179321743680533750911402846266613848320179282548099486666581440864624443009994644911893486452756027755319350043536229950280740939962606153025065579131920359975836988597265370017627541250295978546421979788453105285459062575332204603122698231151460645068749888590734079544872348003711664444197230577054967729</d2><r1>1053970870978855894254790166615137345482987239293942094596692285536090938176434376955881779600492476589576917728124140356913225054023632094918758194842661872898822512226348994373069222779160746893025706631207466930853766132819791122883588323634923161201260011248473101945020084439335538202106241421247382088 9</r1><r2>353347596128743874511712134417909516903795317023621520228295972640222819388127969060973210982264671792418248267164927412364355780819154001100552908439153353500665368067815569672268064562400006532307921809016931289291709010411361762468452888787772616846735584870969697993037273396326498099813698455522578666 39</r2><r>479791226255729363943654640602109999510962953745</r><s>1176236552875941692771539378687260517680333644545</s></value>
```

3.2.5.1 Demostración de correctitud de una papeleta de votación (Chaum-Pedersen)

Para validar la papeleta de votación se utiliza el protocolo de Chaum-Pedersen [18] con el fin de verificar que un voto v sea correcto. Este protocolo es descrito a continuación.

El protocolo puede entenderse como la comunicación entre el "prover" (*P*), en este caso el votante, y el verificador (*V*). Sea el voto $v \in \{1,-1\}$, entonces el algoritmo es el siguiente:

Si v=1, entonces:

1. *P* genera:

   r, w, r_1, d_1 al azar en Z_q
   $(a, b) \leftarrow (g^r, y^r h)$ (ElGamal)
   $a_1 \leftarrow g^{r_1} a^{d_1}$
   $b_1 \leftarrow y^{r_1} (bh)^{d_1}$
   $a_2 \leftarrow g^w$
   $b_2 \leftarrow y^w$

2. *P* envía a *V*: a, b, a_1, b_1, a_2, b_2

3. *V* genera c al azar en Z_q y lo envía a *P*

4. *P* genera:

   $d_2 \leftarrow c - d_1$
   $r_2 \leftarrow w - r d_2$

5. *P* envia a *V*: d_1, d_2, r_1, r_2

6. Prueba de correctitud

   Si todo lo siguiente se cumple:

   $c = d_1 + d_2$
   $a_1 = g^{r_1} a^{d_1}$
   $a_2 = g^{r_2} a^{d_2}$
   $b_1 = y^{r_1} (bh)^{d_1}$
   $b_2 = y^{r_2} (bh^{-1})^{d_2}$

   Entonces la papeleta es correcta, de lo contrario es inválida.

Si v=-1 entonces:

1. *P* genera:

   r, w, r_2, d_2 al azar en Z_q
   $(a, b) \leftarrow (g^r, y^r h^{-1})$ (ElGamal)
   $a_1 \leftarrow g^w$
   $b_1 \leftarrow y^w$
   $a_2 \leftarrow g^{r_2} a^{d_2}$
   $b_2 \leftarrow y^{r_2} (bh^{-1})^{d_2}$

2. *P* envía a *V*: a, b, a_1, b_1, a_2, b_2

3. *V* genera c al azar en Z_q y lo envía a *P*

4. P genera:

   $d_1 \leftarrow c - d_2$
   $r_1 \leftarrow w - rd_1$

5. *P* envía a *V*: d_1, d_2, r_1, r_2

6. La prueba de correctitud es la misma que en el caso de v=1

En [17,18] se demuestra que este protocolo es ZK ante un verificador honesto y además constituye una proof of knowledge.

3.2.5.2 Implementación de prueba de correctitud del voto

La prueba de correctitud del voto requiere que exista interacción entre el votante ("prover") y el verificador ("verifier"). La idea es que el proceso sea independiente, de modo que el votante genere la prueba de correctitud y cualquier entidad externa pueda ejecutar la validación. Del algoritmo de Chaum-Pedersen (Capítulo 3.2.5.1), se aprecia que solo el cálculo del valor "c" requiere interacción entre ambas entidades. Para evitar esto se implementa un valor "c" calculable por ambas partes, lo cual puede realizarse definiendo el valor "c" como: c=Hash(IdVotante,α,β,a1,a2,b1,b2) . Esta modificación se conoce como "heurística Fiat-Shamir" [23,24] y puede justificarse formalmente en el modelo de oráculo aleatorio [25]. La función de Hash utilizada es SHA-1 la cuál entrega una salida de 160 bits. Para conseguir un valor de 1024 bits se realizan sucesivas concatenaciones de la salida de la función, agregando en cada ocasión un valor nuevo (un contador), hasta completar los 1024 bits:

c=Hash(x)=sha1(x|1)||sha1(x|2)||…

Esto se realiza con el fin de garantizar que "c" sea públicamente verificable.

Para validar el voto simplemente deben recuperarse los valores desde el bulletin board y realizar las siguientes validaciones. Por ejemplo el código que valida el voto es el siguiente:

```
Sha1 sha=new Sha1();
c=sha.getsha1(id+c1+c2+a1+a2+b1+b2); //obtiene el hash utilizando sha-1 y
```

```
concatenando
t1=d1.add(d2);
t2=g.modPow(r1,p).multiply(c1.modPow(d1,p));
t3=g.modPow(r2,p).multiply(c1.modPow(d2,p));
t4=y.modPow(r1,p).multiply(c2.multiply(h).modPow(d1,p));
t5=y.modPow(r2,p).multiply((c2.multiply(h.modInverse(p))).modPow(d2,p));

 // Verificación de los test
if (c.compareTo(t1.mod(p)) !=0 || a1.compareTo(t2.mod(p))!=0 ||
a2.compareTo(t3.mod(p))!=0 || b1.compareTo(t4.mod(p))!=0  ||
b2.compareTo(t5.mod(p))!=0 )
       return 2;   // voto no valido
```

La clase Sha1 implementa la función de hash SHA-1 y la función getsha1 entrega el valor de 1024 bits buscado. La instrucción es la siguiente:

```
Sha1 sha=new Sha1();
 c=sha.getsha1(id+c1+c2+a1+a2+b1+b2);
```

Los valores t1 a t5 definen los valores de las pruebas a validar. Finalmente se comprueban que se cumplan las cinco igualdades de la prueba de correctitud. El valor de retorno “2” especifica que la comprobación no fue exitosa.

3.2.6 Cómputo final

3.2.6.1 Descripción general

La obtención del cómputo final se realiza utilizando la clave privada de las autoridades, la cual nunca es formalmente calculada. Cada autoridad A_i posee una clave privada x_i la cual es calculada durante el protocolo DKG (Capítulo 2.1.6). Cada autoridad usa su clave x_i para desencriptar (en forma distribuida) la encriptación del cómputo final.

El algoritmo a implementar consiste en que cada autoridad perteneciente a QUAL genera un cómputo parcial, para ello obtiene del bulletin board los votos emitidos y los verifica.

Para que una papeleta de votación sea correcta debe cumplir con:

1. La firma del voto debe ser verificada.

2. La prueba de correctitud del voto definida en el Capítulo 3.2.6.3 debe ser exitosa.

En caso de que el voto no sea correcto, la autoridad publica una queja en el bulletin board con la razón por la cual no considera el voto.

3.2.6.2 Obtención del cómputo final

Sea (α_j,β_j) el j-ésimo voto encriptado cuya papeleta de votación fue declarada válida en el proceso anterior.

De acuerdo a ElGamal la desencriptación de un mensaje v_j (voto) cifrado como (α_j,β_j) es igual a: $v_j=\beta_j\alpha_j^{-x}$ con x la clave privada de las autoridades.

Se definen los valores: $d = \prod b_j \bmod p$ y $g = (\prod a_j)^x \bmod p$

De acuerdo a la propiedad homomórfica de ElGamal, se tiene que la desencriptación de la multiplicación de los votos es:

$$D(\prod v_j) = dg^{-1} \bmod p$$

La clave privada x no es conocida por ninguna autoridad y aunque es posible reconstruirla como $x = \sum_{i \in QUAL} x_i$, ello no se realiza ya que ninguna autoridad conoce los valores privados de las demás autoridades.

Cada autoridad debe publicar una desencriptación parcial usando su propia clave privada x_i y luego g puede obtenerse combinando los valores resultantes.

En resumen, la autoridad, con los votos que validó como correctos, publica en el bulletin board los siguientes valores:

1. El valor δ correspondiente a la multiplicación de β_j

$$d = \prod_{i \in votosválidos} b_j$$

2. El valor g correspondiente a la multiplicación de los α_j

$$g = (\prod_{i \in votosváidos} a_j)^{x_i}$$

3. La prueba de igualdad de logaritmos descrita en el Capítulo 3.2.6.3 a fin de demostrar que la clave privada x_i usada es consistente con la clave usada el protocolo DKG.

El programa que ejecuta el cómputo parcial debe conectarse al servidor de autoridad (*autserver*) local para recuperar la clave privada x_i de desencriptación y además al bulletin board para obtener todos los votos.

El cómputo final de la votación puede ser calculado por cualquier entidad externa como:

$V=\delta\gamma^{-1} \bmod p$ donde $g = \prod_{i \in QUAL} g_i \bmod p$

Como cada voto $v \in \{1,-1\}$ fue representado por h^v se tiene que: $V = h^{\sum v}$

Si $s=\sum v$ entonces el resultado final se consigue resolviendo la ecuación $V=h^s \bmod p$ donde V y h son conocidos. Dado que el número de votantes es conocido también, se efectúa una resolución iterativa de la ecuación con todos los resultados posibles hasta encontrar el valor de s correcto. Una vez resuelta la ecuación es simple concluir tanto el resultado final como el número de votantes por cada opción.

Nótese además, que el resultado final s puede ser verificado por un observador externo, en el sentido de validar la relación $V=h^s \bmod p$.

3.2.6.3 Demostración de igualdad de logaritmos (Chaum-Pedersen)

Se requiere demostrar que el valor privado x usado por la autoridad para generar el valor y_i (clave pública de la autoridad, Capítulo 2.1.6), que se utilizará en la generación de la clave pública, sea el mismo que se use para desencriptar. El protocolo es el siguiente:

La autoridad ejecuta: $(S_1,S_2)=(g^x,\alpha^x) \bmod p$.

Sean P (prover) y V (verifier), entonces:

1. P genera :

   w al azar en Z_q.

   (a,b) ß $(g^w,\alpha^w) \bmod p$.

2. P envía a,b a V.

3. V genera c al azar en Z_q y lo envía V.

4. P calcula: r ß w + xc mod q y lo envía a V.

Prueba de correctitud: Si lo siguiente se cumple, entonces el procedimiento es correcto:

$g^r=a*S_1^c \bmod p$.
$\alpha^r=b*S_2^c \bmod p$.

Para la implementación se define *c=H*ash(IdAutoridad,S_1,S_2,a,b) y se opera de la misma forma que en el Capítulo 3.2.5.2.

3.2.6.4 Implementación del cómputo parcial

Cada autoridad debe generar un cómputo parcial, el cuál contiene la desencriptación ElGamal del total de votos válidos, utilizando su propia clave privada. Dicho cómputo es enviado al bulletin board además del total de votos válidos y los mensajes que anuncian el descarte de los votos inválidos que encuentre (por firma o correctitud). El mensaje de cómputo tiene la siguiente estructura:

ID	Type	Value
Id Autoridad	COMP	gama,alfas,delta,s1,a,b,r

Figura 8: Formato de mensaje de cómputo parcial en el bulletin board

El valor “COMP” en el campo “Type” define el tipo de mensaje como un cómputo parcial.

Los valores del campo “Value” son los siguientes:

- gama: Corresponde a la multiplicación de los valores c2 de las papeletas de votación.
- alfas: Corresponde a la multiplicación de los valores c1 de las papeletas de votación. Este valor es publicado por transparencia. Tanto gama como alfas deben ser iguales para cada autoridad.
- delta: Corresponde al valor de alfas "desencriptado" con la llave privada de la autoridad.
- S1: Corresponde al valor S1 para la igualdad de logaritmos. $S1=g^x \bmod p$.
- a,b,r: Valores de acuerdo a lo visto en el Capítulo 3.2.6.3 de igualdad de logaritmos.

El programa implementado lee todos los votos, los valida y calcula los valores a publicar. El programa completo que implementa el cómputo parcial se encuentra en el Apéndice B de esta memoria.

3.2.6.5 Implementación del cómputo final

El cómputo final consta de dos pasos previos: primero se comprueba la validez de la prueba ZK de igualdad de logaritmos (Capítulo 3.2.6.3), en caso de que la proof sea incorrecta se envía un mensaje al bulletin board. El segundo paso implica que cada autoridad revise si hay alguna queja contra ella, de ser así la autoridad envía sus valores privados S, S' a todas las demás de modo de poder reconstruir su clave privada del mismo modo que la fase de reconstrucción en el paso 6 de DKG.

Una vez finalizados los pasos previos se calcula el cómputo final. La función principal se implementa con el siguiente código:

```
public void computofinal (ArrayList<Valor> dat)
  {
    String val,c1,c2;
    int nv,res;
    boolean found=false;
    BigInteger alp,v,c2tot=BigInteger.ONE;
    BigInteger z,x;
    BigInteger alfa=BigInteger.ONE;

    for (int i=0;i<dat.size();i++) // recorre todos los computos existentes
        {
          id=dat.get(i).id; // obtengo el id de la autoridad
          if (inqual(id)) {   // si es la autoridad es válida
          val=dat.get(i).value;
          if (valido(dat.get(i).value,id)) {  //valida igualdad de logaritmos
             c1=parsea(val,"gama");  //parsea obtiene el valor entre <gama> y </gama>
             c2=parsea(val,"delta");
             c2tot=c2tot.multiply(new BigInteger(c2));
             c2=parsea(val,"alfas");
             alfa=new BigInteger(c2);
          }
         else {
           System.out.println(id+" computo no verificado");
           z=recontruye_z();  // reconstrucción de Shamir
           if (alfa.compareTo(BigInteger.ONE)!=0) {
              BigInteger d=alfa.modPow(z,p);
              c2tot=c2tot.multiply(d);
              }
            }
```

```
         }
        }
    alp=new BigInteger(c1);
    c2tot=c2tot.mod(p);
    v=alp.multiply(c2tot.modInverse(p)); //desencripta el voto (ElGamal)
    v=v.mod(p);

    nv=total_votantes(); // obtiene el número de votantes desde el bulletin board

    // encuentro el valor del logaritmo recorriendo los posibles valores, desde –nv a
nv
    for (int i=0;i<nv+1;i++)
        if (v.compareTo(h.pow(i).mod(p))==0) {
            found=true;
            res=i;
            break;
         }
     if (!found) {
        for (int i=1;i<nv+1;i++)
          if (v.compareTo(h.pow(i).modInverse(p))==0){
             res=-i;
             break;
           }
      }


     // entrego los resultados finales y porcentajes
     int opa=(nv+res)/2;
     int opb=Math.abs(res-opa);
     float pa=(opa*100)/nv;
     float pb=(opb*100)/nv;

}
```

El primer paso es obtener los cómputos parciales de todas las autoridades en el QUAL. Cada cómputo es verificado de acuerdo a la prueba ZK del Capítulo 3.2.6.3. Si es válido entonces se agrega el valor delta a la multiplicación de valores. El código es el siguiente:

```
c1=parsea(val,"gama");  // Obtiene el valor c1 entre <gama> y </gama>
c2=parsea(val,"delta"); // Obtiene c2
c2tot=c2tot.multiply(new BigInteger(c2)); //Multiplicación de los valores c2
```

Si el cómputo no es verificado, se reconstruye la clave privada de la autoridad usando la Ecuación 1 de reconstrucción de Shamir (Capítulo 2.1.4) y se genera el valor delta para agregarlo a la multiplicación de los votos encriptados. Las instrucciones son las siguientes:

```
z=recontruye_z();  // reconstrucción de Shamir
BigInteger d=alfa.modPow(z,p); // d=alfa^z mod p
c2tot=c2tot.multiply(d);
```

Una vez finalizado el proceso se procede a desencriptar la suma de los votos. Las instrucciones son las siguientes:

```
alp=new BigInteger(c1); // valores gama
c2tot=c2tot.mod(p); // valores delta
v=alp.multiply(c2tot.modInverse(p)); //v=alp*c2tot^-1 mod p (ElGamal)
```

El valor v obtenido corresponde a h elevado a la sumatoria de los votos. Para resolver la ecuación se itera sobre todos los valores posibles de la elección. El código es el siguiente:

```
          nv=total_votantes(); //Obtiene el número de votantes desde el bulletin board

         //Los valores posibles de la sumatoria de los votos van entre nv (si todos
votaron 1) y –nv (si todos votaron -1)
         // primero itero entre 0 -> nv
          for (int i=0;i<nv+1;i++)
            if (v.compareTo(h.pow(i).mod(p))==0) {
              found=true;
              res=i;
              break;
             }
            if (!found) {
           // iteramos ente –nv -> -1
           for (int i=1;i<nv+1;i++)
             if (v.compareTo(h.pow(i).modInverse(p))==0){
                res=-i;
                break;
              }
           }
```

El valor “res” contiene la suma final de la votación (votos 1 + votos -1). El resultado final por cada opción se obtiene con las siguientes instrucciones:

```
         // Total de votantes que optaron por la opcion 1: (nv + res)/ 2
          int opa=(nv+res)/2;

          // Total de votantes que optaron por la opcion -1: abs (res-opa)
          int opb=Math.abs(res-opa);

          // porcentajes por cada opcion
          float pa=(opa*100)/nv;
          float pb=(opb*100)/nv;
```

3.2.7 Auditorías

El sistema de votación permite a un usuario externo realizar las siguientes auditorías:

3.2.7.1 Validación de una papeleta de votación

El algoritmo que implementa el protocolo de Chaum-Pedersen (Capítulo 3.2.5.2) permite que un usuario pueda validar que el voto encriptado en una papeleta de votación sea efectivamente un voto válido. Además se debe verificar la firma del voto. Este algoritmo es ejecutado por las autoridades al momento del cómputo final y puede ser ejecutado por cualquier entidad externa.

3.2.7.2 Validación del proceso de desencriptación

El algoritmo que implementa el protocolo de Chaum-Pedersen de demostración de igualdad de logaritmos (Capítulo 3.2.6.3) permite probar que los valores usados para generar la clave pública en el proceso DKG sean los mismos utilizados al momento de desencriptar. Este algoritmo es utilizado por las autoridades al momento del cómputo final y puede ser ejecutado por cualquier entidad externa.

4 Evaluación de la implementación

El proceso de votación puede ser llevado a cabo ejecutando los siguientes pasos:

1. El servidor de bulletin board levanta su proceso para atender requerimientos (*bbserver*).

2. Cada autoridad levanta su proceso servidor para atender los requerimientos de las otras autoridades (*autserve*r).

3. Se ejecuta el proceso de generación de parámetros iniciales.

4. Cada autoridad ejecuta los pasos de DKG para generar el QUAL y la clave pública de las autoridades.

5. Una vez finalizado DKG puede iniciarse el proceso de votación. Las máquinas clientes ejecutan el programa de votación permitiendo que los usuarios ingresen su preferencia y el voto sea publicado en el bulletin board.

6. Una vez finalizada la votación cada autoridad genera el cómputo parcial con los votos considerados válidos.

7. Finalmente cualquier autoridad puede generar el cómputo final.

Finalizado el proceso es posible validar la corrección del mismo ejecutando los programas de validación de votos y de validación de la desencriptación.

En las votaciones de prueba realizadas se utilizaron 3 máquinas, cada una con dirección IP distinta dentro de una red local. La descripción de los equipos utilizados es la siguiente:

Identificación	Hardware	Tareas
A0	1. CPU: 1 procesador Intel Pentium D 3.0 GHz 2. Memoria: 2GB Ram 3. Dirección IP: 172.17.69.120	1. Servidor de bulletin board 2. Generador de parámetros iniciales 3. Generador de claves de usuario 4. Autoridad 1 5. Máquina de votación
A1	1. CPU:1 procesador Intel Pentium 4 3.0 GHz 2. Memoria: 1GB Ram 3. Dirección IP: 200.9.100.33	1. Autoridad 2

A2	1. CPU: 2 procesadores Intel Xeon 2.5GHz 2. Memoria: 2GB Ram 3. Dirección IP: 172.17.40.7	1. Autoridad 3

Para iniciar el proceso de votación el bulletin board debe contener la identificación de todas las autoridades participantes, en este caso el bulletin board tendría los siguientes valores antes de comenzar el proceso:

```
<id>0</id><type>AUTORIDAD</type><value>172.17.69.120</value>
<id>1</id><type>AUTORIDAD</type><value>200.9.100.33</value>
<id>2</id><type>AUTORIDAD</type><value>172.17.40.7</value>
```

4.1 Tiempo de generación de parámetros iniciales

Se realizaron 100 procesos de generación de todos los parámetros iniciales con el fin de determinar el tiempo que tardaba el proceso. Los resultados son los siguientes (en segundos):

Promedio	Tiempo máximo	Tiempo Mínimo	Desviación estándar
141.07	1893.50	8.72	266.39

4.2 Tiempo de generación de cómputo final

Se simuló una votación generando algunos votos aleatorios y replicándolos con el fin de realizar pruebas de cómputo con un número de votos variable.

Debido a que el proceso de generación del cómputo final implica iterar entre todos los resultados posibles, con el fin de entregar el resultado final (solución del logaritmo discreto), se tomaron los tiempos para diversos resultados, siendo los resultados promedios de 10 ejecuciones los siguientes:

Número de votantes	Tiempo computo parcial (segundos)	Tiempo computo final (segundos)
100	30.85	0.77
500	168.1	0.97
2000	932.8	1.86

Se observa que el mayor costo de tiempo es la lectura y validación de cada voto (cómputo parcial), el cuál es proporcional al tamaño de la muestra. La resolución de la ecuación del logaritmo discreto toma un tiempo despreciable para votaciones del tamaño considerado en esta memoria, esto es, un número máximo aproximado de 2000 votantes.

4.3 Tolerancia a fallas

Se testeó el sistema ante varios supuestos razonables de fallas, tanto para autoridades como

para votantes. Los escenarios probados fueron los siguientes:

1. Autoridad envía valores privados incorrectos

En este escenario, la verificación de los valores S y S' enviados durante el paso 1 del protocolo DKG falla. El sistema es capaz de detectar el evento y las autoridades envían de inmediato una queja, forzando a la autoridad a reenviar los valores correctos (paso 3 del protocolo).

2. Autoridad responde con un nuevo valor privado incorrecto

En este caso la autoridad es automáticamente descalificada del conjunto QUAL, enviándose un mensaje público del tipo DISQ al bulletin board. La autoridad descalificada no continúa participando en el proceso.

3. Autoridad envía valores de clave pública incorrectos (valores A)

Los valores de clave pública (paso 4 del protocolo) son enviados a cada autoridad de modo de poder generar la clave pública DKG con todos los participantes. Cuando una autoridad envía un valor incorrecto, el algoritmo de verificación (paso 5) lo detecta y permite que dicho valor sea reconstruido automáticamente por las demás autoridades, utilizando la ecuación 1 de Shamir (Capítulo 2.1.4), de modo que el proceso no se ve interrumpido.

4. Votante envía voto con firma inválida

Cada votante firma su voto y las autoridades verifican la validez de dicha firma usando la clave pública del usuario, la cual se encuentra en el bulletin board. En caso de que la firma no sea válida (firma DSA), el voto es marcado como inválido, la razón es publicada en el bulletin board y la preferencia no es contabilizada en el cómputo final.

5. Votante envía voto inválido

Cada autoridad verifica que el voto encriptado sea válido (Capítulo 3.2.5.1). En caso de que la validación sea incorrecta, el voto es marcado como inválido, la razón es publicada en el bulletin board y la preferencia no es contabilizada en el cómputo final.

6. Autoridad envía cómputo incorrecto

Cada autoridad debe generar el cómputo parcial junto con la prueba de igualdad de logaritmos (Capítulo 3.2.6.3). En caso de que la prueba enviada sea inválida, las autoridades pueden regenerar el cómputo, usando la ecuación 1 de Shamir (Capítulo 2.1.4), de modo que el proceso siga su curso normal y pueda entregarse el cómputo final.

Estas pruebas no forman una lista exhaustiva de las posibles fallas, y son descritas para ejemplificar como responde el sistema ante las fallas más intuitivas. En el artículo original [17] se demuestra en forma genérica que aún ante fallas arbitrarias (maliciosas), el sistema sigue preservando las propiedades del Capítulo 1.2 (excepto coerción).

5 Consideraciones sobre la implementación

5.1 Restricciones de diseño

5.1.1 Sincronización entre autoridades

Esta implementación considera que todas las autoridades (honestas o no), participan del proceso respondiendo a los requerimientos, con datos que se ajustan a los formatos establecidos. Si una autoridad no envía los valores que esperan las demás autoridades, el proceso no puede finalizar, por lo tanto la elección no es realizable. Esta restricción puede eliminarse implementando mecanismos de expiración de tiempo (timeout).

5.1.2 Fallas de hardware o malware

Esta implementación, si bien provee herramientas de verificación durante todo el proceso, no protege de fallas de hardware o malware, por ejemplo que el equipo “vote mal”, esto quiere decir, que si el usuario voto por la alternativa “A” pero el equipo entregó “B”, el resto del proceso seguirá su curso normal, generando un resultado final que no corresponde a la voluntad de los votantes. Para superar esta restricción el equipo debe ser “supervisado” y el software verificado o certificado.

5.1.3 Autoridades deshonestas

El modelo implementado asume que si n es el número de autoridades, al menos n/2 + 1 de ellas son honestas. Si eso no se cumple, el protocolo no entrega ninguna garantía criptográfica, por lo que no se puede garantizar la correctitud del proceso ni la detección de errores. Este supuesto es un supuesto estándar en los diseños criptográficos de votación electrónica usando criptografía de umbral (threshold).

5.2 Seguridad

5.2.1 Comunicaciones

5.2.1.1 RMI

En la implementación de este trabajo se ha utilizado RMI como canal de comunicación entre los diversos actores del sistema. Existen diversas consideraciones de seguridad cuando se utiliza RMI, entre ellas podemos destacar:

1- Autentificación de un participante: solo los clientes autorizados deben poder acceder al puerto en que el servidor de RMI escucha, de lo contrario un cliente no autorizado podría enviar mensajes y ser atendidos. Esto se soluciona con mecanismos como firewall y SSL.

2- Privacidad: se puede utilizar RMI con SSL para las comunicaciones [26], de modo de

garantizar privacidad de la comunicación.

3- Los programas clientes y servidor se ejecutan bajo el administrador de seguridad de Java, por lo cual se necesitan especificar los archivos de políticas de seguridad de modo de garantizar los permisos de seguridad que el código necesita para ejecutarse.

5.2.1.2 Red física

La comunicación a través de la red puede ser objeto de ataques de tipo DoS (denegación de servicio) o de interrupciones físicas. Para minimizar estos riesgos se deben tener redes redundantes, con enlaces dedicados entre las autoridades y los centros de votación dentro de lo posible. En una elección a pequeña escala este riesgo no es significativo.

5.2.2 bulletin board

En la implementación de este trabajo, el bulletin board es uno de los componentes claves del proceso de votación electrónica. En caso de que éste no se encuentre disponible o su contenido sea alterado, el proceso podría verse interrumpido o cuestionado. La implementación del bulletin board en este sistema es un servicio web que mantiene la información disponible para cualquier persona o entidad que desee consultarla, es por esto que se deben aplicar las políticas y prácticas de seguridad más estrictas de modo de evitar ataques exitosos a este servidor. La información debe mantenerse replicada en uno o más servidores, alojados en redes distintas y con las mismas políticas de seguridad del servidor principal.

5.2.3 Máquinas de votación (clientes y servidores)

El proceso de votación debe desarrollarse en un ambiente controlado, esto con el fin de evitar coerción al emitir el voto o presencia de malware en el equipo del votante. Él o los equipos que permitirán que los votantes emitan su preferencia, deben ser auditados por alguna entidad confiable y deben aplicarse las más estrictas políticas de seguridad de modo de proteger tanto el sistema como el código utilizado. Los servidores deben contar con firewall y sistemas de detección de intrusos (IDS) para minimizar posibilidades de ataques. Todos los equipos deben considerar el uso de procedimientos de redundancia y protección ante fallas (por ejemplo el uso de sistemas de respaldo de energía). En el caso de una elección de pequeña escala es razonable suponer que la amenaza de coerción es mínima.

5.2.4 Código fuente

Es fundamental que el código de los programas se encuentre libre de errores, por ello es altamente recomendable que el código sea abierto, de modo de poder detectar y corregir errores de la forma más rápida y eficiente posible. Otra ventaja del código abierto es que existe verificación pública, de modo de mejorar la confianza en el sistema. Del mismo modo se debe considerar el uso firmas de los programas de modo de garantizar que no haya alteraciones en el código al momento de la votación.

5.3 Aplicabilidad del sistema

5.3.1 En elección de pequeña escala

Los supuestos razonables y condiciones para que una elección de pequeña escala sea factible son los siguientes:

1. Conectividad de red: si no existe conectividad entre las autoridades, entre las autoridades y el bulletin board o entre los votantes y el bulletin board, la elección no puede realizarse.

2. Se requiere una infraestructura para el manejo de claves públicas y privadas de los votantes.

3. Se requiere que al menos t+1 autoridades sean honestas, donde n > 2t, con n el número de autoridades participantes, de modo de poder garantizar la correctitud del proceso y la detección de errores.

5.3.2 En elecciones de gran escala

Los supuestos razonables y condiciones para que una elección de gran escala sea factible son los siguientes:

1. Conectividad de red: si no existe conectividad entre las autoridades o entre los votantes, la elección no puede realizarse. Debe existir protección a ataques de denegación de servicios y mecanismos de redundancia de los enlaces.

2. Se debe implementar un bulletin board en forma distribuida (Capítulo 6.1.3) y si n es el número de participantes en el broadcast, t+1 de ellos deben ser honestos, con n > 2t.

3. Se requieren mecanismos de broadcast seguro entre las autoridades, lo cual es difícil debido al mayor número de participantes.

4. Se requiere infraestructura de manejo de claves públicas y privadas de los votantes.

4. Se requiere que t+1 autoridades, de las n participantes en el proceso DKG, sean honestas para que la elección se lleve a efecto, con n > 2t

5. Se requiere el hardware apropiado y algoritmos eficientes para obtener los cómputos en un tiempo razonable.

6. Se deben aplicar políticas de tolerancia a fallas de hardware para los equipos de votación y equipos de las autoridades.

7. Se debe examinar (certificar) el software que ejecutan los equipos de modo de detectar presencia de malware o código incorrecto.

8. Se deben aplicar estrictas políticas para prevenir y detectar ataques a los equipos involucrados.

9. Se debe efectuar la votación en ambiente controlado.

5.4 Implementaciones conocidas

Implementaciones conocidas que usan el paradigma basado en encriptación homomórfico, como el desarrollado en el presente trabajo, son tres:

i. Proyecto CyberVote [27], financiado por la Comisión Europea, fue construido como prototipo para votar sobre Internet, pero no existe código fuente disponible, ni es posible descargar alguna parte del software

ii. El proyecto E-Vote [28], también fue financiado por la Unión Europea y probado en Grecia en el 2003. No existe una implementación pública disponible del sistema y al parecer el proyecto tiene una orientación comercial

iii. Sistema Adder [29], construido en la Universidad de Connecticut, EEUU, es un sistema gratuito y de código abierto.

El sistema Adder trabaja con principios similares al sistema desarrollado en el presente trabajo, sin embargo tiene algunas diferencias:

a) El sistema permite elecciones de múltiples candidatos.

b) El esquema de autentificación de usuarios se basa en esquema usuario/password.

c) El esquema de generación de claves de las autoridades usa protocolos criptográficos más débiles (protocolo de Feldman, Capítulo 2.1.5) y es menos robusto ante autoridades deshonestas, ya que no verifica ni reconstruye valores incorrectos, por lo que el proceso puede verse abortado con mayor facilidad.

d) Los votos no son firmados digitalmente.

e) No hay verificación de la correctitud en el proceso de desencriptación de los resultados parciales de cada autoridad.

El presente trabajo se pondrá a disposición de la comunidad en forma gratuita y con el código disponible para motivar posteriores revisiones y desarrollos.

6 Conclusiones

El sistema de votación electrónica diseñado e implementado en este trabajo de título, permitió analizar los supuestos para que una elección de pequeña escala sea considerada exitosa y entregar un sistema útil y confiable.

El diseño del sistema es confiable, proveyéndose los mecanismos criptográficos necesarios para la verificación universal del proceso eleccionario.

El sistema permite que un número de autoridades honestas suficientes permita la realización del proceso, siendo el sistema lo suficientemente robusto ante comportamiento malicioso (arbitrario) de una minoría de las autoridades.

El sistema permite que los votantes emitan sus votos en forma correcta y privada, pudiendo verificar que el usuario no fue suplantado (bajo el supuesto que cada usuario controla su clave privada) y que su voto es válido, secreto y que sea contabilizado.

El sistema genera un cómputo que corresponde a los votos correctamente emitidos y el proceso es repetible y verificable.

El sistema es auditable por una entidad externa, pudiendo verificarse que el proceso completo fue realizado correctamente.

El sistema no requiere grandes componentes de hardware, las votaciones de prueba demostraron que con equipos básicos puede efectuarse la elección en tiempos razonables y obtenerse resultados en muy breve plazo.

El código del sistema es abierto, permitiendo que entidades externas verifiquen la implementación de los diversos pasos de la votación y que el sistema pueda ser mejorado y extendido.

Los requerimientos para que una votación electrónica de pequeña escala pueda llevarse a efecto son bastante simples de cumplir, por lo que una organización de tamaño medio (menos de 5000 votos) puede realizarlas a muy bajo costo.

6.1 Extensiones al actual trabajo

6.1.1 DKG y broadcast seguro

El protocolo DKG requiere que las autoridades puedan sincronizar cada paso que ejecutan, de modo que cada una de ellas sepa por ejemplo que el paso 1 finalizó antes de comenzar el paso 2. Una forma de realizar esto es usando un esquema de relojes virtuales, lo cual se implementa vía broadcast seguro [30], el cuál es un protocolo simple de implementar. En este esquema, una autoridad envía un mensaje, un "tic" a las demás anunciando el fin del proceso que ejecutaba (por ejemplo el paso 1). El resto de las autoridades solo responden dicho "tic" cuando finalizan el paso. Cuando se han recibido todas las respuestas al "tic" de un paso entonces todas las autoridades están de acuerdo en comenzar con el paso siguiente.

6.1.2 Interfaz de votación

Se debe construir una interfaz de usuario para la votación que sea adecuada al perfil del votante (votantes de 3era edad, discapacitados, etc.). La interfaz debiese mostrar en pantalla un voto lo más parecido al voto real, pedir confirmación antes de enviar el voto y generar una copia impresa en papel, la cual debiese ser automáticamente depositada en una urna. Esto último es recomendable para efectuar un reconteo manual en casos de votaciones extremadamente estrechas, con motivos razonables para sospechar de alguna irregularidad en el proceso de votación electrónica, o simplemente como mecanismo de auditoría.

6.1.3 Eliminar el bulletin board

Por simplicidad, el bulletin board lo implementa una entidad (autoridad) distinta, la cual almacena los datos en una máquina que es confiable y no se puede corromper. Sin embargo esto no limita el desarrollo del sistema ya que es simple implementar un protocolo para distribuir el bulletin board entre las autoridades de votación y observadores externos, con garantías de seguridad similares a las de un protocolo distribuido tipo DKG.

6.1.4 Múltiples candidatos

Se puede extender el sistema para permitir la selección de múltiples candidatos en la elección. Para ello se deben cambiar los algoritmos de encriptación de votos, de generación de pruebas de correctitud (ZK) y de cómputo (desencriptación de votos) [31].

7 Referencias

[1] “Election Reform and Electronic Voting Systems (DREs): Análisis of Security Issues”, E. Fisher, Nov 2003, Congressional Research Service, The Library of Congress.

[2] “Caltech/MIT Voting Technology Project, Voting: What Is, What Could Be, July 2001, [http://www.vote.caltech.edu/Reports/index.html] (Caltech/MIT Study).

[3] “Electronic Voting”, Rebecca Mercury, Ph.D. http://www.notablesoftware.com/evote.html, enero 2008.

[4] “Fundations of Cryptography, Basic Tools", Oded Goldreich, 2001,
Cambridge University Press.

[5] “Handbook of Applied Cryptography”, by A. Menezes, P. Van Oorshot, and S. Vanstone, CRC Press, 1996.

[6] “Computacional Number Theory”, Bellare and Rogaway, Lecture notes, course 207, University of California, San Diego, 2006.

[7] Entrevistas con el profesor Alejandro Hevia, Marzo-Junio 2007.

[8] “Cryptography meets voting”, Warren D. Smith, Sep 10, 2005.

[9] “How to share a secret”, Adi Shamir, Communications of the ACM, volume 22, pp. 612-613, 1979.

[10] “Non-Interactive and information-theoretic secure verificable secret sharing”, Pedersen, CRYPTO ’91 vol. 576, pp 129-140.

[11] “Secure Distributed Key generation for Discrete-Log Based Cryptosystems”, Gennaro, Jarecki, Krawczyk, Rabin, Communications of the ACM, Volume 20, pp 51-83, 2007.

[12] “Implementation of Distributed Key Generation Algorithms using Secure Sockets”, Chronopoulos, Balbi, Veljkovic, Kolano.

[13] “Towards Secure and Practical Elections in the New Era”, M. Burmester, E. Magkos, Secure Electronic Voting, (Gritzalis editor), pp. 63-76. Kluwer, Boston. 2003.

[14] “Untraceable Electronic Mail, Return Addresses, and Digital Pseudonyms”, D. Chaum , Communications of the ACM, volumen 24, pp. 84-81, 1981.

[15] “Blind Signatures for Untraceable Payments”, D. Chaum, CRYPTO ’82 pp. 199-203, 1982.

[16] “Verificable Secret-Ballot Elections”, J. Benaloh, PhD Thesis, Yale University, 1987.

[17] “A secure and optimally efficient multi-authority election scheme”, R. Cramer, R. Gennaro y B. Schoenmakers. Proc. of EuroCrypt'97, 103--118. Springer-Verlag. LNCS Vol. 1233.

[18] “Wallet databases with observers”, D. Chaum, T. Pedersen, Advances in Cryptography-CRYPTO ’92, pp 89-105.

[19] “Sun Developer Network (SDN)”, http://java.sun.com, diciembre 2007.

[20] “Java Cryptography Arquitecture (JCA) Reference Guide”, http://java.sun.com/javase/6/docs/technotes/guides/security/crypto/CryptoSpec.html, diciembre 2007.

[21] “Java Remote Method Invocation (Java RMI)”, http://java.sun.com/javase/6/docs/technotes/guides/rmi/index.html, diciembre 2007.

[22] “RSA Laboratories”, 7.21 What is tamper-resistant hardware? http://www.rsa.com/rsalabs/node.asp?id=2357, marzo 2008.

[23] “How to prove to yourself: practical solutions to identification and signature problems”, Amos Fiat and Adi Shamir. In Advances in Cryptology–Crypto 86, pp 186-194, Springer, Berlin, 1987.

[24] “Security proofs for signature schemes”, D. Pointcheval and J. Stern, In Advances in Cryptology-EUROCRYPT 96, vol.1070 of Lecture Notes in Compute Science, pp 387-398. Springer-Verlag, 1996.

[25] “Random oracles are practical: a paradigm for designing efficient protocols”, Mihir Bellare and Phillip Rogaway, In Proceedings of the First Annual Conference on Computer and Communications Security. ACM, November 1993.

[26] “Using Java RMI with SSL”, http://java.sun.com/j2se/1.5.0/docs/guide/rmi/socketfactory/SSLInfo.html, diciembre 2007.

[27] “CyberVote”, http://www.eucybervote.org, marzo 2008.

[28] “E-VOTE: An Internet-based Electronic Voting System: Consolidated Prototype 2 Documentation”, Technical Report e-VOTE/WP-7/D7.4/3.0/29-05-2003, May 2003. http://www.instore.gr/evote/evote_end/htm/3public/doc3/public/public_deliverables/d7_4/Consolidated_Docu_final.zip.

[29] “Adder: an Internet-based Voting System”, http://cryptodrm.engr.uconn.edu/adder, marzo 2008.

[30] “Secure and efficient asynchronous broadcast protocols (extended abstract)”, Christian Cachin, Klaus Kursawe, Frank Petzold, and Victor Shoup. In Joe Kilian, editor, Advances in Cryptology: CRYPTO 2001, volume 2139 of Lecture Notes in Computer Science, pages 524-541. Springer, 2001.

[31] “Practical Multi-candidate Election System”, O. Baudron, P.A. Fouque, D. Pointcheval, G. Poupard, and J. Stern. PODC 2001, 20th ACM Symposium on Principles of Distributed Computing, ACM 2001, pp. 274-283.

Apéndice A: implementación del servidor bulletin board

```
import java.rmi.RemoteException;
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;
import java.rmi.server.UnicastRemoteObject;
import java.io.*;
import java.util.ArrayList;
import BB.BulletinBoard;
import BB.Valor;

public class BBServer implements BulletinBoard
{
    String BBFILE="/home/admin/pub_www/bb.txt"; // path al archivo

    public BBServer () {
        super();
    }

    //  Publica value en el BB
public boolean publica ( Valor value )
    throws RemoteException
    {
        FileOutputStream fout;
        PrintStream p;
         try
           {
           fout = new FileOutputStream(BBFILE,true);
           p = new PrintStream(fout );
           String stringout="<id>"+value.id+"</id>";
           stringout=stringout+"<type>"+value.type+"</type>";
           stringout=stringout+"<value>"+value.value+"</value>";
           p.println (stringout);
           p.close();
           }
          catch (Exception e)
           {
            System.err.println ("Error writing to bulletin board File");
           }
      return true;
    }
Valor parseaLinea(String linea)
      {
        int i;
        String id,t,v;

        i=linea.indexOf("</id>");
        id=linea.substring(4,i);
        linea=linea.substring(i+5);
        i=linea.indexOf("</type>");
        t=linea.substring(6,i);
        linea=linea.substring(i+7);
        i=linea.indexOf("</value>");
        v=linea.substring(7,i);

        return(new Valor(id,t,v));
      }

    // envia valor (todos)
    //pat=patron, campo=0..2 (id..value)
    public ArrayList<Valor> envia (String pat,int campo)
    throws RemoteException
    {
```

```
Valor v=new Valor("NULL","NULL","NULL");
      ArrayList<Valor> vect=new ArrayList<Valor>();
      int i=0;

      try {
        BufferedReader in = new BufferedReader(new FileReader(BBFILE));
        String l;
        while ((l = in.readLine()) != null)
          {
            v=parseaLinea(l);
            if(campo==0) {
             if(pat.compareTo(v.id)==0)
                vect.add(v);
             }
            else if (campo==1) {
              if(pat.compareTo(v.type)==0)
                vect.add(v);
            }
            else if (campo==2) {
               if(pat.compareTo(v.value)==0)
                  vect.add(v);
            }
          }
    in.close();

       } catch (FileNotFoundException e) {
            e.printStackTrace();
       } catch (IOException e) {
            e.printStackTrace();
      }

      return(vect);
    }

    public static void main ( String args[] )
    {
        if (System.getSecurityManager() == null)
             System.setSecurityManager(new SecurityManager());

        try {
            String name = "BulletinBoard";
            BulletinBoard bb = new BBServer();
            BulletinBoard stub =
                (BulletinBoard) UnicastRemoteObject.exportObject(bb, 0);
            Registry registry = LocateRegistry.getRegistry();
            registry.rebind(name, stub);
           System.out.println("BBServer bound");
        } catch (Exception e) {
            System.err.println("BBServer exception:");
            e.printStackTrace();
        }
    }
}
```

Apéndice B: implementación del algoritmo de cómputo parcial

```
package computo;

import java.math.BigInteger;
import java.security.*;
import java.util.ArrayList;
import BB.Valor;
import AUT.Dato;
import java.util.Arrays;


public class computop {
 static private SecureRandom srandom=new SecureRandom();
 static public BigInteger p,q,g;
 static public BigInteger x;
 static public BigInteger h;
 static public BigInteger y;
 static public BigInteger qdsa,pdsa,gdsa;
 static public Valor v;
 static public int MYID=0;
 static public String id;
 static public int BITLENGTH=1023;
 static public BigInteger c1tot=BigInteger.ONE;
 static public BigInteger c2tot=BigInteger.ONE;
 static public int n,t;
 static public int[] qual;
 static public int nvot=0;


 public static void main( String args[] ) {

   ArrayList<Valor> d=new ArrayList<Valor>();
   BigInteger tot;

   leeinitparams();
   d=recupera_votos();
   valida(d);
   computoparcial();
 }


 static ArrayList<Valor> recupera_votos()
  {
     Recibe recibe;
     ArrayList<Valor> v=new ArrayList<Valor>();

     recibe=new Recibe();
     v=recibe.leetodo("B",1);
     return v;
  }

 public static void valida (ArrayList<Valor> dat)
  {
```

```
   String val,c1,c2;
   int valido;
   BigInteger yp;


   for (int i=0;i<dat.size();i++)
       {
         val=dat.get(i).value;
         c1=parsea(val,"c1");
         c2=parsea(val,"c2");
         id=dat.get(i).id;
         yp=get_ypub(id);
         valido=voto_valido(yp,val);
         if (valido==2) {
          System.out.println(id+" Voto Invalido");
          v=new Valor(id,"XB","correctitud del voto invalido");
          publica(v);
         }
         else if (valido==3) {
           System.out.println(id+" Firma Invalida");
           v=new Valor(id,"XB","firma invalida");
           publica(v);
          }
         else {
           System.out.println("id "+id+"...");
           c1tot=c1tot.multiply(new BigInteger(c1));
           c2tot=c2tot.multiply(new BigInteger(c2));
           nvot++;
         }

       }

}

public static int voto_valido(BigInteger ypub, String val)
// retorna 1 valido, 2 proof incorrecta, 3 firma invalida
 {
   String dat,dsa_msg;
   BigInteger c1,c2,a1,a2,b1,b2,d1,d2,r1,r2,c;
   BigInteger t1,t2,t3,t4,t5;
   BigInteger r_dsa,s_dsa;
   BigInteger w,u1,u2,v,hash;

   Sha1 sha=new Sha1();

   dat=parsea(val,"c1");
   c1=new BigInteger(dat);
   dat=parsea(val,"c2");
   c2=new BigInteger(dat);
   dat=parsea(val,"a1");
   a1=new BigInteger(dat);
   dat=parsea(val,"a2");
   a2=new BigInteger(dat);
   dat=parsea(val,"b1");
   b1=new BigInteger(dat);
   dat=parsea(val,"b2");
   b2=new BigInteger(dat);
   dat=parsea(val,"d1");
   d1=new BigInteger(dat);
   dat=parsea(val,"d2");
   d2=new BigInteger(dat);
   dat=parsea(val,"r1");
   r1=new BigInteger(dat);
   dat=parsea(val,"r2");
   r2=new BigInteger(dat);
```

```
    c=sha.getbigsha1(id+c1+c2+a1+a2+b1+b2);
    c=c.mod(q);

    dat=parsea(val,"r");
    r_dsa=new BigInteger(dat);
    dat=parsea(val,"s");
    s_dsa=new BigInteger(dat);
    int i=val.indexOf("<r>");
    dsa_msg=val.substring(0,i);

// test proof voto
    t1=d1.add(d2);
    t2=g.modPow(r1,p).multiply(c1.modPow(d1,p));
    t3=g.modPow(r2,p).multiply(c1.modPow(d2,p));
    t4=y.modPow(r1,p).multiply(c2.multiply(h).modPow(d1,p));
    t5=y.modPow(r2,p).multiply((c2.multiply(h.modInverse(p))).modPow(d2,p));

// test firma
    w=s_dsa.modInverse(qdsa);
    hash=sha.getsha1(dsa_msg);
    hash=hash.mod(qdsa);
    u1=(w.multiply(hash)).mod(qdsa);
    u2=(r_dsa.multiply(w)).mod(qdsa);
    v=gdsa.modPow(u1,pdsa).multiply(ypub.modPow(u2,pdsa));
    v=v.mod(pdsa);
    v=v.mod(qdsa);


    if (c.compareTo(t1.mod(q)) !=0 || a1.compareTo(t2.mod(p))!=0 || a2.compareTo(t
3.mod(p))!=0 || b1.compareTo(t4.mod(p))!=0  || b2.compareTo(t5.mod(p))!=0 )
        return 2;
    if (v.compareTo(r_dsa) != 0 )
        return 3;
    return 1;

  }

 static String parsea(String val, String pat)
   {
      int i, i2, size;
      String ret;

      size=pat.length();
      i=val.indexOf(pat+">");
      i2=val.indexOf("/"+pat+">");
      ret=val.substring(i+size+1,i2-1);
      return ret;
   }

 static void computoparcial()
  {
    BigInteger gama,delta,s1,s2,a,b,r,c,w;
    Leelocal lee;
    Sha1 sha=new Sha1();
    ArrayList<Dato> d=new ArrayList<Dato>();

    lee=new Leelocal();
    d=lee.leetodo("A"+MYID,0);
    x=recupera_x(d);
    gama=c2tot.mod(p);
    delta=c1tot.mod(p);

    //proof
    s1=g.modPow(x,p);
    s2=delta.modPow(x,p);
```

```
    do {
      w=new BigInteger(BITLENGTH,srandom);
     } while (!(w.compareTo(BigInteger.ONE)>0 && w.compareTo(q)<0));
    a=g.modPow(w,p);
    b=delta.modPow(w,p);
    c=sha.getbigsha1("A"+MYID+s1+s2+a+b);
    c=c.mod(q);
    r=w.add(x.multiply(c));
    r=r.mod(q);

    String value="<gama>"+gama+"</gama><alfas>"+delta+"</alfas><delta>"+s2+"</delt
a><s1>"+s1+"</s1><a>"+a+"</a><b>"+b+"</b><r>"+r+"</r>";
     v=new Valor("A"+MYID,"COMP",value);
     publica(v);
     v=new Valor("A"+MYID,"NVOTANTES",String.valueOf(nvot));
     publica(v);

  }

 static BigInteger recupera_x(ArrayList<Dato> dat) {
  String type;
   BigInteger y=BigInteger.ZERO;

   for (int i=0;i<dat.size();i++)
        {
          type=dat.get(i).type;
          if(type.compareTo("a0")==0) {
             y=new BigInteger(dat.get(i).value);
             break;
         }
       }
    return y;
 }

 static void publica (Valor val) {
   String BBAddr="172.17.69.120";
   Publica pub=new Publica(BBAddr);
   pub.publica(v);
 }

 public static BigInteger get_ypub(String id) {
     BigInteger ret=BigInteger.ONE;
     Recibe recibe;

     recibe=new Recibe();
     ArrayList<Valor> v;
     v=recibe.leetodo("pubk",1);

     for (int i=0;i<v.size();i++)
        if (id.compareTo(v.get(i).id)==0)  {
           ret=new BigInteger(v.get(i).value);
           break;
       }

     return ret;
  }

 public static void leeinitparams() {
     Init ini;
     ini=new Init();
     p=ini.p;
     q=ini.q;
     qdsa=ini.qdsa;
     pdsa=ini.pdsa;
     gdsa=ini.gdsa;
```

```
      g=ini.g;
      h=ini.h;
      n=ini.n;
      t=ini.t;
      y=ini.y;

      Recibe recibe;
      recibe=new Recibe();
      ArrayList<Valor> v;
      int hits;

      v=recibe.leetodo("QUAL",1);
      qual=new int[t+1];
      int[][] cand=new int[n][t+1];
      for (int i=0;i<n;i++) {
        cand[i]=get_q(v.get(i).value);
       }
     out:
      for (int i=0;i<n;i++) {
        hits=0;
        for (int j=i+1;j<n;j++)  {
          if (Arrays.equals(cand[i],cand[j]))
             hits++;
          if (hits==t) {
             qual=cand[i];
             break out;
            }

          } //j
      } // i
   }

   static public int[] get_q(String line) {
    int j;
    String aux;
    int[] ret=new int[t+1];

    for (int k=0; k<t+1;k++) {
     aux=Character.toString(line.charAt(3));
     j=Integer.parseInt(aux);
     ret[k]=j;
     line=line.substring(8);
   }

   return ret;
  }

 }
```