Facultad de Ciencias Empresariales
Departamento de Sistemas de Información

Selección e Implementación de Librería Java de Algoritmo para **E-Voting**

Memoria para optar al título de Ingeniero Civil en Informática

---

María Olga Aravena Bravo
moaraven@alumnos.ubiobio.cl

Profesor Guía:

Patricio Alejandro Galdames Sepúlveda

Dedicado a mi gato Thor y a mi "Abuelita Mamma",
Quienes no entienden nada de lo que estoy haciendo
y aun así me han apoyado incondicionalmente…

Índice.

Índice de Tablas.

Índice de Figuras.

Índice de Gráficos.

Capítulo 1: Introducción.

En la vida universitaria existen instancias en que los alumnos no han de poder participar en las elecciones “democráticas” que en instancias como las asambleas se generan.

En el presente informe se encontrarán los argumentos, basados en la experiencia universitaria de quien ha desarrollado este documento, para la implementación de un sistema E-Voting a implementarse en las asambleas de la carrera de Ingeniería Civil en Informática de la Universidad del Bío-Bío.

Se podrá encontrar la documentación de nueve esquemas criptográficos, junto a su correspondiente referencia, y una comparación de estos visualizando la implementación de un sistema Android de votación para dichas asambleas.

Se encontrará la implementación de cuatro de aquellos nueve sistemas criptográficos en lenguaje java, junto a diversas pruebas realizadas de su funcionalidad y una prueba en una aplicación Android sencilla, comprobando los requerimientos que dicho sistema de votación poseerían al momento de implementarse en un sistema más completo para las asambleas de la carrera.

Se desea que los resultados obtenidos sirvan como base teórica y/o práctica para la implementación de un sistema democrático para las asambleas de la carrera, el cual permita superar los problemas “baja representatividad” que han de poseer ante la baja asistencias a las mismas instancias.

Sección 1.1: Objetivo General del proyecto

Implementar una librería para Java de al menos un algoritmo criptográfico que permita el desarrollo de un sistema de votación online, ambientado en el sufragio estudiantil.

Sección 1.2: Objetivos Específicos

1. Estudiar diferentes tipos de algoritmos criptográficos para realizar una comparación entre estos, considerando sus diferencias y sus semejanzas.
2. Comparar algoritmos estudiados para la selección de al menos uno de ellos para implementarse en lenguaje java.
3. Evaluar el o los algoritmos implementados en librería Java para la realización de pruebas en ambiente Android.

Capítulo 2: ¿Por qué la necesidad de implementar un sistema E-voting?

A lo largo de mi trayectoria educativa en la Universidad del Bío-Bío han existido instancias en que el proceso de sufragio en las asambleas se vuelve antidemocrático ante las irregularidades que este posé:

- El proceso de votación actual es manual, existiendo instancias en que los alumnos levantan su mano o se les pregunte uno a uno por su opción de sufragio, lo cual no permite el anonimato y facilita el cohecho además el que un alumno pudiera sentirse obligado a votar a favor de una postura con la que no está de acuerdo ante la presión social.
- La extensión de las votaciones en las asambleas y los problemas de topes de horario motivan al común del alumnado a prescindir de ellas.
- Pese a que se realizan esfuerzos para que las asambleas sean en horarios con mayor facilidad de acceso para los alumnos, quienes no hayan podido asistir a las asambleas, sin importar el motivo, quedan vetados del sufragio de la misma.
- Los Quórums de las asambleas no son representativos del total de personas inscritas, en diversas ocasiones se han tomado decisiones de las cuales no se puede concluir que sean representativas del pensamiento de la mayoría.

Por estas razones he podido concluir que la correcta implementación de un sistema de E-Voting no sólo beneficiaría al proceso del sufragio a nivel carrera, también se lograría en el alumnado una mayor aceptación de la votación ante los beneficios que se expondrán a continuación.

Sección 2.1: Beneficios de Implementar un sistema E-Voting

2.1.1: Privacidad del Votante.

Actualmente en las asambleas, dado a que los votos se realizan a mano alzada o de forma oral, cuando existen opiniones poco aceptadas por la mayoría del alumnado, es común que un votante pueda ceder a la presión de la mayoría y votar en contra de su respectiva ideología.

Un sistema E-Voting permitiría mantener el anonimato del voto ejercido, respetando el derecho del alumno correspondiente a no sufrir prejuicios por su voto.

2.1.2: Exactitud y disminución de tiempos de los conteos de votos.

Eliminando el margen de error generado por un conteo manual, además de omitir el tiempo que conlleva la recepción manual de los votos, se conseguirá mayor exactitud en el resultado obtenido mediante el sufragio.

2.1.3: Reducción del trabajo de los integrantes del Centro de Alumnos.

Si el sistema logra desarrollarse de una manera eficiente, la generación de un proceso de sufragio se realizaría a la brevedad, permitiéndoles a los integrantes del Centro de Alumnos una mejor distribución de los tiempos en los otros tópicos a tratar en las asambleas.

Reduciéndose el tiempo que estos mismos deben de invertir en el conteo manual al realizarse la votación a mano alzada, o en escuchar el voto individual de cada uno de los participantes de la asamblea.

2.1.4: Masificación inmediata del resultado del sufragio.

Al culminar el plazo de la votación, el resultado de la misma podrá ser visualizado de manera instantánea por todos los sufragantes.

Esta ventaja podría de implementarse en un sistema mayor para una notificación masiva tanto a los alumnos, docentes y afectados correspondiente por la decisión elegida.

Sección 2.2: Futuras implementaciones.

La implementación de un sistema E-voting podría sentar base en la implementación de un sistema más completo para las asambleas de la universidad del Bío-Bío: Un sistema de Streaming de las asambleas para los alumnos que no sean capaces de presenciar físicamente las asambleas y que aun así deseen ser parte del proceso democrático que en estas se generan.

De esta manera, de implementarse dicho sistema, se podría garantizar un mayor quorum en las asambleas: Masificando los temas tratados en estas; Y, ante el incremento de las votaciones, volviendo más evidente la voluntad de una auténtica mayoría sufragante.

Capítulo 3: Historia del arte, Algoritmos Criptográficos

Sección 3.1: Introducción

En esta sección del documento se pretende explicar de manera detallada los diferentes tipos de algoritmos Criptográficos capaces de ser utilizados en la implementación de un sistema de E-Voting.

El objetivo de esta sección es servir como fundamento teórico para futuros proyectos de esta envergadura.

Sección 3.2: Elementos Comunes.

A continuación se procederá a describir los elementos en comunes que poseen los sistemas criptográficos introducidos en [1].

3.2.1: Personajes

- **Alice:**

  Se considera como Alice a la persona que inicia la emisión de un mensaje, con el objetivo de comunicarse con un determinado receptor.

- **Bob:**

  Se considera como Bob a quien recibe el mensaje enviado por “Alice”, estos roles no son fijos puesto que se asume que existe una comunicación entre ambos personajes.

- **Charlie:**

  Se considera como Charlie a un tercer participante en la comunicación realizada entre “Alice” y “Bob”, estos dos están consciente de la presencia de Charlie y pueden comunicarse con él de ser necesario.

- **Trudy:**

  Antagonista, persona u entidad que desea intervenir en la comunicación establecida entre Alice y Bob, ya sea:

  - Escuchando la información enviada.
  - Hurtando y/o Borrando la información enviada.
  - Editando la información enviada.

3.2.2: Mensaje

Es la información que Alice y Bob desean de transmitirse mutuamente, la cual no debe de ser vista ni modificada por algún agente externo.

En el caso de este estudio, el mensaje a transmitir será la elección realizada mediante el proceso de sufragio: El voto.

Sección 3.3: Encriptaciones bases.

3.3.1: Encriptación Asimétrica.

La encriptación Asimétrica estudiada en [1] contempla la existencia de un conjunto de usuarios los cuales son poseedores de dos claves preestablecidas las cuales se denominan “Llaves”:

- Una llave es pública (Public Key), conocida por todos los integrantes de aquel conjunto, la cual estará a disposición de todos en el sistema.
- La segunda llave es privada (Private Key), conocida únicamente por el usuario a quien se le ha asignado, utilizada por su dueño con el objetivo de asegurarse de que la información que envíe proviene únicamente de él.

Estas llaves poseen diferentes implementaciones, pero ambas individualmente cifrarán la información en un mensaje y colectivamente descifrarán el mensaje, el cual es el contenido que se desea transmitirse sin que este pueda ser visto o modificado por algún agente externo.

- La notación $\{M\}_{Usuario\_1}$ se implementa para denotar que un mensaje ha sido cifrado mediante el uso de la llave pública de un “Usuario 1”.
  - Este tipo de cifrado se implementan cuando se desea que únicamente “Usuario 1” sea el único capaz de leer tal mensaje, puesto que es necesario la implementación de su llave privada para obtener el mensaje enviado.
  - Algebraicamente esto implica que:
    $[\{M\}_{Usuario\_1}]_{Usuario\_1} = M$.
- La notación $[M]_{Usuario\_1}$ se implementa para denotar que un mensaje ha sido cifrado mediante el uso de la llave privada de un “Usuario 1”.
  - Este tipo de cifrado se utiliza cuando el “Usuario 1” desea garantizar que tal mensaje es de su propiedad, dado que todo usuario puede acceder a su contenido utilizando su llave pública.
  - Algebraicamente esto implica que:
    $\{[M]_{Usuario\_1}\}_{Usuario\_1} = M$.

3.3.2: *Encriptación Homomórfica.*

El cifrado Homomórfico estudiada en [1] y mencionada en los papers [3], [4] consiste en que al encriptar el mensaje **m** este adquiere la propiedad de poder interactuar con otros mensajes sin la necesidad de ser desencriptado mediante el uso de funciones homomórficas. De esta manera, al contenido del mensaje M se le puede aplicar un cálculo algebraico.

Una función homomórfica es una función que preserva las operaciones definidas entre un objeto matemático a otro con la misma estructura algebraica, en el caso de la encriptación homomórfica, los mensajes a encriptar deben poseer una estructura similar que permita dicha relación.

Ejemplo:

> Supongamos que en el mensaje A, M_a almacenamos el valor 102, el cual deseamos sumar al contenido de un mensaje B, M_b poseedor del valor 203.
>
> La suma de ambos mensajes posee un valor de 305, el cual se obtiene a partir de la implementación de una función homomórfica aplicada a M_a y a M_b.

3.3.3: *Criptografía de Umbral.*

La encriptación de umbral estudiada en [1] y mencionada en [16] y [30] consiste en la repartición de una llave privada en *l* cantidades, en donde tan sólo son necesarias w pares de la llave para reconstruirla (donde se cumple que w < l) de la llave para reconstruir el secreto.

Este tipo de criptografía se implemente en la encriptación asimétrica, por lo en el caso de la implementación una "Entidad Organizacional" genera un par de llaves una pública para que los usuarios de esta entidad puedan enviar sus mensajes **m**, mientras que la llave privada es dividida entre los **l** integrantes de la Entidad, estableciendo la necesidad de **w** partes para reconstruir la llave privada evitando así que cada llave generada sirva para la reconstrucción de la llave original.

Un usuario, asignémosle tal rol a Alice, para enviar un mensaje a la "Entidad", encripta su mensaje **m** con la llave pública de la Entidad.

Algebraicamente sería:

$$\text{Mensaje de Alice} = \{M\}_{\text{Entidad\_Organizacional}}$$

Para que la Organización pueda desencriptar el mensaje de Alice, sólo se necesitan las w partes de la llave.

Supongamos que en este caso la **Organización** se dividió mediante "Secret Sharing" en tres partes (l =3) y que tanto Bob y Charlie poseen las piezas para reconstruir la llave privada (w =2).

Bob y Charlie combina sus partes w_1 y w_2 para para restaurar la llave privada, de esta manera se tiene que:

$$[\{M\}_{\text{Entidad\_Organizacional}}]_{\text{Entidad\_Organizacional}} = \text{Mensaje de Alice.}$$

3.3.4: Secret Sharing.

Secret Sharing mencionado en [4] y profundizado en [27] es un método de distribución de un secreto entre un grupo de participantes con la característica de que se necesita una determinada cantidad de partes del secreto para reconstruirlo.

Considerando *n* participantes y un secreto *t*, podemos decir que existen las siguientes clasificaciones de Secret Sharing:

- **Trivial**

  Este tipo posee la forma $t = 1$, el secreto se distribuye entre todos los participantes.

- $\boldsymbol{t = n}$

  En estos esquemas, denominados tambien de la forma (t, n) en donde se posee un secreto *S*, el cual queremos dividir en *n* partes, del cual se puede decir que:

  - Con *k* o más elementos de S_i partes podemos reconstruir S
  - Con *k-1* o menos elementos S_i partes no podemos reconstruir S

  Este (k, n) en donde si $k = n$ el secreto S se descubre.

Shamir Secret Sharing

Shamir Secret Sharing mencionado en [12] y detallado con mayor profundidad en [2] es una variación del Secret Sharing en donde para realizar la separación, se debe elegir (k-1) coeficientes los cuales poseerán el valor a_i de rango $1 \leq i \leq k-1$, y considerando el valor $a_0 = S$ se realiza un polinomio:

$$f(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_{k-1} x^{k-1}$$

Con esta función logramos, para separar el secreto S en *n* partes

$$S_i = (i\ , f(i)), \text{ para todo } 1 \leq i \leq n.$$

Para unir el secreto *S* con *k* partes se utiliza un polinomio de Lagrange al seleccionarse $a, (a = k)$ elementos S_i, con de los cuales tendrán la forma $(x_0, y_0), (x_1, y_1), \dots, (x_a, y_a)$, valores que se aplicarán en el polinomio de Lagrange $l_j\ (x)$ para volver a calcular el valor de $f(x)$:

$$f(x) = \sum_{j=0}^{2} y_i * l_j\ (x)$$

Donde:

$$l_j\ (x) = \prod_{\substack{0 \le m \le k \\ m \ne j}} \frac{x - x_m}{x_j - x_m}$$

Dado que en este sistema se sabe que $f(0) = S$, para calcular el valor del secreto se puede utilizar la función:

$$L\ (0) = \sum_{j=0}^{k-1} f(x_j) \prod_{\substack{m=0 \\ m \ne j}}^{k-1} \frac{x_m}{x_m - x_j}$$

3.3.5: Digital Signature.

Las firmas digitales estudiadas en [1], y mencionadas en [18] son un tipo de validación y se usan para autentificar un mensaje proveniente de un emisor en particular al adjuntar un código que actúa como firma, lo cual garantiza tanto la integridad del mensaje como la fuente del mismo.

Las firmas digitales implementan una encriptación asimétrica, en donde se deben generar las llaves públicas y privadas:

- Primero generamos dos números primos grandes y distanciados p y q, que son multiplicados para obtener un módulo *n*.
- Tenemos que $\varphi(n) = (p-1)(q-1)$, donde $\varphi(n)$ es la Función Phi de Eulier. además debemos poseer un número *e* que es menor que $\varphi(n)$, siendo primo relativo de este y un valor *d*, tal que $d = e^{-1} \bmod \varphi(n)$.
- De esta manera, las llaves generadas son:
  - Public Key: {n, e}
  - Private Key: {n, d}
- Considerando M como el mensaje que deseamos encriptar, tenemos que las Ecuaciones a utilizar son:
  - Encriptación: $C = M^e \ mod \ n$
  - Desencriptación: $M = C^d \ mod \ n$

Blind Signature

Blind Signature [3] es un protocolo de firma digital aplicando criptografía asimétrica en donde el contenido de un mensaje generado por una persona se oculta antes de que sea firmado por una entidad.

Ejemplo:

Supongamos que Alice quiere que Bob firme su mensaje pero sin que este pueda leerlo, al aplicar Blind Signature, Alice puede encriptar su mensaje M agregando un factor aleatorio r formando un mensaje M', entendiéndose como $M' = E(M, r)$, el cual será encriptado con la clave privada de Alice antes de que sea enviado hacia Bob.

Bob recibirá: $[M']_{Alice}$

Puesto que Bob puede acceder a la llave pública de Alice, él puede desencriptar *M'* para luego firmar el mensaje con su llave privada.

Alice recibirá: $[M']_{Bob}$

Tras remover el factor aleatorio de *M'*, Alice será capaz de obtener una firma validad de su por Bob sin que este sea capaz de desencriptar su mensaje.

Sección 3.4: Esquemas de Encriptación.

3.4.1: Esquema de Encriptación de Paillier.

Paillier es un esquema de encriptación asimétrica probabilísticos con propiedades homomórficas inventado por Pascal Paillier en 1999 que permiten la suma y multiplicación de sus textos cifrados.

Tradicionalmente, como es expuesto en [9], este criptosistema compuesto por dos claves públicas q y n, compuesta por los números primo p y q distintos entre ellos. Y dos claves privadas μ y λ, donde $\boldsymbol{\lambda} = lcm((p-1),(q-1))$.

En simples palabras el proceso de Encriptación un mensaje M como $\boldsymbol{c} = g^M r^n \ mod \ n^2$, donde r es un entero aleatorio con $r \in Z_{n^2}^*$

Siendo el proceso de desencriptación del mensaje M la siguiente ecuación:
$\boldsymbol{M} = L\left(c^\lambda \ mod \ n^2\right) * \mu \ mod \ n$, con $c \in Z_{n^2}^*$

Donde la ecuación L es: $L(u) = \frac{u-1}{n}$

En Paillier, la Generación de las Llaves se lleva a cabo de la siguiente manera:

1) Se eligen dos números primos largos, p y q, en donde se debe cumplir que el máximo común divisor (gcd) sea igual a uno, visualizado en la siguiente expresión:
$$\gcd(pq,((p-1)(q-1))) = 1$$
2) Se debe computar un Módulo n equivalente a $n = pq$, para luego calcular la función de Carmichael utilizando la expresión:
$$\boldsymbol{\lambda} = \frac{(p-1)(q-1)}{\gcd((p-1),(q-1))}$$
3) Luego se selecciona un número aleatorio g donde $g \in Z_{n^2}^*$
$$\gcd\left(\frac{g^\lambda \ mod \ n^2 - 1}{n}\right) = 1$$
4) Finalmente hay que aplicar el inverso multiplicativo con:
$$\mu = (\mathrm{L}(g^\lambda \ mod \ n^2))^{-1} \ mod \ n$$
Donde L es: $L(u) = \frac{u-1}{n}$

5) De esta manera tenemos las claves:
   - Públicas: (n, g)
   - Privadas: (μ, λ)

No obstante en [8] se propone una optimización en la generación de las llaves, posibilitando la implementación de una desencriptación parcial para la cual se realizan ciertas modificaciones en los procesos de Generación de llave, Cifrado y Descifrado de las mismas.

1. Generación de la llave.

Para la generación de la llave debemos considerar cuatro números primos largos: *p*, *q*, *p'* y *q'* en donde se debe cumplir que:

$$p \neq q, \qquad p \neq p', \qquad q \neq q'$$

$$\boldsymbol{p} = 2p' + 1$$

$$\boldsymbol{q} = 2q' + 1$$

Con los valores de *p* y *q* se puede determinar que:

$$\boldsymbol{n} = pq$$

$$\boldsymbol{m} = p'q'$$

En este paper se considera la utilización de un módulo computacional n^{s+1} para todo $s \geq 1$ con un espacio de texto plano correspondiente a $Z_{n^s}^*$.

Además de ello se agregará la implementación de un número d que debe cumplir la siguiente propiedad:

$$d = 0 \; mod \; m$$

$$d = 1 \; mod \; n^s$$

Considerando *w* como el subconjunto de en los que se puede desencriptar eficientemente, y *l* como el número de autoridades se realiza una función polinómica f(X):

$$f(\boldsymbol{X}) = \sum_{i=0}^{w-1} a_i \boldsymbol{X}^i \; mod \; n^s m$$

En donde

a_i, cuyo i cumple con $0 < i < w$, corresponde a un valor aleatorio calculado en:

$$a_i = \begin{cases} \{0, \dots, n^s * m - 1\}, & i > 0 \\ d, & i = 0 \end{cases}$$

De esta manera, el secreto compartido por la iésima autoridad será $s_i = f(i)$ desde $1 \leq i \leq l$. Y la llave pública (n, s).

No obstante, para la verificación de la desencriptación en los servidores se necesitará un valor público *v* generado en el conjunto $Z^*_{n^{s+1}}$. Permitiendo así que la llave de verificación de cada servidor como: $v_i = v^{\Delta s_i} \ mod \ n^{s+1}$ con $\Delta = l!$

2. Encriptación.

El proceso de encriptación no posee mayores cambios.

Se posee un mensaje *M,* un número *r* aleatorio $r \in Z^*_{n^2}$ y un texto cifrado *c*, cuya fórmula es:

$$\boldsymbol{c} = (n+1)^M r^{n^s} \ mod \ n^{s+1}$$

3. Desencriptación Compartida

En el proceso de desencriptación compartida, la iésima autoridad computará $\boldsymbol{c_i} = c^{2\Delta s_i}$, donde *c* es el texto cifrado.

4. Combinación de Desencriptaciones.

En esta parte es donde se realiza la desencriptación Parcial, para ello debe existir un subconjunto *S* de tamaño *w* en donde se formará:

$$c' = \prod_{i \in S} c_i^{2\boldsymbol{\lambda}^S_{0,i}} \ mod \ n^{s+1}$$

En donde:

$$\lambda^S_{0,i} = \Delta \prod_{i' \in S \setminus i} \frac{-i}{i - i'} \in Z$$

Con lo cual, siendo M el texto deseado, *c'* tendrá la forma:

$$c' = (1+n)^{4\Delta^2 \boldsymbol{M}} \ mod \ n^{s+1}$$

Computando M:

$$M = c' * (4\Delta^2)^{-1} \ mod \ n^s$$

3.4.2: Esquema de Encriptación Okamoto-Uchiyama.

Okamoto-Uchiyama es un esquema inventado por Tatsuaki Okamoto y Shigenori Uchiyama en 1998 de criptografía asimétrica probabilístico cuya llave privada se compone de dos valores *p* y *q*, y su llave pública de tres valores *n*, *g*, y *h*, además de ello se caracteriza por sus propiedades homomórficas.

Este sistema ha sufrido modificaciones desde su postulación [10], no obstante se el proceso de generación de la llave, encriptación y desencriptación se ha mantenido.

- **Generación de la llave:**

  Siendo *p* y *q* dos números primos largos elegidos aleatoriamente, se calcula un valor *n*, tal que $n = p^2q$.

  Se genera un valor g de manera aleatoria $g \in (Z/nZ)^*$, donde $g^p \neq 1 \; mod \; p^2$.

  Finalmente se calcula el valor *h* como:

  $$h = g^n mod \; n$$

  La llave pública es: $< n, g, h >$

  La llave privada es: $< p, q >$

- **Encriptación:**

  En el proceso de encriptación se realiza de un mensaje M, el cual debe ser un número entero que cumpla la propiedad de $M \in Z/nZ$.

  Se genera un número *r* aleatorio $r \in Z_n^*$.

  La generación del texto cifrado C corresponde a

  $$C = g^M h^r \; mod \; n$$

- **Desencriptación:**

  Para el proceso de desencriptación se debe considerar la expresión, L(X), tal que:

  $$L(X) = \frac{x-1}{p}$$

  Con la ayuda de dicha función, para recuperar el mensaje M del texto cifrado C, utilizamos la siguiente expresión.

$$m = \frac{L(C^{p-1} \bmod p^2)}{L(g^{p-1} \bmod p^2)} \bmod p$$

Donde g corresponde al valor de la llave pública.

En el caso del resultado de $L(g^{p-1} \bmod p^2)$ hay que aplicar un módulo inverso de p, para luego multiplicarlo con $L(C^{p-1} \bmod p^2)$ y al resultado de dicha multiplicación se le aplica módulo de p.

- **Proceso de Homomorfismo:**

  En el caso en que dos textos válidos encriptados C_0 y C_1, hayan sido generados mediante el uso de las mismas claves, al ser ambos valores números enteros se puede obtener la suma de los valores originales aplicando la propiedad homomórfica:

$$E(m_0, r_0) * E(m_1, r_1) \bmod n = E(m_0 + m_1, r_0)$$

  Si se cumple que $m_0 + m_1 < p$

  Considerando:

  - Llave Pública: < n, g, h>
  - Llave Privada: < p, q>
  - $C_0 = g^{M_0} h^{r_0}$
  - $C_1 = g^{M_1} h^{r_1}$

  La suma de los textos M_0 y M_1 sin encriptar corresponde a.

$$C_0 C_1 = g^{M_0 + M_1} h^{r_0 + r_1} \bmod n$$

En el paper [11] queda demostrado que este esquema posee aplicaciones en sistemas e-voting, exponiéndose en este un caso práctico con un ejemplo de prueba de la implementación de este sistema.

3.4.3: *Esquema de Encriptación ElGammal.*

ElGammal [12], [15] es un criptosistema con encriptación asimétrica Taher Elgamal en 1984 que está compuesto por las llaves públicas $< h, G, q, g >$ donde G es un grupo cíclico de orden q con el generador g, mientras que la llave privada corresponde al valor $< x >$ en donde x es un valor aleatorio elegido entre 1 y q.

Además de que se debe cumplir que $h = g^x$

Para realizar la encriptación de un mensaje m, con las llaves públicas $< h, G, q, g >$, se genera un valor r que es un número aleatorio entre 1 y q.

$$E(m) = (C_1, C_2)$$

$$C_1 = m * h^r$$
$$C_2 = g^r$$

La encriptación de ElGammal es semánticamente segura bajo la asunción del Decisional Diffie-Hellman (DDH), donde se considera intratable el subgrupo residual cuadrático $Q_p \in Z_p^*$, donde p y q son dos números primos y $p = 2q + 1$.

Otra característica de ElGammal es que este soporta la rerandomización: Considerando a m' como un texto cifrado, $m' = E(m) = (C_1 C_2)$, se calcula r' como un valor aleatorio existente $1 \leq r' \leq q$.

$$E'(m') = (C_1 h^{r'}, C_2 g^{r'})$$

Mientras que para realizar la desencriptación del mensaje m es: $m = C_1 (C_2^x)^{-1}$, considerando que $h = g^x$, esto implica que $h^r = g^{xr}$.

Tenemos que

$$C_1 (C_2^x)^{-1} = m * h^r (g^{rx})^{-1}$$
$$= m * g^{rx} * g^{-rx}$$
$$= m$$

3.4.3.1: ElGammal aplicando Homomorfismo

En el paper [13] se considera la existencia de un subgrupo G_q de orden q de Z_p^*, donde p y q son números primos largos donde $q \mid p-1$, donde p y q junto a al menos un generador g de G_q son tratados como parámetros del sistema aplicando propiedades del Shamir Secret Sharing.

Considerando como base que existen:

- Llave privada s
- Llave pública, $h = g^s$
- El mensaje es $m \in G_q$.
- El votante elije un valor $\propto$ aleatorio, $\propto \in Z_q$.
- Siendo (x, y) el texto cifrado, el proceso de encriptación es: $(x, y) = (g^{\propto}, h^{\propto} m)$.
- El proceso de desencriptación de(x, y) es: $m = y/x^s$

Se realizan dos procedimientos:

- **Generación de Llave.**

  Considerando la existencia de autoridades, a cada autoridad A_j poseerán un valor $s_j \in Z_q$ de un secreto s, además de que se vuelve el valor $h_j = g^{s_j}$ como un valor público.

  Los valores de s_j permiten que el secreto s se puede reconstruir a partir de cualquier conjunto Λ de t partes mediante el uso apropiado de un coeficiente de Lagrange:

$$s = \sum_{j\ e\ \Lambda} s_j \lambda_i, \Lambda$$

$$\lambda_i, \Lambda = \prod_{l \in \Lambda\{j\}} \frac{l}{l-j}$$

  La llave pública $h = g^s$ es compartida con todos los participantes del sistema, con lo cual hay que tener en cuenta de que el secreto s no es compartido con los participantes.

- **Desencriptación.**

  Para la desencriptación del texto $(x, y) = (g^{\propto}, h^{\propto}m)$ evitando la reconstrucción del secreto s se sigue el siguiente protocolo:

  1. Cada autoridad A_j emite un valor $w_j = x^{s_j}$ con el cual se realiza una prueba en cero-conocimiento para comprobar que: $log_g\, h_j = log_x w_j$.

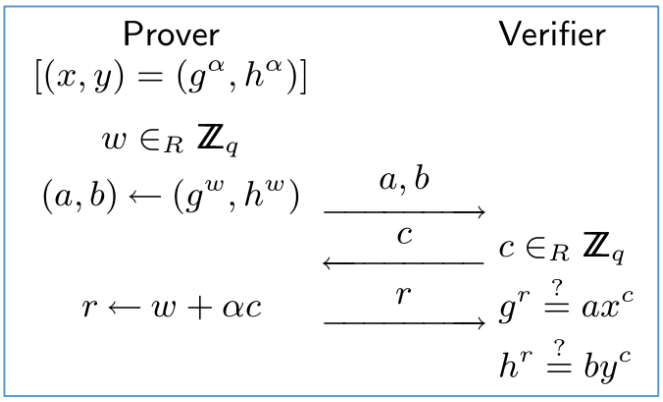


Ilustración 1: *Prueba de conocimiento de logaritmo [13].*

  2. Dejamos que Λ denote cualquier subconjunto de t autoridades que pasen la prueba de cero-conocimiento del paso anterior.

En donde podemos determinar que el texto plano es:

$$m = y / \prod_{j \in \Lambda} w_j^{\lambda_{i,\Lambda}}$$

Además, en este paper se explica que la aplicación del Homomorfismo se consigue volviendo el grupo del mensaje $m \in Z_q$, aplicándole una operación módulo q.

Agregando un generador arreglado $G \in G_q$, la encriptación del mensaje $m \in Z_q$ será en la criptografía de ElGammal de G^m, esta aplicación de la operación modular se explica en el paper [10]. , con mayor detalle en [12]. , en donde considerando que existe un valor n como el total de votantes existentes, y que cada votante, denominado como U_i, posee los siguientes pares de llaves:

- $\left[(x_i \, mod_q), \left(X_i mod_q \; = (g_i^x mod_q)\right)\right]$
- $\left[(y_i \, mod_q), \left(Y_i mod_q \; = (g_i^y mod_q)\right)\right]$

En donde x_i e y_i corresponden a las llaves privadas, mientras que X_i e Y_i, valores que no pueden ser reutilizados en diferentes ejecuciones del algoritmo.

Tenemos que

- $(X \, mod_q) = \prod_{i=1}^{n} (X_i \, mod_q)$
- $(Y \, mod_q) = \prod_{i=1}^{n} (Y_i \, mod_q)$

Cada votante U_i genera un voto d_i puede poseer un valor numérico o booleano, el cual será encriptado utilizando las llaves privadas x_i e y_i y las claves públicas compartidas X e Y con la siguiente descripción:

$$E(d_i) = \begin{cases} (m_i \, mod_q) = (g^{d_i} \, mod_q) * (X^{y_i} \, mod_q) \\ (h_i \, mod_q) = (Y^{x_i} \, mod_q) \end{cases}$$

El sistema recibe todos los votos y procede a computar los valores m y h:

- $m = \prod_{i=1}^{n} (m_i \, mod_q)$
- $h = \prod_{i=1}^{n} (h_i \, mod_q)$

Estos valores serán utilizados por el sistema para calcular el valor de todos los d_i como d, mediante la ecuación:

$$m \, mod_q = (g^d \, mod_q) * (h \, mod_q)$$

En la cual se determina que se encuentra el posible valor de d con un algoritmo minero:

$for \; i = 1 \; to \; \max(d)$

$if \left(m \, mod_q = (g^d \, mod_q) * (h \, mod_q)\right) \; then \; return \; (d)$

Para determinar que $d = \sum_{i=1}^{n} d_i$

$$m \bmod_q = (g^d \bmod_q) * (h \bmod_q) \Rightarrow \prod_{i=1}^{n}(m_i \bmod_q) = (g^d \bmod_q) * \prod_{i=1}^{n}(h_i \bmod_q)$$

$$\Rightarrow (g^d \bmod_q) = \frac{\prod_{i=1}^{n}(m_i \bmod_q)}{\prod_{i=1}^{n}(h_i \bmod_q)} = \frac{\prod_{i=1}^{n}(g^{d_i} \bmod_q) * (X^{y_i} \bmod_q)}{\prod_{i=1}^{n}(Y^{x_i} \bmod_q)}$$

$$= \frac{\prod_{i=1}^{n}(g^{d_i} \bmod_q)\prod_{i=1}^{n}(X^{y_i} \bmod_q)}{\prod_{i=1}^{n}(Y^{x_i} \bmod_q)}$$

$$= \prod_{i=1}^{n}(g^{d_i} \bmod_q)\frac{\prod_{i=1}^{n}(X^{y_i} \bmod_q)}{\prod_{i=1}^{n}(Y^{x_i} \bmod_q)}$$

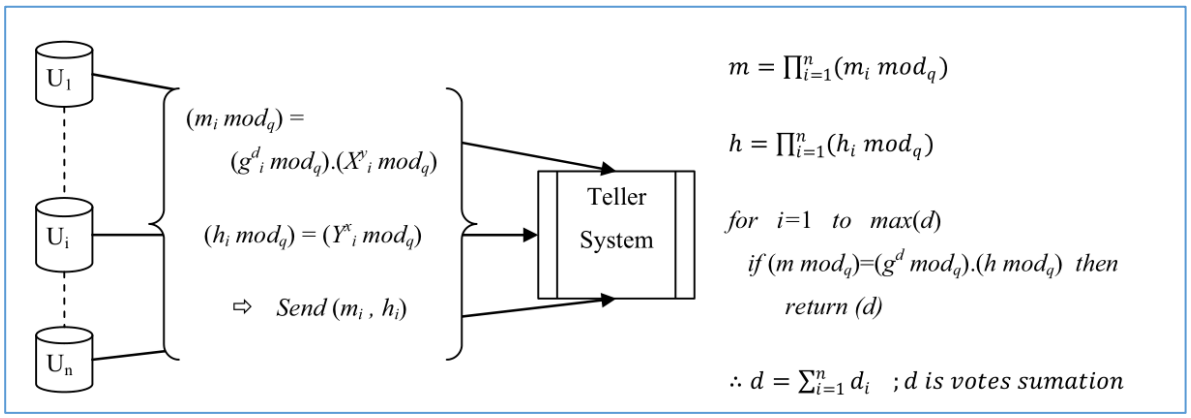
$$= g^{\sum_{i=1}^{n}(d_i \bmod_q)} * \prod_{i=1}^{n}\frac{\left(\left(\prod_{j=1}^{n}(X_j \bmod_q)\right)^{y_i} \bmod_q\right)}{\left(\left(\prod_{j=1}^{n}(Y_j \bmod_q)\right)^{x_i} \bmod_q\right)}$$

$$= g^{\sum_{i=1}^{n}(d_i \bmod_q)} * \prod_{i=1}^{n}\frac{\left(\left(g^{\sum_{i=1}^{n}(x_i \bmod_q)}\right)^{y_i} \bmod_q\right)}{\left(\left(g^{\sum_{i=1}^{n}(y_i \bmod_q)}\right)^{x_i} \bmod_q\right)}$$

$$= g^{\sum_{i=1}^{n}(d_i \bmod_q)} * \frac{g^{\sum_{i=1}^{n}\sum_{j=1}^{n}(x_j\, y_i \bmod_q)}}{g^{\sum_{i=1}^{n}\sum_{j=1}^{n}(y_j\, x_i \bmod_q)}} = g^{\sum_{i=1}^{n}(d_i \bmod_q)}$$

$$\Rightarrow (g^d \bmod_q) = g^{\sum_{i=1}^{n}(d_i \bmod_q)} \qquad \therefore d = \sum_{i=1}^{n} d_i$$

Lo cual se aplica en la siguiente arquitectura de e-Voting:



En el paper [15] se especifica la implementación completa de un sistema e-voting mediante el uso del criptosistema ElGammal, mientras que en los papers [16] y [17] se visualizan sistemas de e-voting basados en este criptosistema.

3.4.4: *Esquema de Encriptación Blind Signature.*

El esquema de Blind Signature inventado por David Chaum en 1983 aplicado en el contexto de E-Voting implica que los votantes obtengan la firma de un validador cuando realice su voto.

En el paper [18] se puede visualizar que se puede aplicar el esquema Blind Signature como complemento a un sistema de encriptación para permitir la autentificación del voto mediante un seudónimo, de esta manera al realizarse un voto *B*, este será ocultado con un número aleatorio y enviarlo al validador.

Teniendo en cuenta que las llaves de la entidad del validador:

- Llave Pública: {n, e}
- Llave Privada: {n, d}

Un votante genera un número *r* aleatorio tal que al buscar el gran común divisor, $gcd(r, n) = 1$, de esta manera se puede enviar el ocultamiento del voto *B*, *B'*:

$$B' = r^e \, B \; mod \; n$$

El número *r* permite ocultar el valor del voto del validador, el cual toma *B'* para verificarlo:

$$S' = (B')^d \; = r \, B^d \; mod \; n$$

Luego de recibir el voto validado el votante des-oculta el voto obteniendo la verdadera firma al computar:

$$S = S' \, r^{-1} \; mod \; n \; = B^d$$

Por ende, al implementar la implementación de

1. El Votante genera su voto, lo oculta, firma, encripta y envía al validador.
2. El Validador firma el voto oculto después de verificarlo y se lo reenvía al Votante.
3. El Votante verifica la integridad del voto al des-ocultarlo y compararlo con el original.
4. El voto validado y el original se envían a un Contador, el cual será protegido con una llave de sesión.
5. El Contador revisará la validez del voto usando la llave pública del Validador para su pronto almacenamiento.

Como se especifica en el paper [18] se encuentra documentadas las librerías necesarias para realizar la implementación de este sistema:

- Java language versión jdk1.3.1_01, que posee Java Crytography Architecture.
- Tambien se requiere Java Cryptography Extension (JCE), el remomendado es el BouncyCastle JCE puesto que es una licencia de código abierto además de ser el proveedor criptográfico más completo.
- Además de ello se encuentra la documentación de cómo se realizó el sistema.

3.4.5: Esquema de Encriptación Blowfish.

Blowfish es un algoritmo inventado por Bruce Schneier en 1993 de encriptación simétrica además de ser un codificador de bloques, mencionado en el paper [5], detallado en el paper [29] cual divide un mensaje en bloques de una longitud de 64 bits durante la encriptación y la desencriptación.

El algoritmo de Blowfish descrito en [26] consiste en dos partes:

1. Expansión de la llave

   En esta etapa, se convierte una llave de hasta 448 bits en 18 subllaves almacenadas en un arreglo P, cada una de 32 bits, además de ello se utiliza una matriz S[4][256].

   Para Encriptar generar las subllaves se deben seguir los siguientes pasos:

   - Inicializar P y S, en orden, con un Sting fijo que consta de los dígitos hexadecimales de p.
   - Aplicar XOR a P[1] con los primeros 32 bits de la llave, aplicar XOR P[2] con los segundos 32 bits de la llave, siguiendo así para todos los bits de la llave: Se genera un ciclo hasta que el arreglo P se le haya aplicado XOR con los bits de la llave.
   - Cifre la cadena de ceros con el algoritmo de Blowfish conservando las subllaves descritas en los pasos anteriores.
   - Reemplace P[1] y P[2] con la salida del tercer paso.
   - Cifre la salida del paso tres con el algoritmo Blowfish con las subclaves modificadas.
   - Reemplace P[3] y P[4] con la salida del paso anterior.
   - Continue el proceso reemplazando todos los elementos del arreglo P y luego los de la matriz S en orden con la salida del algoritmo Blowfish que cambia continuamente.

   En total se requieren de 521 iteraciones para generar las subllaves, no obstante las aplicaciones pueden almacenarlas por lo cual no es necesario ejecutar este procedimiento de manera constante.

2. Encriptación de los datos.

Una vez se posea el arreglo con las subllaves P[18] y considerando *x* como el texto a cifrar de un tamaño de 64 bits.

- Proceso de Encriptación:

Dividimos x en dos mitades de 64 bits: x_L, x_R

$Desde\ i = 1\ hasta\ i = 16$

$x_L = x_L$ XOR $P[i]$

$x_R = f(x_L)$ XOR x_R

Intercambiar x_R y x_L

Al finalizar el ciclo, se deshace el último intercambio entre x_R y x_L.

$x_R = x_R$ XOR $P[17]$

$x_L = x_L$ XOR $P[18]$

Finalmente se reconvinan x_R y x_L

La función f consiste en que se divide x_L en cuatro elementos de 8bits: a, b, c y d.

$$f(x_L) = \big((S[1][a] + S[2][b]\ mod\ 2^{32})\ \text{XOR}\ S[3][c]\big) + S[4][d] mod\ 2^{32}$$

- Proceso de Desencriptación:

El proceso de descencriptado es idéntico al proceso de encriptación, salvo que se invierte el uso de las subllaves de P[18].

Dividimos x en dos mitades de 64 bits: x_L, x_R

$Desde\ i = 16\ hasta\ i = 1$

$x_L = x_L$ XOR $P[i]$

$x_R = f(x_L)$ XOR x_R

Intercambiar x_R y x_L

Al finalizar el ciclo, se deshace el último intercambio entre x_R y x_L.

$x_R = x_R$ XOR $P[2]$

$x_L = x_L$ XOR $P[1]$

Finalmente se reconvinan x_R y x_L

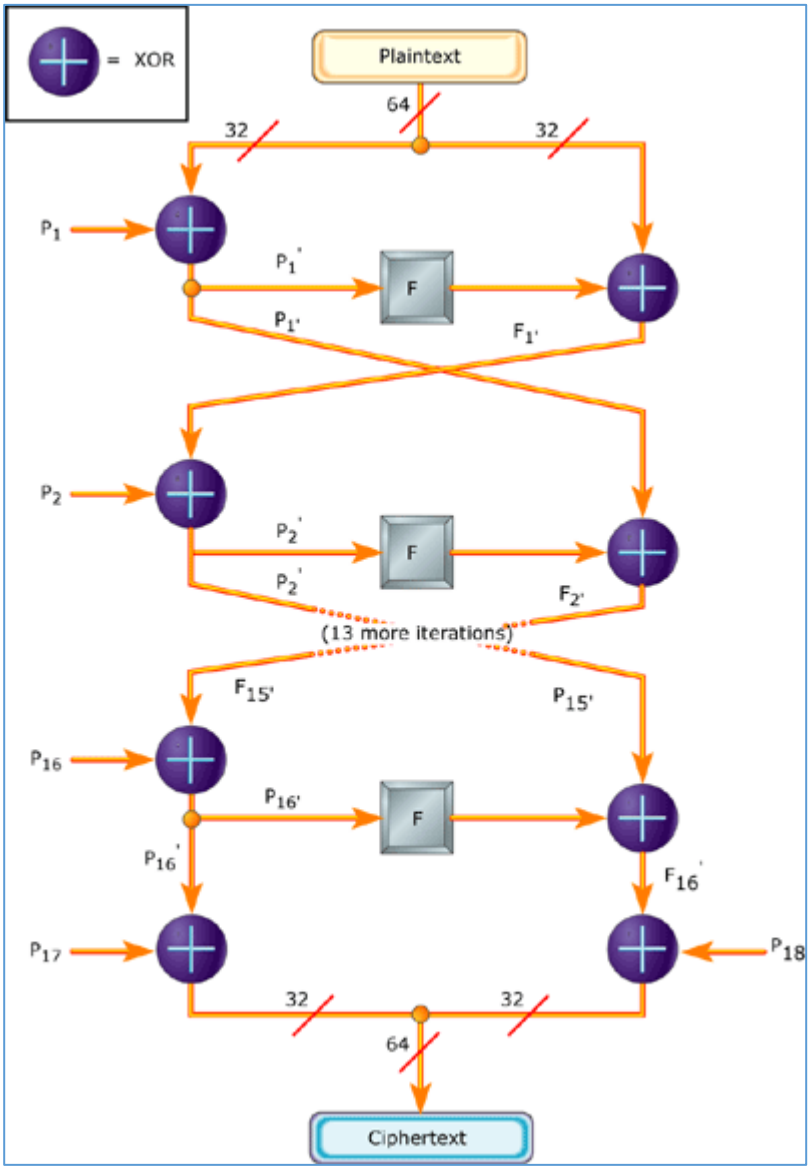


Ilustración 2: *Algoritmo de Blowfish, [25].*

Según [23] este algoritmo no se encuentra patentado además de ser de licencia libre permitiendo el acceso gratuito a todo usuario que desee implementarlo mediante el uso de las librerías java.cypto and java.security[CryptoSpec] las cuales se utilizan como complementos en JCE (Java Cryptography Extension) y JCA (Java Cryptography Architecture), en donde la Clase Cipher provee la funcionalidad criptográfica.

3.4.6: *Esquema de Encriptación ElGammal Blind Signature & Secret Sharing.*

Este esquema, propuesto en [27] por Chin-Ling Chen, Yu-Yi Chen, Jinn-Ke Jan y Chih-Cheng Chen, consiste en la implementación de elementos de Secret Sharing y de Blind Signature aplicando propiedades de ElGammal en una estructura de cuatro partes:

II. Inicialización.

i. Tenemos tres números conocidos por todos los usuarios del sistema:

- p, un número primo largo.
- q, un número primo factor de $(p-1)$.
- g, un número primitivo, (mod p).

ii. Se genera una llave privada para un Centro de Autentificación (AC), Centro de Conteo (TC) y para un Centro de Supervisión (SC) entre números en el rango $[1, q-1]$, SK_{AC}, SK_{TC} y SK_{SC}, para luego generar con estos las llaves públicas de dichos centros:

$$PK_{AC} = g^{SK_{AC}} \ mod\ p$$
$$PK_{TC} = g^{SK_{TC}} \ mod\ p$$
$$PK_{SC} = g^{SK_{SC}} \ mod\ p$$

iii. Antes de realizarse una votación se valida el estado de los electores y se genera un "Certificado Personal" a la Autoridad Certificadora (CA) la cual puede ser utilizada en otras votaciones.

III. Autentificación.

En el día de la votación, los votantes que posean un "Certificado Personal" válido podrán acceder al sitio de la entidad certificadora en donde deberán de entregar su "Firma Seudónima de Votante" siguiendo los siguientes pasos:

i. La entidad AC selecciona un número aleatorio $\acute{K}_i$ donde $\acute{K}_i \in Z_q^*$ para luego computar un número $\check{r}_i = g^{\acute{K}_i} \ mod\ p$ que se envía al votante.

ii. El votante elige un seudónimo v_i, y dos factores ciegos $\alpha, \beta \in Z_q^*$.

El seudónimo del votante es ocultado de la siguiente manera:

$$r_i = \check{r}_i \, g^{\beta} \, mod \, p$$

$$\tilde{v}_i = \alpha \, v_i \, \check{r}_i r_i^{-1} \, mod \, q$$

En donde $\tilde{v}_i$ es el pseudónimo ocultado que será enviado al AC.

iii. AC recibe el mensaje y lo firma usando su llave secreta SK_{AC}:

$$\acute{s}_i = SK_{AC} * \check{r}_i + \acute{K}_i * \tilde{v}_i \, mod \, q$$

En donde el mensaje $\acute{s}_i$ es enviado de regreso al votante.

iv. El votante recibe $\acute{s}_i$ y de él logra recuperar la verdadera firma s_i, finalmente se obtiene que la "Firma Seudónima de Votante" v_i es el par $[r_i, s_i]$.

IV. Votación.

i. Para el proceso de votación, el votante genera un voto m, el cual será encriptado en un valor w:

$$w = PK_{TC}^{a} \, PK_{SC}^{b} \; m \, mod \, p$$

Donde a y b son números aleatorios elegidos por el votante para randomizar m.

Además de ello, se generan dos nuevos números Y_a y Y_b

$$Y_a = g^a \, mod \, p$$

$$Y_b = g^b \, mod \, p$$

El votante deberá de enviar al TC los valores (v_i, r_i, s_i, w, Y_a) y al SC los valores (v_i, r_i, s_i, w, Y_b) mediante el uso de un servidor proxy confiable.

ii. TC y SC validarán la "Firma Seudónima de Votante" mediante la comparación entre g^{s_i} y $PK_{AC}^{r_i} \, r_i^{v_i} \, mod \, p$.

$$g^{s_i} \, ? = PK_{AC}^{r_i} \, r_i^{v_i} \, mod \, p$$

De cumplirse esta validación, tanto TC como SC guardarán en sus respectivas bases de datos (v_i, r_i, s_i, w, Y_a) y (v_i, r_i, s_i, w, Y_b).

V. Resolución.

i. Cuando el periodo de votación culmina, Tc y SC ya no receptarán votos, con lo cual ambas entidades intercambiarán llaves privadas y harán las siguientes comparaciones:

En TC: $PK_{SC} ?= g^{SK_{SC}} \ mod \ p$

En SC: $PK_{TC} ?= g^{SK_{TC}} \ mod \ p$

ii. Si ambas comparaciones son verdaderas, TC y SC liberarán sus correspondientes Y_a y Y_b para formar dos parámetros de Secret Sharing

$$\lambda_a = (Y_a)^{SK_{TC}} \ mod \ p$$

$$\lambda_b = (Y_b)^{SK_{SC}} \ mod \ p$$

Con ambas entidades, cada podrá ser desencriptado como será demostrado a continuación:

$$\frac{w}{(\lambda_a * \lambda_b)} = \frac{w}{((Y_a)^{SK_{TC}} * (Y_b)^{SK_{SC}}) \ mod \ p}$$

$$= \frac{w}{(g^{a\,SK_{TC}} * g^{b\,SK_{SC}}) \ mod \ p}$$

$$= \frac{w}{\left(PK_{TC}^{a}\ PK_{SC}^{b}\right) mod \ p} = \frac{PK_{TC}^{a}\ PK_{SC}^{b}\ m \ \ mod \ p}{PK_{TC}^{a}\ PK_{SC}^{b} \ mod \ p}$$

$$= m$$

iii. Una vez contabilizado todos los votos, se entrega el resultado.

Como se ha logrado visualizar, en la primera parte de este esquema se implementan cualidades del esquema ElGammal y de Secret Sharing, en la segunda y tercera de Blind Signature, mientras que en la cuarta se utiliza una implementación de Secret Sharing para validar los votos, permitiendo de esta manera que en la votación los votos ingresados en las bases de datos sean contados.

3.4.7: *Esquema de Encriptación CAST-128.*

Es un algoritmo de encriptación simétrica inventado por Carlisle Adams y Stafford Tavares en 1996 que utiliza llaves cuyo tamaño pueden variar entre los 40 bits y los 128 bits en incrementos de 8 bits.

CAST, como se especifica en [29] posee una estructura de una clásica Red de Feistel con 16 rondas y operaciones de bloques de texto de 64bit para producir bloques de texto cifrado de 64bits, además de ello se implementan dos sub llaves en cada ronda de bloque: Km_i de 32 bits y Kr_i de 5 bits.

En la encriptación, en este algoritmo se implementan cuatro operaciones primitivas:

- Suma y resta.
    - (+), adición de palabras aplicando módulo de 2^{32}.
    - (-), sustracción de palabras aplicando módulo de 2^{32}-
- OR-Exclusivo ($\oplus$)
- Rotación circular a la izquierda.
- Rotación cíclica de una palabra x en y bits, denotado: $x <<< y$.

Para realizar la encriptación el texto original es dividido en dos mitades de 32 bits L_0 y R_0, siendo L_i y R_i los valores de cada mitad luego de realizarse cada una de las 16 rondas a ser aplicadas. Finalmente la concatenación de R_{16} y L_{16} el texto cifrado.

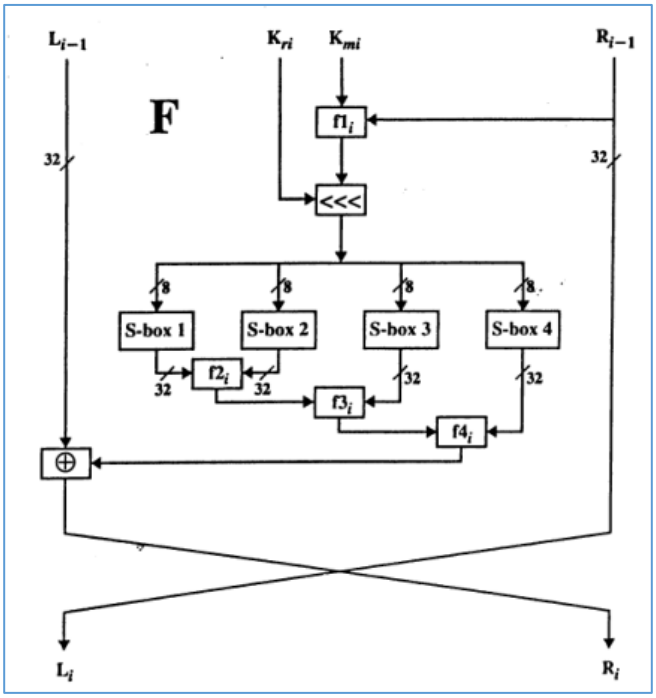


Ilustración 3: *Detalle de una ronda CAST-128, [29].*

Esto en Pseudocódigo [29] corresponde a:

$$L_0 \,||\, R_0 = Plaintext$$
$$For\ i = 1\ to\ 16\ do$$
$$L_i = R_{i-1}$$
$$R_i = L_{i-1} \oplus F_i[R_{i-1}, Km_i, Kr_i]$$
$$Ciphertext = R_{16} \,||\, L_{16}$$

Mientras que el proceso de desencriptación es idéntico al algoritmo de cifrado anteriormente expuestas, salvo que en las rondas y el par de subclaves se utilizan en orden inverso para calcular $L_0 \,||\, R_0$ desde $R_{16} \,||\, L_{16}$.

La función F incluye el uso de cuatro 8x32 S-boxes, la función de rotación circular a la izquierda y cuatro funciones que van dependiendo del número de la ronda en la que se encuentre el texto, las cuales son denominadas como: $f1_i$, $f2_i$, $f3_i$ y $f4_i$.

Se utiliza un valor I para refiriese al valor intermedio de 32 bit, generado después de la ronda circular a la izquierda y a las etiquetas I_a, I_b, I_c, I_d para referirse a 4bits de I, en donde I_a es la más significativa y I_d la etiqueta menos significativa.

Teniendo esto presente, la función F se define como:

Rounds 1, 4, 7, 10, 13, 16	$I = ((Km_i + R_{i-1}) <<< Kr_i)$ $F = ((S1[Ia] \oplus S2[Ib]) - S3[Ic]) + S4[Id]$
Rounds 2, 5, 8, 11, 14	$I = ((Km_i \oplus R_{i-1}) <<< Kr_i)$ $F = ((S1[Ia] - S2[Ib]) + S3[Ic]) \oplus S4[Id]$
Rounds 3, 6, 9, 12, 15	$I = ((Km_i - R_{i-1}) <<< Kr_i)$ $F = ((S1[Ia] + S2[Ib]) \oplus S3[Ic]) - S4[Id]$

Ilustración 4: *Definición de la función F, [29].*

CAST-128 utiliza ocho 8x32 S-Box:

- Cuatro de ellas, desde S-Box1 hasta S-Box4, en el proceso de encriptación y desencriptación.
- S-Box5 hasta S-Box8 son utilizadas para la generación de las subllaves.

Cada S-Box es un arreglo de 32 columnas con 256 filas. La entrada de 8 bits selecciona una fila en la matriz; el valor de 32 bits en esa fila es la salida, todas las S-Boxes contienes valores arreglados.

La generación de Subllaves es un proceso complejo, etiquetando los bytes de una llave de 128bits de la forma: $x0x1x2x3x4x5x6x7x8x9xAxBxCxDxExF$.

Donde $x0$ es el bit más significativo y xF es el bit menos significativo, tambien deben de implementarse las siguientes definiciones:

- $Km_1 \dots Km_{16}$ 16 subclaves de enmascaramiento de 32 bits (una por ronda)
- $Kr_1 \dots Kr_{16}$ 16 subclaves de rotación de 32 bits, una por ronda, de los cuales solo se utilizan los 5bits menos significativos de cada uno.
- $z0 \dots zF$ Bytes intermedios, temporales.
- $K_1 \dots K_{16}$ Palabras de 32 bits intermedias, temporal.

Estos valores se calculan mediante el uso del S-Box5 hasta S-Box8 expresando en la siguiente imagen:

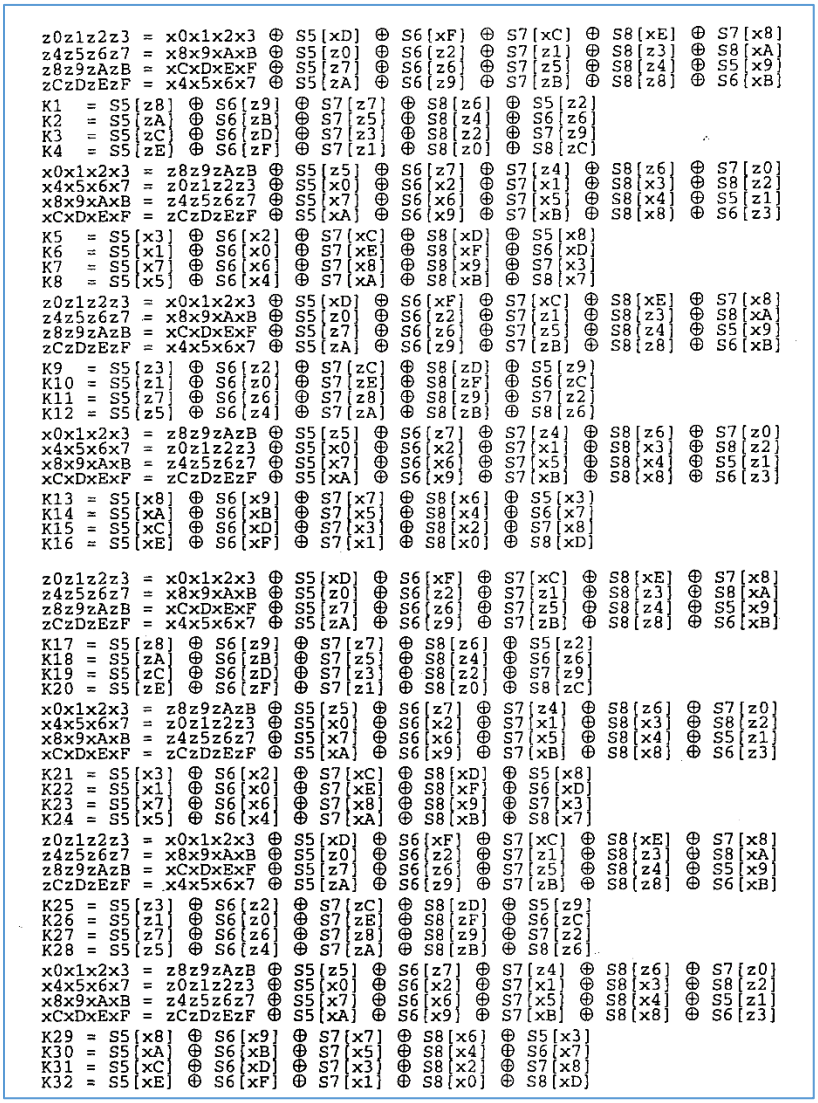


Ilustración 5: *Generación de subllaves de CAST-128, [29].*

Mientras que la definición de las subllaves corresponde a:

$$For\ i = 1\ to\ 16\ do$$
$$Km_i = K_i$$
$$Kr_i = K_{16+i}$$

Dado lo expuesto con anterioridad, se puede apreciar que existe una estructura similar entre las funcionas de ronda de CAST-128 y la utilizada en el criptosistema Blowfish.

3.4.8: *Esquema de Encriptación Goldwasser-Micali (GM).*

GM es un sistema criptográfico asimétrico inventado por Shafi Goldwasser y Silvio Micali en 1982, mencionado en el paper [30], con propiedades homomórficas, en el cual consta de tres etapas.

1. Generación de Llaves.

   GM implementa un criptosistema RSA, en el cual considerando los valores p y q como números primos aleatorios e independientes que cumplen con la siguiente expresión: $p = 1 = 3(mod 4)$.

   Existe un valor N, donde $N = p * q$.

   Se calcula un valor a, el cual debe cumplir con las siguientes propiedades:

   - $a_p^{(p-1)/2} = -1(mod\ p)$
   - $a_q^{(q-1)/2} = -1(mod\ q)$

   De esta manera el conjunto de llave pública es (a, N), mientras que la llave privada es (p, q).

2. Encriptación.

   Para la encriptación de un mensaje m, de largo n el cual consiste en un Sting de compuesto de los bits $(m_1, \dots, m_n)$.

   En cada bit m_i se debe generar un valor b_i aleatorio del grupo unitario modulo N o debe de cumplir $gcd(b_i, N) = 1$.

   De esta manera cada c_i, bit cifrado de m_i, es:

   $$c_i = b_i^2\ a^{m_i}\ (mod\ N)$$

   Finalmente, el texto cifrado c se compone de: $c = (c_1, c_2, \dots, c_n)$

3. Desencriptación.

   Con el texto cifrado c, se puede recuperar el mensaje m aplicando en cada c_i una factorización prima (p, q) donde c_i es un residuo cuadrático de esta con lo cual se obtiene un valor: $m_i = 1$ o $m_i = 1$.

   Una vez se aplique este método en todos los c_i se puede recobrar el mensaje $m = (m_1, \dots, m_n)$.

No obstante, en el paper [30] se indica que no es eficiente dado que el tamaño del texto cifrado puede llegar a ser superior al del texto original.

3.4.9: *Esquema de Encriptación Boneh-Goh-Nissim (BGN).*

El esquema BGN es un sistema criptográfico asimétrico inventado por Dan Boneh, Eu-Jin Goh y Kobbi Nissim en 2006 mencionado en [31], que permite la multiplicación de textos cifrados con un valor constante, para lo cual utiliza una curva elíptica con el objetivo de garantizar la multiplicación homomórfica.

Este criptosistema se compone en tres etapas:

1. Generación de Llaves.

   Considerando un valor de seguridad valores λ, donde $\lambda \in Z^+$, se genera una tupla (q_1, q_2, G, G_1, e), donde:

   - q_1 y q_2 son dos números primos de gran tamaño.
   - G es un grupo cíclico de orden $q_1 q_2$
   - Y e es un mapa de emparejamiento, $e: G \times G \rightarrow G_1$

   Definimos un valor N como $N = q_1 q_2$, para luego generar dos generadores aleatorios de G, g y u, para definir a h como un generador aleatorio del subgrupo G de orden q_1 de valor $h = u^{q_2}$.

   Por lo cual, el conjunto de llave pública es (N, G, G_1, e, g, h), mientras que la llave privada es (q_1).

2. Encriptación.

   Para la encriptación de un mensaje m, se genera un valor aleatorio r perteneciente al rango $[1, N]$ para generar un texto cifrado C.

   $$C = g^m h^r \in G$$

   En el caso en que el mensaje a encriptar m posea un espacio de bits como enteros del set $\{0,1, \ldots, T\}, T < q_2$ se encripta cada bits que compone al mensaje m.

3. Desencriptación.

   Para la desencriptación del texto cifrado C, se utiliza la llave privada q_1, $C^{q_1} = (g^m h^r)^{q_1} = (g^{q_1})^m$, siendo necesario la computación de un logaritmo de C^{q_1} a la base g^{q_1} aplicando “Pollard's Kangaroo algorithm”.

Para que la multiplicación homomórfica pueda realizarse, implementando una curva elíptica, se considera que existe un G_1 y G_2 ambos pertenecientes a un grupo aditivo y a G_T como un grupo multiplicativo para todos los primos de orden p.

Tenemos los generadores de G_1 y G_2, $P \in G_1$ y $Q \in G_2$ respectivamente y un mapa de emparejamiento e: $G_1 \times G_2 \rightarrow G_T$, en donde e por fines prácticos deberá de ser computable de una manera eficiente, se buscan las siguientes propiedades:

- Bilinealidad: $\forall a, b \in Z_p^*$:

$$e\left(P^a, Q^b\right) = e(P, Q)^{ab}$$

- No degeneración:

$$e(P, Q) \neq 1$$

En el caso en que el emparejamiento sea simétrico, cumpliéndose que $G_1 = G_2 = G$, donde G sea cíclico, el mapa e deberá de ser conmutativo:

$$e(P, Q) = e(Q, P)$$

Además de ello, deberá de considerarse un generador $g \in G$, donde existan enteros p y q, tal que $P = g^p$ y $Q = g^q$, permitiéndose el cumplimiento de que:

$$e(P, Q) = e(g^p, g^q) = e(g, g)^{pq} = e(g^q, g^p) = e(Q, P)$$

En el paper [32] se encuentra detallado la implementación de la curva elíptica en este criptosistema, además de encontrarse un análisis a las propiedades homomórficas que este criptosistema posee.

3.4.10: Clasificación comparativa de los algoritmos

Los algoritmos estudiados comparten algunos elementos en común por lo cual, con el objetivo de seleccionar una cantidad específica de los sistemas de encriptación anteriormente explicados, se han optado por generar una lista de cualidades a tener en cuenta para dicha selección.

3.4.10.1: Tipo de criptografía

Como se ha hecho mención con anterioridad, existen dos tipos de criptografía a aplicar Asimétrica y Simétrica, al cual se ha optado por brindar mayor prioridad a los sistemas cuya encriptación sea del tipo Asimétrica dada a la mayor seguridad que estos criptosistemas brindan al momento de realizarse el mensaje.

3.4.10.2: Homomorfismo

Para la elección de los algoritmos a utilizar se consideró que la propiedad de Homomorfismo especialmente el homomorfismo Aditivo, es una cualidad deseable en el sistema de encriptación a utilizarse en el posterior desarrollo de un sistema E-Voting para las elecciones de las asambleas de la carrera, dado a que esta propiedad permite la suma de votos aun encriptados, permitiendo el ahorro de tiempo a la hora de realizar el conteo de los votos.

3.4.10.3: Desencriptación Parcial

La desencriptación Parcial es una propiedad que permite obtener información del texto cifrado sin la necesidad de realizarse una completa desencriptación del mismo.

3.4.10.4: Encriptación Probabilística

La encriptación probabilística consiste en la agregación de un valor aleatorio calculable a una ecuación de encriptación para evitar que, en la encriptación de un mismo valor en más de una oportunidad, se obtenga como texto cifrado dos textos completamente diferentes entre sí.

3.4.10.5: Conservación del anonimato e Integridad del Voto

Como en todo sistema de votación, la conservación del anonimato ambos aspectos posee una relevante importancia al momento de elegir un sistema de encriptación.

3.4.10.6: Tamaño de la llave

El valor que posee la llave privada de estos algoritmos como una guía para determinar si estos criptosistemas podrán ser implementados en un ambiente móvil.

3.4.10.7: Tamaño Texto Cifrado (C) vs Texto plano (M)

El conocer el tamaño que poseerá un texto cifrado contra el texto plano servirá como una guía para determinar si estos criptosistemas serán aptos para implementarse en un ambiente móvil.

3.4.10.8: Tamaño Texto Cifrado (C) vs Texto plano (M) en un Vector

Atributo encontrado en el Paper [30] del criptosistema Boneh-Goh-Nissim el cual se considerará en la realización de las pruebas a los criptosistemas seleccionados.

3.4.10.9: Número de Exponenciación en el algoritmo de Encriptación

El conocer la cantidad de exponenciales que los algoritmos de encriptación permitirá tener una idea inicial del tiempo en que estos tardarán en ejecutarse.

Teniendo en cuenta de que el objetivo final de este estudio es lograr la implementación de alguno de los sistemas estudiados en un dispositivo Android de brindará prioridad a los sistemas que posean un tipo de criptografía Asimétrica, en los que pueda aplicarse homomorfismo.

3.4.11: Tabla resumen

	PAILLIER **[9].**	OKAMOTO-UCHIYAMA **[10].**	ELGAMMAL **[13].**	BLIND SIGNATURE RSA **[18].**	BLOWFISH **[26].**	BLIND SIGNATURE & SECRET SHARING **[27].**	CAST-128 **[29].**	GOLDWASSER-MICALI (GM). **[30]**.	BONEH-GOH-NISSIM (BGN) **[32].**
TIPO DE CRIPTOGRAFÍA	Asimétrica	Asimétrica	Asimétrica	Asimétrica	Simétrica	Asimétrica	Simétrica	Asimétrica	Asimétrica
HOMOMORFISMO	Si	Si	Si [14].	No	No	No	No	Si	Si [30].
HOMOMORFISMO ADITIVO	Total	Total	Aplicable [36].	-	-	-	-	Total	Total
HOMOMORFISMO. MULTIPLICATIVO	Total	-	Parcial	-	-	-	-	-	1
DESENCRIPTACIÓN PARCIAL	Aplicable	No	No	No	No	No	No	No	No
ENCRIPTACIÓN PROBABILISTICA	Si	Si	Si	Si	No	Si	No	Si	Si
CONSERVACIÓN DEL ANONIMATO	Si	Si	Si	Si	No	Si	No	Si	Si
INTEGRIDAD DEL VOTO	Si	Si	Si	Si	Si	Si	Si	Si	Si
TAM. LLAVE	16 bits	Máx. 256 en bloques AES	?	512, 1024, 2048 bits	Llave: 448 bits Subllaves: 4168 bytes	?	40 bits - 128 bits	?	?
TAM. TEXTO CIFRADO (C) VS TEXTO PLANO (M)	Cuatro veces mayor [9].	3 veces mayor, casi similar al RSA	Dos veces mayor [30].	?	Igual	?	Igual	Varios cientos de veces más grande que el texto plano. [30].	?
TAM. TEXTO CIFRADO (C) VS TEXTO PLANO (M) EN UN VECTOR	-	-	-	-	-	-	-	-	2m= sqtr(l) l: Tamaño del vector
NÚMERO DE EXPONENCIACIÓN EN EL ALGORITMO DE ENCRIPTACIÓN.	3	2	2	1	XOR	2	XOR	2 x cada bit del mensaje.	2

Tabla 1: *Tabla comparativa de los algoritmos estudiados, fuente propia.*

En la presente tabla se puede observar un resumen de los sistemas estudiados, visualizando la existencia de elementos en común entre estos algoritmos, siendo los casos en que existe un símbolo "?" en los cuales no se encontró información asociada, mientras que en el caso de la fila "Tam. Texto Cifrado (C) vs Texto Plano (M) en un Vector" es una propiedad expuesta en el paper [32] es una propiedad exclusiva del esquema de Boneh-Goh-Nissim en donde no se encontraba especificado su implementación.

Sección 3.5: Conclusiones

Se puede apreciar una gran similitud con respecto al funcionamiento de la gran mayoría de los algoritmos, esto se debe a que la gran mayoría de estos criptosistemas poseen como base teórica la implementación de ElGammal.

Por ello, se ha determinado que los algoritmos a implementarse en la siguiente sección serán:

- Paillier.
- Boneh-Goh-Nissim
- Okamoto-Uchiyama
- ElGammal.

Estos cuatro esquemas fueron elegidos dado que los cuatro criptosistemas son del tipo de Encriptación Asimétrica, además de que puede implementarse el Homomorfismo, lo cual representa una ventaja en el proceso de implementación de la librería: Dado que la realización de operaciones de Sumas Homomórficas permite la obtención de una única desencriptación; Mientras que en los casos de los esquemas en los que no es posible aplicar Homomorfismo se debe de desencriptar cada voto para obtener un resultado final, con lo cual se estarían consumiendo más recursos ante las constantes desencriptaciones.

En el caso exclusivo de Boneh-Goh-Nissim, este esquema fue elegido sobre el esquema Goldwasser-Micali dado que el último posee un tamaño de texto cifrado, según [32] extremadamente superior.

Capítulo 4: Implementación de Librerías & Desarrollo de Pruebas.

Sección 4.1: Introducción

En esta sección se pretende realizar la implementación en lenguaje java de los esquemas de encriptación seleccionados del estudio anterior, Boneh-Goh-Nissim (BGN), Okamoto-Uchiyama y ElGammal, tomando como base el código de la librería Paillier desarrollada por Murat Kantarcioglu, James Garrity y Sean Hall encontrada durante el periodo de investigación.

Los códigos en lenguaje Java de las librerías generadas podrán ser encontrados en el Capítulo “Anexos”, sección 9.1.

Posteriormente se realizarán un conjunto de pruebas para determinar las diferencias en tiempo de los cuatro criptosistemas seleccionados, las cuales serán:

- Pruebas de Encriptación & Desencriptación a un valor 1 en primera instancia.
- Pruebas de Encriptación & Desencriptación a dos valores (1 y 0) almacenados en un arreglo.
- Pruebas de Encriptación & Desencriptación a un arreglo compuesto por dos valores.
- Pruebas de Homomorfismo & Desencriptación de dos valores.
- Pruebas de Encriptación y Desencriptación del valor 1 con diferentes intervalos.

Los códigos generados para la realización de las pruebas podrán ser encontrados en el Capítulo “Anexos”, sección 9.3.1.

Para posteriormente la realización de una última prueba en aplicación Android con al cual determinar los recursos necesarios del sistema móvil necesarios para la funcionalidad del sistema de prueba e-voting.

Sección 4.2: Descripción de los algoritmos seleccionados e implementados.

4.2.1: Paillier

Para el análisis de este Esquema, se utilizó una librería java desarrollada por Murat Kantarcioglu, James Garrity y Sean Hall [33] la cual cuenta con tres paquetes, uno principal y dos subpaquetes:

- Paillierp
- Paillierp.key
- Paillierp.zkp

En el paquete principal Paillierp se puede encontrar la implementación de Paillier tanto en su proceso de encriptación como en su proceso de desencriptación.

Este paquete se compone por cinco clases, las cuales son:

- **AbstractPaillier.java:** Clase Abstracta de Paillier.
- **ByteUtils.java:** Clase que contiene un conjunto de funciones para manipular Bytes.
- **Paillier.java:** Implementación simple de Paillier basado en la generalización dada en *"Generalization of Paillier Public-Key System with Applications to Electronic Voting"* por Damgård et al, tomando los métodos previamente definidos en "AbstractPaillier".
- **PaillierThreshold.java:** Clase en donde se implementa de manera simple el umbral del esquema de Paillier basado en la generalización dada en *"Generalization of Paillier Public-Key System with Applications to Electronic Voting"* por Damgård et al.
- **PartialDecryption.java:** Clase en donde se implementa de manera simple la desencriptación parcial aplicado en el esquema de umbral de Paillier.

En el subpaquete Paillierp.key se encuentra todas las clases relacionadas con la generación de llaves.

Este paquete se compone por cinco clases, las cuales son:

- **KeyGen.java:** Clase encargada de generar las llaves del esquema Paillier.
- **PaillierKey.java:** Clase encargada de implementar las llaves del esquema Paillier.
- **PaillierPrivateKey.java:** Clase que implementa las llaves privadas del esquema Paillier.
- **PaillierPrivateThresholdKey.java:** Clase que implementa las llaves privadas para la implementación de la desencriptación parcial.
- **PaillierThresholdKey.java:** Clase que implementa las llaves para la implementación de la desencriptación parcial.

En el subpaquete Paillierp.zkp se encuentra la implementación del criptosistema Paillier aplicando pruebas de cero conocimientos (Zero Knowledge Proof)

Este paquete se compone por cuatro clases, las cuales son:

- **DecryptionZKP.java:** Clase que permite la desencriptación de las pruebas de cero conocimientos.
- **EncryptionZKP.java:** Clase que permite la encriptación de las pruebas de cero conocimientos.
- **MultiplicationZKP.java:** Clase que permite la implementación de la multiplicación de valores encriptados pruebas de cero conocimientos.
- **ZKP.java:** Clase abstracta para pruebas de cero conocimientos no interactivas, pruebas que usan el hash de valores como entradas.

4.2.1.1: Código Fuente:

http://cs.utdallas.edu/dspl/cgi-bin/pailliertoolbox/javadoc/index.html?paillierp/Paillier.html

4.2.1.2: Observación:

- La librería no permite encriptar letras, sólo números importando la clase BigInteger.

4.2.2: Boneh-Goh-Nissim (BGN)

Tomando como base la distribución realizada en la librería de Paillier desarrollada por Murat Kantarcioglu, James Garrity y Sean Hall para la realización de la librería de la criptografía de Boneh-Goh-Nissim se han realizado cinco clases más dos clases de pruebas, una clase (.java) donde se realizaron pruebas a la funcionalidad de la librería [38] y una segunda clase (test.java) en la cual se realizaron revisiones del funcionamiento del criptosistema.

- **AbstractBGN.java:**

  Clase Abstracta del criptosistema Bohen-Goh-Nissim, en la cual se encuentra el método de encriptación, y el método que permite la implementación de la suma homomórfica de dos valores cifrados aplicando las mismas llaves. Sus funciones son:

  - `public BGN_key getPublicKey():` Función que permite obtener las llaves públicas del criptosistema.
  - `public ECPoint encrypt(BigInteger m):` Función inicial para realizar la encriptación de un mensaje “m”.
  - `public ECPoint encrypt(BigInteger m, BigInteger random, BGN_key key):` Función llamada por “encrypt(BigInteger m)” para resguardar la seguridad de las llaves públicas
  - `public ECPoint encrypt(BigInteger m, BigInteger random, ECPoint h, ECPoint g):` Función en donde se realiza la encriptación del mensaje m el cual es almacenado en un punto de la Curva Elíptica generada en la clase “BGNKeyGen”.
  - `public ECPoint add_BGN(ECPoint c1, ECPoint c2):` Función donde se implementa la suma homomórfica, actualmente esta función presenta problemas en su funcionamiento.

- **BGNKeyGen.java:**

  Clase que permite la generación de las llaves del criptosistema Boneh-Goh-Nissim.

  - `public static BGN_privateKey BGNKey(int s, long seed)`: Función que permite generar tanto las llaves privadas como públicas del esquema de Boneh-Goh-Nissim como la Curva Elíptica que se utilizará en el criptosistema.
  - `public static LinkedList<BigInteger> CalcularQ_N_q1_q2(int s, Random rnd)`: Función que genera los valores Q, valor relevante para formar la Curva Elíptica utilizada por las llaves privadas "g" y “h”, N, q_1 y q_2

- `public static boolean esPrimo(BigInteger numero)`: Función que permite determinar si un número ingresado es un número primo, obtenida de http://lineadecodigo.com/java/numeros-primos-en-java/ .
- `public static ECPoint CalcularPunto(ECCurve curve, BigInteger field)`: Función que permite calcular un punto aleatorio existente en la Curva Elíptica generada en la función "".
- `public static BigInteger getPrime(int length, Random random)`: Función proveniente de la librería Paillier, la cual permite generar números primos con un largo determinado.

- **BGN_key.java:**

  Clase que permite la generación y almacenado de las llaves públicas del criptosistema.

  - `public BGN_key()`: Constructor de la clase.
  - `public BGN_key(BigInteger N, long seed)`: A
  - `public BGN_key(BigInteger N, Random rnd)`:A
  - `public BGN_key(BigInteger N, BigInteger q2, ECPoint g, ECPoint h, long seed)`:a
  - `public BGN_key(BigInteger N, BigInteger q2, ECPoint g, ECPoint h, Random rnd)`: Función que almacena los valores de la llave pública generada por la clase "BGNKeyGen".
  - `public BGN_key getPublicKey()`: Función Get que retorna los valores de la llave pública.
  - `public BigInteger getN()`: Función Get que retorna la llave "N" del conjunto de las llaves públicas.
  - `public BigInteger getQ2()`: Función Get que retorna la llave "q2" del conjunto de las llaves públicas.
  - `public ECPoint getG()`: Función Get que retorna la llave "g" del conjunto de las llaves públicas.
  - `public ECPoint getH()`: Función Get que retorna la llave "h" del conjunto de las llaves públicas.

- **BGN_privateKey.java:**

  Clase de la librería que permita el almacenado de los valores de las llaves, tanto privadas como públicas, generadas mediante la clase BGNKeyGen.

  - `public BGN_privateKey(BigInteger N, BigInteger q1, BigInteger q2, BigInteger q, ECPoint g, ECPoint h, ECCurve curve, long seed)`: Función encargada de almacenar los valores de la llave privada como permitir el acceso a la clase "BGN_key" para el almacenado de las llaves públicas generadas por la clase "BGNKeyGen".
  - `public BigInteger getQ1()`: Función Get que retorna la llave "q1" del conjunto de las llaves privadas.
  - `public ECCurve getCurve()`: Función Get que retorna la Curva Elíptica "e" generada por el Criptosistema.
  - `public BigInteger getQ()`: Función Get que retorna la llave "q" del conjunto de las llaves privadas.

- **BGN_p.java:**

  Clase hija de AbstractBGN, llamada para realizar tanto la implementación de la encriptación, como de la desencriptación.

  - `public BGN_p()`: Constructor vacío de la clase.
  - `public BGN_p(BGN_key key)`: Constructor con llave pública del criptosistema.
  - `public BGN_p(BGN_privateKey key)`: Constructor con llave privada del criptosistema.
  - `public void setEncryption(BGN_key key)`: Función que permite activar sólo el modo de encriptación de la variable "BGN_p" generada.
  - `public void setDecryption(BGN_privateKey key)`: Función que permite activar sólo el modo de desencriptación de la variable "BGN_p" generada.
  - `public void setDecryptEncrypt(BGN_privateKey key)`: Función que permite activar tanto el modo de encriptación como el modo de desencriptación de la variable "BGN_p" generada.
  - `public BGN_privateKey getPrivateKey())`: Función get que permite obtener la llave privada del criptosistema.
  - `public BigInteger decrypt_BGN(ECPoint encrypted)`: Función de desencriptación de una variable "ECPoint" previamente generada, esta función posee dos resultados:
    - El valor almacenable en BigInteger del mensaje previamente encriptado.
    - O un valor almacenable en BigInteger de un -1, el cual significa que no encontró el resultado de la función.

Uso de la librería:

1. Se debe generar un elemento de la clase *"BGNKeyGen.java"*.

```
BGNKeyGen BGNkg =new BGN_p();
```

2. Se genera un nuevo elemento de la clase *"BGN_p.java"*

```
BGN_p esystem= new BGN_p();
```

3. Se genera un nuevo valor de la clase *"BGN_privateKey.java"* la cual se agrega al elemento de la clase *"BGN_p.java.*

```
BGN_privateKey key= BGNkg.BGNKey(24,25165);
esystem.setDecryptEncrypt(key);
```

4. Para encriptar un valor, por ejemplo el número diez, se debe generar una variable "ECPoint" de la librería [38] para almacenar el valor cifrado, y utilizar el elemento generado de la clase *"BGN_p.java"*, llamando a la función *"encrypt(BigInteger m)"*.

```
ECPoint c;
c=esystem.encrypt(m);
```

5. Para desencriptar un valor cifrado "C" se debe generar una variable BigInteger para almacenar el valor del texto plano, y utilizar el elemento generado de la clase *"BGN _p.java"*, llamando a la función *"BigInteger decrypt_BGN(BigInteger c)"*.

```
BigInteger m= esystem.decrypt_BGN(c);
```

4.2.2.1: Diagrama de Clases

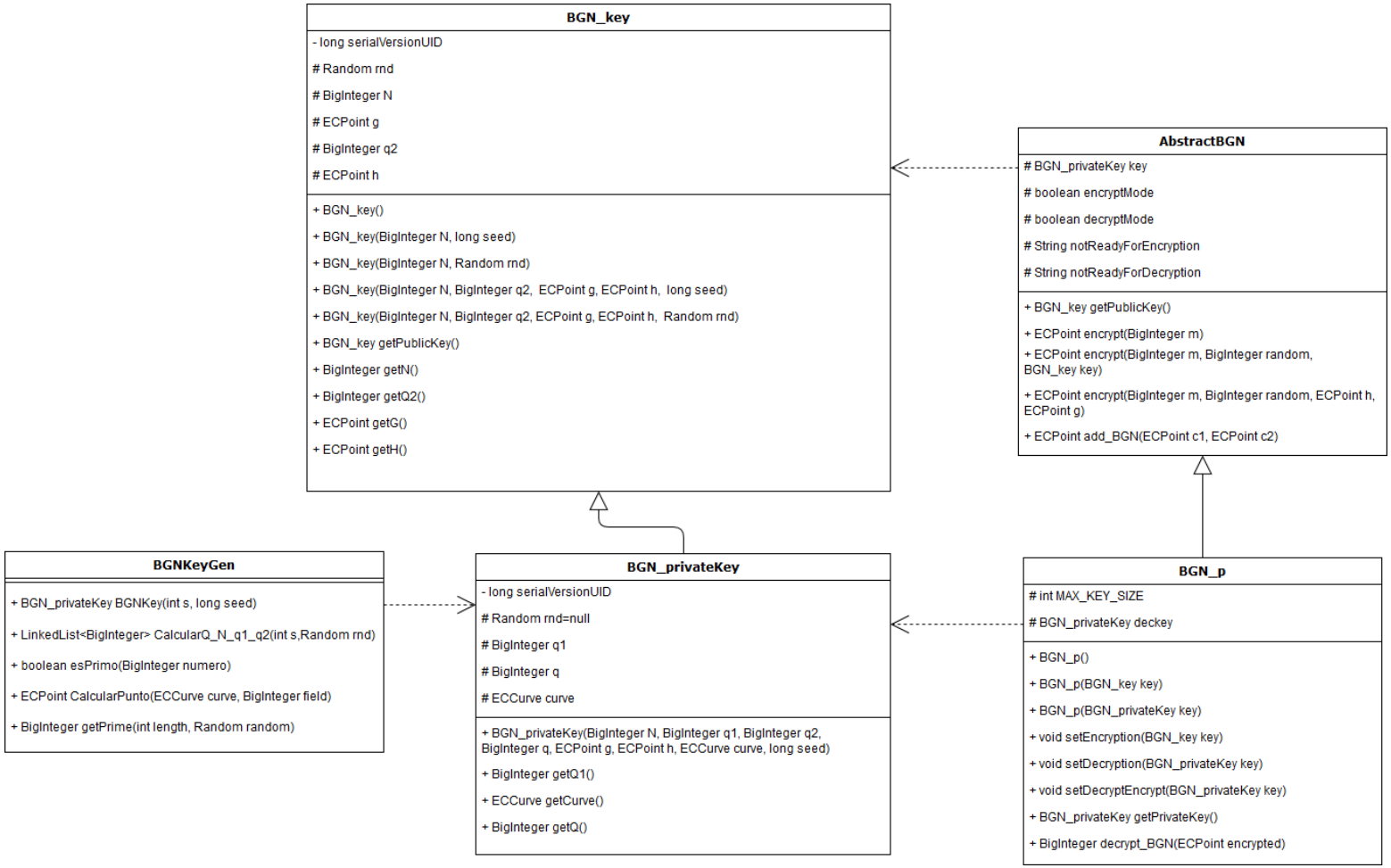


Ilustración 6: *Diagrama de Clases de Librería Boneh-Goh-Nissim, fuente propia.*

4.2.2.2: Código Fuente:

El código fuente de esta librería se encuentra en el Capítulo de Anexos, Sección 9.1.1.

4.2.2.3: Observación:

- La complejidad de Boneh-Goh-Nissim se encuentra en que, para realizar la encriptación, desencriptación, y generación de algunos de los valores de las llaves del sistema, se necesita implementar sumas y multiplicaciones vectoriales.
  - Para implementar dichas operaciones, se utilizó una librería externa enfocada en curvas elípticas [38].
- El tiempo de generación de llaves del criptosistema es extremadamente extenso y no garantiza el correcto funcionamiento de la función generada.
  - Se ha determinado de que el funcionamiento del criptosistema, cuando se utiliza la función KeyGen puede o no funcionar correctamente, punto explicado en la siguiente sección.
- La implementación de este criptosistema posee problemas al realizarse la encriptación del valor 0, retornando un valor "-1" el cual tambien es la respuesta del sistema ante la imposibilidad de encontrar el mensaje encriptado mediante el proceso actual de desencriptación.

4.2.2.4: Problema de la Librería.

Actualmente existe un problema en la implementación del sistema, para lo cual, en el archivo “test.java” se implementó el siguiente código en la prueba:

```
BGN_p esystem = new BGN_p();
System.out.println("Prueba funcionalidad");
BGNKeyGen bgn_kg= new BGNKeyGen();
long startT = System.currentTimeMillis();
BGN_privateKey pr_k = bgn_kg.BGNKey(24,25165);
long stopT = System.currentTimeMillis();
System.out.println("Tiempo Generacion Llaves: "+ (stopT-startT) + " milisegundos");
System.out.println("\nValores Llaves Privadas");
System.out.println("\t q1:" + pr_k.getQ1().toString()+"\tq:" + pr_k.getQ().toString());
System.out.println("Valores Llaves Públicas");
System.out.println("\t N: "+pr_k.getN().toString()+"\t Q2: "+
     pr_k.getQ2().toString()+"\t g(x:"+
     pr_k.getG().getXCoord().toBigInteger().toString()+",y:"+
     pr_k.getG().getYCoord().toBigInteger().toString()          +")\t h(x:" +
     pr_k.getH().getXCoord().toBigInteger().toString()          +",y:"+
     pr_k.getH().getYCoord().toBigInteger().toString()+")");

/*Prueba de encriptación*/
System.out.println("\nPrueba de Encriptación");
esystem = new BGN_p();
esystem.setDecryptEncrypt(pr_k);
startT = System.currentTimeMillis();
ECPoint encoded1 =  esystem.encrypt(BigInteger.valueOf(2));
System.out.print("Texto cifrado: ");
System.out.print(encoded1.getXCoord().toBigInteger().toString(10)+","+
     encoded1.getYCoord().toBigInteger().toString(10));
stopT = System.currentTimeMillis();
System.out.println("\nTiempo Encriptación: "+ (stopT-startT) + " milisegundos");
System.out.println("Tamaño
valores:"+encoded1.getXCoord().toBigInteger().bitLength()+","+encoded1.getYCoord().toBigInteg
er().bitLength());
startT = System.currentTimeMillis();
BigInteger m = esystem.decrypt_BGN(encoded1);
     System.out.println("Texto descifrado: ");        System.out.print(m.toString());
stopT = System.currentTimeMillis();
System.out.println("Tiempo Desencriptación: "+ (stopT-startT) + " milisegundos");

/*Prueba de funcionalidad encriptación*/
System.out.println("\nPrueba de encriptación en un ciclo for ");
ECPoint temp, temp2;
BigInteger mtemp;
int cont= 0;
for(int i=1;i<100; i++){
     temp = esystem.encrypt(BigInteger.valueOf(i));
     mtemp= esystem.decrypt_BGN(temp);
     int test = mtemp.intValue();
     if(test != -1){//System.out.println(i +" corresponde: "+ mtemp.toString());
               cont++;
     }
}
System.out.println("Valores de 1 a 99 desencriptados correctamente: " + cont);
```

```
/*Prueba suma Homomórfica*/
System.out.println("\nPrueba de Homomorfismo en un ciclo for ");
cont= 0;
temp2 = esystem.encrypt(BigInteger.valueOf(1));
for(int i=2;i<100; i++){
     temp = esystem.encrypt(BigInteger.valueOf(1));
     mtemp= esystem.decrypt_BGN(temp);
     int test = mtemp.intValue();
     if(test != -1){ //System.out.println(i +" corresponde. ");
               temp2 = temp2.add(temp);     cont++;
     }
}
mtemp = esystem.decrypt_BGN(temp2);
System.out.println(cont+" La suma homomórfica corresponde: "+ mtemp.toString());
```

Al ejecutarse la ejecución de la prueba en diversas ocasiones se pudo determinar de que existe un problema de estabilidad al momento de generarse las llaves públicas “g” y “h”, las cuales al ser puntos válidos de la Curva Elíptica, por razones desconocidas no aseguran el correcto funcionar de la suma homomórfica ni de la correcta encriptación de diferentes mensajes con un rango $[\,1, q_2]$.

Esta suposición se pudo observar en la ejecución puesto que, al momento de generar la llave privada "BGN_privateKey pr_k = bgn_kg.*BGNKey*(24,25165)" los valores de las llaves "N", "q_1", "q_2", "q" se mantuvieron constantes al ser valores "BigInteger" generados por la misma semilla "$s = 24$", existiendo diferencias en los valores de las llaves "g" y "h".

Generándose, al momento de ejecutarse el código, tres tipos de resultados en la prueba:

- **Resultado de Funcionamiento Total.**

```
Prueba funcionalidad
Tiempo Generacion Llaves: 86471 milisegundos

Valores Llaves Privadas
         q1:2137        q:27430531
Valores Llaves Públicas
         N: 6857633      Q2: 3209        g(x:6331871,y:6730421)  h(x:17588282,y:14119986)

Prueba de Encriptación
Texto cifrado: 15732019,8120572
Tiempo Encriptación: 5 milisegundos
Tamaño valores:24,23
Texto descifrado:
2Tiempo Desencriptación: 288 milisegundos

Prueba de encriptación en un ciclo for
Valores de 1 a 99 desencriptados correctamente: 99

Prueba de Homomorfismo en un ciclo for
99 La suma homomórfica corresponde: 99
```

Ilustración 7: *Resultado prueba BGN donde la suma homomórfica* ***Funciona*** *correctamente ante el funcionamiento* ***TOTAL*** *del proceso de encriptación y desencriptación, fuente propia.*

- **Resultado de Funcionamiento Parcial, con falla en la suma homomórfica.**

```
Prueba funcionalidad
Tiempo Generacion Llaves: 31958 milisegundos

Valores Llaves Privadas
         q1:2137        q:27430531
Valores Llaves Públicas
         N: 6857633      Q2: 3209        g(x:19238590,y:12770268)        h(x:5753165,y:5243511)

Prueba de Encriptación
Texto cifrado: 10455711,6314240
Tiempo Encriptación: 4 milisegundos
Tamaño valores:24,23
Texto descifrado:
2Tiempo Desencriptación: 308 milisegundos

Prueba de encriptación en un ciclo for
Valores de 1 a 99 desencriptados correctamente: 25

Prueba de Homomorfismo en un ciclo for
27 La suma homomórfica corresponde: -1
```

Ilustración 8: *Resultado prueba BGN donde la suma homomórfica* ***No Funciona*** *correctamente ante el funcionamiento* ***PARCIAL*** *del proceso de encriptación y desencriptación, fuente propia.*

- **Resultado de Funcionamiento Parcial, con un correcto funcionamiento de la suma homomórfica.**

```
Prueba funcionalidad
Tiempo Generacion Llaves: 32277 milisegundos

Valores Llaves Privadas
         q1:2137        q:27430531
Valores Llaves Públicas
         N: 6857633      Q2: 3209        g(x:15720416,y:11530029)        h(x:23820832,y:17279677)

Prueba de Encriptación
Texto cifrado: 4630082,5228472
Tiempo Encriptación: 4 milisegundos
Tamaño valores:23,23
Texto descifrado:
2Tiempo Desencriptación: 223 milisegundos

Prueba de encriptación en un ciclo for
Valores de 1 a 99 desencriptados correctamente: 55

Prueba de Homomorfismo en un ciclo for
53 La suma homomórfica corresponde: 53
```

Ilustración 9: *Resultado prueba BGN donde la suma homórfica* ***Funciona*** *correctamente ante el funcionamiento* ***PARCIAL*** *del proceso de encriptación y desencriptación, fuente propia.*

Lo cual, junto al tiempo de generación de las llaves, dificulta la realización de las pruebas del funcionamiento del sistema además de volver complejo el desarrollo de las pruebas de homomorfismos ante lo inestable que se vuelve el criptosistema con respecto a la suma homomórfica.

Por ello, y ante la imposibilidad existente de desencriptar correctamente el valor 0.

- Las pruebas de homomorfismo serán omitidas.
- Para las restantes pruebas se asumirá que el criptosistema generado entrega los resultados esperados.

4.2.3: Okamoto-Uchiyama

Tomando como base la distribución realizada en la librería de Paillier desarrollada por Murat Kantarcioglu, James Garrity y Sean Hall para la realización de la librería de la criptografía de Okamoto-Uchiyama se han realizado cinco clases más una clase de prueba (test.java) en la cual se realizaron revisiones del funcionamiento del criptosistema.

- **AbstractOkamotoUchiyama.java:**

  Clase Abstracta del criptosistema Okamoto-Uchiyama, en la cual se encuentra el método de encriptación, y el método que permite la implementación de la suma homomórfica de dos valores cifrados aplicando las mismas llaves. Sus funciones son:

  - `public OkamotoUchiyama_key getPublicKey:` Función pública que retorna las llaves públicas del criptosistema de Okamoto-Uchiyama.
  - `public BigInteger encrypt_ou(BigInteger m):` Función inicial de la encriptación de un mensaje m en la cual se genera un valor aleatorio para pasar a la siguiente función.
  - `public static BigInteger encrypt_ou(BigInteger m, BigInteger r, OkamotoUchiyama_key key):` Función secundaria de la encriptación de un mensaje m, un valor aleatorio y la llave del sistema, en esta función se ingresaran los valores de la llave pública para finalmente ingresar a la función de encriptación.
  - `public static BigInteger encrypt_ou(BigInteger m, BigInteger r, BigInteger h, BigInteger g, BigInteger n):` Función en la cual se realiza el proceso de encriptación del mensaje m, aplicando la fórmula de cifrado del criptosistema.

    $$C = g^M h^r \ mod \ n$$

  - `public BigInteger addOU(BigInteger c1, BigInteger c2):` Función en la cual se realiza la suma de dos textos cifrados por las mismas llaves públicas.

- **OUKeyGen.java:**

  Clase que permite la generación de las llaves del criptosistema Okamoto-Uchiyama.

  - `public static OkamotoUchiyama_privateKey OkamotoUchiyamaKey(int s, long seed)`: Función en la cual se generan los valores de la llaves del criptosistema, en la cual el valor "s", para un óptimo funcionamiento debe poseer un valor $0 < s < 25$, mientras que el valor "seed" es para el inicializar el randómico del sistema.
  - `public static BigInteger primitiveRootModulo_Gen(BigInteger n, Random rnd, BigInteger p)`: Función que retorna un valor aleatorio el cual permite cumplir la propiedad de $g \in (Z/nZ)^*$.
  - `public static boolean primitiveRoot(BigInteger p, int a)`: Función [32]. que permite determinar si un valor calculado es un número primitivo.
  - `public static BigInteger getPrime(int length, Random random)`: Función proveniente de la librería Paillier, la cual permite generar números primos con un largo determinado.

- **OkamotoUchiyama_key.java:**

  Clase que permite la generación y almacenado de las llaves públicas del criptosistema.

  - `public OkamotoUchiyama_key()`: Constructor simple de la clase.
  - `public OkamotoUchiyama_key(BigInteger n, BigInteger g, BigInteger h, long seed)`: Función que permite almacenar los valores generados en OUKeyGen.java para el almacenado de los valores de la llave pública del criptosistema.
  - `public OkamotoUchiyama_key(BigInteger n, BigInteger g, BigInteger h, Random rnd)`: Función que permite almacenar los valores generados como elementos de la llave pública, junto al valor aleatorio.
  - `public OkamotoUchiyama_key getPublicKey()`: Función que retorna los valores de las llaves públicas como un elemento de la clase OkamotoUchiyama_key.
  - `public BigInteger getG()`: Función que retorna el valor de la llave pública "g".
  - `public BigInteger getH()`: Función que retorna el valor de la llave pública "h".
  - `public BigInteger getN()`: Función que retorna el valor de la llave pública "n".
  - `public static boolean inModN(BigInteger a, BigInteger n)`: Función perteneciente a la librería Paillierkey, la cual permite determinar si un valor pertenece al conjunto existen en el conjunto Z_n.
  - `public static boolean inModNStar(BigInteger a, BigInteger n)`: Función perteneciente a la librería Paillierkey, la cual permite determinar si un valor pertenece al conjunto existen en el conjunto Z_n^*.
  - `public BigInteger getRandomModNStar()`: Función perteneciente a la librería Paillierkey, la cual obtener un valor perteneciente al conjunto existen en el conjunto Z_n^*.

- **OkamotoUchiyama_privateKey.java:**

  Clase de la librería que permita el almacenado de los valores de las llaves, tanto privadas como públicas, generadas mediante la clase OUKeyGen.

  - `public OkamotoUchiyama_privateKey(BigInteger p, BigInteger q, BigInteger n, BigInteger g, BigInteger h, long seed)`: Función que permite almacenar las llaves privadas y públicas que han sido generadas en la clase OUKeyGen.java.
  - `public BigInteger getQ()`: Función que retorna el valor de la llave pública "q".
  - `public BigInteger getP()`: Función que retorna el valor de la llave pública "p".

- **OkamotoUchiyama_p.java:**

  Clase hija de AbstractOkamotoUchiyama, llamada para realizar tanto la implementación de la encriptación, como de la desencriptación.

  - `public OkamotoUchiyama_p()`: Constructor simple de la clase.
  - `public OkamotoUchiyama_p(OkamotoUchiyama_key key)`: Constructor de clase que ingresa la llave pública de este criptosistema
  - `public OkamotoUchiyama_p(OkamotoUchiyama_privateKey key)`: Constructor de clase que ingresa la llave privada de este criptosistema
  - `public void setEncryption(OkamotoUchiyama_key key)`: Función que permite ingresar un conjunto de llave pública para habilitar el proceso de encriptación.
  - `public void setDecryption(OkamotoUchiyama_privateKey key)`: Función que permite ingresar un conjunto de llave pública para habilitar el proceso de desencriptación.
  - `public void setDecryptEncrypt(OkamotoUchiyama_privateKey key)`: Función que, al ingresar una llave privada, permite habilitar tanto el proceso de encriptación como el de desencriptación.
  - `public OkamotoUchiyama_privateKey getPrivateKey()`: Función que regresa los valores de la llave privada del criptosistema.
  - `public BigInteger decrypt_ou(BigInteger c)`: Función que permite desencriptar un texto cifrado "C" para recobrar el mensaje original utilizando la fórmula:

    $$m = \frac{L(C^{p-1} \bmod p^2)}{L(g^{p-1} \bmod p^2)} \bmod p$$

    En donde, para obtener el mensaje original al resultado de $L(g^{p-1} \bmod p^2)$, hay que aplicarle módulo inverso de p, para luego multiplicarlo con el valor resultante de $L(C^{p-1} \bmod p^2)$ y al resultado de esta multiplicación se le aplica módulo de p.

Uso de la librería:

1. Se debe generar un elemento de la clase *"OUKeyGen.java"*.

```
OUKeyGen OUkg = new OUKeyGen();
```

2. Se genera un nuevo elemento de la clase *"OkamotoUchiyama_p.java"*

```
OkamotoUchiyama_p esystem= new OkamotoUchiyama_p();
```

3. Se genera un nuevo valor de la clase *"OkamotoUchiyama_privateKey.java"* la cual se agrega al elemento de la clase*"OkamotoUchiyama_p.java.*

```
OkamotoUchiyama_privateKey key=
    OUkg.OkamotoUchiyamaKey(24,122333356);
esystem.setDecryptEncrypt(key);
```

4. Para encriptar un valor, ejemplo el número diez, se debe generar una variable BigInteger para almacenar el valor cifrado, y utilizar el elemento generado de la clase *"OkamotoUchiyama_p.java"*, usando a la función *"BigInteger encrypt_ou(BigInteger m)"*.

```
BigInteger c = esystem.encrypt_ou(BigInteger.valueOf(10));
```

5. Para desencriptar un valor cifrado "C", se debe generar una variable BigInteger para almacenar el valor del texto plano, y utilizar el elemento generado de la clase *"OkamotoUchiyama_p.java"*, llamando a la función *"BigInteger decrypt_ou(BigInteger c)"*.

```
BigInteger m= esystem.decrypt_ou(c);
```

6. Para realizar la suma homomórfica de dos valores encriptados se deben generar dos valores BigInteger, “c1” y “c2”, los cuales deben de almacenar el texto cifrado de dos valores numéricos, para luego almacenar en un tercer BigInteger el resultado de la suma que se obtiene utilizar el elemento generado de la clase *“OkamotoUchiyama_p.java”,* llamando a la función *“BigInteger addOU(BigInteger c1, BigInteger c2”*, finalmente este tercer valor debe ser desencriptado para comprobar si se ha cumplido la suma homomórfica.

```
BigInteger c1 = esystem.encrypt_ou(BigInteger.valueOf(105));
BigInteger c2= esystem.encrypt_ou(BigInteger.valueOf(151));
BigInteger result = esystem.addOU(c1, c2);
BigInteger mplus= esystem.decrypt_ou(result);
```

4.2.3.1: Diagrama de Clases

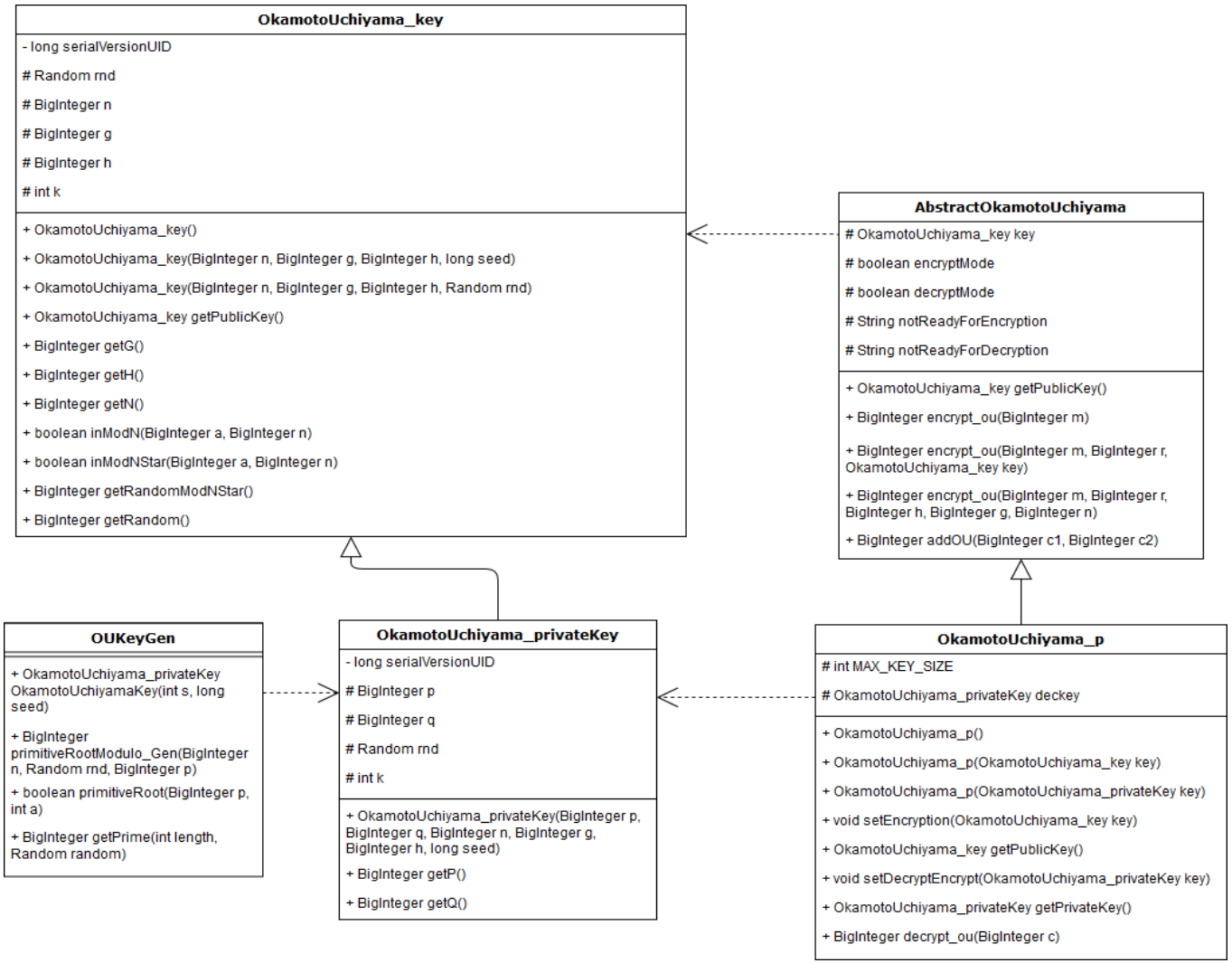


Ilustración 10: *Diagrama de Clases de Librería Okamoto-Uchiyama, fuente propia.*

4.2.3.2: Código Fuente:

El código fuente de esta librería se encuentra en el Capítulo de "Anexos", Sección 9.1.2.

4.2.3.3: Observación:

- En esta implementación, el tiempo que el criptosistema tarda en generar las llaves es mayor al tiempo que tarda la librería de Paillier en generarlas, este tiempo varía del valor del "int s" (cuyo rango es $0 < s < 25$) elegido para la función "`OkamotoUchiyamaKey(int s, long seed)`".
  - Si el valor de s es más cercano al 25, el tiempo de cálculo de las llaves públicas será mayor que de poseer un s cercano al 0.
  - Por otro lado, si el tamaño de s es cercano al valor 0, las llaves p y q poseerán valores muy pequeños volviéndose inseguras.
- En el proceso de Generación de Llaves, la generación de la llave pública g es el valor que más tiempo toma en generar ante el uso de la función "`primitiveRootModulo_Gen(BigInteger n, Random rnd, BigInteger p)`" por ende, de lograrse mejorar dicha función el tiempo de generación de llaves sería considerablemente menor.

4.2.4: ElGammal

Tomando como base la distribución realizada en la librería de Paillier desarrollada por Murat Kantarcioglu, James Garrity y Sean Hall, junto a la implementación simplificada de [35], se procedió a realizar la implementación de la librería de ElGammal.

No obstante se puede observar que mantener la fórmula de encriptación:

$$E(m) = (C_1, C_2)$$
$$C_1 = m * h^r$$
$$C_2 = g^r$$

El Homomorfismo se aplica tan sólo de forma multiplicativa y, como se especifica en [36], para implementar el homomorfismo de forma aditiva, se deberá de editar la fórmula de $C_1 = g^m * h^r$ lo cual implica una modificación en la desencriptación del esquema. Modificación que no se realizó por motivos de tiempo.

La librería desarrollada basándose en el formato implementado en la librería de Paillier, consta de cinco clases, las cuales serán descritas a continuación.

- **AbstractElGammal.java:**

  Clase Abstracta del criptosistema ElGammal, en la cual se encuentra el método de encriptación, y el método que permite la implementación de la multiplicación homomórfica de dos valores cifrados aplicando las mismas llaves. Sus funciones son:

  - `public ElGammal_key getPublicKey():` Función pública que retorna las llaves públicas del criptosistema de ElGammal.
  - `public LinkedList<BigInteger> encrypt_EG(BigInteger m):` Función inicial de la encriptación de un mensaje m en la cual se genera un valor aleatorio para pasar a la siguiente función.
  - `public LinkedList<BigInteger> encrypt_EG(BigInteger m, BigInteger r, ElGammal_key key):` Función secundaria de la encriptación de un mensaje m y un valor aleatorio r, en la cual se realiza un llamado a la siguiente función adjuntándole las llaves públicas generadas por el sistema.

- `public LinkedList<BigInteger> encrypt_EG(BigInteger m,BigInteger r, BigInteger p, BigInteger g, BigInteger y):` Función en la cual se realiza el proceso de encriptación del mensaje m, aplicando la fórmula de cifrado del criptosistema., aplicando:

$$E(m) = (C_1, C_2)$$
$$C_1 = m * h^r$$
$$C_2 = g^r$$

- `public LinkedList<BigInteger> mul_EG(LinkedList<BigInteger> c1, LinkedList<BigInteger> c2):` Función en la cual se realiza la multiplicación de dos textos cifrados por las mismas llaves públicas.

- **EGKeyGen.java:**

  Clase que permite la generación de las llaves del criptosistema ElGammal.

  - `public static ElGammal_privateKey ElGammalKey(int s, long seed):` Función en la cual se generan los valores de la llaves del criptosistema. En la cual el valor "s", para un óptimo funcionamiento, debe poseer un valor $s > 0$, mientras que el valor "seed" es para el inicializar el randómico del sistema.
  - `public static BigInteger generarG(BigInteger p, Random random):` Función que permite la generación de un valor aleatorio g, que se encuentre en el rango [1,p].
  - `public static BigInteger getPrime(int length, Random random):` Función proveniente de la librería Paillier, la cual permite generar números primos con un largo determinado.

- **ElGammal_key.java:**

  Clase que permite la generación y almacenado de las llaves públicas del criptosistema:

  - `public ElGammal_key()`: Constructor Simple de la clase.
  - `public ElGammal_key(BigInteger p,BigInteger g, BigInteger y, long seed)`: Función que permite almacenar los valores generados en EGKeyGen.java para el almacenado de los valores de la llave pública del criptosistema.
  - `public ElGammal_key(BigInteger p,BigInteger g,BigInteger y, Random rnd)`: Función que permite almacenar los valores generados como elementos de la llave pública, junto al valor aleatorio.
  - `public ElGammal_key getPublicKey()`: Función que retorna los valores de las llaves públicas como un elemento de la clase ElGammal_key.
  - `public BigInteger getG()`: Función que retorna el valor de la llave pública "g".
  - `public BigInteger getY()`: Función que retorna el valor de la llave pública "y".
  - `public BigInteger getP()`: Función que retorna el valor de la llave pública "p".
  - `public BigInteger getRandom()`: Función que permite generar un valor aleatorio.

- **ElGammal_privateKey.java:**

  Clase de la librería que permita el almacenado de los valores de las llaves, tanto privadas como públicas, generadas mediante la clase EGKeyGen. Sus funciones son:

  - `public ElGammal_privateKey(BigInteger p, BigInteger g, BigInteger y, BigInteger x, long seed)`: Función que, además de permitir el futuro almacenado de los valores de las llaves públicas, permite el almacenado de la llave privada "x" generada en la clase "EGKeyGen".
  - `public BigInteger getX()`: Función que retorna el valor de la llave privada "x".

- **ElGammal_p.java:**

  Clase hija de AbstractElGammal, llamada para realizar tanto la implementación de la encriptación, como de la desencriptación.

  Sus funciones son:

  - `public ElGammal_p()`: Constructor vacío de la clase.
  - `public ElGammal_p(ElGammal_key key)`: Constructor de clase que ingresa la llave pública de este criptosistema.
  - `public ElGammal_p(ElGammal_privateKey key)`: Constructor de clase que ingresa la llave privada de este criptosistema.
  - `public void setEncryption(ElGammal_key key)`: Función que permite ingresar un conjunto de llave pública para habilitar el proceso de encriptación.

- `public void setDecryption(ElGammal_privateKey key)`: Función que permite ingresar un conjunto de llave pública para habilitar el proceso de desencriptación.
- `public void setDecryptEncrypt(ElGammal_privateKey key)`: Función que, al ingresar una llave privada, permite habilitar tanto el proceso de encriptación como el de desencriptación.
- `public ElGammal_privateKey getPrivateKey()`: Función que regresa los valores de la llave privada del criptosistema.
- `public BigInteger decrypt_EG(LinkedList<BigInteger> c)`: Función de desencriptación de los valores C_1 y C_2, que se encuentran almacenados en la variable c, específicamente en: $c[0] = C_1$ y $c[1] = C_2$. Para lo cual se ha de implementar la fórmula $m = C_1(C_2^x)^{-1}$.

Uso de la librería:

1. Se debe generar un elemento de la clase *"EGKeyGen.java"*.

```
EGKeyGen EGkg = new EGKeyGen();
```

2. Se genera un nuevo elemento de la clase *"ElGammal_p.java"*

```
ElGammal_p esystem= new ElGammal_p();
```

3. Se genera un nuevo valor de la clase *"ElGammal_privateKey.java"* la cual se agrega al elemento de la clase*"OkamotoUchiyama_p.java.*

```
ElGammal_privateKey key=
     EGkg.ElGammalKey(245,14656546);
esystem.setDecryptEncrypt(key);
```

4. Para encriptar un valor, por ejemplo el número diez, se debe generar una variable LinkedList<BigInteger> para almacenar el valor cifrado, y utilizar el elemento generado de la clase *"ElGammal_p.java"*, llamando a la función *"BigInteger encrypt_EG(BigInteger m)"*.

```
LinkedList<BigInteger> c = esystem.encrypt_EG(BigInteger.valueOf(10));
```

5. Para desencriptar los valores cifrados almacenados en la variable LinkedList "C", se debe generar una variable BigInteger para almacenar el valor del texto plano, y se debe utilizar el elemento generado de la clase *"ElGammal _p.java"*, llamando a la función *"BigInteger decrypt_EG(LinkedList<BigInteger> c)"*.

```
BigInteger m= esystem.decrypt_EG(c);
```

4.2.4.1: Diagrama de Clases

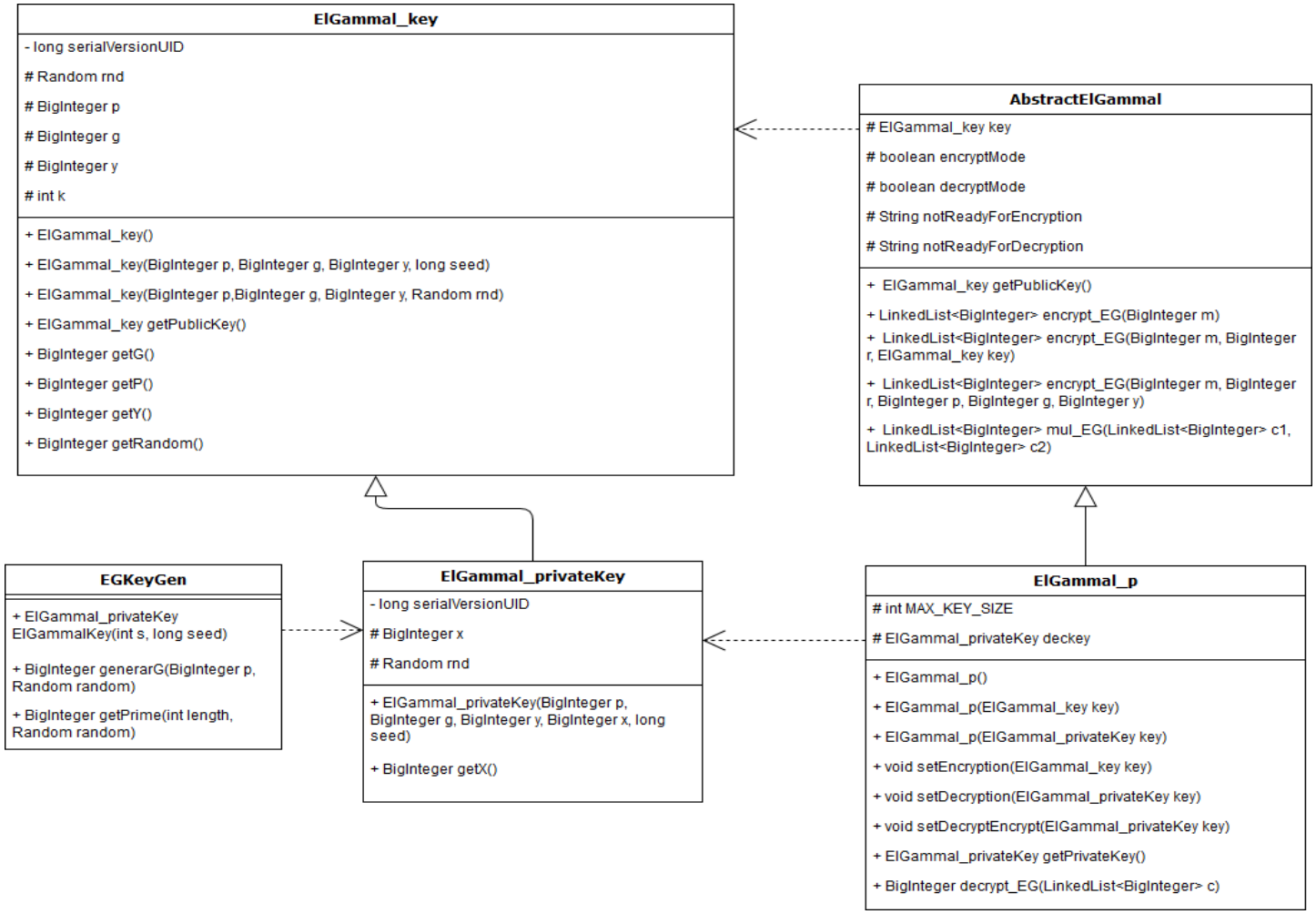


Ilustración 11: *Diagrama de Clases de Librería ElGammal, fuente propia.*

4.2.4.2: Código Fuente:

El código fuente de esta librería se encuentra en el capítulo de Anexos.

4.2.4.3: Observación

- Dado que la forma de encriptación del valor C_1 del código es $C_1 = m * h^r$, en el instante en que el valor a encriptar sea $m = 0$, el sistema queda vulnerable puesto que se podrá obtener un valor de C_1, $C_1 = 0$.

Sección 4.3: Descripción de las pruebas realizadas.

4.3.1: Pruebas de criptosistemas.

A continuación, en esta sección se registrarán tanto el código implementado, junto a los resultados obtenidos al realizarse en las siguientes pruebas:

- Pruebas de Encriptación & Desencriptación a un valor 1 en primera instancia.
- Pruebas de Encriptación & Desencriptación a dos valores (1 y 0) almacenados en un arreglo.
- Pruebas de Encriptación & Desencriptación a un arreglo compuesto por dos valores.
- Pruebas de Homomorfismo & Desencriptación de dos valores.
- Pruebas de Encriptación y Desencriptación de valores almacenados en un vector contabilizados mediante homomorfismo.
- Pruebas de Encriptación y Desencriptación del valor 1 con diferentes intervalos.

Además de ello se adjuntarán tablas y/o gráficos comparativos para analizar las diferencias existentes entre los diferentes esquemas criptográficos en cada prueba.

En la codificación de estas pruebas, los valores utilizados se encuentran implementado mediante el uso de una variable BigInteger cuyo rango de valores se encuentra en el rango Binario, (1 y 0).

4.3.1.1: Pruebas de Encriptación & Desencriptación a un valor 1 en primera instancia:

En esta prueba, se busca obtener los tiempos de ejecución media tanto del proceso de encriptación y del proceso de desencriptación, además de buscarse obtener un tamaño medio del texto cifrado, para lo cual se implementó el uso de la función "`bitLength()`".

4.3.1.1.1: Tabla de la prueba

Media	*Paillier*	*Boneh-Goh-Nissim*	*Okamoto Uchiyama*	*ElGammal*
Tiempo de encriptación [Milisegundos]	4 ~ 26	0~4	0 ~ 23	0
Tamaño texto Cifrado [Función `bitLength()`]	872	(x:24~25 y: 24~25)	70	C1: 244 C2: 239
Tiempo de Desencriptación [Milisegundos]	4 ~ 10	107 ~ 259	0 ~ 2	0 ~ 16

Tabla 2: *Tabla Resumen de Prueba de Encriptación y Desencriptación de un valor 1, fuente propia.*

4.3.1.1.2: DFD de la Prueba.

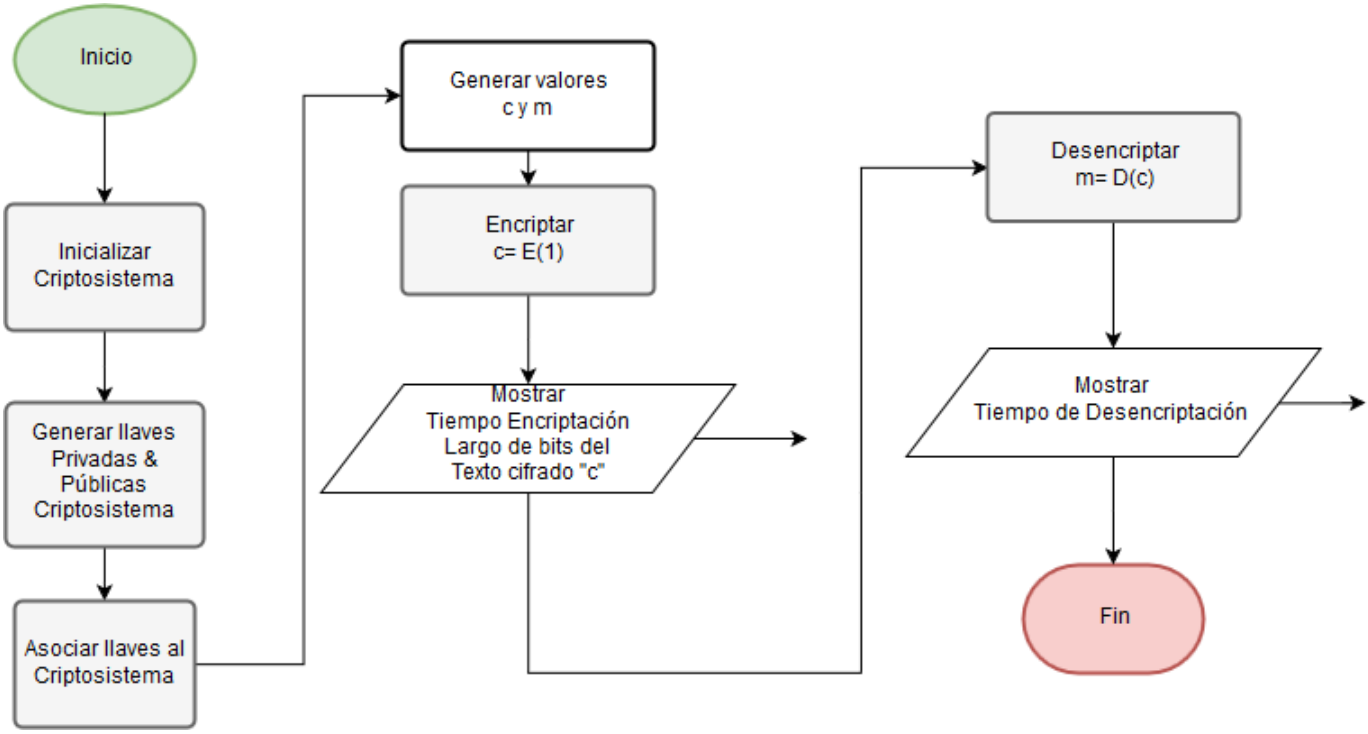


Ilustración 12: *DFD Base de Prueba Encriptación & Desencriptación a un valor "1", fuente propia.*

4.3.1.2: Pruebas de Encriptación & Desencriptación a dos valores (1 y 0) almacenados en un arreglo:

En esta prueba se considerará el valor de bit del Arraylist como la suma de los bits que corresponden a los valores 1 y 0 cifrados e ingresados posteriormente, en esta prueba se calculará el tiempo que tarda el encriptar y almacenar en un Arraylist dos valores.

4.3.1.2.1: Tabla de la prueba

	Paillier	*Boneh-Goh-Nissim*	*Okamoto Uchiyama*	*ElGammal*
Tiempo de encriptación [Milisegundos]	13 ~ 20	0 ~ 13	0 ~ 16	0 ~ 3
Tamaño texto Cifrado (Suma de ambos valores cifrados)	1741	(x: 43 ~ 50 , y: 43~50)[1]	140	C1: 483 C2:239[2]
Tiempo de Desencriptación [Milisegundos]	12 ~ 18	188 ~ 1022	0 ~ 2	1 ~ 5

Tabla 3: *Tabla Prueba de Encriptación y Desencriptación de dos valores (1 y 0) almacenados en un arreglo, fuente propia.*

[1] La Encriptación de Boneh-Goh-Nissim está conforadas corresponde a un punto (ECPoint) de una curva elíptica, por lo que, para realizar el cálculo de la suma se realizó una suma de los valores en enteros de los valores "x" e "y" de los puntos almacenados y se utilizó la función `.bitLength()` para determinar el valor que dichos puntos ocupan en el arreglo generado por la correspondiente prueba.

[2] La Encriptación de ElGammal forma un arreglo compuesto por los valores C1, C2.
En la implementación realizada de tal esquema criptográfico, ambos valores son almacenados en un elemento de la clase LinkedList, por lo cual el valor C1 y C2 ingresado en la tabla son los valores obtenidos de la suma de:

- Valor determinado con la función `.bitLength()` de un número 1 encriptado: c1:244 c2:239
- Valor determinado con la función `.bitLength()` de un número 0 encriptado: c1:239 c2:0

4.3.1.2.3: DFD de la Prueba.

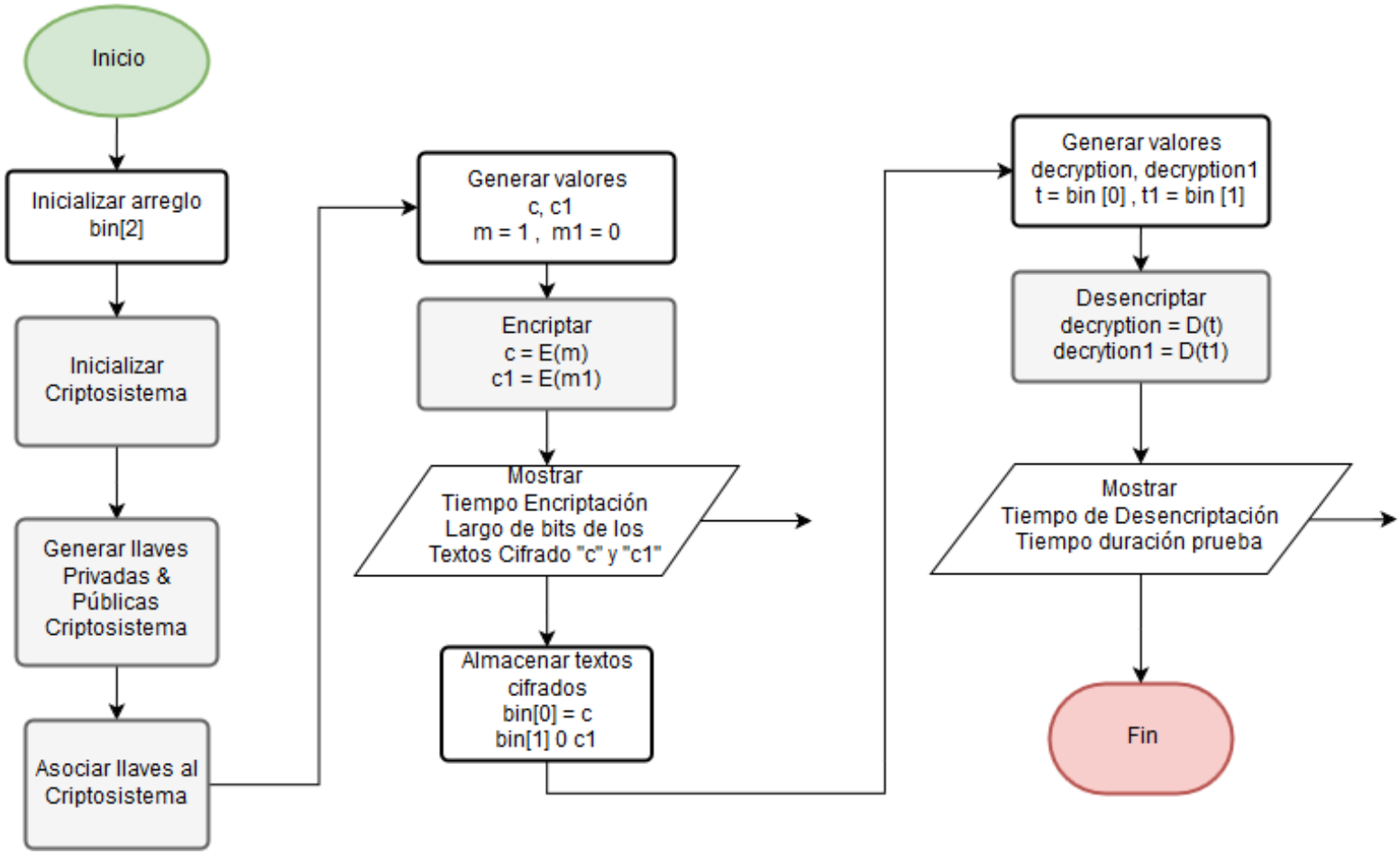


Ilustración 13: *DFD Base de Prueba Encriptación & Desencriptación a dos valores (1 y 0) almacenados en un arreglo, fuente propia.*

4.3.1.3: Pruebas de Homomorfismo & Desencriptación de dos valores:

En esta prueba, se realizará la suma homomórfica de dos BigInteger cuyo valor es 1 tras ser ambos encriptados, con el objetivo de determinar el tamaño del texto resultante.

4.3.1.3.1: Tabla de la prueba

	Paillier	*Boneh-Goh-Nissim*	*Okamoto Uchiyama*	*ElGammal*
Tiempo de encriptación [Milisegundos]	7 ~ 27	-	0 ~ 25	-
Tamaño suma de los dos Textos Cifrados [Función `bitLength()`]	871	-	70	-
Tiempo de la Suma Homomórfica [Milisegundos]	0	-	0	-
Tiempo de Desencriptación [Milisegundos]	3 ~ 11	-	0 ~ 3	-

Tabla 4: *Tabla de prueba de Homomorfismo y Desencriptación de dos valores, fuente propia.*

4.3.1.3.3: DFD de la Prueba.

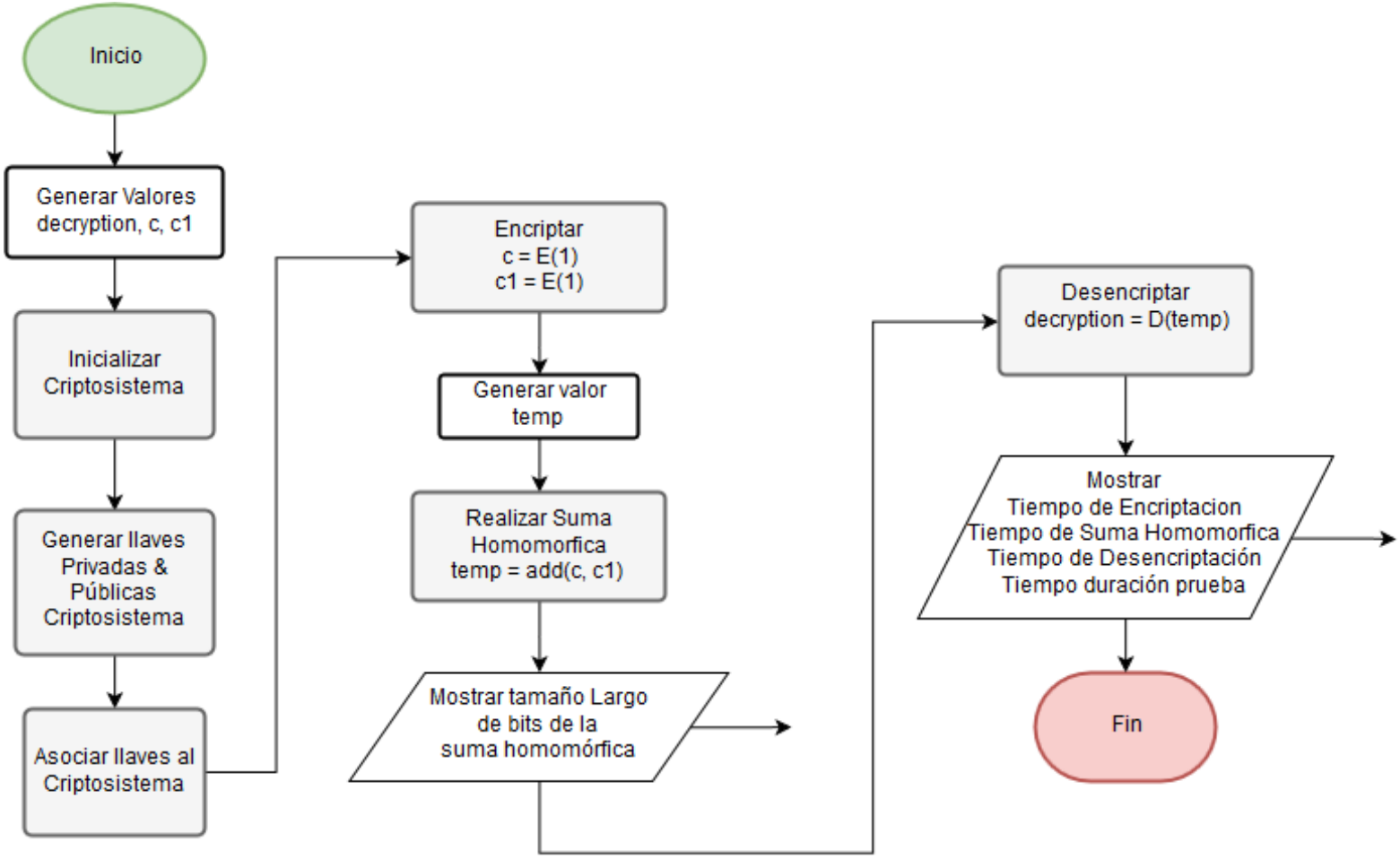


Ilustración 14: *DFD Base de Pruebas Homomorfismo & Desencriptación de dos valores, fuente propia.*

4.3.1.4: Pruebas de Encriptación y Desencriptación de valores almacenados en un vector, contabilizados mediante homomorfismo:

El objetivo de estas pruebas es determinar las variaciones de tiempo que posee la implementación de homomorfismo en la suma de dos valores almacenado en un vector binario cuyos valores son generados al azar en los diferentes criptosistemas a analizarse al momento de realizarse el proceso de encriptación y desencriptación una cantidad "v" de votos a sumar, donde "v" posee los valores: [1, 5, 25, 125, 625, 3125, 15625].

4.3.1.4.1: Gráficas de la Prueba

4.3.1.4.1.1: Gráfica de Media del Tiempo de realización de la Encriptación de la Prueba.

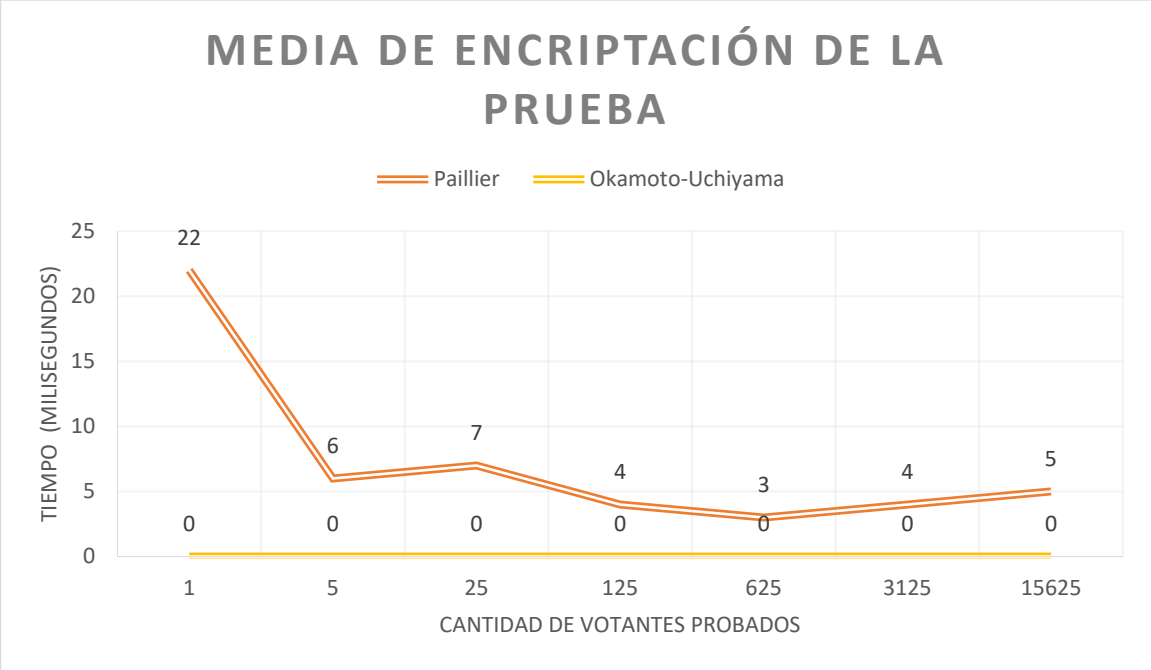


Gráfico 1: *Tiempo de realización de Encriptación de la prueba "Encriptación y Desencriptación de valores almacenados en un vector, contabilizados mediante homomorfismo", fuente propia.*

En este gráfico se visualiza el tiempo medio de la encriptación de los votos durante la realización de la prueba, pudiéndose apreciar que, cuando la cantidad de votos a encriptar aumenta:

- En el caso de Paillier, el tiempo de encriptación tiende a disminuir.
- En el caso de Okamoto-Uchiyama, el tiempo permanece constante en un tiempo de 0 milisegundos.

4.3.1.4.1.2: Gráfica de Media del Tiempo de realización de la Desencriptación de la Prueba.

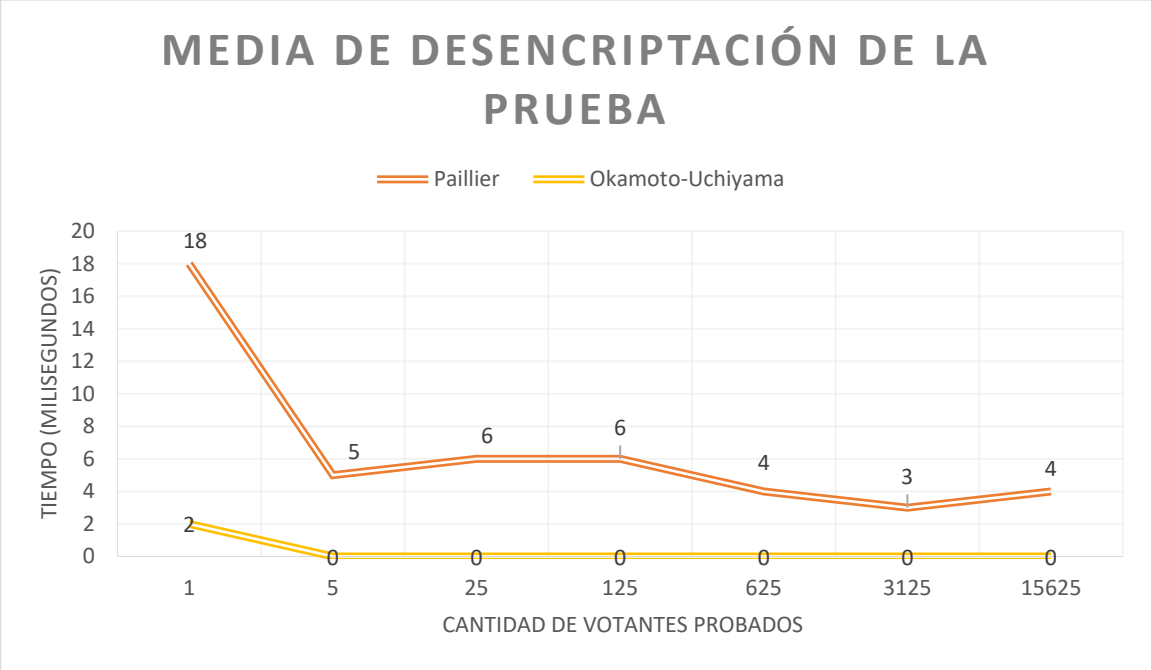


Gráfico 2: *Tiempo de realización de Desencriptación de la prueba "Encriptación y Desencriptación de valores almacenados en un vector, contabilizados mediante homomorfismo", fuente propia.*

En este gráfico se visualiza el tiempo medio de la desencriptación de los votos durante la realización de la prueba, pudiéndose apreciar que, cuando la cantidad de votos a desencriptar, ocurre que:

- En el caso de Paillier, el tiempo de desencriptación tiende a disminuir.
- En el caso de Okamoto-Uchiyama, el tiempo disminuye y permanece constante en un tiempo de 0 milisegundos.

4.3.1.4.1.3: Gráfica de Media del Tiempo de realización de la Suma Homomórfica de la Prueba.

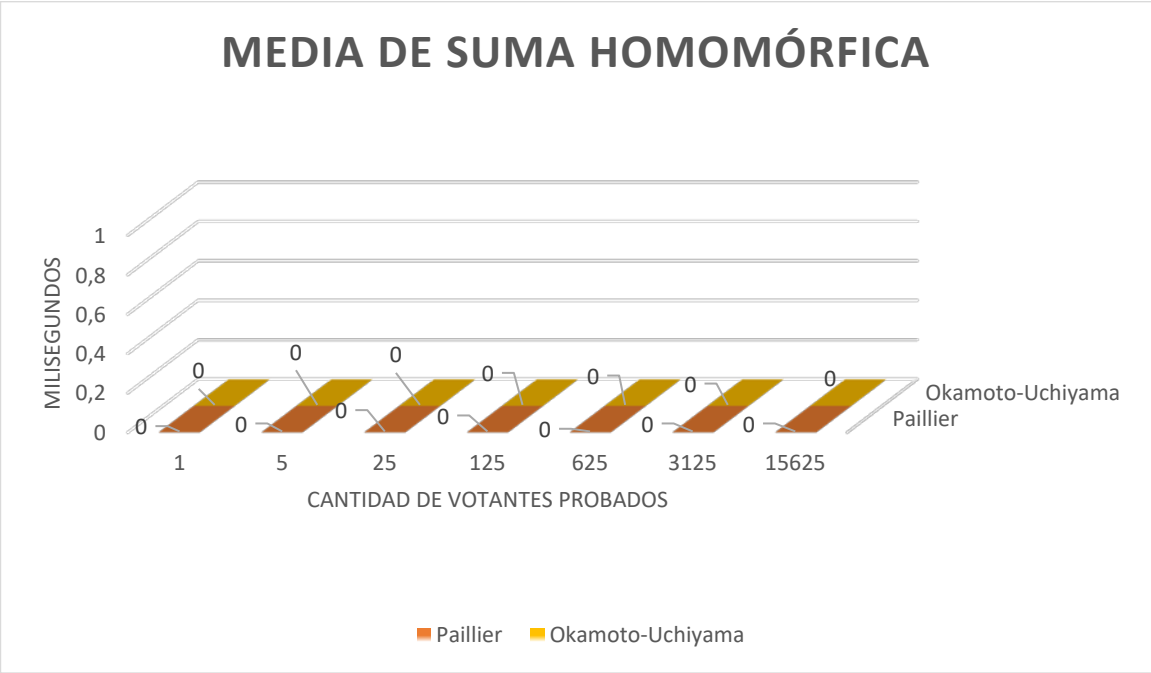


Gráfico 3: *Tiempo de realización de Suma Homomórfica de la prueba "Encriptación y Desencriptación de valores almacenados en un vector, contabilizados mediante homomorfismo", fuente propia.*

En este gráfico se visualiza el tiempo medio de la realización de la Suma Homomórfica de los votos en la realización de la prueba, pudiéndose apreciar que, en ambos casos dicho cálculo posee un tiempo de 0 milisegundos de ejecución.

4.3.1.4.1.4: Gráfica de la Media del Tiempo de Duración de la Prueba.

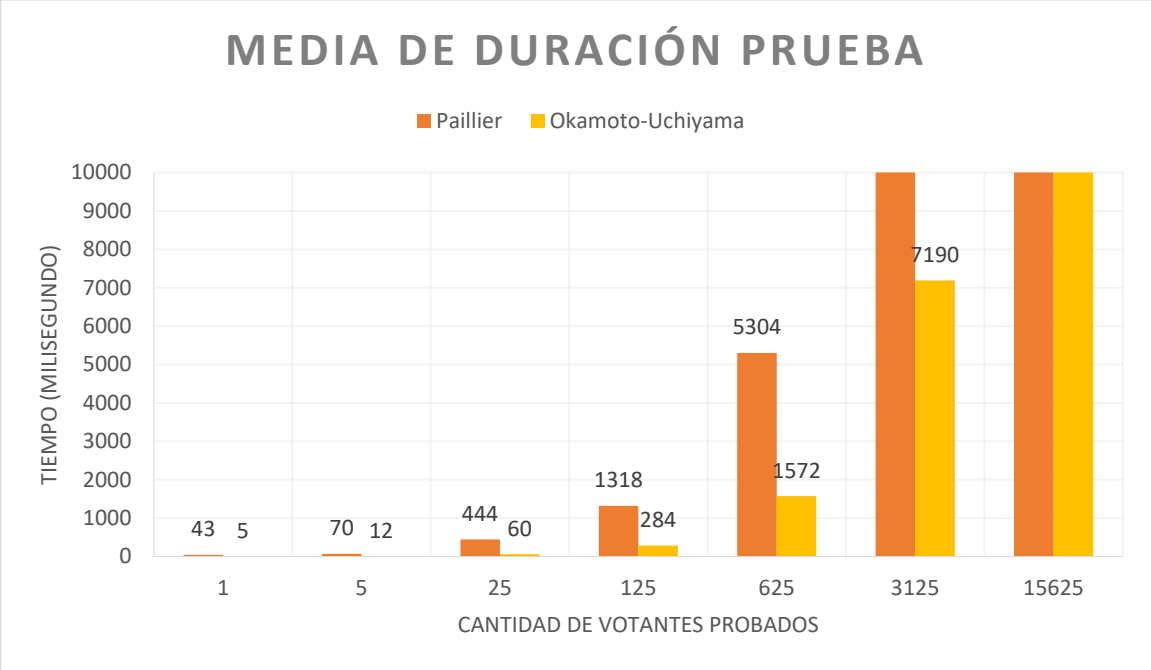


Gráfico 4: *Duración de la Prueba "Encriptación y Desencriptación de valores almacenados en un vector, contabilizados mediante homomorfismo", fuente propia.*

En este gráfico se visualiza el tiempo de ejecución de las pruebas realizadas según la cantidad de votos simulados, donde se puede apreciar que, cuando la cantidad de votos probados incrementa:

- En el caso de Paillier, el tiempo de ejecución de la prueba incrementa considerablemente.
- En el caso de Okamoto-Uchiyama, el tiempo de ejecución de las pruebas, posee un incremento considerablemente menor comparado con la prueba ejecutada con el esquema de Paillier.

Tabla asociada al tiempo en Milisegundos de duración de la prueba

Votos contabilizados	*Paillier*	*Boneh-Goh-Nissim*	*Okamoto Uchiyama*	*ElGammal*
1	43 milisegundos.	-	5 milisegundos.	-
5	70 milisegundos	-	12 milisegundos.	-
25	444 milisegundos	-	60 milisegundos.	-
125	1318 milisegundos	-	284 milisegundos.	-
625	5304 milisegundos	-	1572 milisegundos.	-
3125	33072 milisegundos	-	7190 milisegundos.	-
15625	197991 milisegundos	-	34715 milisegundos.	-

Tabla 5: *Tabla de Prueba de Encriptación y Desencriptación de valores almacenados en un vector, contabilizados mediante homomorfismo, fuente propia.*

4.3.1.4.2: DFD de la Prueba.

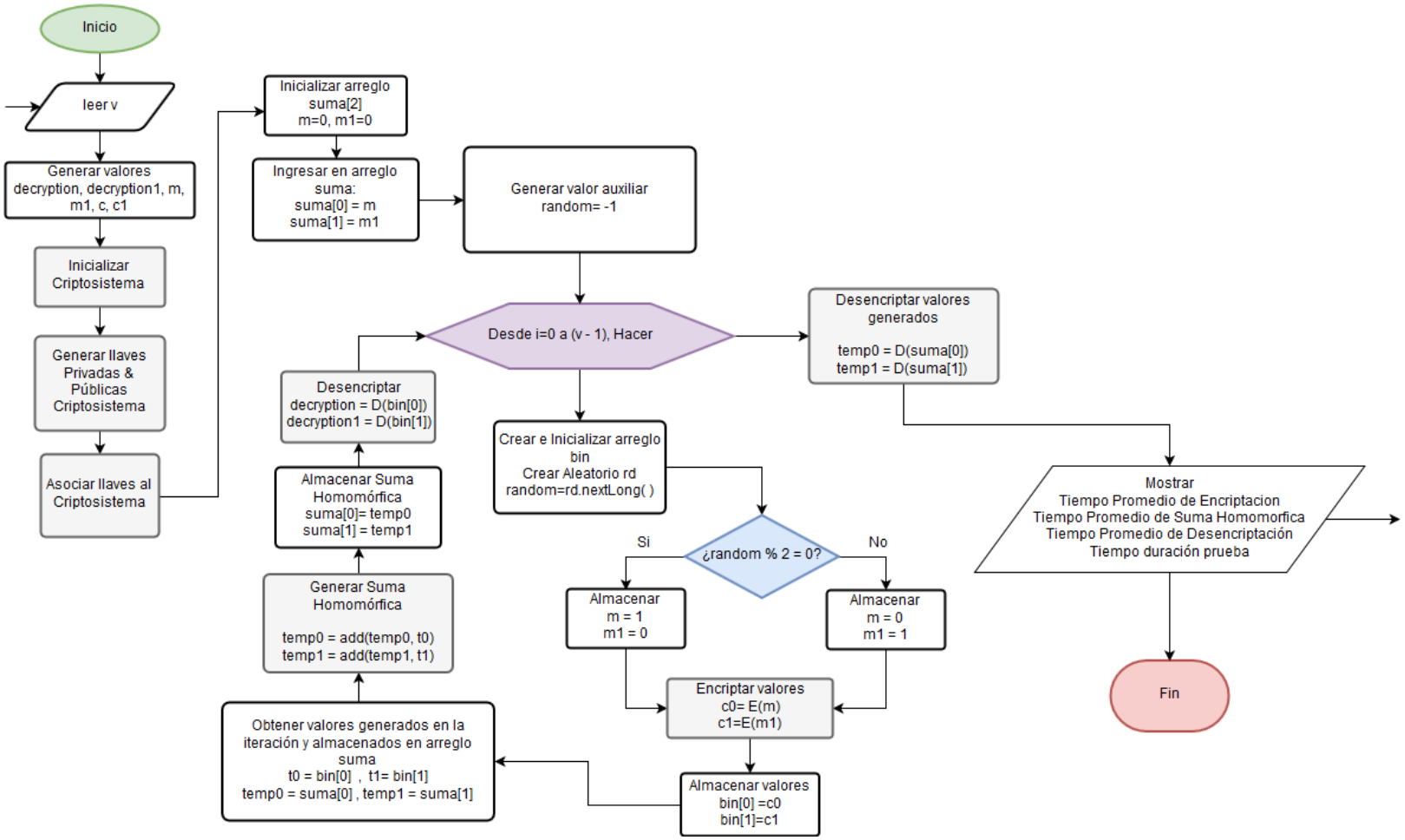


Ilustración 15: *DFD Base Prueba de Encriptación y Desencriptación de valores almacenados en un vector, contabilizados mediante homomorfismo, fuente propia.*

4.3.1.5: Pruebas de Generación de Llaves, Encriptación y Desencriptación del valor 1 ingresando una cantidad de repeticiones de dichos procesos:

El objetivo de estas pruebas es determinar las variaciones que poseen los diferentes criptosistemas a analizarse al momento de realizarse el proceso de generación de llaves, encriptación y desencriptación una cantidad "n" de veces, brindándole a n los valores: [1, 5, 25, 125, 625, 3125, 15625].

4.3.1.5.1: Observación:

En esta prueba, la ejecución del esquema de Boneh-Goh-Nissim se debió de detener, un mayor detalle de los motivos se dicha suspensión se encuentran en el Capítulo "Anexos", sección 9.3.

4.3.1.5.2: Gráficas de la Prueba

4.3.1.5.2.1: Gráfica de Media del Tiempo de realización de la Encriptación de la Prueba.

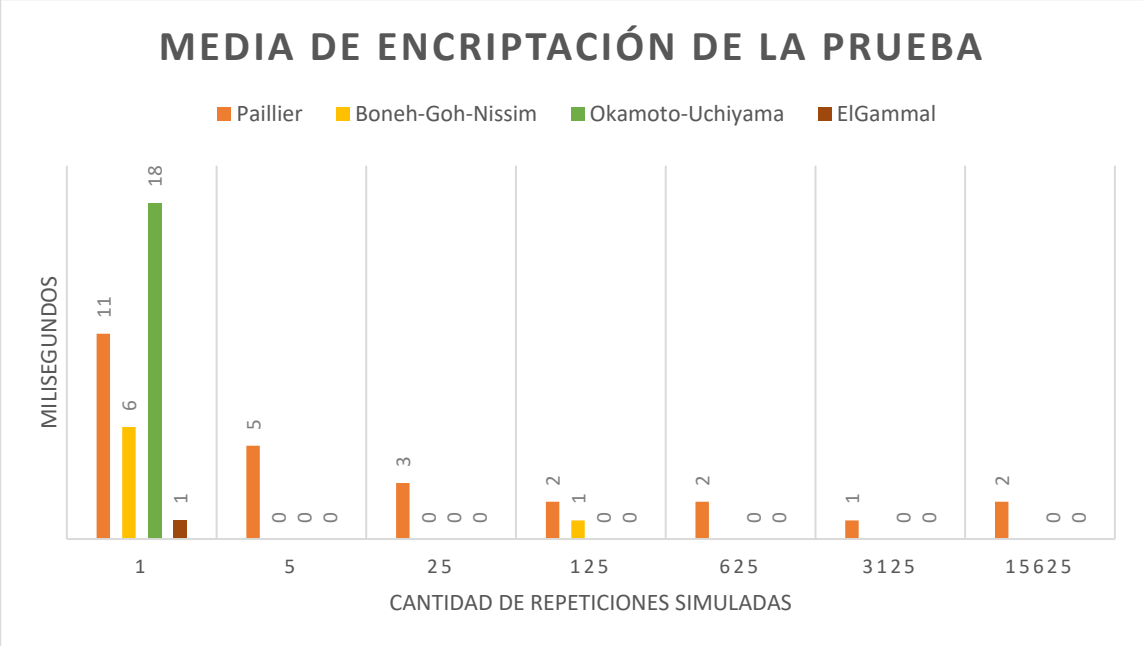


Gráfico 5:*Tiempo de realización de Encriptación de la prueba "Generación de Llaves, Encriptación y Desencriptación del valor 1 ingresando una cantidad de repeticiones de dichos procesos", fuente propia.*

En este gráfico se visualiza el tiempo medio de la encriptación de los votos durante la realización de la prueba, pudiéndose apreciar que, cuando la cantidad de votos a encriptar aumenta:

- En el caso de Paillier, el tiempo de encriptación tiende a disminuir, siendo este el tiempo el esquema con mayor duración media de encriptación en las pruebas donde la cantidad de repeticiones es mayor a 1.
- En el caso de Okamoto-Uchiyama en la simulación de un voto, el tiempo de encriptación es el mayor.
- En el caso de Boneh-Goh-Nissim, su ejecución es relativamente menor, no obstante su prueba no se continuó con los valores de n= 625, 3125 y 15625.
- ElGammal en todas las pruebas quien posee el tiempo de encriptación menor.

4.3.1.5.2.2: Gráfica de Media del Tiempo de realización de la Desencriptación de la Prueba.

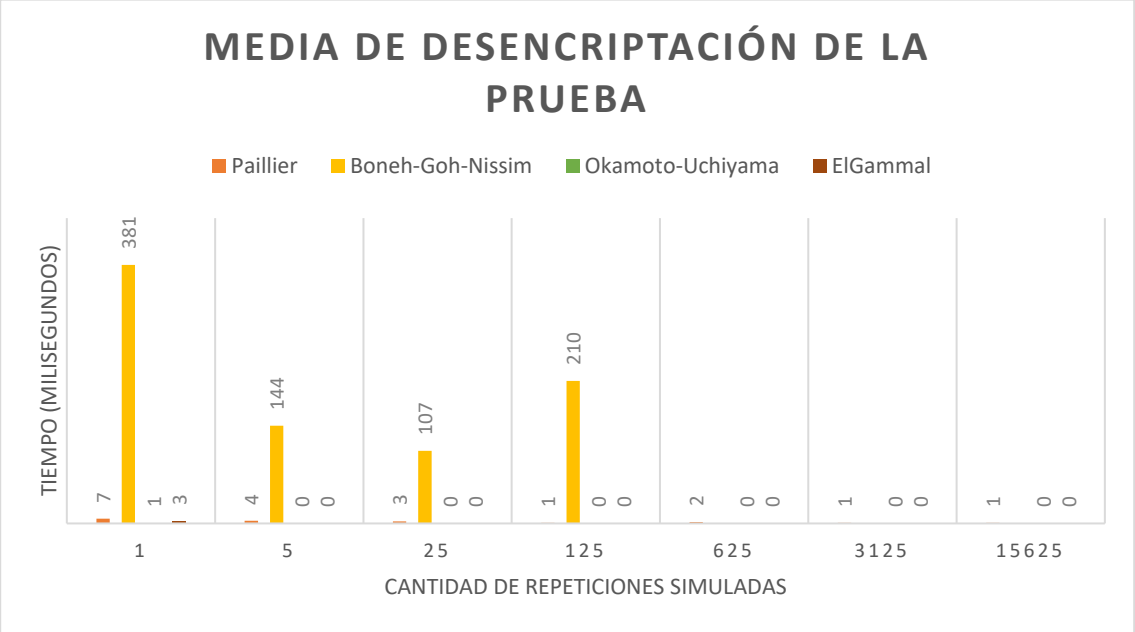


Gráfico 6: *Tiempo de realización de Desencriptación de la prueba "Generación de Llaves, Encriptación y Desencriptación del valor 1 ingresando una cantidad de repeticiones de dichos procesos", fuente propia.*

En este gráfico se visualiza el tiempo medio de la desencriptación de los votos durante la realización de la prueba, pudiéndose apreciar que, cuando la cantidad de votos a encriptar aumenta:

- El esquema Boneh-Goh-Nissim es el esquema que mayor tiempo de realización de la desencriptación requiere, además de que su prueba no se continuó con los valores de n= 625, 3125 y 15625.
- Por otro lado, los otros esquemas presentan un tiempo de desencriptación considerablemente menor al obtenido por Boneh-Goh-Nissim.

4.3.1.5.2.3: Gráfica de la Media del Tiempo de Duración de la Prueba.

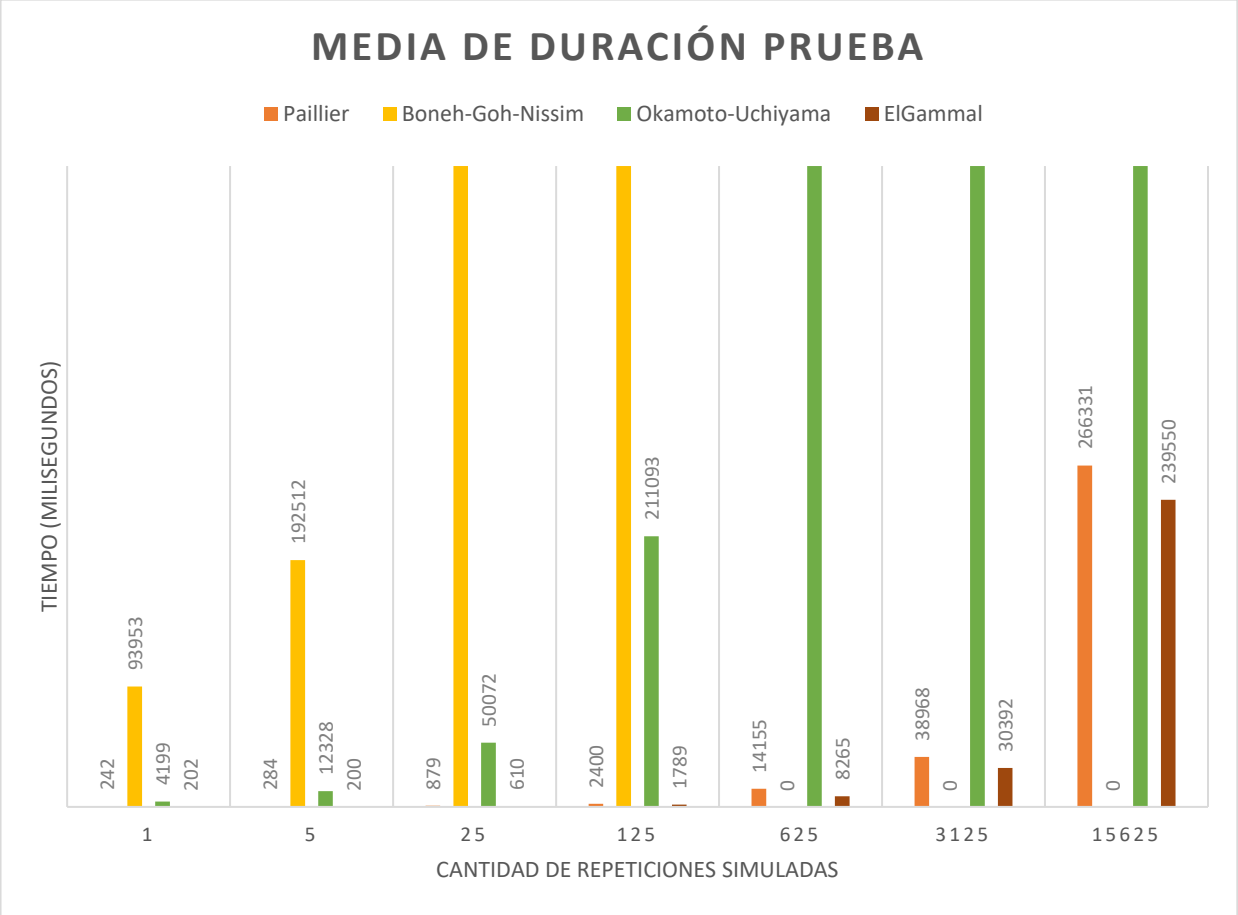


Gráfico 7: *Tiempo de Duración de la prueba "Generación de Llaves, Encriptación y Desencriptación del valor 1 ingresando una cantidad de repeticiones de dichos procesos", fuente propia.*

En este gráfico se visualiza el tiempo de duración total de las pruebas realizadas en esta sección, pudiéndose apreciar que, cuando la cantidad de votos a simular aumenta:

- El esquema Boneh-Goh-Nissim es el esquema que mayor tiempo de ejecución posee, además de que su prueba no se continuó con los valores de n= 625, 3125 y 15625.
- El esquema de Okamoto-Uchiyama presenta el segundo mayor tiempo de ejecución debido al proceso de generación de llaves.
- Por otro lado ElGammal presenta el menor tiempo de ejecución seguido por el esquema de Paillier.

Tabla asociada al tiempo en milisegundos de duración de la prueba

Repeticiones del Proceso Realizadas	*Paillier*	*Boneh-Goh-Nissim*	*Okamoto Uchiyama*	*ElGammal*
1	242 milisegundos.	93.953 milisegundos.	4.199 milisegundos.	202 milisegundos.
5	284 milisegundos.	192.512 milisegundos.	12.328 milisegundos.	200 milisegundos.
25	879 milisegundos.	1.117.895 milisegundos.	50.072 milisegundos.	610 milisegundos.
125	2.400 milisegundos.	14.602.221 milisegundos.	211.093 milisegundos.	1.789 milisegundos.
625	14.155 milisegundos.	**Mayor a 14.602.221** milisegundos.	1.193.211 milisegundos.	8.265 milisegundos.
3125	38.968 milisegundos.	**Mayor a 14.602.221** milisegundos.	6.848.750 milisegundos.	30.392 milisegundos.
15625	266.331 milisegundos.	**Mayor a 14.602.221** milisegundos.	32.968.287 milisegundos.	239.550 milisegundos.

Tabla 6: *Pruebas de Generación de Llaves, Encriptación y Desencriptación del valor 1 ingresando una cantidad de repeticiones de dichos procesos, fuente propia.*

4.3.1.5.3: DFD de la Prueba.

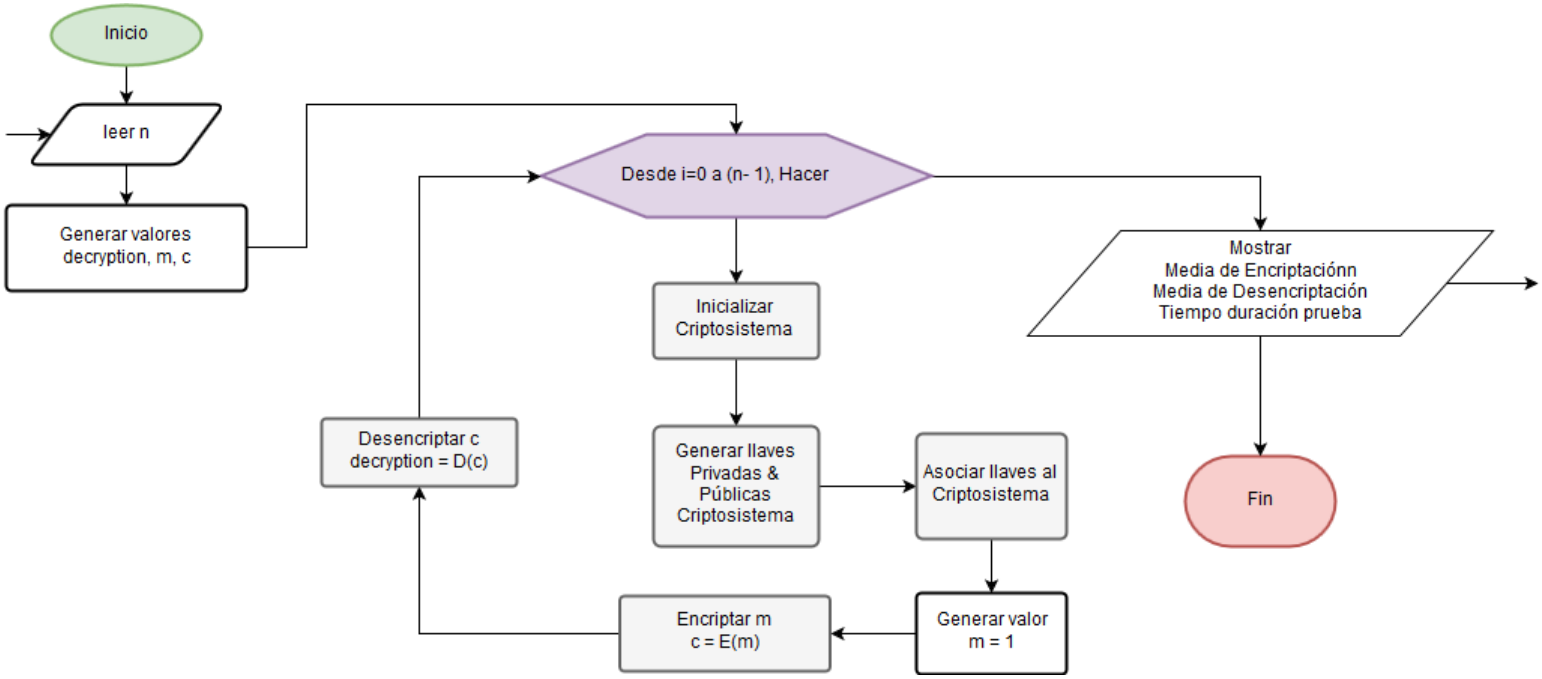


Ilustración 16: *DFD Base Prueba de Encriptación y Desencriptación del valor 1 con diferentes intervalos, fuente propia.*

4.3.1.6: Conclusión de las Pruebas a los Criptosistemas.

El criptosistema de ElGammal es el esquema cuyo tiempo de generación de llaves es menor, seguido de cerca por el esquema de Paillier, mientras que el Esquema de Okamoto-Uchiyama puede tardar más de dos segundos lo cual no deja de ser considerablemente menor al tiempo que el Esquema de Boneh-Goh-Nissim necesita para la realización de las mismas.

Sin embargo, la aplicación del esquema de ElGammal en un ambiente Android, al no poder realizarse la suma homomórfica, implicaría que se deben desencriptar cada uno de los votos y sumarlos uno a uno lo que significa un mayor uso de recursos del sistema.

El esquema Boneh-Goh-Nissim actualmente además de poseer un tiempo de generación de sus llaves considerablemente extenso (lo cual no lo vuelve el criptosistema idóneo para implementarse en un dispositivo móvil ante lo limitado de sus recursos), de depender de una librería extra para su implementación especialmente y la inestabilidad que la generación de sus llaves impiden un correcto funcionamiento del mismo, lo descartan como candidato para la siguiente prueba.

Por otro lado, tanto el esquema de Okamoto-Uchiyama y el esquema de Paillier poseen implementadas la suma homomórfica, siendo el primero el cual demuestra una mayor velocidad al realizarse la suma de elementos encriptados mientras que el segundo es el que posee las llaves más grandes y, por consiguiente, más seguras.

Por ende, se ha determinado que los dos esquemas criptográficos a probar en un ambiente Android serán Paillier y Okamoto-Uchiyama.

4.3.1.6.1: Tabla Comparativa Librerías

	Paillier[3]	*Boneh-Goh-Nissim*	*Okamoto Uchiyama*	*ElGammal*
Valor ingresado variable S	218	24	24	245
Tiempo de Generación de llaves **[Milisegundos]**	328	33170	2996	220
Estabilidad del Criptosistema al ser implementado	Estable	No Estable	Estable	Estable
Tamaño Llaves Privadas Generadas	N: 436 G:436	q1:12 q:25	P: 24 Q: 24	X: 63
Tamaño Llaves Públicas Generadas	D: 435 Inverso de D: 436	N: 23 Q2: 12 g(x: 24 , y: 25) h(x :25 , y: 25)	N: 71 G: 6 H: 70	P: 245 G: 60 Y: 244
Tiempo de Encriptación **[Milisegundos]**	4 ~ 26	8	0 ~ 23	0
Tamaño Texto Cifrado	872	(x: 25 , y: 24)	70	C1: 244 C2: 239
Tiempo de Desencriptación **[Milisegundos]**	4 ~ 10	332	0 ~ 2	0 ~ 16
Tamaño Resultado Suma Homomórfica.	871	-	70	-
Tiempo de Suma Homomórfica **[Milisegundos]**	0	-	0	-

Tabla 7: *Tabla resumen con las cualidades de los sistemas implementados, fuente propia.*

[3] Los valores de las llaves generadas en el caso de la Librería de Paillier corresponden a los valores utilizados por el código los cuales son almacenados en dicha librería.

En esta tabla se pretende entregar la información correspondiente a los valores generados por las mismas librerías, tales como:

- Tiempo en Milisegundos que poseen las librerías para:
    - Generación de llaves del sistema.
    - El proceso de Encriptación.
    - El proceso de Desencriptación.
    - Implementación de Suma Homomórfica.
- Tamaños Llaves Públicas y Privadas Generadas.
- Tamaño del Texto Cifrado generado y de la Suma Homomórfica.

Mientras que *"Valor ingresado variable S"* corresponde a un valor semilla que permite determinar el largo de las llaves, mientras que la *"Estabilidad del Criptosistema al ser implementado"* corresponde a la existencia de anomalías que impidan un resultado fidedigno o posee algún inconveniente al momento de implementarse.

4.3.2: *Pruebas de criptosistemas en Android.*

A continuación, en esta sección se realizará una sencilla aplicación en Android para comprobar la funcionabilidad de dos de los cuatro sistemas implementados en la sección anterior para comprobar el consumo de los recursos en un ambiente más realista, calculando el costo de la encriptación, desencriptación y de la realización de suma homomórfica utilizando un aparato móvil.

Para lo cual, antes de comenzar con la programación de la misma prueba, se ha realizado tanto un diseño lógico implementando diagrama de flujo de la prueba como una previo a la realización de la interfaz.

4.3.2.1: Diagrama de Flujo del proceso de Sufragio.

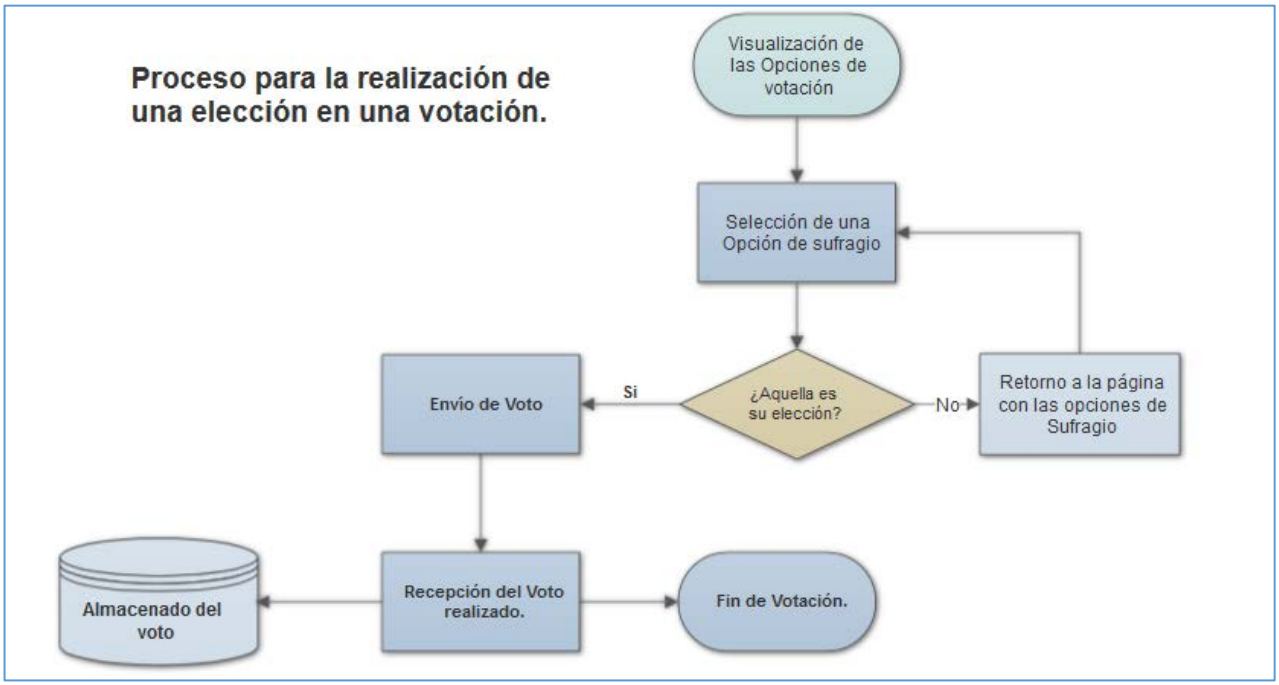


Ilustración 17: *Diagrama de Flujo proceso de Sufragio, fuente propia.*

La prueba consiste en la visualización de dos opciones, en la cual el usuario deberá elegir una de ellas, opción que deberá de confirmar dicha elección: Al confirmarse la elección internamente el sistema realizará la encriptación de la opción elegida:

- De elegirse la Opción 1 en un arreglo se almacenarán los valores [1][0].
- De elegirse la Opción 2 en un arreglo se almacenarán los valores [0][1].

Estos valores, en el sistema, serán almacenados terminando con el proceso de sufragio del usuario.

4.3.2.2: Diseño de la Interfaz de Prueba.

Antes de comenzar con la implementación, utilizando la página web https://ninjamock.com/, se procuró realizar un diseño simple de la interfaz a realizar a la prueba.

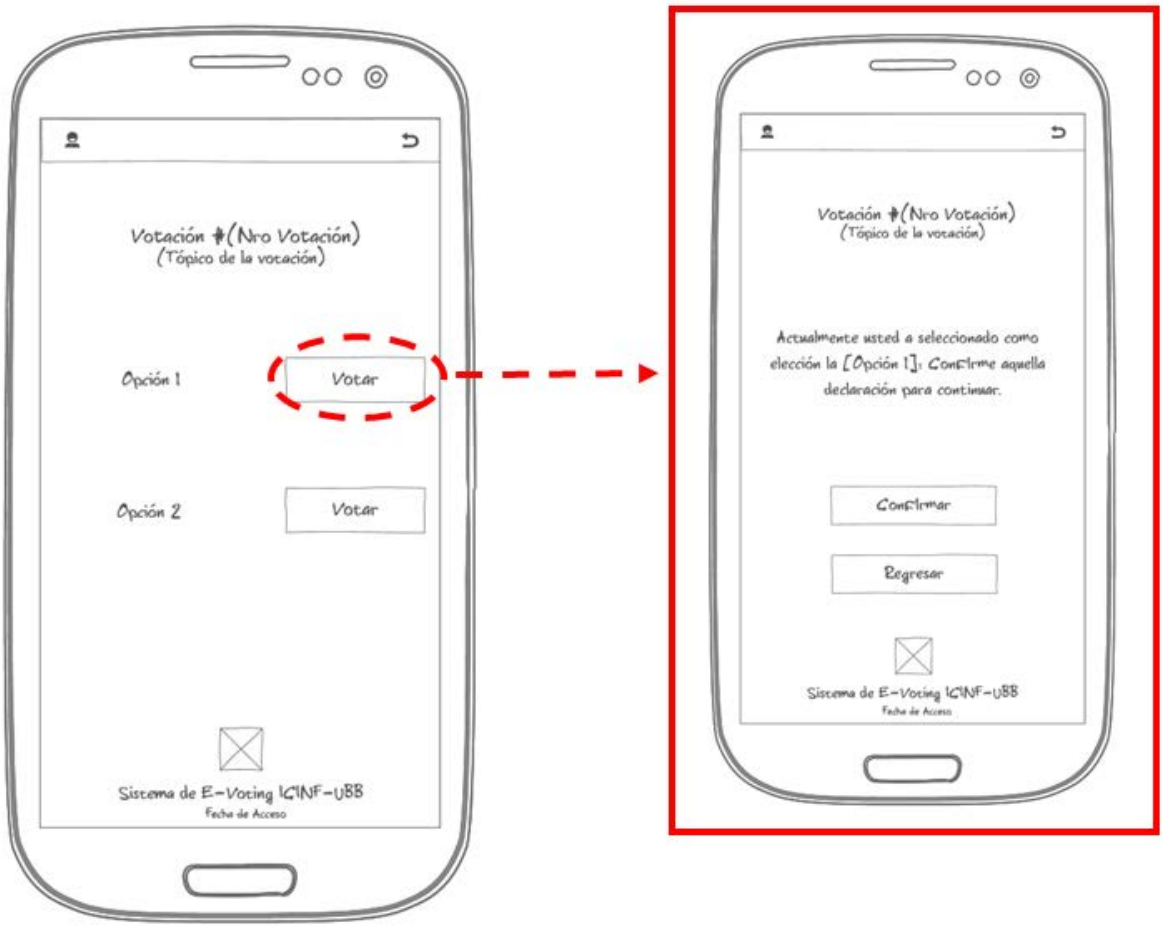


Ilustración 18: *Selección y Confirmación del Sufragio, fuente propia.*

Para la prueba se implementa una ventana sencilla, en donde se mostrará una votación de dos valores, para lo cual se espera que el usuario seleccione una de las opciones entregadas para luego acceder a una ventana de confirmación de su elección:

- De no encontrarse el usuario conforme con su elección se puede presionar el botón “Regresar” para retornar a la pantalla anterior.
- De confirmar la opción, el voto será Encriptado y “Enviado”.

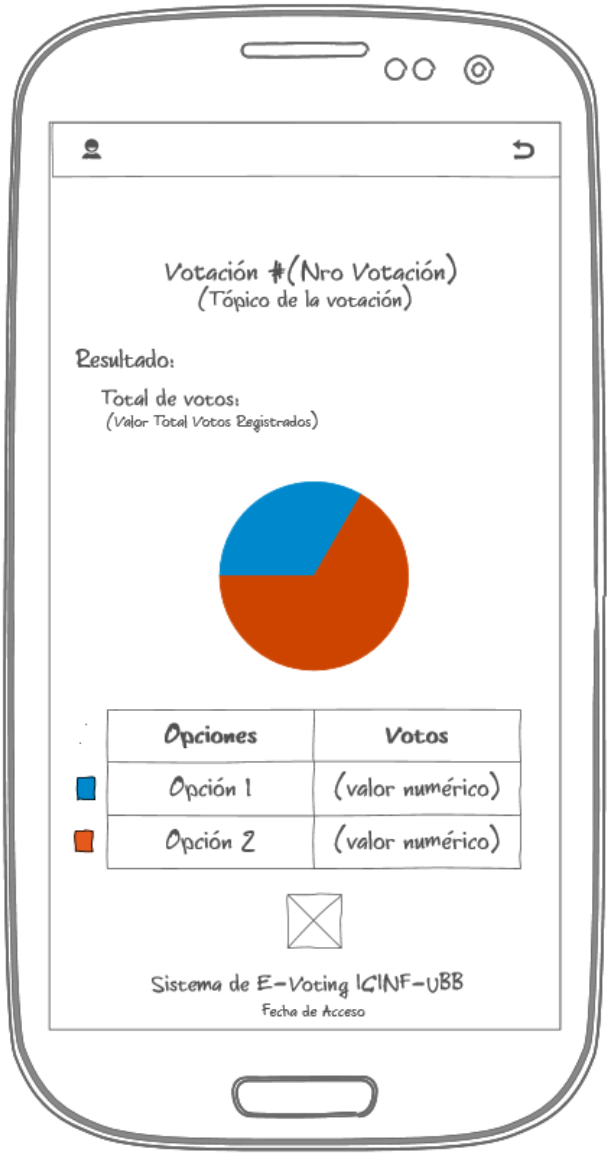


Ilustración 19: *Resultados Votación, fuente propia.*

En esta prueba, en el contexto de la finalización del proceso de sufragio, se espera realiza una visualización de votos con el objetivo de calcular el uso de los recursos en un sistema Android en la suma homomórfica.

Para esta prueba se generarán valores aleatorios para determinar el consumo de recursos del dispositivo con sistema operativo Android de prueba.

Considerando que es importante determinar del uso de los recursos tanto en el momento de encriptar un voto generado por un usuario como la capacidad de un sistema Android para realizar la suma homomórfica y la desencriptaciones de los valores de una votación se ha decidido seguir la siguiente lógica para la implementación de la prueba:

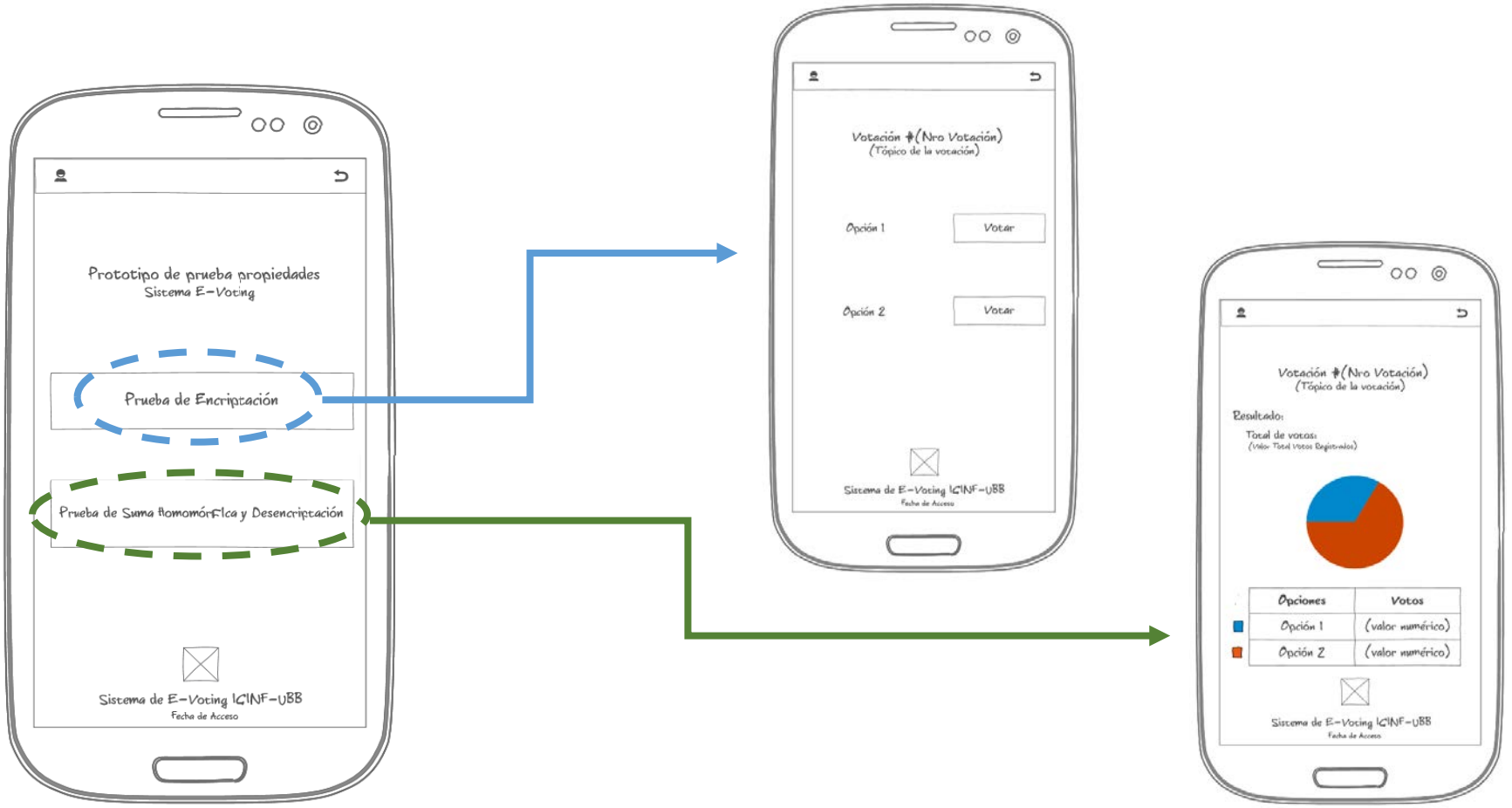


Ilustración 20: *Interfaz de Prueba, fuente propia.*

4.3.2.3: Características del Dispositivo de Prueba.

Para la realización de estas pruebas se usó un dispositivo Samsung Galaxy J7 Neo, Número de Modelo SM-J701 M, por lo que a continuación se adjuntará información técnica del mismo relativa a la prueba.

- La **Información de Hardware** se obtuvo de la página web oficial en español.
- La **Información del Software** proveniente del mismo dispositivo móvil accesible desde el menú de *"Ajustes"* ingresando en el menú *"Acerca del teléfono"* y seleccionado la opción "*Información de Software*".

Información de Hardware[4]	**Batería**	
	Tiempo de uso de Internet (3G) (Horas)	Hasta 11
	Tiempo de uso de Internet (LTE) (Horas)	Hasta 13
	Tiempo de uso de Internet (Wi-Fi) (Horas)	Hasta 13
	Tiempo de reproducción de video (Horas)	Hasta 17
	Capacidad de batería (mAh)	3000
	Removible	Si
	Tiempo de Reproducción de Audio (Horas)	Hasta 80
	Tiempo de conversación (3G WCDMA) (Horas)	Hasta 21
	Memoria	
	RAM (GB)	2
	Memoria Total (ROM en GB)	16 GB
	*Memoria Disponible (GB)**	9.9 GB
	Memoria Externa	MicroSD (Hasta 256GB)
	Procesador	
	Velocidad de CPU	1.6GHz
	Tipo de CPU	Octa-Core

Tabla 8: *Tabla con la Información del Hardware del dispositivo de Prueba, fuente: https://www.samsung.com/ar.*

[4] En la especificación de la información de Hardware se consideraron los elementos de la ficha técnica que serán afectados en la realización de la prueba.

INFORMACIÓN DEL SOFTWARE	
VERSIÓN DE ANDROID	7.0
VERSIÓN DE SAMSUNG EXPERIENCIE	8.1
VERSIÓN DE BANDA BASE	J701MUBU3ARD1
VERSIÓN DE KERNEL	3.18.14-13408364 dpi@SWDG9701 #1 Wed Apr 11 23:07:39 KST 2018
NÚMERO DE COMPILACIÓN	NRD90M.J707MUBU3ARD1
ESTADO DE SE PARA ANDROID	Enforcing SEPF_SECMOBILE_7.0_0010 Wed Apr 23:24:47 2018
VERSIÓN DE KNOX	Knox 2.8 Standard SDK 5.8.0 Premium SDK 2.8.0 Customizator SDK 2.8.0 - CEEP 2.2.0 Enterprise Billing 1.3.0 OPT 2.6.0 SE for Android 2.4.3 Shared Device 2.8.0 TIMA 3.3.015 - VPN 2.4.0
VERSIÓN DE SEGURIDAD	ASKS v1.4 Release 180122 SMR Apr-2018 Release 1
REVISIÓN DE SEGURIDAD ANDROID	1 de abril de 2018

Tabla 9: *Tabla con la Información del Software del dispositivo de Prueba, fuente dispositivo móvil a utilizar.*

4.3.2.4: Implementación del Prototipo y Ejecución de Pruebas.

En la siguiente sección, previo a la realización de las pruebas se detallará el diseño del Prototipo desarrollado para la prueba de la funcionalidad de los algoritmos seleccionados considerando elementos tales como la Navegación del Sistema, el Diagrama de Clases, el Código desarrollado del Prototipo e Imágenes asociadas al mismo, para luego visualizar los resultados obtenidos en las pruebas de desempeño en los procesos de “Encriptación” y “Suma Homomórfica & Desencriptación” de dichos sistemas.

4.3.2.4.1: Diseño del Prototipo para la prueba

4.3.2.4.1.1: Navegación del Sistema

El prototipo Android inicia su ejecución en la página principal, como se ha visualizado en la sección de "Diseño de la Interfaz de la prueba", existen dos botones:

- El primero de ellos permitirá el acceso a una simulación de un voto (Página de Elección Voto), de una votación de prueba con dos opciones "Sí" o "No", opciones que permitirán acceso a la sección de Confirmación del voto (Página de confirmación Voto), en donde:
  - De NO desear Confirmar el voto el usuario será enviado a la sección anterior.
  - De Confirmar el voto, este será encriptado y el usuario será regresado al menú principal.
- Por otra parte, la segunda opción como a la simulación de un proceso de votación con una cantidad determinada de votantes generando votos aleatorios que serán sumados para luego ser desencriptados, y posteriormente se dará paso a la sección habilitada para la visualización del resultado de la votación (Página de Prueba de Suma Homomórfica & Desencriptación).

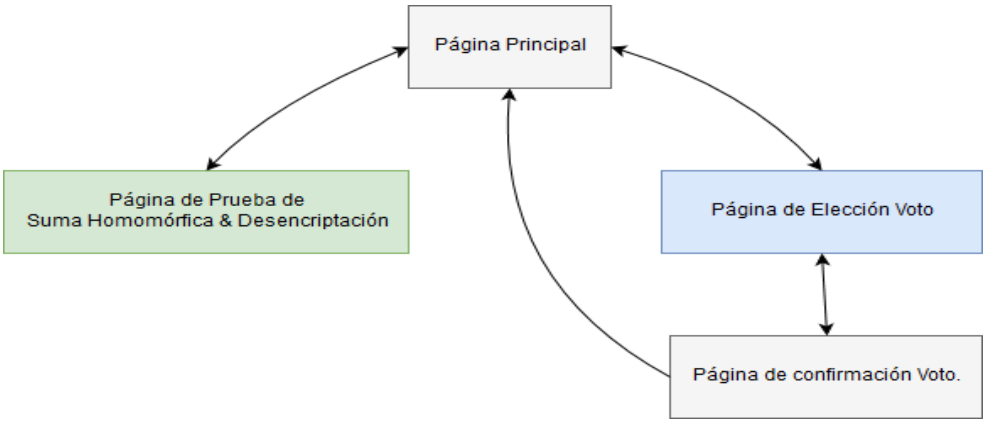


Ilustración 21: *Navegación del Sistema del Prototipo, fuente propia.*

4.3.2.4.1.2: Diagrama de Clases

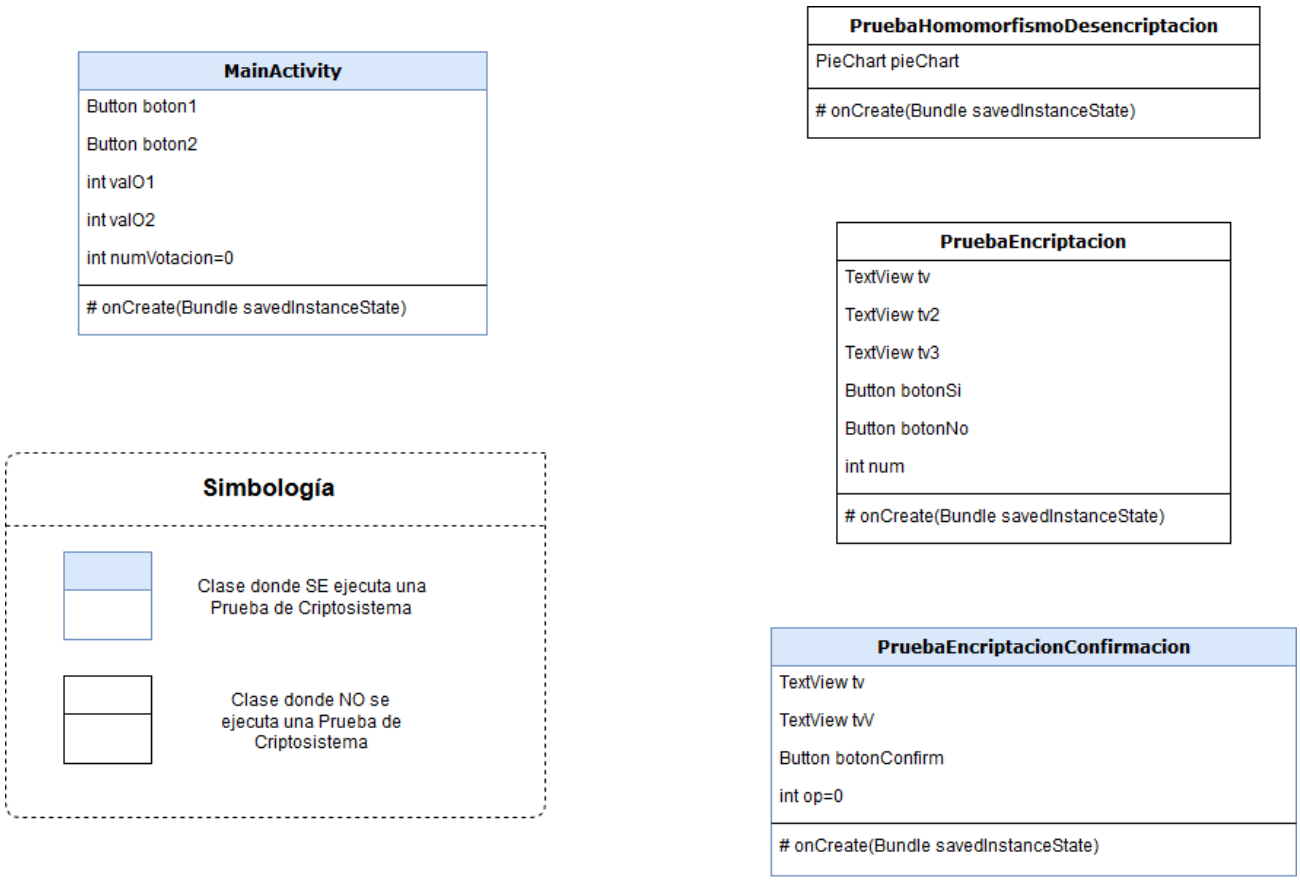


Ilustración 22: *Diagrama de Clases del Prototipo desarrollado, fuente propia.*

([5])

[5] En este diagrama se excluye los métodos asociados a las pruebas de los Criptosistemas elegidos.

4.3.2.4.1.3: Código Prototipo

En el desarrollo de una aplicación para Sistemas Android, al momento de realizar una ventana en el programa “Android Studio” se deben de programar dos archivos:

- Uno donde se implementa la funcionalidad de la ventana creada (.java)
- Y otro archivo con la visualización de la ventana creada (.xml)

En la realización de este prototipo se realizaron cuatro ventanas, las cuales serán descritas a continuación.

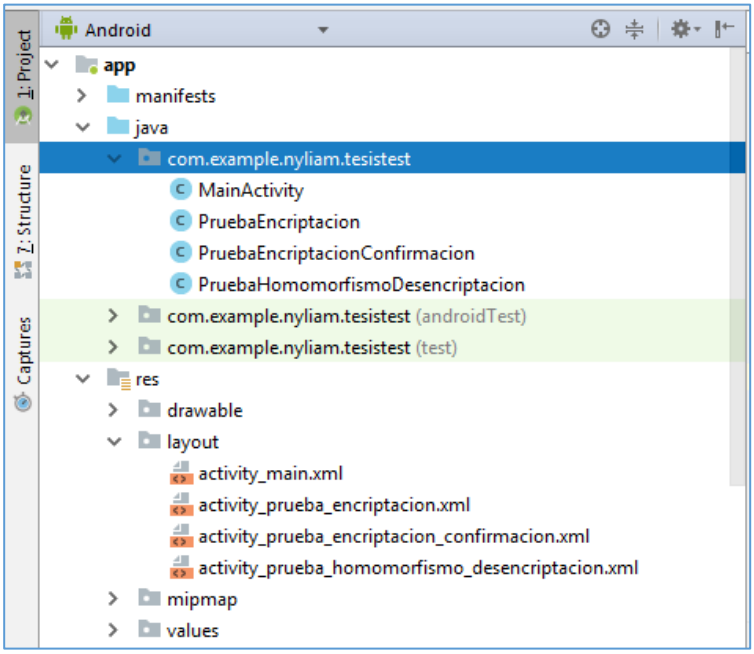


Ilustración 23: *Orden de Archivos en una aplicación Android, fuente propia.*

El detalle de esta sección se encontrará registrada en el capítulo de Anexos.

4.3.2.4.1.4: Imágenes Prototipo

En esta sección se procederá a adjuntar cuatro imágenes del Prototipo desarrollado en su aplicación móvil, imágenes tomadas directamente del dispositivo con sistema Android de prueba descrito con anterioridad.

Menú Inicial

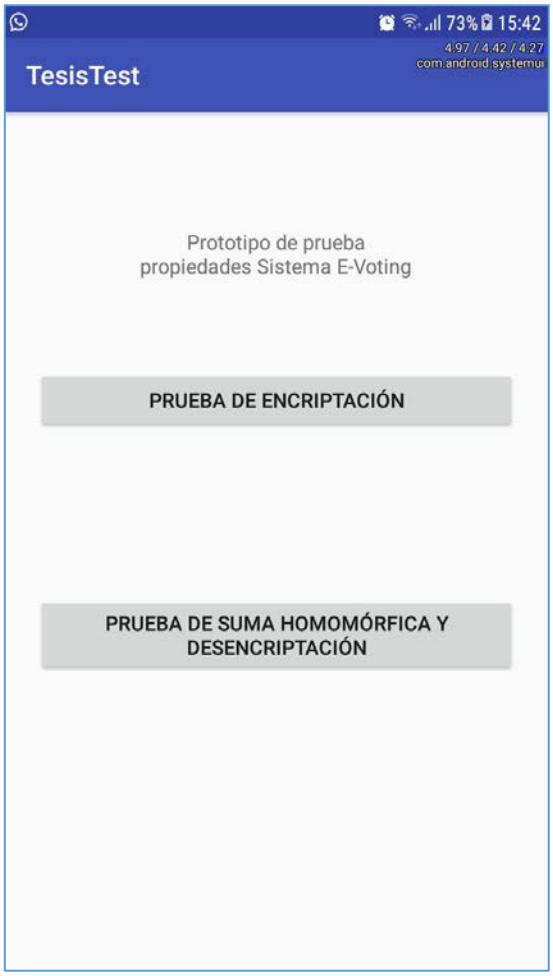


Ilustración 24: *Captura de Pantalla de la Ventana Inicial de la Aplicación, fuente propia.*

Ventana visible al seleccionar el botón "Prueba de Suma Homomórfica y Desencriptación"

Ilustración 25: *Captura de Pantalla de la Ventana con los Resultados de la Simulación de Votación realizada en la ventana principal, fuente propia.*

Ventana visible al seleccionar el botón "Prueba de Encriptación"

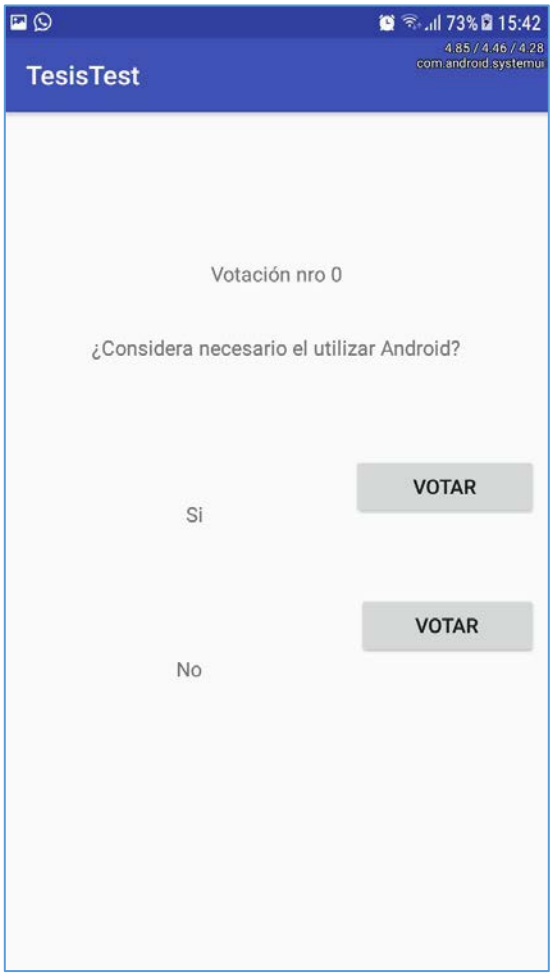


Ilustración 26: *Captura de Pantalla de la Ventana de Simulación de Voto, fuente propia.*

Ventana de confirmación de selección de preferencia.

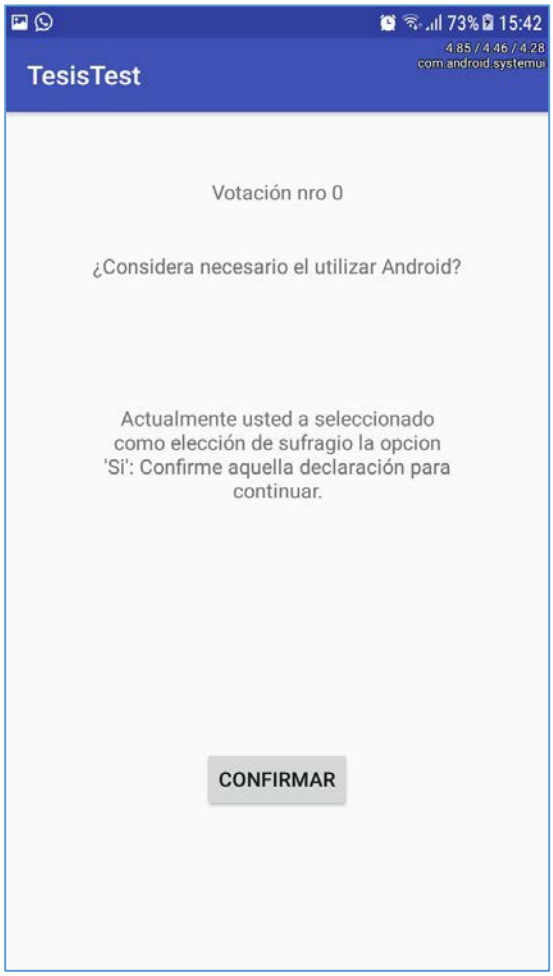


Ilustración 27: *Captura de Pantalla de la Ventana de Confirmación Elección, fuente propia.*

4.3.2.4.1.5: Funciones Utilizadas por ambas pruebas

4.3.2.4.1.5.1: Función para Generar el Reporte

Función que, tras ingresarle un nombre de archivo "`sFileName`" con formato "(nombre).txt" y un cuerpo del reporte "`sBody`" permite que, si no existe se genere una carpeta "`TestFolder`" en el dispositivo Android, en donde se almacenará el archivo generado de reporte.

```
public void generarReporte(String sFileName, String sBody){
        try{
                String path = Environment.getExternalStorageDirectory() +
                        File.separator  + "TestFolder";
              // Create the folder.
               File folder = new File(path);
               folder.mkdirs();
               File file = new File(folder,sFileName);
               OutputStreamWriter outputStreamWriter =
                         new OutputStreamWriter(new FileOutputStream(file));
                    outputStreamWriter.write(sBody);
                    outputStreamWriter.close();
        }
        catch (IOException e) {
                Log.e("Exception", "File write failed: " + e.toString());
        }
}
```

4.3.2.4.1.5.2: Función para Calcular el Uso de la RAM

Función que permite obtener el espacio libre de RAM en el dispositivo Android, para calcular el uso de RAM de las pruebas, se calcula la RAM libre antes de iniciar el procedimiento del sistema de encriptación y la RAM libre luego de finalizar el procedimiento del criptosistema.

Por tanto, se considera que la RAM utilizada por la prueba como:

$$RAM_{Usada} = RAM\_Libre_{final} - RAM\_Libre_{inicial}$$

```
public Integer getFreeRAM() {
        ActivityManager.MemoryInfo mi = new ActivityManager.MemoryInfo();
        ActivityManager activityManager =
                (ActivityManager) getSystemService(ACTIVITY_SERVICE);
        activityManager.getMemoryInfo(mi);
        Integer mem = (int) (mi.availMem / 1048576L);
        return mem;
}
```

4.3.2.4.1.5.3: Función para Calcular el Uso de CPU

Función que permite obtener el uso de la CPU en el dispositivo Android, para calcular el uso de RAM de las pruebas, se calcula el uso de la CPU antes de iniciar el procedimiento del sistema de encriptación y el uso de la CPU luego de finalizar el procedimiento del criptosistema.

```
public float getCPU_Usage() {
    try {
        RandomAccessFile reader = new RandomAccessFile("/proc/stat", "r");
        String load = reader.readLine();
        String[] toks = load.split(" +");  // Split on one or more spaces
        long idle1 = Long.parseLong(toks[4]);

        long cpu1 = Long.parseLong(toks[2]) + Long.parseLong(toks[3]) +
        Long.parseLong(toks[5])+ Long.parseLong(toks[6]) +
        Long.parseLong(toks[7]) + Long.parseLong(toks[8]);

        try {
            Thread.sleep(360);
        } catch (Exception e) {}
        reader.seek(0);
        load = reader.readLine();
        reader.close();
        toks = load.split(" +");
        long idle2 = Long.parseLong(toks[4]);
        long cpu2 = Long.parseLong(toks[2]) + Long.parseLong(toks[3]) +
Long.parseLong(toks[5])
                + Long.parseLong(toks[6]) + Long.parseLong(toks[7]) +
Long.parseLong(toks[8]);
        return (float)(cpu2 - cpu1) / ((cpu2 + idle2) - (cpu1 + idle1));
    } catch (IOException ex) {
        ex.printStackTrace();
    }
    return 0;
}
```

4.3.2.4.1.5.4: Función para Calcular el Consumo de Energía

Función que permite calcular el porcentaje de batería actual del dispositivo móvil se calcula la Batería del Sistema antes de iniciar el procedimiento del sistema de encriptación y la Batería del Sistema luego de finalizar el procedimiento del criptosistema.

```
public int getBatteryPercentage(Context context) {
      IntentFilter iFilter = new
      IntentFilter(Intent.ACTION_BATTERY_CHANGED);
      Intent batteryStatus = context.registerReceiver(null, iFilter);
      int level = batteryStatus != null ?
      batteryStatus.getIntExtra(BatteryManager.EXTRA_LEVEL, -1) : -1;
      int scale = batteryStatus != null ?
      batteryStatus.getIntExtra(BatteryManager.EXTRA_SCALE, -1) : -1;
      float batteryPct = level / (float) scale;
      return (int) (batteryPct * 100);
}
```

4.3.2.4.2: Implementación Prueba de Encriptación

4.3.2.4.2.1: DFD de la Prueba

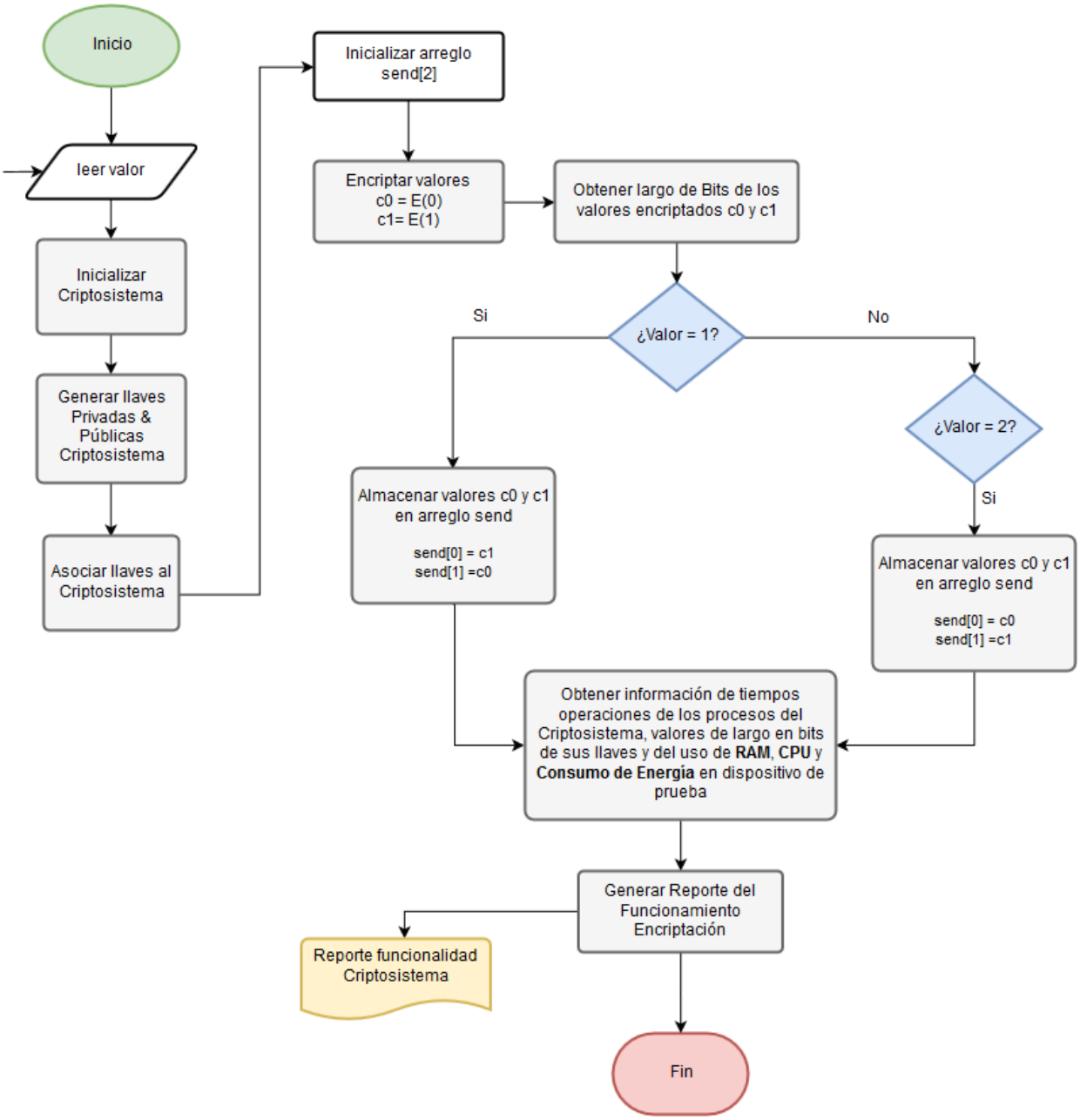


Ilustración 28: *DFD prueba Encriptación – Android, fuente propia.*

4.3.2.4.2.2: Código utilizado por Ambos Esquemas en esta Prueba

4.3.2.4.2.2.1: Función para Generar Cuerpo del Reporte

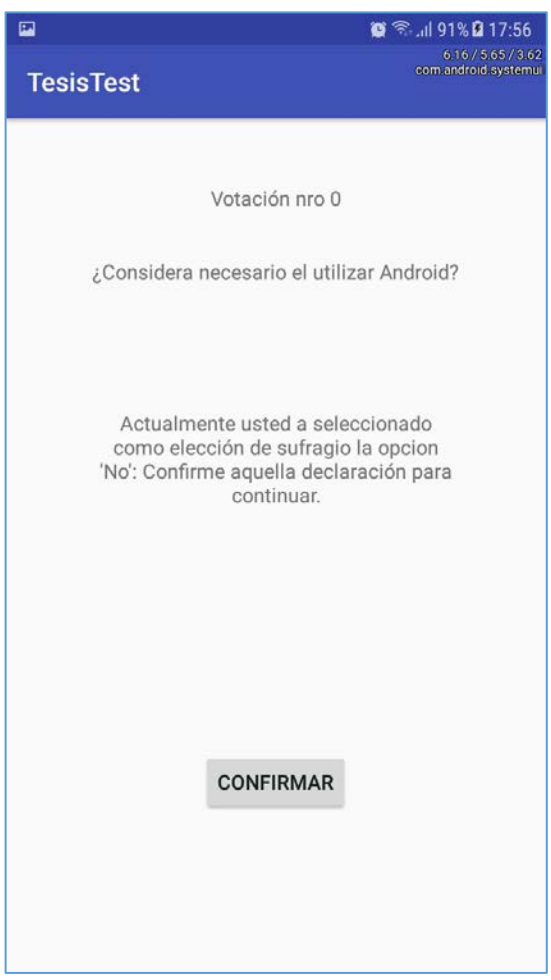
```
public void generarReporte(long tiempoKG, ArrayList<String> PrivateKey,
ArrayList<String> PublicKey, long tEncry, int c1BitL, int c0BitL,
long tiempoTest, int usoRAM, int usoBateria, float usoCPU,
String nombreArchivo){

    String llavesPrivadas ="Tamaño llaves privadas:\t";
    for(int i=0; i<PrivateKey.size();i++) {
        if (i == PrivateKey.size() - 1) {
            llavesPrivadas = llavesPrivadas.concat(PrivateKey.get(i));
        } else {
            llavesPrivadas = llavesPrivadas.concat(PrivateKey.get(i)+" - ");
        }
    }
    String llavesPublicas="Tamaño llaves publicas:\t";
    for(int i=0; i<PublicKey.size();i++) {
        if(i==PublicKey.size()-1) {
            llavesPublicas = llavesPublicas.concat(PublicKey.get(i));
        }else{
            llavesPublicas = llavesPublicas.concat(PublicKey.get(i)+" - ");
        }
    }
    String pendiente= "-";
    String cuerpo;
    cuerpo = "Tiempo Generacion Llaves:\t" + tiempoKG +"\n"
        +llavesPrivadas +"\n"
        +llavesPublicas+"\n"
        +"Tiempo Encriptacion:\t"+tEncry+"\n"
        +"Tamaño textos cifrados:\t C0:"+ c0BitL + " - C1:"+ c1BitL+"\n"
        +"Tiempo total de la prueba:\t"+tiempoTest+"\n"
        +"Uso de la RAM:\t"+usoRAM+"MB"+"\n"
        +"Consumo Bateria:\t"+usoBateria+""+"\n"
        +"Uso de CPU:\t"+usoCPU+""+"\n"
    ;
      Toast.makeText(PruebaEncriptacionConfirmacion.this, cuerpo,
      Toast.LENGTH_LONG).show();
    generarReporte( nombreArchivo, cuerpo);

}
```

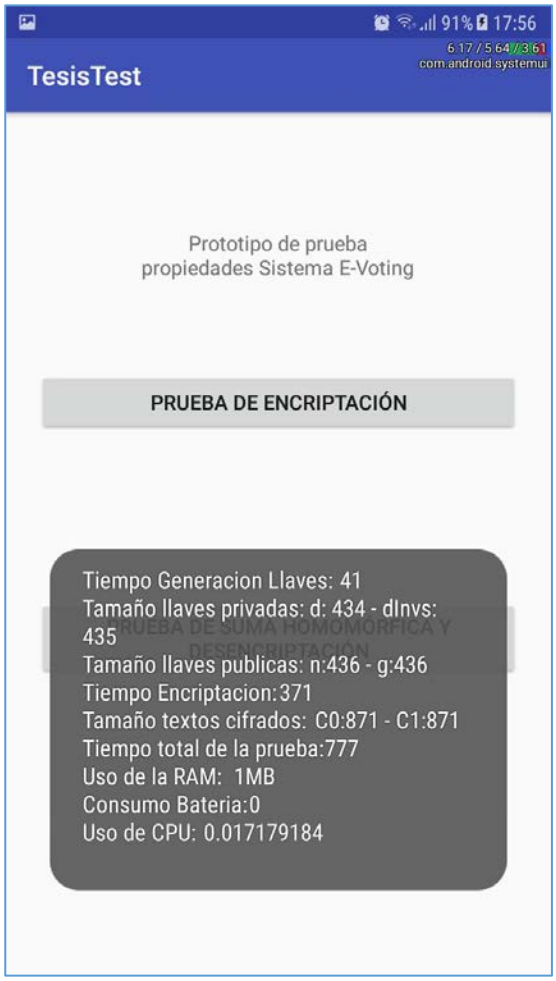
4.3.2.4.2.3: Paillier

4.3.2.4.2.3.1: *Código de la Prueba*

```
public void Encrypt_Paillier(int valor){
    Integer megaStart = getFreeRAM();
    int batteryStart
=getBatteryPercentage(PruebaEncriptacionConfirmacion.this);
    long startT = System.currentTimeMillis();
    float cpuStart =getCPU_Usage();
    Paillier esystem= new Paillier();
    long start = System.currentTimeMillis();
    PaillierPrivateKey key= KeyGen.PaillierKey(218,122333356);
    long stop = System.currentTimeMillis();
    long tkeyGen= stop - start;
    esystem.setDecryptEncrypt(key);
    ArrayList<BigInteger> send= new ArrayList<BigInteger>();
    start=System.currentTimeMillis();
    BigInteger c1=  esystem.encrypt(BigInteger.ONE) , c0=
esystem.encrypt(BigInteger.ZERO);
    float cpuStop =getCPU_Usage();
    stop = System.currentTimeMillis();
    int bitC1 = c1.bitLength(), bitC0= c0.bitLength();
    long tEncry = stop - start;
    if(valor==1){
        //Almacenar en orden [1][0]
        send.add(c1);
        send.add(c0);

    }else if(valor==2){
        //Almacenar en orden [0][1]
        send.add( c0);
        send.add( c1);
    }
    Integer megaFinish = getFreeRAM();
    int batteryStop
=getBatteryPercentage(PruebaEncriptacionConfirmacion.this);
    long stopT = System.currentTimeMillis();
    long timeTest= stopT-startT;
    ArrayList<String> PrivateKey = new ArrayList<String>();
    PrivateKey.add("d: "+key.getD().bitLength());
    PrivateKey.add("dInvs: "+ key.getDInvs().bitLength());
    ArrayList<String> PublicKey = new ArrayList<String>();
    PublicKey.add("n:" + key.getK());
    PublicKey.add("g:" + key.getNPlusOne().bitLength());
    int usoRAM = megaFinish- megaStart;
    int usoBateria= batteryStop-batteryStart;
    float usoCPU = cpuStop - cpuStart;

generarReporte(tkeyGen,PrivateKey,PublicKey,tEncry,bitC1,bitC0,timeTest,
            usoRAM,usoBateria,usoCPU,
"ReporteTestEncriptacion_Paillier.txt");
}
```

4.3.2.4.2.3.2: *Imágenes de la Prueba*



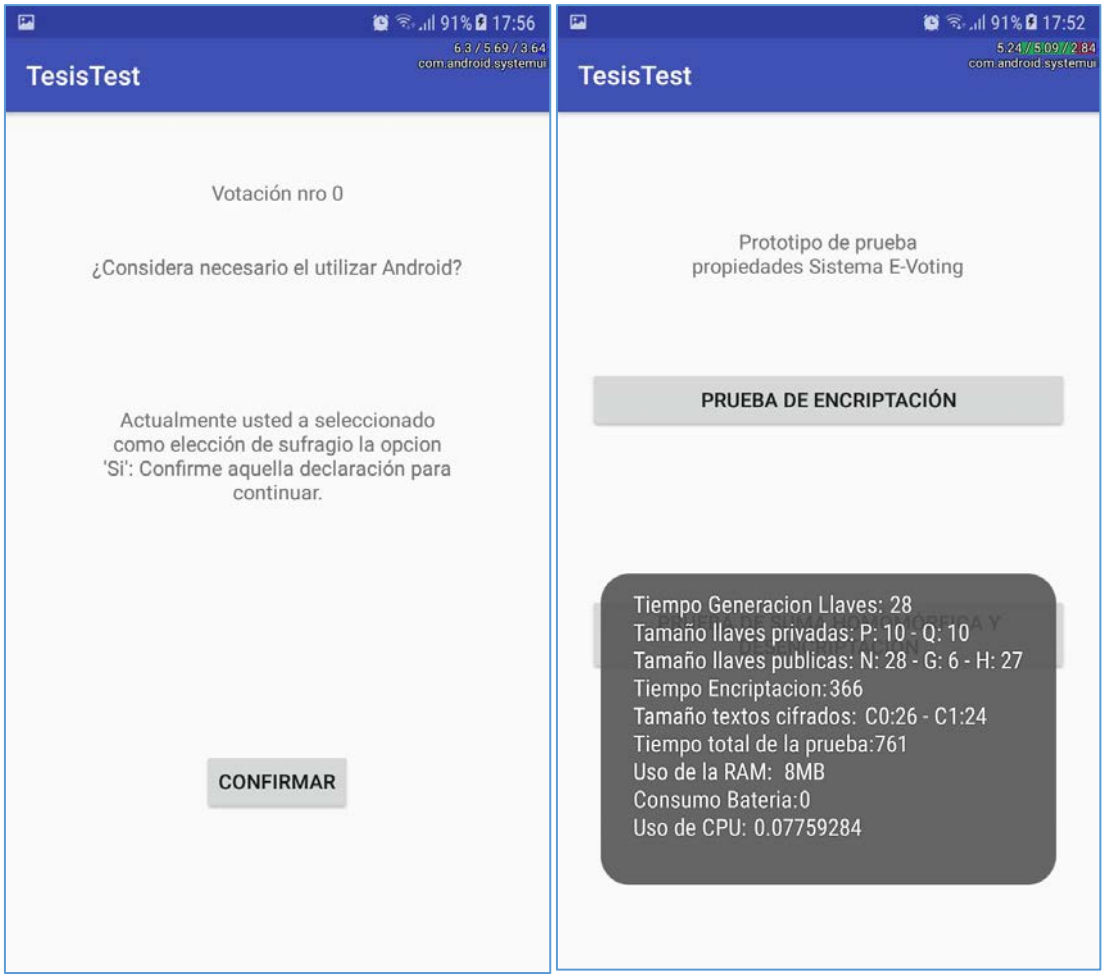


4.3.2.4.2.4: Okamoto-Uchiyama

4.3.2.4.2.4.1: Código de la Prueba

```
public void Encrypt_OkamotoUchiyama(int valor){
    Integer megaStart = getFreeRAM();
    int batteryStart
=getBatteryPercentage(PruebaEncriptacionConfirmacion.this);
    long startT = System.currentTimeMillis();
    float cpuStart =getCPU_Usage();
    OkamotoUchiyama_p esystem= new OkamotoUchiyama_p();
    long start = System.currentTimeMillis();
    OkamotoUchiyama_privateKey key;
    key = OUKeyGen.OkamotoUchiyamaKey(10,12233356);
    long stop = System.currentTimeMillis();
    long tkeyGen= stop - start;
    esystem.setDecryptEncrypt(key);
    ArrayList<BigInteger> send= new ArrayList<BigInteger>();
    start=System.currentTimeMillis();
    BigInteger c1=  esystem.encrypt_ou(BigInteger.ONE), c0=
esystem.encrypt_ou(BigInteger.ZERO);
    float cpuStop =getCPU_Usage();
    stop = System.currentTimeMillis();
    int bitC1 = c1.bitLength(), bitC0= c0.bitLength();
    long tEncry = stop - start;
    if(valor==1){
        //Almacenar en orden [1][0]
        send.add(c1);
        send.add(c0);
    }else if(valor==2){
        //Almacenar en orden [0][1]
        send.add( c0);
        send.add( c1);
    }
    Integer megaFinish = getFreeRAM();
    int batteryStop
=getBatteryPercentage(PruebaEncriptacionConfirmacion.this);
    long stopT = System.currentTimeMillis();
    long timeTest= stopT-startT;
    ArrayList<String> PrivateKey = new ArrayList<String>();
    PrivateKey.add("P: " + key.getP().bitLength());
    PrivateKey.add("Q: " + key.getQ().bitLength());
    ArrayList<String> PublicKey = new ArrayList<String>();
    PublicKey.add("N: " + key.getN().bitLength());
    PublicKey.add("G: " + key.getG().bitLength());
    PublicKey.add("H: " + key.getH().bitLength());
    int usoRAM = megaFinish- megaStart;
    int usoBateria= batteryStop-batteryStart;
    float usoCPU = cpuStop - cpuStart;

generarReporte(tkeyGen,PrivateKey,PublicKey,tEncry,bitC1,bitC0,timeTest,
            usoRAM, usoBateria,usoCPU,
"ReporteTestEncriptacion_OkamotoUchiyama.txt");
}
```

4.3.2.4.2.4.2: Imágenes de la *Prueba*



4.3.2.4.3: Implementación Prueba de Homomorfismo & Desencriptación

4.3.2.4.3.1: DFD de la Prueba

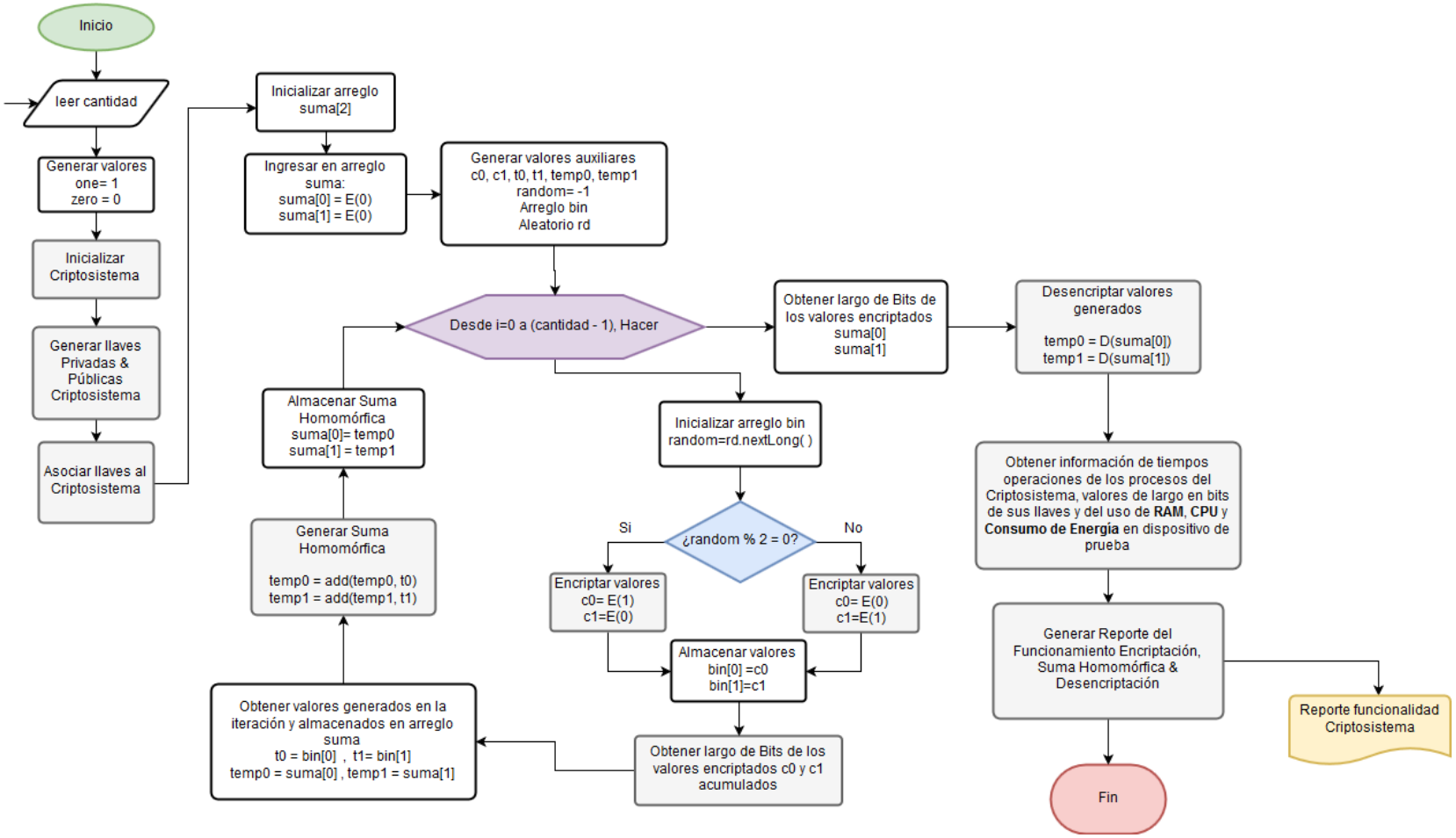


Ilustración 29: *DFD prueba Suma Homomórfica y Desencriptación – Android, fuente propia.*

4.3.2.4.3.2: Código utilizado por Ambos Esquemas en esta Prueba

4.3.2.4.3.2.1: Función para Generar Cuerpo del Reporte

Función que recibe la información generada por ambos esquemas y la organiza en lo que será el cuerpo de un archivo "`sFileName`" con formato "(nombre).txt".

```
public void generarReporte(long tiempoKG, ArrayList<String> PrivateKey,
ArrayList<String> PublicKey, long tEncry, int c1BitL, int c0BitL,
long tSHomo, int c1BitLuSH, int c0BitLuSH,long tDencry, long tiempoTest,
int usoRAM, int usoBateria, float usoCPU, String nombreArchivo){
    String llavesPrivadas ="Tamaño llaves privadas:\t";
    for(int i=0; i<PrivateKey.size();i++) {
        if (i == PrivateKey.size() - 1) {
            llavesPrivadas = llavesPrivadas.concat(PrivateKey.get(i));
        } else {
            llavesPrivadas= llavesPrivadas.concat(PrivateKey.get(i)+" - ");
        }
    }
    String llavesPublicas="Tamaño llaves publicas:\t";
    for(int i=0; i<PublicKey.size();i++) {
        if(i==PublicKey.size()-1) {
            llavesPublicas = llavesPublicas.concat(PublicKey.get(i));
        }else{
            llavesPublicas = llavesPublicas.concat(PublicKey.get(i)+" - ");
        }
    }
    String cuerpo;
    String pendiente= "-";
    cuerpo = "Tiempo Generacion Llaves:\t" + tiempoKG +"\n"
            +llavesPrivadas +"\n"
            +llavesPublicas+"\n"
            +"Tiempo Encriptacion:\t"+tEncry+"\n"
            +"Tamaño textos cifrados:\t C0:"+ c0BitL + " - C1:"+
            c1BitL+"\n"
            +"Tiempo SumaHomomórfica:\t"+tSHomo+"\n"
            +"Tamaños Arreglos cifrados:\t A[0]:"+ c0BitLuSH + " -
            A[1]:"+ c1BitLuSH+"\n"
            +"Tiempo Desencriptacion:\t"+tDencry+"\n"
            +"Tiempo total de la prueba:\t"+tiempoTest+"\n"
            +"Uso de la RAM:\t"+usoRAM+"MB"+"\n"
            +"Consumo Bateria:\t"+usoBateria+""+"\n"
            +"Uso de CPU:\t"+usoCPU+""+"\n";
    Toast.makeText(MainActivity.this, cuerpo, Toast.LENGTH_LONG).show();
    generarReporte( nombreArchivo, cuerpo);
}
```

Observación:

- Para esta prueba, se considerará una cantidad de votos a generar de 1000, que es un valor mayor al aproximado existente de alumnos que actualmente existen en la carrera Ingeniería Civil en Informática (370).[6]

[6] Información entregada por Marcia Andrea Aguayo Aguayo, maraguayo@ubiobio.cl el 7 de agosto de 2018 a las 9:50 am.

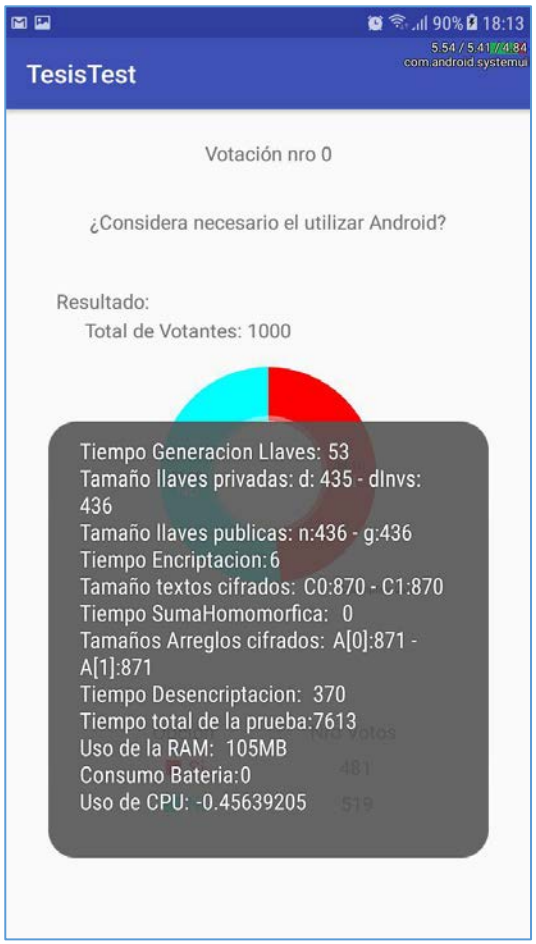
4.3.2.4.3.3: Paillier

4.3.2.4.3.3.1: *Código de la Prueba*

```
public void simularVotacionPaillier(int cant){
    BigInteger  one = BigInteger.ONE, zero = BigInteger.ZERO;
    Integer megaStart = getFreeRAM();
    int batteryStart =getBatteryPercentage(MainActivity.this);
    float cpuStart =getCPU_Usage();
    long startT = System.currentTimeMillis();
    Paillier esystem= new Paillier();
    long start = System.currentTimeMillis();
    PaillierPrivateKey key= KeyGen.PaillierKey(218,122333356);
    long stop = System.currentTimeMillis();
    long tkeyGen= stop - start;
    esystem.setDecryptEncrypt(key);
    //Inicialización arreglo suma
    ArrayList<BigInteger> suma= new ArrayList<BigInteger>();
    suma.add(esystem.encrypt(zero));
    suma.add(esystem.encrypt(zero));
    long miliEnc=0, miliDes=0, miliSum=0;
    long random=-1;
    BigInteger c0, c1, t0, t1, temp0, temp1;
    int c1BitL=0, c0BitL=0;
    Random rd=new Random();
    ArrayList<BigInteger> bin;
    long tempStart, tempStop;
    for(int i=0; i< cant;i++){
        //Encriptación:
        bin= new ArrayList<BigInteger>();
        tempStart = System.currentTimeMillis();
        random= rd.nextLong();
        if (random%2==0){
            c0=esystem.encrypt(one);
            c1=esystem.encrypt(zero);
        }else{
            c0=esystem.encrypt(zero);
            c1=esystem.encrypt(one);
        }
        bin.add(c0);        bin.add(c1);
        tempStop = System.currentTimeMillis();
        c0BitL = c0BitL+ c0.bitLength();
        c1BitL = c1BitL+ c1.bitLength();
        miliEnc= miliEnc +(tempStop -tempStart);
        //SumaHomomórfica
        tempStart = System.currentTimeMillis();
        t0= bin.get(0);
        t1= bin.get(1);
        temp0= suma.get(0);
        temp1= suma.get(1);
        temp0 = esystem.add(temp0,t0);
        temp1 = esystem.add(temp1,t1);
        suma.set(0,temp0);
        suma.set(1,temp1);
        tempStop = System.currentTimeMillis();
        miliSum=miliSum+(tempStop -tempStart);
    }
```

```
    //Desencriptación sumatoria de votos almacenados
    BigInteger tmp1 = null, tmp2=null;
    temp0 = suma.get(0);
    int c0BitLuSH = temp0.bitLength();
    temp1 = suma.get(1);
    int c1BitLuSH = temp1.bitLength();
    tempStart = System.currentTimeMillis();
    tmp1= esystem.decrypt(temp0);
    tmp2= esystem.decrypt(temp1);
    float cpuStop =getCPU_Usage();
    tempStop = System.currentTimeMillis();
    miliDes= (tempStop -tempStart);
    //Almacenamiento en enteros
    valO1 = tmp1.intValue();
    valO2 = tmp2.intValue();
    Integer megaFinish = getFreeRAM();
    int batteryStop =getBatteryPercentage(MainActivity.this);
    long stopT = System.currentTimeMillis();
    long timeTest= stopT-startT;
    ArrayList<String> PrivateKey = new ArrayList<String>();
    PrivateKey.add("d: "+key.getD().bitLength());
    PrivateKey.add("dInvs: "+ key.getDInvs().bitLength());
    ArrayList<String> PublicKey = new ArrayList<String>();
    PublicKey.add("n:" + key.getK());
    PublicKey.add("g:" + key.getNPlusOne().bitLength());
    int usoRAM = megaFinish- megaStart;
    int usoBateria = batteryStop-batteryStart;
    float usoCPU = cpuStop - cpuStart;

generarReporte(tkeyGen,PrivateKey,PublicKey,(miliEnc/cant),c1BitL/cant,c0
BitL/cant,miliSum/cant,c1BitLuSH,c0BitLuSH,miliDes,timeTest,usoRAM,
usoBateria, usoCPU, "ReporteTestSumaHomomórfica-
Desencriptacion_Paillier.txt");
}
```

4.3.2.4.3.3.2: Imágenes de la Prueba



4.3.2.4.3.4: Okamoto-Uchiyama

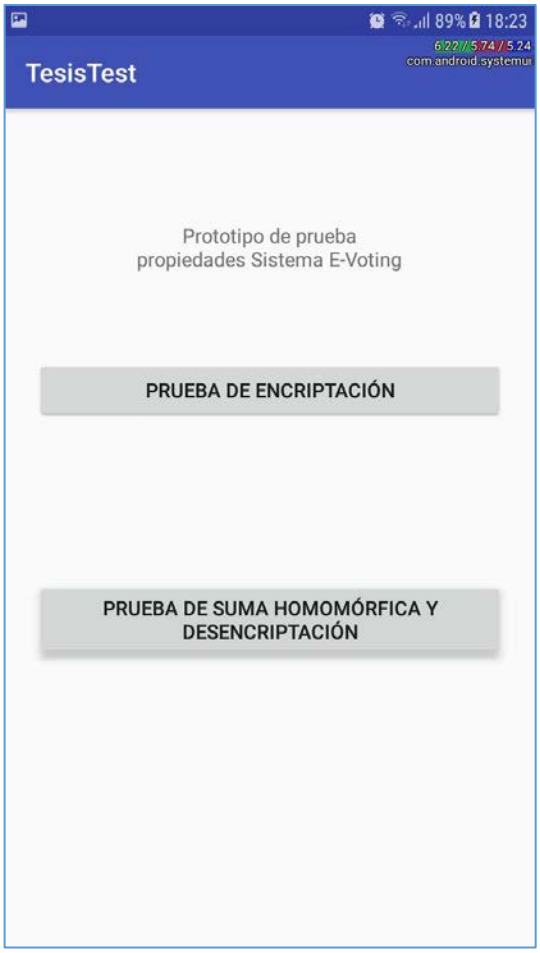
4.3.2.4.3.4.1: *Código de la Prueba*

```
public void simularVotacionOU(int cant){
    BigInteger  one = BigInteger.ONE, zero = BigInteger.ZERO;
    Integer megaStart = getFreeRAM();
    int batteryStart =getBatteryPercentage(MainActivity.this);
    float cpuStart =getCPU_Usage();
    long startT = System.currentTimeMillis();
    OkamotoUchiyama_p esystem= new OkamotoUchiyama_p();
    OUKeyGen = new OUKeyGen();
    long start = System.currentTimeMillis();
    OkamotoUchiyama_privateKey key=
ouKeyGen.OkamotoUchiyamaKey(10,564564);
    long stop = System.currentTimeMillis();
    long tkeyGen= stop - start;
    esystem.setDecryptEncrypt(key);
    ArrayList<BigInteger> suma= new ArrayList<BigInteger>();
    suma.add(esystem.encrypt_ou(zero));
    suma.add(esystem.encrypt_ou(zero));
    long miliEnc=0, miliDes=0, miliSum=0;
    long random=-1;
    BigInteger c0, c1, t0, t1, temp0, temp1;
    int c1BitL=0, c0BitL=0;
    Random rd=new Random();
    ArrayList<BigInteger> bin;
    long tempStart, tempStop;
    for(int i=0; i< cant;i++){
        //Encriptación:
        bin= new ArrayList<BigInteger>();
        tempStart = System.currentTimeMillis();
        random= rd.nextLong();
        if (random%2==0){
            c0=esystem.encrypt_ou(one);
            c1=esystem.encrypt_ou(zero);
        }else{
            c0=esystem.encrypt_ou(zero);
            c1=esystem.encrypt_ou(one);
        }
        bin.add(c0);
        bin.add(c1);
        tempStop = System.currentTimeMillis();
        c0BitL = c0BitL+ c0.bitLength();
        c1BitL = c1BitL+ c1.bitLength();
        miliEnc= miliEnc +(tempStop -tempStart);
        //SumaHomomórfica
        tempStart = System.currentTimeMillis();
        t0= bin.get(0);        t1= bin.get(1);
        temp0= suma.get(0);    temp1= suma.get(1);
        temp0 = esystem.addOU(temp0,t0);
        temp1 = esystem.addOU(temp1,t1);
        suma.set(0,temp0);
        suma.set(1,temp1);
        tempStop = System.currentTimeMillis();
        miliSum=miliSum+(tempStop -tempStart);
    }
```

```
    //Desencriptación sumatoria de votos almacenados
    BigInteger tmp1 = null, tmp2=null;
    temp0 = suma.get(0);
    int c0BitLuSH = temp0.bitLength();
    temp1 = suma.get(1);
    int c1BitLuSH = temp1.bitLength();
    tempStart = System.currentTimeMillis();
    tmp1= esystem.decrypt_ou(temp0);
    tmp2= esystem.decrypt_ou(temp1);
    float cpuStop =getCPU_Usage();
    tempStop = System.currentTimeMillis();
    miliDes= (tempStop -tempStart);
    //Almacenamiento en enteros
    valO1 = tmp1.intValue();
    valO2 = tmp2.intValue();
    Integer megaFinish = getFreeRAM();
    int batteryStop =getBatteryPercentage(MainActivity.this);
    long stopT = System.currentTimeMillis();
    long timeTest= stopT-startT;
    ArrayList<String> PrivateKey = new ArrayList<String>();
    PrivateKey.add("P: " + key.getP().bitLength());
    PrivateKey.add("Q: " + key.getQ().bitLength());
    ArrayList<String> PublicKey = new ArrayList<String>();
    PublicKey.add("N: " + key.getN().bitLength());
    PublicKey.add("G: " + key.getG().bitLength());
    PublicKey.add("H: " + key.getH().bitLength());
    int usoRAM = megaFinish- megaStart;
    int usoBateria = batteryStop-batteryStart;
    float usoCPU = cpuStop - cpuStart;

generarReporte(tkeyGen,PrivateKey,PublicKey,(miliEnc/cant),c1BitL/cant,c0
BitL/cant, miliSum/cant,c1BitLuSH,c0BitLuSH,miliDes,timeTest,usoRAM,
usoBateria, usoCPU, "ReporteTestSumaHomomórfica-
Desencriptacion_OkamotoUchiyam.txt");
}
```

4.3.2.4.3.4.2: Imágenes de la *Prueba*

Actualmente esta prueba está fallando en el dispositivo.



4.3.2.4.4: Tablas Comparativas Librerías

4.3.2.4.4.1: Primera prueba

	Paillier	*Okamoto Uchiyama*
Valor ingresado variable S	218	10
Estabilidad en Ejecución de Pruebas	Si	No[7]
Tiempo de Generación de llaves **[Milisegundos]**	41	28
Tamaño llaves Privadas Generadas	D:434 Inverso de D: 435	P: 10 Q: 10
Tamaño Llaves Públicas Generadas	n:436 g:436	N: 28 G: 6 H: 27
Tiempo de Encriptación **[Milisegundos]**	371	366
Tamaño Texto Cifrado	$C_0 = 871$ $C_1 = 871$	$C_0 = 26$ $C_1 = 24$
Tiempo de Duración de la Prueba	777	761
Uso de RAM Dispositivo	1MB	8MB
Consumo de Energía	0	0
Uso de CPU Dispositivo	~ 0.017	~ 0.775

Tabla 10: *Tabla con los resultados de encriptación de un voto en el Prototipo Android, fuente propia.*

[7] En la realización de las pruebas en Android se determinó que el ingreso de un valor de la variable "s" mayor a 13 provocaba que el programa no respondieses, por lo cual se optó por la selección de un valor más estable ($s = 10$).

En esta tabla se pretende entregar la información correspondiente a los valores generados por las mismas librerías, tales como:

- Tiempo en Milisegundos que poseen las librerías para:
    - Generación de llaves del sistema.
    - El proceso de Encriptación.
    - El proceso de Desencriptación.
    - Duración de la Prueba.
- Tamaños Llaves Públicas y Privadas Generadas.
- Tamaño Textos cifrados
- Usos estimativos de RAM, CPU y Consumo de Energía.

Mientras que *"Valor ingresado variable S"* corresponde a un valor semilla que permite determinar el largo de las llaves, mientras que la *"Estabilidad del Criptosistema al ser implementado"* corresponde a la existencia de anomalías que impidan un resultado fidedigno o posee algún inconveniente al momento de implementarse.

4.3.2.4.4.2: Segunda prueba

	Paillier	*Okamoto Uchiyama*[8]
Valor ingresado variable S	218	10
Estabilidad en Ejecución de Pruebas	Si	No[9]
Tiempo de Generación de llaves **[Milisegundos]**	53	-
Tamaño llaves Privadas Generadas	D:435 Inverso de D: 436	-
Tamaño Llaves Públicas Generadas	n:436 g:436	-
Tiempo de Encriptación Promedio **[Milisegundos]**	6	-
Tamaño Texto Cifrado Promedio	$C_0 = 870$ $C_1 = 870$	-
Tiempo Promedio Suma Homomórfica **[Milisegundos]**	0	-
Tamaño Resultado Suma Homomórfica.	A[0] = 871 A[1] = 871	-
Tiempo de Desencriptación **[Milisegundos]**	370	-
Tiempo de Duración de la Prueba	7613	-
Uso de RAM Dispositivo	105 MB	-
Consumo de Energía	0	-
Uso de CPU Dispositivo	-0.456	-

Tabla 11: *Tabla con los resultados de la simulación de un proceso de Votación en el Prototipo Android, fuente propia.*

[8] Esta prueba ha fallado en el prototipo.

[9] En la realización de las pruebas en Android se determinó que el ingreso de un valor de la variable "s" mayor a 13 provocaba que el programa no respondieses, por lo cual se optó por la selección de un valor más estable ($s = 10$).

En esta tabla se pretende entregar la información correspondiente a los valores generados por las mismas librerías, tales como:

- Tiempo en Milisegundos que poseen las librerías para:
    - Generación de llaves del sistema.
    - El proceso de Encriptación.
    - El proceso de Desencriptación.
    - Tiempo implementación de la Suma Homomórfica.
    - Duración de la Prueba.
- Tamaños Llaves Públicas y Privadas Generadas.
- Tamaño Textos cifrados.
- Usos estimativos de RAM, CPU y Consumo de Energía.

Mientras que *"Valor ingresado variable S"* corresponde a un valor semilla que permite determinar el largo de las llaves, mientras que la *"Estabilidad del Criptosistema al ser implementado"* corresponde a la existencia de anomalías que impidan un resultado fidedigno o posee algún inconveniente al momento de implementarse.

4.3.2.5: Conclusión de las Pruebas en ambiente Android.

Como se puede apreciar en la tabla comparativa, la implementación de Okamoto-Uchiyama en ambiente Android es menos inestable debido a la complejidad existente en la implementación de las llaves privadas lo cual puede explicar el motivo de que, teniendo ambas pruebas un código similar en la prueba con la simulación de votos, la prueba con Okamoto-Uchiyama falle en su implementación.

Si bien, los cálculos tanto del Uso de la RAM, el consumo de Energía y el Uso de CPU fueron estimativos, y el prototipo ha presentado una inesperada falla en la generación de los reportes de las pruebas, en lo que corresponde a la implementación de los sistemas criptográficos probados en esta sección se puede concluir que:

- La encriptación de un único voto para su posterior envío a un sistema de votación puede aplicarse con la librería generada de Okamoto-Uchiyama en un ambiente Android.
- En el caso de la librería de Paillier, sería posible utilizar sus librerías tanto para el envío de un voto, como para la recepción de un conjunto de votos correspondientes a una votación y permitir el proceso de sumar todos los votos para realizar la desencriptación de los resultados totales posteriormente.

Capítulo 5: Conclusión.

La conclusión general de este proyecto de titulación es que logró el objetivo de "Implementar una librería para Java de al menos un algoritmo criptográfico que permita el desarrollo de un sistema de votación online, ambientado en el sufragio estudiantil.", con lo cual esta tesis investigativa podrá servir de base teórica para futuros proyectos a desarrollarse en el contexto de la realización de un "Sistema E-Voting" para las asambleas de la carrera Ingeniería Civil en Informática.

La finalidad de esta Tesis fue la de realizar una comparación de al menos dos criptosistemas en un ambiente Android para verificar la posible implementación de estos de forma móvil, para sentar las bases de un posible futuro desarrollo de un Sistema E-voting que pueda implementarse en las asambleas de la carrera: Finalidad que, de manera teórica y práctica, considero que se ha logrado ante lo anteriormente expuesto a lo largo de este informe.

En las siguientes secciones del capítulo se expresará las conclusiones que se obtuvieron respecto de los "Objetivos Específicos" y, de manera más subjetiva, las conclusiones respecto a las "Dificultades", "Descubrimientos", "Recomendaciones" y "Trabajos Futuros" de este proyecto.

Sección 5.1: Conclusiones Respecto a los Objetivos Específicos.

5.1.1: *Estudiar diferentes tipos de algoritmos criptográficos para realizar una comparación entre estos, considerando sus diferencias y sus semejanzas.*

Como conclusión de este objetivo, la vitalidad de su realización fue fundamental para el desarrollo completo del proyecto. La comprensión de estos algoritmos fue de vital importancia para la obtención un marco teórico con el cual desarrollar los siguientes objetivos, enfocándose una gran parte del desarrollo de esta tesis en la realización de este.

5.1.2: *Comparar algoritmos estudiados para la selección de al menos uno de ellos para implementarse en lenguaje java.*

Como conclusión de este objetivo se puede obtener que, pese a que no se consideraron todos los criptosistemas existentes, los sistemas criptográficos estudiados comparten características similares además de poder compartir un origen en común como lo es en el caso de ElGammal: Siendo este la base teórica de los Esquemas de Paillier y de Okamoto-Uchiyama, pudiendo ser complementado con otros esquemas criptográficos como en el caso de su implementación junto a Blind Signature & Secret Sharing.

5.1.3: *Evaluar el o los algoritmos implementados en librería Java para la realización de pruebas en ambiente Android.*

Como conclusión a este objetivo se puede obtener la gran importancia que posee la comprensión de los fundamentos matemáticos que afectan a los procesos del esquema a codificar, del diseño previo a la implementación de clases y funciones, tanto en lenguaje java como para las posteriores pruebas en un ambiente Android.

Siendo este objetivo en el cual finalmente se visualizó un resultado palpable al trabajo desarrollado en el proyecto, y considerando lo anteriormente mencionado, se puede concluir que la planificación, constancia y la proactividad son elementos fundamentales para la realización de este tipo de proyectos.

Sección 5.2: Conclusiones Respecto a las Dificultades.

Desde el inicio de este proyecto las dificultades no se encontraron ausentes durante el desarrollo de esta tesis, estando presentes en la complejidad del tema elegido, la decisión de realizar este trabajo por cuenta propia, incluso ante la ausencia de alguna pauta de evaluación de Proyectos de Titulación de carácter investigativos como el presente.

Además de ello existieron factores externos e internos a mi persona que entorpecieron y retrasaron el resultado de este extenso proyecto.

Comenzando con los externos, originalmente este proyecto había estimado culminarlo en el segundo semestre del 2017, semestre en que paralelamente estaba realizando el ramo de "Ingeniería de Software" con el profesor Pedro Campos. Resumiendo, aquel semestre los avances en este proyecto fueron ínfimos ante la necesidad de poder rendir correctamente para dicho ramo ante la realización de un sistema al cual personalmente buscaba brindarle calidad en un grupo que, por decir menos, inició con seis participantes de los que tan sólo quedamos dos en la última instancia de entrega.

Desarrollo que debió de mantenerse limitado en el receso de verano del 2018, ante la realización de la segunda Práctica Profesional para la obtención del grado de "Licenciado en Ciencias de la Ingeniería".

Mientras que, en el periodo correspondiente al primer semestre del 2018, los movimientos radicales de feminismos impulsados en la universidad dificultaron las comunicaciones con mi profesor Guía en las últimas semanas del desarrollo del proyecto dado a las responsabilidades que este poseía con los alumnos de pregrado al estos deponer el paro, paro votado "democráticamente" con el voto de sesenta estudiantes de la carrera.

Por otro lado, uno de los mayores factores internos que afectaron al desempeño de este proyecto fue la constante autocrítica que no me permitía visualizar el extenso trabajo realizado dado a la mentalidad de "Podría haberlo hecho mejor".

En lo que respecta a las dificultades encontradas a lo largo del desarrollo de este proyecto fueron el objetivo de la realización de este proyecto dado al rechazo que sentía por la idea de desarrollar un sistema convencional.

Sinceramente opté a este tema como Proyecto de Titulación con la idea de adquirir nuevos conocimientos, arriesgándome conscientemente a la posibilidad de no ser capaz de cumplir con todos los objetivos planteados.

Con el objetivo de desafiarme a mí misma a la vez de realizar un trabajo que pueda contribuir a la carrera que me formó ante todo lo que aprendí en los años que he cursado, por ello he de ahondar con profundidad en las “dificultades” encontradas al momento del desarrollo en la siguiente sección.

Sección 5.3: Conclusiones Respecto a los Descubrimientos.

Como alumna tesista puedo concluir que la realización de este proyecto ha sido sumamente enriquecedora y gratificante de realizar por más que la incertidumbre de un resultado no satisfactorio para la evaluación de este mismo.

Comenzando con el principal análisis de todos los Esquemas Criptográficos estudiados de los cuales tan sólo se logró profundizar de manera superficial en cuatro de ellos, limitado en el aspecto informático del mismo proyecto, sin poder ahondar en el análisis más matemático por limitantes de tiempo.

La implementación de los esquemas seleccionados, tomando como base la estructura del esquema de Paillier estudiada, fue un proceso arduo. Requiriendo de la comprensión de la estructura utilizada en Paillier y su funcionamiento, además de un entendimiento de cómo se realizaban las interacciones entre las clases: Mas, los restantes criptosistemas tenían sus propias características, por lo que el entendimiento del funcionamiento teóricos de los mismos fue fundamental, por lo que debí de desarrollar las pruebas al código java en lo que lograba comprender los restantes sistemas.

La librería de ElGammal, expresada de manera simple por el tiempo y el trabajo que aún debía de realizar, es la librería menos compleja de las tres implementadas. Y, considerando que la encriptación de un texto plano “0” genera como resultado un texto cifrado “0”, pese a sus resultados en tiempo no recomendaría el utilizar esta versión en un ambiente de e-voting.

Por otro lado sinceramente he de admitir que de los dos esquemas restantes, Boneh-Goh-Nissim y Okamoto-Uchiyama, han de ser mis dos predilectos debido a la complejidad matemática que representó el llevar a cabo sus implementaciones, lo cual resulta irónico ante la incertidumbre que me generó (especialmente el primero) la realización del código de ambas librerías.

La complejidad del esquema de Okamoto-Uchiyama radicaba meramente en el desconocimiento de un concepto matemático, el “Módulo Inverso”, con lo cual en sus primeras pruebas no lograba obtener resultados correctos: Al conocer tal concepto e implementarlo este fue el primer criptosistema implementado por mi persona que logró funcionar correctamente.

Lamentablemente, por limitantes de tiempos, no logré optimizar la función "primitiveRootModulo_Gen(BigInteger n, Random rnd, BigInteger p)" que permite generar la llave pública "g" del sistema la cual extiende el tiempo de generación de las llaves de dicha clase y que, teorizo, es la razón por la cual las pruebas en Android de esta librería tuvieron problemas.

Por otra parte, los motivos que vuelven al esquema de Boneh-Goh-Nissim mi esquema favorito inician con que este fue un resto desde el comienzo: Ante la complejidad del concepto de Curvas Elípticas junto a su implementación en código java, llegando al punto de procurar avanzar con la implementación en Android ante la incertidumbre de poder siquiera implementar algún elemento de este esquema.

Cada paso exitoso de la implementación fue satisfactorio a medida que lograba funcionar y, sinceramente, no esperaba que realmente pudiese implementarlo: El momento de generarse por primera vez las llaves estaba sorprendida; Luego, tras implementar en código los procesos de Encriptación y Desencriptación y comprobar que estos funcionaban, la sorpresa, y el alivio se incrementó; Finalmente, cuando quise comprobar la implementación de la Suma Homomórfica fue cuando descubrí el "irregular" comportamiento que poseía la generación de dos de las llaves públicas "g" y "h" (que deben de ser puntos válidos de la Curva Elíptica), con lo cual estaba completamente limitada ante mis superficiales conocimientos sobre el cálculo de dichos puntos.

Me habría gustado tener el tiempo de solucionar tal "irregularidad" y poder comprobar la efectividad de este criptosistema en su total funcionalidad, sin embargo el comportamiento azaroso generado por estas dos llaves públicas[10] me resulta fascinante, llegando a denominar a la librería como "Un código que funciona y no funciona a la vez" a quienes se lo he comentado.

He de agregar que, pese a que mi computador personal logró limitarme con respecto al desarrollo Android (ante los recursos que Android Studio requiere para su funcionamiento), la experiencia de desarrollar una aplicación para este tipo de sistemas operativos resultó más grato de lo esperado.

[10] Irregularidad ahondada en la sección 4.2.2.4, "Problema de la Librería" del informe.

Por ende, en materias de descubrimientos brindados a lo largo de este proyecto he de concluir que, ha sido una ardua labor, más los conocimientos y experiencias adquiridas son de por sí un aspecto invaluable de este proyecto ante los conocimientos teóricos de los criptosistemas, los conocimientos matemáticos adquiridos al implementar las librerías y el haber aprendido elementos básicos de la programación en Android para desarrollar el prototipo para las pruebas.

Sección 5.4: Conclusiones Respecto a las Recomendaciones.

En este aspecto del proyecto, enfocado únicamente a "Recomendaciones de modificaciones a la información existente en este proyecto", me gustaría el sugerir a quien decida tomar el proyecto del desarrollo de un sistema E-Voting para las asambleas de la carrera Ingeniería Civil en Informática la optimización de los esquemas desarrollados en este proyecto limitados por el tiempo y el trabajo investigativo que estos requirieron:

- Probar una implementación más sofisticada del esquema de ElGammal, de preferencia donde se implementar la Suma Homomórfica.
- Optimizar la función "primitiveRootModulo_Gen(BigInteger n, Random rnd, BigInteger p)" de la librería de Okamoto-Uchiyama.
- Solucionar el problema existente en la librería de Boneh-Goh-Nissim desarrollada que afecta tanto al proceso de encriptación, desencriptación como a la suma homomórfica.

Todo esto con el objetivo de realizar nuevas pruebas comparativas, basadas en las desarrolladas en este proyecto, para comprobar si algunos de estos esquemas modificados podría funcionar de mejor manera que el criptosistema de Paillier en un ambiente de programación Android.

En caso contrario, dado a los resultados obtenidos en las pruebas, la implementación de la librería de Paillier para el desarrollo de un Sistema E-Voting para dispositivos Android aplicables a las asambleas de la carrera Ingeniería Civil en Informática, considerando que la Simulación de votación se realizó con "1000" votantes y se sabe que existen actualmente un aproximado de 370 alumnos cursando la carrera.

No obstante, librerías como la Okamoto-Uchiyama y Boneh-Goh-Nissim podrían ser de utilidad en la futura formación de dicho sistema E-Voting, con evidentes limitaciones:

> En el caso de la implementación del esquema de Okamoto-Uchiyama, la recepción y conteo de los votos deberá realizarse preferentemente en un dispositivo cuyo sistema operativo no se encuentre tan limitado como lo son el de los dispositivos móviles, limitando el tamaño de las llaves como se puede apreciar en la sección 4.3.2 del Capítulo "Implementación de Librerías & Desarrollo de Pruebas".

En el caso de la implementación del esquema Boneh-Goh-Nissim, de solucionarse el problema funcional y comprobar su funcionamiento, de ser este óptimo se recomienda el probar su funcionalidad en un ambiente Android: De funcionar correctamente en un dispositivo con sistema operativo Android, se recomendaría que las llaves del sistema fueran generadas en la nube.

Como conclusión a esta sección puedo tan sólo agregar que este proyecto puede ser utilizado como una base tanto teórica como práctica para la futura elaboración de un Sistema E-Voting para las asambleas de la carrera Ingeniería Civil en Informática que facilite el proceso democrático que en estas ha de desarrollarse.

Sección 5.5: Conclusiones Respecto al Trabajo Futuro.

El trabajo futuro que concierne a la elaboración de un Sistema E-Voting para las asambleas de la carrera Ingeniería Civil en Informática podría considerarse extenso dado a que el enfoque de este proyecto tan sólo alcanzó a comparar la funcionalidad de cuatro esquemas de criptografía Asimétrica que implementan Homomorfismo para analizar la posibilidad de la implementación de las votaciones utilizando dispositivos móviles, es decir si en un dispositivo móvil se podría realizar el proceso de encriptación y el proceso de recepción lineal y conteo de votos.

Para el desarrollo de dicho sistema se debe realizar un análisis de las tecnologías disponibles para su implementación, estudios de factibilidad del sistema, implementar el uso de servidores web, analizar las interacciones que el sistema deberá de poseer, seleccionar un modelo de desarrollo que permita una mayor eficiencia de los recursos disponibles.

Tareas como diseño de las interfaces, la realización de pruebas de recepción de los votos en forma paralela para comprobar los límites de los esquemas estudiados al momento, quizás hasta realizar nuevas funciones que utilicen la funciones de Suma Homomórfica para la realización de recepción de votos en paralelo son algunas de los trabajos que quedan pendiente para quien decida continuar este proyecto.

Incluso está la posibilidad de complejizar aún más el sistema ante la posibilidad de implementar un sistema Streaming de las mismas asambleas para que las limitaciones físicas de espacio en la universidad no sean un inconveniente para la realización del mismo proceso de sufragio.

La única conclusión que puedo poseer en este aspecto es que, pese a que aún existe un arduo trabajo para la realización de este sistema, se espera de que la información reunida en este trabajo pueda servir como una base para futuros proyectos, ya sea en el contexto de carrera Ingeniería Civil en Informática, como su posible implementación o a nivel de nivel de facultad o, incluso, a nivel de la universidad del Bío-Bío.

Capítulo 6: Referencias.

Introducción

[1]. Material entregado a lo largo de la asignatura electiva *"Seguridad Informática"* Código 620481 (malla antigua) cursada en el periodo académico 2017-01.

[2]. Información introductoria del Esquema Sharim's Secret Sharing - https://en.wikipedia.org/wiki/Shamir%27s_Secret_Sharing

Papers de E-Voting

[3]. E-Voting a survey and Introduction – Thomas Rössler.

[4]. Survey on Remote Electronic Voting – Alexander Schneider, Christian Meter, Philipp Hagemeister. (2017)

[5]. A Survey on Crypt-Algorithms in Voting System – P. Ashok, P. Annadurai, R. Lavanya, P. Raghuvara Pandian (2016).

[6]. A Secure E-voting Architecture – A. S. Sodiya, S. A. Onashoga, D. I. Adelani.

[7]. A Survey of Remote Internet Voting Vulnerabilities – Okediran O. O., Omodiora E. O., Olabiyisi S. O., Ganiyu R. A.

Papers de Paillier

[8]. A Generalization of Paillier's Public-Key System with Applications to Electronic Voting – Ivan Damgård, Mads Jurik, Jesper Buus Nielsen.

[9]. Advanced E-Voting System Using Paillier Homomorphic Encryption Algorithm – Shifa Manaruliesya Anggriane, Surya Michrandi Nasution, Fairuz Azmi.

Papers de Okamoto-Uchiyama

[10]. A new Public-Ket Cryptosystem as Secure as Factoring – Tatsuki Okamoto, Shigenori Uchiyama.

[11]. Okamoto-Uchiyama homomorphic Encryption Algorithm Implementation in E-Voting Sistem – Rifki Suwandi, Surya Michrandi Nasution, Fairuz Azmi.

Papers de ElGammal

[12]. A Robust and High Speed E-Voting Algorithm Using ElGammel CrypoSystem – Abdolreza Rasouli Kenari, Javad Hosseinkhani, Mahboubeh Shamsi, Majid Harouni

[13]. A Secure and Optimally Efficient Multi-Authority Election Scheme – Ronald Cramer, Rosario Gennaro, Berry Schoenmakers.

[14]. Efficient receipt-free voting based on homomorphic encryption – Hirt Martin, Sako Kazue

[15]. Desig and Implementatio of Secure Remote e-voting System Using Homomorphic Encryption. – Ihsan Jabbar, Saad Najim Alsaad.

[16]. Practical Internet Voting System –Xun Yi, Eiji Okamoto.

[17]. An elementary Electronic Voting Protocol Using RFID – Xiangdong Li, Michael Carlisle, Andin C. Kwan, Lin Leung, Amara Enemuo, Michael Anshel.

Papers de Blind Signature

[18]. Secure E-Voting With Blind Signature – Subariah Ibrahim, Maznah Kamat, Mazleena Salleh, Shah Rizan Abdul Aziz.

[19]. Blind Signatures for untraceable payments. – David Chaum.

[20]. Untraceable electronic mail, return adresses and digital pseudonyms. – David Chaum.

Papers de Blowfish

[21]. Implementation of AES and Blowfish Algorithm. – Haldankar C, Kuwelkar S.

[22]. Investigating Efficiency of Blowfish and Rijndael (AES) Algorithms. – Kumar MA, Karthikeyan S.

[23]. DES, AES and Blowfish: Symmetric Key Cryptography Algorithms Simulation Based Performance Analysis – Jawahar Thakur, Nagesh Kumar.

[24]. Performance enhancement of Blowfish and CAST-128 algorithms and Security analysis of improved Blowfish algorithm using avalanche effect – Krishnamurthy G.N, Dr V. Ramaswamy, Leela G.H and Ashalatha M.E.

[25]. Blowfish Algorithm – Ms NehaKhatri Valmik , Prof. V. K Kshirsagar.

[26]. Bruce Schneier, “Applied Cryptography”, sección 14.3.

Papers de ElGammal Blind Signature & Secret Sharing

[27]. A secure anonymous E-Voting System based on Discrete Logarithm Problem. – Chen, Chin-ling; Chen, Yu-Yi; Jan, Jinn-Ke; Chen, Chih-Cheng

Papers de Cast-128

[28]. The Cast-128 encryption algorithm – Adams C.

[29]. Cryptography and network security principles and practice 2nd edition, Section 4.5 – W. Stallings

Papers de Goldewasser-Micali

[30]. Homomorphic Encription and Applications, Chapter 2 – Xun Yi, Russell Paulet, Elisa Bertino

Papers de Boneh-Goh-Nissim (BGN)

[31]. Homomorphic Encryption Schemes and Application for a Secure Digital World – Diana-Ștefania MAIMUȚ, Alecsandru PĂTRAȘCU, Emil SIMION

[32]. Homomorphic Encryption and the BGN Cryptosystem – David Mandell Freeman

Librerías, Códigos y documentaciones externos

[33]. Librería Paillier implementada por Murat Kantarcioglu, James Garrity y Sean Hall http://cs.utdallas.edu/dspl/cgi-bin/pailliertoolbox/javadoc/index.html?paillierp/Paillier.html

[34]. Código de generación de una variable primitiva a utilizarse en el código de Okamoto-Uchiyama. https://stackoverflow.com/questions/27764667/how-to-generate-public-key-for-okamoto-uchiyama-cryptosystem

[35]. Código que implementa una versión simplificada del esquema criptográfico ElGammal sin considerar el homomorfismo. http://faculty.washington.edu/moishe/javademos/Security/ElGamal.java

[36]. Código para implementar las suma homomórfica en ElGammal. https://nvotes.com/multiplicative-vs-additive-homomorphic-elgamal/

[37]. Explicación de curvas elípticas para la implementación de la librería de Boneh-Goh-Nissim. http://andrea.corbellini.name/2015/05/17/elliptic-curve-cryptography-a-gentle-introduction/

[38]. Librería con implementación de curvas elípticas utilizada en el desarrollo de la librería de Boneh-Goh-Nissim (BGN). https://www.bouncycastle.org/latest_releases.html

Capítulo 7: Agradecimientos.

A lo largo del desarrollo de esta tesis ha habido instantes en que los avances parecían ser insignificantes por lo cual deseo agradecer en primera instancia a mi pareja Cristóbal Dreau que, pese a la distancia, siempre me ha apoyado en mi vida universitaria y por quien finalmente culminé entrando a la carrera de Ingeniería Civil en Informática al verlo programando, y a mi gato Thor por el incondicional apoyo que me ha brindado a lo largo de su vida sin tener idea de qué es lo que estoy haciendo.

Deseo agradecer a la infinita paciencia de mi profesor guía Patricio Galdames, además de a todo docente con el cual tuve el gusto de conocer en mi vida universitaria y que lograron que creciesen mis conocimientos y pudiese crecer como persona.

A mi familia y a mis escasos amigos por el apoyo, especialmente a mi compañero de generación Alfredo Baquedano, que me enseñó lo necesario para implementar el Prototipo Android.

Finalmente deseo agradecer a los diferentes integrantes del Centro de Alumnos que, a lo largo de mi vida universitaria, han asumido los cargos que en estas instancias se generan y a su oligarquía disfrazada de pseudo-democracia, cuyo nepotismo sirvió de inspiración para la realización de este proyecto.

Capítulo 8: Bibliografía.

- Columna con reflexión sobre la funcionalidad de la votación online - <2 de Abril de 2018> http://www.elmercurio.com/blogs/2018/04/01/59165/Voto-electronico-Sin-papel-no-funciona.aspx
- Página para la generación de diagrama de flujos online - <30 de Julio del 2018> https://www.smartdraw.com/flowchart/diagramas-de-flujo.htm
- Página para la generación de diseños de interfaz Android online - <30 de Julio del 2018> https://ninjamock.com/
- Librería para la implementación de gráficos en Android. - <31 de Julio del 2018> https://github.com/PhilJay/MPAndroidChart
- Página con la consulta de cómo ejecutar las aplicaciones de Android. - <31 de Julio del 2018> https://developer.android.com/studio/run/?utm_source=android-studio
- Página con la consulta de cómo pasar variables a la interfaz Android. - <31 de Julio del 2018> https://stackoverflow.com/questions/768969/passing-a-bundle-on-startactivity
- Página para la generación de diagramas de clases. - <1 de Agosto del 2018> https://www.draw.io/
- Página con información de implementación de Objetos pertenecientes a una misma Clase de una interfaz Android a otra - <2 de Agosto del 2018> https://www.techjini.com/blog/passing-objects-via-intent-in-android/
- Página oficial de Samsung con la información del dispositivo Android en el cual se instaló y probó el prototipo de prueba Android - <2 de Agosto del 2018>. https://www.samsung.com/ar/smartphones/galaxy-j7-neo-core-sm-j701/SM-J701MZKPARO/
- Página oficial con documentación de gráficos aplicados en Android. - <3 de Agosto del 2018>. https://github.com/PhilJay/MPAndroidChart
- Página con información para calcular el uso de RAM de una aplicación Android en un dispositivo. - <3 de Agosto del 2018>. https://forum.xda-developers.com/showthread.php?t=2230102
- Página con información de cómo agregar librerías ".jar" a una aplicación Android. - <3 de Agosto del 2018>. https://es.stackoverflow.com/questions/56429/c%C3%B3mo-agregar-librer%C3%ADas-en-android-studio

- Página con información sobre cómo realizar la escritura de archivos ".txt" en Android. - <3 de Agosto del 2018>. http://instinctcoder.com/read-and-write-text-file-in-android-studio/
- Página oficial de Android con información de la clase ActivityManager - <4 de Agosto del 2018>. https://developer.android.com/reference/kotlin/android/app/ActivityManager
- Función que permite determinar si un número ingresado es primo. - <7 de Agosto del 2018>. http://lineadecodigo.com/java/numeros-primos-en-java/
- Página con información para calcular el nivel de la Batería de una aplicación Android en un dispositivo. - <7 de Agosto del 2018>. https://sackoverflow.com/questions/3291655/get-battery-level-and-state-in-android
- Página con información para calcular el uso de CPU de una aplicación Android en un dispositivo. - <7 de Agosto del 2018>. https://stackoverflow.com/questions/3118234/get-memory-usage-in-android

Capítulo 9: Anexos.

Sección 9.1: Código Librerías

9.1.1: Boneh-Goh-Nissim (BGN)

9.1.1.1: Clase "AbstractBGN.java"

```
import java.math.BigInteger;
import org.bouncycastle.math.ec.ECPoint;

public abstract class AbstractBGN {
	protected BGN_key key= null;
	protected boolean encryptMode=false;
	protected boolean decryptMode=false;
	protected String notReadyForEncryption =
		"You must first call setEncrypt or
		setDecryptEncrypt before calling this method";
	protected String notReadyForDecryption =
		"You must first call setDecrypt or
		setDecryptEncrypt before calling this method";

	public BGN_key getPublicKey(){
		if(key==null){
			return null;
		}
		return key.getPublicKey();
	}

	public ECPoint encrypt(BigInteger m){
		BigInteger q2=key.getQ2();
		if (m.compareTo(q2) == 1) {
			throw new IllegalArgumentException("El mensaje a
			encriptar no puede ser mayor a: "+q2.toString());
		}
		BigInteger random= BigInteger.valueOf(
		  key.getPublicKey().rnd.nextLong());
		random= random.mod(key.N).add(BigInteger.ONE);
		return encrypt(m, random, key);
	}

	public ECPoint encrypt(BigInteger m, BigInteger random, BGN_key key){
		return encrypt(m, random, key.getH(), key.getG());
	}

	public ECPoint encrypt(BigInteger m, BigInteger random,
      ECPoint h, ECPoint g){
		ECPoint t1, t2, encoded;
		t1 = g.multiply(m);
		t2 = h.multiply(random);
		encoded = t1.add(t2);
		return encoded;
	}
```

```
        //Función de suma homomórfica.
        public ECPoint add_BGN(ECPoint c1, ECPoint c2){
                BGN_key key = getPublicKey();
                ECPoint h = key.getH();
                BigInteger r = BigInteger.valueOf(
                key.getPublicKey().rnd.nextLong());
                r= r.mod(key.N).add(BigInteger.ONE);
                ECPoint h_r = h.multiply(r).normalize();
                ECPoint suma = c1.add(c2).normalize().add(h_r);
                return suma;
        }

        //Función de prueba
        public ECPoint encrypt(BigInteger m, BigInteger random){
                BigInteger q2=key.getQ2();
                if (m.compareTo(q2) == 1) {
                        throw new IllegalArgumentException("El mensaje a
                        encriptar no puede ser mayor a: "+q2.toString());
                }

                return encrypt(m, random, key);
        }

}
```

9.1.1.2: Clase "BGNKeyGen.java"

```
import java.math.BigInteger;
import java.security.SecureRandom;
import java.util.LinkedList;
import java.util.Random;
import org.bouncycastle.math.ec.ECCurve;
import org.bouncycastle.math.ec.ECPoint;

public class BGNKeyGen {

	public static BGN_privateKey BGNKey(int s, long seed){
		if (s<=0) {
			throw new IllegalArgumentException(
			"Number of bits set is less than 0");
		}
		/*Valores para formar la llave*/
			BigInteger q1, q2, N, q;
			SecureRandom rnd;
			rnd = new
SecureRandom(BigInteger.valueOf(seed).toByteArray());
		/**Cálculo de las llaves q1,q2, N, q*/
			LinkedList<BigInteger> temp = new LinkedList();
			temp= CalcularQ_N_q1_q2(s, rnd);
			q = temp.get(0);  N= temp.get(1);
			q1 = temp.get(2); q2= temp.get(3);

		  ECCurve curve = new ECCurve.Fp(q, BigInteger.ONE,
		  BigInteger.ZERO);
		/*Calcular llaves g y auxiliar u*/
			ECPoint g =CalcularPunto(curve, q);
			ECPoint u =CalcularPunto(curve, q);
		/*Calcular llave h*/
			ECPoint h =u.multiply(q2).normalize();
		return new BGN_privateKey(N, q1, q2, q, g, h, curve, seed);
	}
```

```
public static LinkedList<BigInteger> CalcularQ_N_q1_q2(int s,Random rnd){
            BigInteger q=BigInteger.ZERO, N=BigInteger.ZERO,
            q1=BigInteger.ZERO, q2=BigInteger.ZERO;
                 boolean termine=false;
                 //Generar q1 y q2, N y q
                 while(termine==false){
                          q1 = getPrime(s/2, rnd);
                          q2 = getPrime(s/2, rnd);
                          N= q1.multiply(q2);
                          q= N.multiply(BigInteger.valueOf(4));
                          q= q.subtract(BigInteger.ONE);
                          if(esPrimo(q)==true){
                          if( q.mod(
                          (BigInteger.valueOf(4))).compareTo(BigInteger.valueOf(
                          3)) == 0){
                                            termine=true;
                                   }
                          }
                 }
                 LinkedList<BigInteger> bin = new LinkedList<BigInteger>();
                 bin.add(q);       bin.add(N);       bin.add(q1);
        bin.add(q2);
                 return bin;
        }

    public static boolean esPrimo(BigInteger numero){
        int contador = 2;
        boolean primo=true;
        while ((primo) && (numero.compareTo(BigInteger.valueOf(contador))!=0)){
         BigInteger  temp = numero.mod(BigInteger.valueOf(contador));
          if (temp.compareTo(BigInteger.ZERO) == 0)
            primo = false;
          contador++;
        }
        return primo;
      }

        public static ECPoint CalcularPunto(ECCurve curve, BigInteger field){
                 BigInteger X , Y;
                 Random rnd = new Random();
                 ECPoint temp;
                 do{
                          X = BigInteger.valueOf(rnd.nextLong()).mod(field);
                          Y= BigInteger.valueOf(rnd.nextLong()).mod(field);
                          temp = curve.createPoint(X, Y);
                 }while(temp.isValid()==false);
                 return temp;
        }

        public static BigInteger getPrime(int length, Random random) {
                 return BigInteger.probablePrime(length, random);
        }

}
```

9.1.1.3: Clase "BGN_key.java"

```
import java.io.Serializable;
import java.math.BigInteger;
import java.security.SecureRandom;
import java.util.Random;
import org.bouncycastle.math.ec.ECPoint;

public class BGN_key implements Serializable {
	private static final long serialVersionUID = 3769119843993218341L;
	protected Random rnd;
	protected BigInteger N, q2;
	protected ECPoint g, h;

	public BGN_key(){
		N= null; g= null; q2= null;	h= null;
	}

	public BGN_key(BigInteger N, long seed){
		this(N, new
SecureRandom(BigInteger.valueOf(seed).toByteArray()));
	}

	public BGN_key(BigInteger N, Random rnd){
		this.N= N;
		this.rnd = rnd;
	}

	public BGN_key(BigInteger N, BigInteger q2, ECPoint g,
       ECPoint h,long seed) {
		this(N, q2, g, h, new SecureRandom(
		  BigInteger.valueOf(seed).toByteArray()));
	}

	public BGN_key(BigInteger N, BigInteger q2, ECPoint g, ECPoint h,
       Random rnd) {
		this.N=N;	this.g=g;	this.h=h; this.q2 = q2;
this.rnd=rnd;
	}
//Getters y Setters:
	public BGN_key getPublicKey(){
		return new BGN_key(N,q2,g,h,rnd);
	}
	public BigInteger getN(){
		return N;
	}
	public BigInteger getQ2(){
		return q2;
	}

	public ECPoint getG(){
		return g;
	}
```

```
        public ECPoint getH(){
                return h;
        }
}
```

9.1.1.4: Clase "BGN_privateKey.java"

```
import java.math.BigInteger;
import java.util.Random;
import org.bouncycastle.math.ec.ECCurve;
import org.bouncycastle.math.ec.ECPoint;

public class BGN_privateKey extends BGN_key {
	private static final long serialVersionUID = 7588211371579735487L;
	protected Random rnd=null;
	protected BigInteger q1=null;
	protected BigInteger q=null;
	protected ECCurve curve;

	public BGN_privateKey(BigInteger N, BigInteger q1, BigInteger q2,
      BigInteger q, ECPoint g, ECPoint h, ECCurve curve, long seed) {
		super(N, q2,g,h, seed);
		this.q1= q1;
		this.q=q;
		this.curve =curve;
	}

	public BigInteger getQ1(){
		return q1;
	}
	public ECCurve getCurve(){
		return curve;
	}
	public BigInteger getQ(){
		return q;
	}
}
```

9.1.1.5: Clase "BGN_p.java"

```
import java.math.BigInteger;
import java.util.LinkedList;
import org.bouncycastle.math.ec.ECPoint;

public class BGN_p extends AbstractBGN{
	protected BGN_privateKey deckey=null;

	public BGN_p(){
	}

	public BGN_p(BGN_key key){
		this.key = key;
		this.encryptMode= true;
	}

	public BGN_p(BGN_privateKey key){
		this(key.getPublicKey());
		this.encryptMode=true;
	}
	public void setEncryption(BGN_key key){
		this.key=key;
		this.encryptMode=true;
		return;
	}
	public void setDecryption(BGN_privateKey key){
		this.deckey=key;
		this.decryptMode=true;
		return;
	}

	public void setDecryptEncrypt(BGN_privateKey key){
		setDecryption(key);
		setEncryption(key);
		return;
	}

	public BGN_privateKey getPrivateKey(){
		if(decryptMode){
			return deckey;
		}
		return null;
	}
```

```
//desencriptar:
        public BigInteger decrypt_BGN(ECPoint encrypted){
                BigInteger q1 = deckey.getQ1(), q2 = key.getQ2(),
                        m=BigInteger.valueOf(-1);
                ECPoint g = key.getG(), CpowQ1, G_Cap, g_capt;
                //C^q1
                CpowQ1 = encrypted.multiply(q1);
                G_Cap = g.multiply(q1);
                //Para determiner el valor del mensaje se debe
            //realizar una multiplicación con el exponente i
            // y comprobar si el resultado es igual a C^q1
                for(int i=1; i<q2.intValue(); i++){
                        g_capt = G_Cap.multiply(BigInteger.valueOf(i));
                        if(g_capt.equals(CpowQ1)){
                                m= BigInteger.valueOf(i);
                        }
                }
                return m;
        }
}
```

9.1.2: Okamoto-Uchiyama

9.1.2.1: Clase "AbstractOkamotoUchiyama.java"

```
import java.math.BigInteger;

public abstract class AbstractOkamotoUchiyama {
         protected  OkamotoUchiyama_key key=null;
         protected boolean encryptMode=false;
         protected boolean decryptMode=false;
         protected String notReadyForEncryption =
         "You must first call setEncrypt or setDecryptEncrypt before calling this
method";
         protected String notReadyForDecryption =
         "You must first call setDecrypt or setDecryptEncrypt before calling this
method";

       public OkamotoUchiyama_key getPublicKey(){
         if(key==null)
                  return null;
         return key.getPublicKey();
       }

       //Encriptación de Okamoto Uchiyama
       public  BigInteger encrypt_ou(BigInteger m){
         BigInteger random = key.getRandom();
         return encrypt_ou(m, random,key);
       }

       public static BigInteger encrypt_ou(BigInteger m, BigInteger r,
       OkamotoUchiyama_key key){
         return encrypt_ou(m, r, key.getH(),key.getG(),key.getN());
       }

       public static BigInteger encrypt_ou(BigInteger m, BigInteger r,
       BigInteger h, BigInteger g, BigInteger n){
         if(!(OkamotoUchiyama_key.inModNStar(r,n))) {
                  throw new IllegalArgumentException("r must be relatively prime to
       n and 0 <= r < n");
         }        //c= g^m h^r mod n
         BigInteger c=null, gModN= BigInteger.ONE, hModN= BigInteger.ZERO;
         gModN= g;
         BigInteger t1= gModN.modPow(m, n);
         hModN= h;
         hModN=hModN.modPow(r,n);
         c= (t1.multiply(hModN)).mod(n);
         return c;
       }
       //Suma de valores cifrados de Okamoto-Uchiyama
       public BigInteger addOU(BigInteger c1, BigInteger c2){
         if(encryptMode==false)
                  throw new IllegalStateException(this.notReadyForEncryption);
         return (c1.multiply(c2).mod(key.getN()));
       }
}
```

9.1.2.2: Clase "OUKeyGen.java"

```
import java.math.BigInteger;
import java.security.SecureRandom;
import java.util.Random;

public class OUKeyGen {
        public static OkamotoUchiyama_privateKey
        OkamotoUchiyamaKey(int s, long seed){
                if (s<=0) {
                throw new IllegalArgumentException(
                  "Number of bits set is less than 0");
                }
                /*valores llaves*/
                BigInteger n, g, h, p, q;
                SecureRandom rnd;
                rnd= new SecureRandom(BigInteger.valueOf(seed).toByteArray());
                p = getPrime(s, rnd);      q = getPrime(s, rnd);
                //cálculo de n
                        BigInteger p2= p.pow(2);
                        n= p2.multiply(q);
                //cálculo de g
                        g= primitiveRootModulo_Gen(n, rnd, p);
                  if (g.compareTo(BigInteger.ZERO)==0)
                        throw new IllegalArgumentException("\nEl valor del int
                        s ingresado ("+s+"), no permite que la llave g difiera
                        de 0.  Utilize un valor s<25 al llamar esta
                        función.");
                //cálculo de h= gn mod n
                        h= g.modPow(n,n);
                return new OkamotoUchiyama_privateKey(p, q, n, g, h, seed);
        }


        public static BigInteger primitiveRootModulo_Gen(BigInteger n, Random rnd,
BigInteger p){
                BigInteger g= null;
                g= BigInteger.ONE;
                BigInteger r= BigInteger.ZERO;
                for (int idx=1; idx<= n.intValue(); ++idx){
                        int rand = rnd.nextInt(100);
                        if(primitiveRoot(p, rand)==true){
                                r= r.add(BigInteger.valueOf(rand)); break;
                        }
                }
                g=g.multiply(r);
                return g;
        }
```

```
//a= valor aleatorio, p llave p
        public static boolean primitiveRoot(BigInteger p, int a ) {
            int i;
            BigInteger val = BigInteger.valueOf(a);
            for (i=2; i<p.intValue(); i++) {
                val = val.multiply(val).mod(p);
                if (val.equals(BigInteger.ONE))
                 break;
            }
           int test=BigInteger.valueOf(i).compareTo(p);
                 if(test==0)       return true;
            return false;
        }
        public static BigInteger getPrime(int length, Random random) {
                 return BigInteger.probablePrime(length, random);
        }

}
```

9.1.2.3: Clase "OkamotoUchiyama_key.java"

```
import java.io.Serializable;
import java.math.BigInteger;
import java.security.SecureRandom;
import java.util.Random;

public class OkamotoUchiyama_key implements Serializable{
	private static final long serialVersionUID = 7296569067937928113L;
	protected Random rnd;
	protected BigInteger n;
	protected BigInteger g;
	protected BigInteger h;
	protected int k=0;

	public OkamotoUchiyama_key(){
		  n=null;  g=null;		h=null;
	}
	public OkamotoUchiyama_key(BigInteger n,BigInteger g,BigInteger h, long
seed){
		this(n, g,h, new
SecureRandom(BigInteger.valueOf(seed).toByteArray()));
	}

	public OkamotoUchiyama_key(BigInteger n,BigInteger g,BigInteger h, Random
rnd){
		this.n=n;			this.h=h;			this.g=g;

		this.rnd=rnd;	k= n.bitLength();
	}
	public OkamotoUchiyama_key getPublicKey(){
		return new OkamotoUchiyama_key(n,g,h,rnd);
	}

	public BigInteger getG() {
		return g;
	}
	public BigInteger getH() {
		return h;
	}
	public BigInteger getN() {
		return n;
	}

			/** Metodos de Paillierkey.**/
	/*Metodo  de que si un valor a existe en Zn */
	public static boolean inModN(BigInteger a, BigInteger n) {
		return (a.compareTo(n) < 0 && a.compareTo(BigInteger.ZERO) >=
0);
	}

	/*Método de que si un valor a existe en Zn* */
	public static boolean inModNStar(BigInteger a, BigInteger n) {
		return (a.gcd(n).equals(BigInteger.ONE) && inModN(a, n));
	}
```

```
        /*Generador aleatorio de r existente en  Zn* */
        public BigInteger getRandomModNStar() {
                BigInteger r;
                do {
                        r = new BigInteger(k,rnd);
                } while (!inModNStar(r, getN()));
                return r;
        }

        public BigInteger getRandom(){
                BigInteger ret, key=getN();
                do{
                        int i=rnd.nextInt();
                        if (i<0) i=(i*-1);
                        ret= BigInteger.valueOf(i);
                }while(ret.compareTo(key)!=-1);
                return ret;
        }
}
```

9.1.2.4: Clase "OkamotoUchiyama_privateKey.java"

```
import java.math.BigInteger;
import java.util.Random;

public class OkamotoUchiyama_privateKey extends OkamotoUchiyama_key{
         private static final long serialVersionUID = -5099676862115044452L;
         protected BigInteger p;
         protected BigInteger q;
         protected Random rnd=null;
         protected int k=0;

       public OkamotoUchiyama_privateKey(BigInteger p,BigInteger q,BigInteger
       n,BigInteger g,BigInteger h, long seed){
                  super(n,g, h, seed);
                  this.p=p;
                  this.q=q;
         }

         public BigInteger getP() {
                  return p;
         }

         public BigInteger getQ() {
                  return q;
         }

}
```

9.1.2.5: Clase "OkamotoUchiyama_p.java"

```
import java.math.BigInteger;

public class OkamotoUchiyama_p extends AbstractOkamotoUchiyama {
        /** Maximum number of bits allowed for keysize. */
        protected static final int MAX_KEY_SIZE = 512;
        protected OkamotoUchiyama_privateKey deckey= null;

        public OkamotoUchiyama_p(){
      }

        public OkamotoUchiyama_p(OkamotoUchiyama_key key){
                this.key = key;
                this.encryptMode=true;
        }

        public OkamotoUchiyama_p(OkamotoUchiyama_privateKey key){
                this(key.getPublicKey());
                setDecryption(key);
        }

        public void setEncryption(OkamotoUchiyama_key key){
                this.key= key;
                this.encryptMode=true;
                return;
        }

        public void setDecryption(OkamotoUchiyama_privateKey key){
                this.deckey = key;
                this.decryptMode=true;
                return;
        }

        public void setDecryptEncrypt(OkamotoUchiyama_privateKey key){
                setDecryption(key);
                setEncryption(key);
                return;
        }

        public OkamotoUchiyama_privateKey getPrivateKey(){
                if(decryptMode){
                        return deckey;
                }
                return null;
        }
```

```
        public BigInteger decrypt_ou(BigInteger c){
                //System.out.println("Desencriptacion de "+ c.toString());
                BigInteger m= null;
                if(decryptMode==false)
                  throw new IllegalStateException(this.notReadyForDecryption);
                //Obtener las claves
                        BigInteger p= deckey.getP();
                        BigInteger g= super.key.getG();
                //p-1
                        BigInteger pRest1= p;
                        pRest1= pRest1.subtract(BigInteger.ONE);
                //p^2
                        BigInteger p2=p;
                        p2= p2.pow(2);
                //formacion de argumentos funcion L
                        BigInteger c1= c.modPow(pRest1, p2);
                        BigInteger c2= g.modPow(pRest1, p2);
                //Resultados funcion L.
                        BigInteger l1 =
(c1.subtract(BigInteger.ONE)).divide(p);
                        BigInteger l2 =
(c2.subtract(BigInteger.ONE)).divide(p);
                //Aplicación de modulo inverso a L(g^(p-1) mod p^2)
                        BigInteger l2_i= l2.modInverse(p);
                //Obtención de texto plano m
                        m= (l2_i.multiply(l1)).mod(p);
                return m;
        }


}
```

9.1.3: ElGammal

9.1.3.1: Clase "AbstractElGammal.java"

```
import java.math.BigInteger;
import java.util.LinkedList;

public abstract class AbstractElGammal {
         protected ElGammal_key key= null;
         protected boolean encryptMode=false;
         protected boolean decryptMode=false;
         protected String notReadyForEncryption =
         "You must first call setEncrypt or setDecryptEncrypt before calling this
method";
         protected String notReadyForDecryption =
         "You must first call setDecrypt or setDecryptEncrypt before calling this
method";

         public ElGammal_key getPublicKey(){
                  if(key==null){
                           return null;
                  }
                  return key.getPublicKey();
         }

public LinkedList<BigInteger> encrypt_EG(BigInteger m){
                  BigInteger random = key.getRandom();
                  return encrypt_EG(m, random, key);
         }

       public LinkedList<BigInteger> encrypt_EG(BigInteger m, BigInteger r,
       ElGammal_key key){
                  return encrypt_EG(m, r, key.getP(), key.getG(), key.getY());
         }

         public LinkedList<BigInteger> encrypt_EG(BigInteger m,BigInteger r,
       BigInteger p, BigInteger g, BigInteger y){
                  LinkedList<BigInteger> a= new LinkedList<BigInteger>();
                  BigInteger C1, C2;
                           C1= g.modPow(r, p); //Valor C1
                           C2 = m.multiply(y.modPow(r, p).mod(p)); //Valor C2
                  a.add(0, C1);
                  a.add(1, C2);
                  return  a;
         }

       public LinkedList<BigInteger> mul_EG(LinkedList<BigInteger>
       c1,LinkedList<BigInteger> c2){
                  LinkedList<BigInteger> c = new LinkedList();
                  c.add(0, c1.get(0).multiply(c2.get(0)));
                  c.add(1, c1.get(1).multiply(c2.get(1)));
                  return c;
         }
}
```

9.1.3.2: Clase "EGKeyGen.java"

```
import java.math.BigInteger;
import java.security.SecureRandom;
import java.util.Random;

public class EGKeyGen {
        public static ElGammal_privateKey ElGammalKey(int s, long seed){
                if (s<=0) {
                        throw new IllegalArgumentException("Number of bits"
                        + "set is less than 0");
                }
                /*valores llaves*/
                        BigInteger p=null, g=null, y=null, x=null;
                        SecureRandom rnd;
                        rnd = new
                        SecureRandom(BigInteger.valueOf(seed).toByteArray());
                //cálculos de llave privada (x)
                        long a = rnd.nextLong();
                        if(a<0)
                                a= a*(-1);
                        x= BigInteger.valueOf(a);
                //cálculo de llaves públicas (p,g,y)
                        p= getPrime(s,rnd);
                        g= generarG(p,rnd);
                        y= g.modPow(x,p);
                return new ElGammal_privateKey(p, g, y, x, seed);
        }

        public static BigInteger generarG(BigInteger p, Random random){
                BigInteger g=null;
                long t;
                do{
                        t= random.nextLong();
                        if(t<0){
                                t= t*(-1);
                        }
                }while(t > p.longValue()-1 || t<2);

                g= BigInteger.valueOf(t);
                return g;
        }

        public static BigInteger getPrime(int length, Random random) {
                return BigInteger.probablePrime(length, random);
        }
}
```

9.1.3.3: Clase "ElGammal_key.java"

```
import java.io.Serializable;
import java.math.BigInteger;
import java.security.SecureRandom;
import java.util.Random;

public class ElGammal_key implements Serializable {
	private static final long serialVersionUID = -754829726309719224L;
	protected Random rnd;
	protected BigInteger p;
	protected BigInteger g;	protected BigInteger y;
	protected int k=0;

	ElGammal_key(){
		p=null; g=null; y=null;
	}

	public ElGammal_key(BigInteger p,BigInteger g,BigInteger y, long seed){
		this(p, g,y,
		  new SecureRandom(BigInteger.valueOf(seed).toByteArray()));
	}

	public ElGammal_key(BigInteger p,BigInteger g,BigInteger y, Random
rnd){
		this.p=p;			this.y=y;
		this.g=g;			this.rnd=rnd;	k= p.bitLength();
	}

	public ElGammal_key getPublicKey(){
		return new ElGammal_key(p,g,y,rnd);
	}

	public BigInteger getG(){
		return g;
	}

	public BigInteger getP(){
		return p;
	}

	public BigInteger getY(){
		return y;
	}

	public BigInteger getRandom(){
		BigInteger ret, key=getP();
			do{
				int i=rnd.nextInt();
				if (i<0) i=(i*-1);
				ret= BigInteger.valueOf(i);
			}while(ret.compareTo(key)!=-1);
		return ret;
	}
}
```

9.1.3.4: Clase "ElGammal_privateKey.java"

```
import java.math.BigInteger;
import java.util.Random;

public class ElGammal_privateKey extends ElGammal_key {
	private static final long serialVersionUID = -5015127893793267112L;
	protected BigInteger x;
	protected Random rnd=null;

	public ElGammal_privateKey(BigInteger p, BigInteger g, BigInteger y,
BigInteger x, long seed){
		super(p,g,y,seed);
		this.x= x;
	}

	public BigInteger getX(){
		return x;
	}
}
```

9.1.3.5: Clase "ElGammal_p.java"

```
import java.math.BigInteger;
import java.util.LinkedList;

public class ElGammal_p extends AbstractElGammal{
	protected static final int MAX_KEY_SIZE = 512;
	protected ElGammal_privateKey deckey= null;

	public ElGammal_p(){
	}

	public ElGammal_p(ElGammal_key key){
		this.key = key;
		this.encryptMode=true;
	}

	public ElGammal_p(ElGammal_privateKey key){
		this(key.getPublicKey());
		this.encryptMode=true;
	}

	public void setEncryption(ElGammal_key key){
		this.key=key;			this.encryptMode=true;
		return;
	}
	public void setDecryption(ElGammal_privateKey key){
		this.deckey=key;
		this.decryptMode=true;
		return;
	}

	public void setDecryptEncrypt(ElGammal_privateKey key){
		setDecryption(key);
		setEncryption(key);
		return;
	}

	public ElGammal_privateKey getPrivateKey(){
		if(decryptMode){
			return deckey;
		}
		return null;
	}
	public BigInteger decrypt_EG(LinkedList<BigInteger> c){
		BigInteger C1, C2, d=null, x= deckey.getX(), p= deckey.getP();
		C1= c.get(0);
		C2= c.get(1);
		BigInteger temp1 = C1.modPow(x,p);
		BigInteger ad= temp1.modInverse(p);
		d= ad.multiply(C2).mod(p);
		return d;
	}
}
```

Sección 9.2: Código Prototipo Android

9.2.1: Código de Actividades (.java)

9.2.1.1: MainActivity.java

Clase principal que implementa la funcionalidad de la ventana de la aplicación en el cual, además de desarrollarse la funciones de las pruebas de Suma Homomórfica & Desencriptación, se le permite el acceso a la simulación de una votación y a la visualización del resultado de la simulación de una votación con una determinada cantidad de votantes.

```
package com.example.nyliam.tesistest;

import android.app.ActivityManager;
import android.content.Context;
import android.content.Intent;
import android.content.IntentFilter;
import android.os.BatteryManager;
import android.os.Environment;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.util.Log;
import android.view.View;
import android.widget.Button;
import android.widget.Toast;

import java.io.File;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.OutputStreamWriter;
import java.io.RandomAccessFile;
import java.math.BigInteger;
import java.util.ArrayList;
import java.util.Random;

import okamotoUchiyama.OUKeyGen;
import okamotoUchiyama.OkamotoUchiyama_p;
import okamotoUchiyama.OkamotoUchiyama_privateKey;
import paillierp.Paillier;
import paillierp.key.KeyGen;
import paillierp.key.PaillierPrivateKey;
```

```
public class MainActivity extends AppCompatActivity {

    Button boton1;
    Button boton2;
    int valO1;
    int valO2;
    int numVotacion=0;


    //Acceso menú de prueba
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        boton1 = findViewById(R.id.btn1);
        boton2 = findViewById(R.id.btn2);

        //Acceso prueba de Encriptación
        boton1.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
               Intent intent =
               new Intent( MainActivity.this, PruebaEncriptacion.class);
                intent.putExtra("numero",numVotacion);
                startActivity(intent);
            }
        });

        //Acceso prueba de Simulación de Homomorfismo y Desencriptación
        boton2.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
             Intent intent = new Intent( MainActivity.this,
                       PruebaHomomorfismoDesencriptacion.class);

             simularVotacion(1000);[11]
             intent.putExtra("numero",numVotacion);
             intent.putExtra("TotalVoto",1000);
             intent.putExtra("TotO1",valO1);
             intent.putExtra("TotO2",valO2);
             startActivity(intent);
            }
        });

    }

    /** MÉTODOS DE PRUEBAS **/

}
```

[11] Nombre referencia para el método de la prueba de Homomorfismo y Desencriptación.

9.2.1.2: PruebaEncriptacion.java

Clase que implementa la funcionalidad de la ventana de la aplicación que permite simular el proceso de una votación de dos opciones “Si” y “No” de la aplicación la cual permitirá el acceso a una ventana de confirmación de la elección realizada.

```
package com.example.nyliam.tesistest;


import android.content.Intent;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.TextView;

public class PruebaEncriptacion extends AppCompatActivity {

    TextView tv, tv2, tv3;
    Button botonSi;
    Button botonNo;
    int num;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_prueba_encriptacion);

        tv = findViewById(R.id.NroVot);
        Bundle bundle = getIntent().getExtras();
        if(bundle != null){
            num = bundle.getInt("numero");
            tv.setText("Votación nro "+num);
        }

        botonSi = findViewById(R.id.BtonOp1);
        botonSi.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                Intent intent = new Intent( PruebaEncriptacion.this,
PruebaEncriptacionConfirmacion.class);
                intent.putExtra("Opcion",1);
                intent.putExtra("numero",num);
                tv2 = findViewById(R.id.Op1);
                intent.putExtra("ValorOp", tv2.getText());
                //Envío valor de la opcion a la siguiente ventana
                startActivity(intent);
            }
        });
```

```
        botonNo = findViewById(R.id.BtonOp2);
        botonNo.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                Intent intent = new Intent( PruebaEncriptacion.this,
PruebaEncriptacionConfirmacion.class);
                intent.putExtra("Opcion",2);
                tv3= findViewById(R.id.Op2);
                intent.putExtra("ValorOp", tv3.getText());
                //Envío valor de la opcion a la siguiente ventana
                startActivity(intent);
            }
        });
    }
}
```

9.2.1.3: PruebaEncriptacionConfirmacion.java

Clase que implementa la funcionalidad de la ventana de la aplicación que permite simular la confirmación de la opción seleccionada en la ventana anterior, además de desarrollarse la funciones de las pruebas de Encriptación de la Votación.

```
package com.example.nyliam.tesistest;

import android.app.ActivityManager;
import android.content.Context;
import android.content.Intent;
import android.content.IntentFilter;
import android.os.BatteryManager;
import android.os.Bundle;
import android.os.Environment;
import android.support.v7.app.AppCompatActivity;
import android.util.Log;
import android.view.View;
import android.widget.Button;
import android.widget.TextView;
import android.widget.Toast;

import java.io.File;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.OutputStreamWriter;
import java.io.RandomAccessFile;
import java.math.BigInteger;
import java.util.ArrayList;

import okamotoUchiyama.OUKeyGen;
import okamotoUchiyama.OkamotoUchiyama_p;
import okamotoUchiyama.OkamotoUchiyama_privateKey;
import paillierp.Paillier;
import paillierp.key.KeyGen;
import paillierp.key.PaillierPrivateKey;
```

```
public class PruebaEncriptacionConfirmacion extends AppCompatActivity {
    Button botonConfirm;
    TextView tv, tvV;
    int op=0;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(
                R.layout.activity_prueba_encriptacion_confirmacion);

        Bundle bundle = getIntent().getExtras();
        if(bundle != null){
            tvV = findViewById(R.id.NroVot);
            int num = bundle.getInt("numero");
            tvV.setText("Votación nro "+num);

            op = bundle.getInt("Opcion");
            String option = bundle.getString("ValorOp");
            tv = findViewById(R.id.Confirmas);
            tv.setText("Actualmente usted a seleccionado como elección
               de sufragio la opcion '"+option+"': Confirme aquella
           declaración para continuar.");
            }

            botonConfirm = findViewById(R.id.Confirm);
            botonConfirm.setOnClickListener(new View.OnClickListener() {
                @Override
                public void onClick(View view) {
                //Prueba encriptación
                  Encrypt_(op);[12]
                Intent intent = new Intent(
                   PruebaEncriptacionConfirmacion.this, MainActivity.class);
                 //Mensaje emergente
                    Toast.makeText(PruebaEncriptacionConfirmacion.this,
                           "Voto enviado satisfactoriamente",
                           Toast.LENGTH_SHORT).show();

                // Volver a la ventana inicial
                    startActivity(intent);
                }
            });
    }
    /** MÉTODOS DE PRUEBAS **/
}
```

[12] Nombre referencia para el método de la prueba de Encriptación.

9.2.1.4: PruebaHomomorfismoDesencriptacion.java

Clase que implementa la funcionalidad de la ventana de la aplicación que permite visualizar el gráfico con el resultado de la votación simulada en la Clase "`MainActivity`", para lo cual hay que ingresar una librería externa:

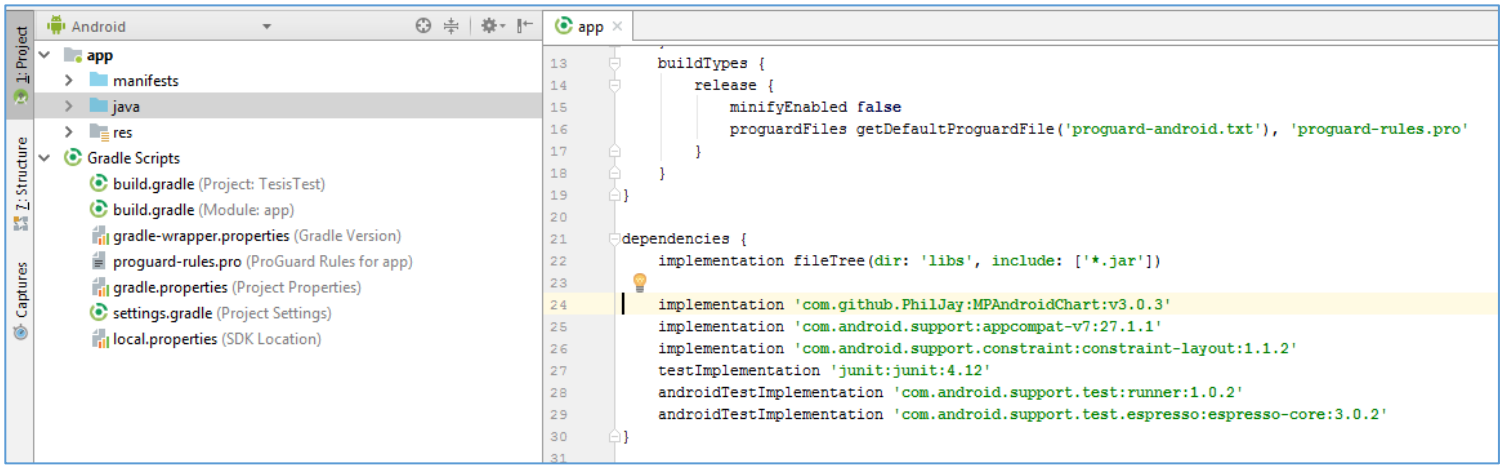


Ilustración 30: *Ingreso de Librería externa en Android, fuente propia.*

```
package com.example.nyliam.tesistest;

import android.graphics.Color;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.widget.TextView;

import com.github.mikephil.charting.charts.PieChart;
import com.github.mikephil.charting.components.Legend;
import com.github.mikephil.charting.data.PieData;
import com.github.mikephil.charting.data.PieDataSet;
import com.github.mikephil.charting.data.PieEntry;

import java.util.ArrayList;
```

```
public class PruebaHomomorfismoDesencriptacion extends AppCompatActivity{
    PieChart pieChart;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(
               R.layout.activity_prueba_homomorfismo_desencriptacion);
        pieChart = findViewById(R.id.chart);
        Grafico(2, 1, 1);
        TextView tv = findViewById(R.id.NroVot);
   /Recepcion valores gráfico
        Bundle bundle = getIntent().getExtras();
        if (bundle != null) {
            int num = bundle.getInt("numero");
            tv.setText("Votación nro "+num);
   //Recepción de los valores generados
             int TotalVotos = bundle.getInt("TotalVoto");
             int VotosOp1 = bundle.getInt("TotO1");
             int VotosOp2 = bundle.getInt("TotO2");
             if (VotosOp1 + VotosOp2 == TotalVotos) {
                 TextView  tvTV = findViewById(R.id.TotVota);
                 tvTV.setText("Total de Votantes: "+TotalVotos);
                 Grafico(TotalVotos, VotosOp1, VotosOp2);
             }
        }
    }

    public void Grafico(int TotalVotos, int VotosOp1, int VotosOp2) {
        ArrayList<PieEntry> entrys = new ArrayList<>();
        entrys.add(new PieEntry(
               (float) VotosOp1 *100 / TotalVotos, "Si"));
        entrys.add(new PieEntry(
               (float)VotosOp2 *100 / TotalVotos, "No"));

        ArrayList<Integer> colors = new ArrayList<>();
        colors.add(Color.RED);
        colors.add(Color.CYAN);
        PieDataSet dataSetChart = new PieDataSet(entrys, "Grafico");
        dataSetChart.setColors(colors);
        PieData dataChart = new PieData(dataSetChart);

        Legend legend = pieChart.getLegend();
        legend.setEnabled(false);
        TextView Option1Name = findViewById(R.id.OptionName1);
        TextView Option1Votos = findViewById(R.id.OptionValue1);
        TextView Option2Name = findViewById(R.id.OptionName2);
        TextView Option2Votos = findViewById(R.id.OptionValue2);
           Option1Name.setText("■ Si");
           Option1Name.setTextColor(Color.RED);
           Option2Name.setText("■ No");
           Option2Name.setTextColor(Color.CYAN);
           Option1Votos.setText("" + VotosOp1);
           Option2Votos.setText("" + VotosOp2);
        pieChart.setData(dataChart);
    }
}
```

9.2.2: Código de Layout (.xml)

9.2.2.1: activity_main.xml

Código de la ventana principal de la aplicación, en la cual se permitirá el acceso tanto a la ventana de la "Prueba de Encriptación" y a la ventana con el resultado de la "Prueba de Suma Homomórfica & Desencriptación", en ambos casos con el uso de botones.

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">


    <TextView
        android:id="@+id/textView3"
        android:gravity="center"
        android:layout_width="220dp"
        android:layout_height="39dp"
        android:layout_marginEnd="8dp"
        android:layout_marginLeft="8dp"
        android:layout_marginRight="8dp"
        android:layout_marginStart="8dp"
        android:layout_marginTop="88dp"
        android:text="Prototipo de prueba propiedades Sistema E-Voting"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

    <Button
        android:id="@+id/btn1"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginEnd="24dp"
        android:layout_marginLeft="24dp"
        android:layout_marginRight="24dp"
        android:layout_marginStart="24dp"
        android:layout_marginTop="68dp"
        android:text="Prueba de Encriptación"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/textView3" />
```

```
    <Button
        android:id="@+id/btn2"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginBottom="8dp"
        android:layout_marginEnd="24dp"
        android:layout_marginLeft="24dp"
        android:layout_marginRight="24dp"
        android:layout_marginStart="24dp"
        android:layout_marginTop="8dp"
        android:text="Prueba de Suma Homomórfica y Desencriptación"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/btn1"
        app:layout_constraintVertical_bias="0.349" />
</android.support.constraint.ConstraintLayout>
```

9.2.2.2: activity_ prueba_encriptacion.xml

Código de la ventana con la simulación de una votación, en donde se visualiza el número de la votación, el tópico de la votación, las opciones (“Si” y “No”).

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".PruebaEncriptacion">

    <TextView
        android:id="@+id/NroVot"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginTop="112dp"
        android:text="Votación nro"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />
    <TextView
        android:id="@+id/TopicVota"
        android:layout_width="wrap_content"
        android:layout_height="75dp"
        android:layout_marginTop="8dp"
        android:gravity="center"
        android:text="¿Considera necesario el utilizar Android?"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/NroVot" />

    <TextView
        android:id="@+id/Op1"
        android:layout_width="223dp"
        android:layout_height="48dp"
        android:layout_marginBottom="8dp"
        android:layout_marginEnd="8dp"
        android:layout_marginLeft="28dp"
        android:layout_marginRight="8dp"
        android:layout_marginStart="28dp"
        android:layout_marginTop="8dp"
        android:gravity="center"
        android:text="Si"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toStartOf="@+id/BtonOp2"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/TopicVota"
        app:layout_constraintVertical_bias="0.154" />
```

```
    <TextView
        android:id="@+id/Op2"
        android:layout_width="223dp"
        android:layout_height="50dp"
        android:layout_marginBottom="8dp"
        android:layout_marginEnd="8dp"
        android:layout_marginLeft="28dp"
        android:layout_marginRight="8dp"
        android:layout_marginStart="28dp"
        android:layout_marginTop="32dp"
        android:gravity="center"
        android:text="No"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toStartOf="@+id/BtonOp2"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/Op1"
        app:layout_constraintVertical_bias="0.155" />

    <Button
        android:id="@+id/BtonOp1"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_marginEnd="8dp"
        android:layout_marginLeft="8dp"
        android:layout_marginRight="8dp"
        android:layout_marginStart="8dp"
        android:layout_marginTop="44dp"
        android:text="Votar"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.0"
        app:layout_constraintStart_toEndOf="@+id/Op1"
        app:layout_constraintTop_toBottomOf="@+id/TopicVota"
        tools:ignore="MissingConstraints" />

    <Button
        android:id="@+id/BtonOp2"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_marginBottom="8dp"
        android:layout_marginEnd="8dp"
        android:layout_marginLeft="8dp"
        android:layout_marginRight="8dp"
        android:layout_marginStart="8dp"
        android:layout_marginTop="44dp"
        android:text="Votar"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.0"
        app:layout_constraintStart_toEndOf="@+id/Op2"
        app:layout_constraintTop_toBottomOf="@+id/BtonOp1"
        app:layout_constraintVertical_bias="0.056"
        tools:ignore="MissingConstraints" />

</android.support.constraint.ConstraintLayout>
```

9.2.2.3: activity_prueba_encriptacion_confirmacion.xml

Código de la ventana con la visualización de la ventana de confirmación del voto seleccionado.

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".PruebaEncriptacionConfirmacion">

    <TextView
        android:id="@+id/NroVot"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginTop="50dp"
        android:text="Votación nro"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

    <TextView
        android:id="@+id/TopicVota"
        android:layout_width="wrap_content"
        android:layout_height="75dp"
        android:layout_marginTop="8dp"
        android:gravity="center"
        android:text="¿Considera necesario el utilizar Android?"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/NroVot" />

    <TextView
        android:id="@+id/Confirmas"
        android:layout_width="293dp"
        android:layout_height="163dp"
        android:layout_marginEnd="8dp"
        android:layout_marginLeft="8dp"
        android:layout_marginRight="8dp"
        android:layout_marginStart="8dp"
        android:layout_marginTop="24dp"
        android:gravity="center"
        android:text="Actualmente usted a seleccionado como elección de
sufragio la opcion [Valor]: Confirme aquella declaración para continuar."
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.506"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/TopicVota" />
```

```
    <Button
        android:id="@+id/Confirm"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginBottom="54dp"
        android:layout_marginEnd="148dp"
        android:layout_marginLeft="148dp"
        android:layout_marginRight="148dp"
        android:layout_marginStart="148dp"
        android:layout_marginTop="75dp"
        android:text="Confirmar"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/Confirmas" />

</android.support.constraint.ConstraintLayout>
```

9.2.2.4: activity_prueba_homomorfismo_desencriptacion.xml

Código de la ventana con el resultado de la votación simulada, tanto en un gráfico de torta como en una tabla con la cantidad de votos correspondientes a la elección.

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".PruebaHomomorfismoDesencriptacion">

    <TextView
        android:id="@+id/NroVot"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginTop="24dp"
        android:text="Votación nro"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

    <TextView
        android:id="@+id/TopicVota"
        android:layout_width="wrap_content"
        android:layout_height="71dp"
        android:layout_marginTop="8dp"
        android:gravity="center"
        android:text="¿Considera necesario el utilizar Android?"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/NroVot" />

    <TextView
        android:id="@+id/Result"
        android:layout_width="84dp"
        android:layout_height="19dp"
        android:layout_marginEnd="8dp"
        android:layout_marginLeft="4dp"
        android:layout_marginRight="8dp"
        android:layout_marginStart="4dp"
        android:layout_marginTop="16dp"
        android:gravity="center"
        android:text="Resultado:"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.097"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/TopicVota" />
```

```
<com.github.mikephil.charting.charts.PieChart
     android:id="@+id/chart"
     android:layout_width="243dp"
     android:layout_height="190dp"
     android:layout_marginEnd="8dp"
     android:layout_marginLeft="8dp"
     android:layout_marginRight="8dp"
     android:layout_marginStart="8dp"
     android:layout_marginTop="8dp"
     app:layout_constraintEnd_toEndOf="parent"
     app:layout_constraintStart_toStartOf="parent"
     app:layout_constraintTop_toBottomOf="@+id/TotVota">

 </com.github.mikephil.charting.charts.PieChart>

 <LinearLayout
     android:id="@+id/llinearLayoutContainer"
     android:layout_width="267dp"
     android:layout_height="85dp"
     android:layout_marginBottom="16dp"
     android:layout_marginEnd="8dp"
     android:layout_marginLeft="8dp"
     android:layout_marginRight="8dp"
     android:layout_marginStart="8dp"
     android:layout_marginTop="16dp"
     android:gravity="center"
     android:orientation="vertical"
     app:layout_constraintBottom_toBottomOf="parent"
     app:layout_constraintEnd_toEndOf="parent"
     app:layout_constraintStart_toStartOf="parent"
     app:layout_constraintTop_toBottomOf="@+id/chart">

     <LinearLayout
         android:layout_width="match_parent"
         android:layout_height="wrap_content"
         android:layout_weight="1"
         android:gravity="center"
         android:orientation="horizontal">

         <TextView
             android:id="@+id/header1"
             android:layout_width="match_parent"
             android:layout_height="wrap_content"
             android:layout_weight="1"
             android:gravity="center"
             android:text="Opción" />

         <TextView
             android:id="@+id/header2"
             android:layout_width="match_parent"
             android:layout_height="wrap_content"
             android:layout_weight="1"
             android:gravity="center"
             android:text="Nro Votos" />
     </LinearLayout>
```

```
    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:gravity="center"
        android:orientation="horizontal">

        <TextView
            android:id="@+id/OptionName1"
            android:layout_width="wrap_content"
            android:layout_height="match_parent"
            android:layout_weight="1"
            android:gravity="center"
            android:text="TextView" />

        <TextView
            android:id="@+id/OptionValue1"
            android:layout_width="wrap_content"
            android:layout_height="match_parent"
            android:layout_weight="1"
            android:gravity="center"
            android:text="TextView" />

    </LinearLayout>

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:gravity="center"
        android:orientation="horizontal">

        <TextView
            android:id="@+id/OptionName2"
            android:layout_width="wrap_content"
            android:layout_height="match_parent"
            android:layout_weight="1"
            android:gravity="center"
            android:text="TextView" />

        <TextView
            android:id="@+id/OptionValue2"
            android:layout_width="wrap_content"
            android:layout_height="match_parent"
            android:layout_weight="1"
            android:gravity="center"
            android:text="TextView" />

    </LinearLayout>
</LinearLayout>
```

```
    <TextView
        android:id="@+id/TotVota"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginEnd="8dp"
        android:layout_marginLeft="8dp"
        android:layout_marginRight="8dp"
        android:layout_marginStart="8dp"
        android:layout_marginTop="4dp"
        android:text="Total de Votantes"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.23"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/Result" />

</android.support.constraint.ConstraintLayout>
```

Sección 9.3: Código aplicado a las Pruebas

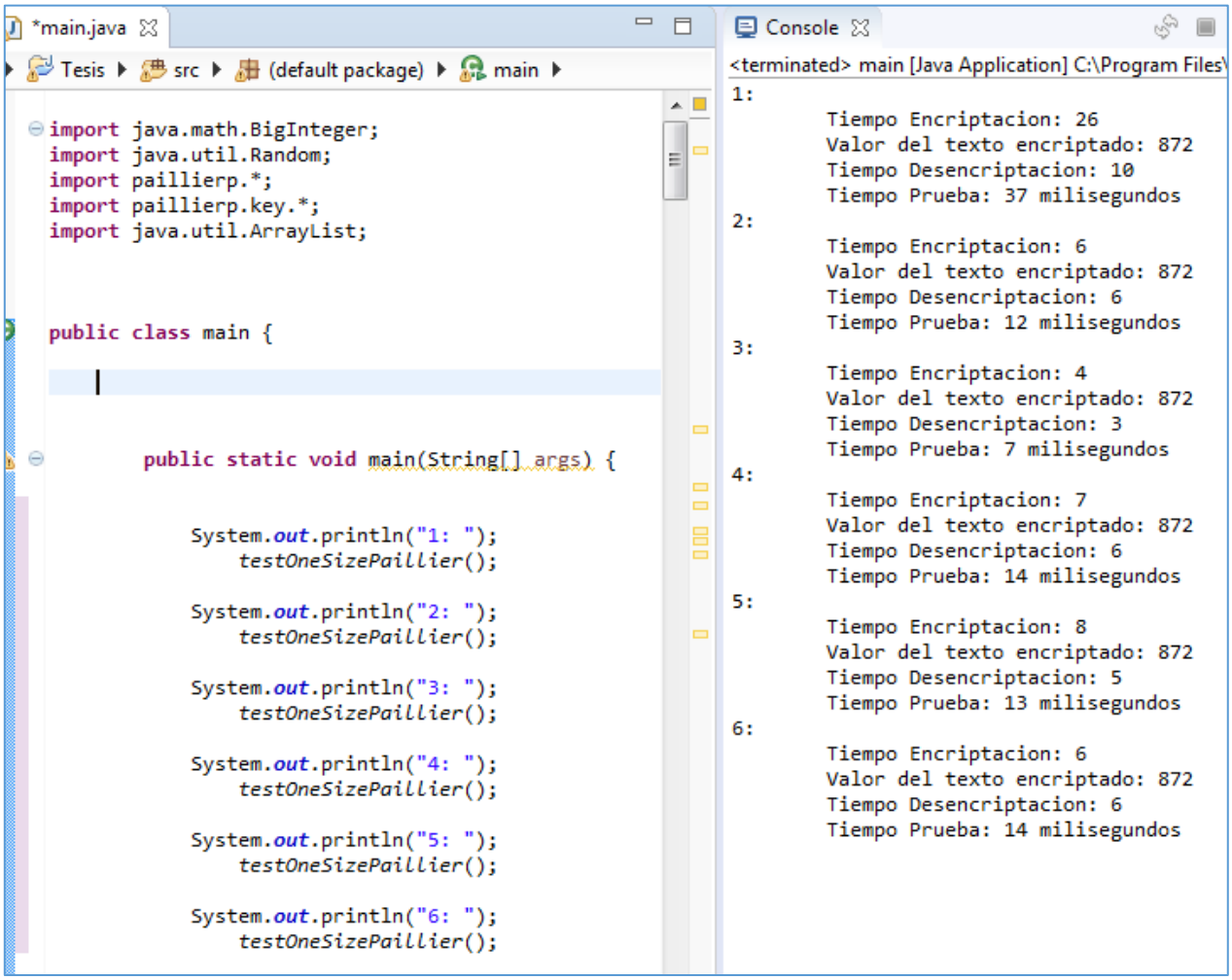
9.3.1: Pruebas de Criptosistemas.

9.3.1.1: Pruebas de Encriptación & Desencriptación a un valor 1 en primera instancia

9.3.1.1.1: Paillier

```
public static void testOneSizePaillier(){
        /** Inicialización de Paillier **/
                Paillier esystem= new Paillier();
                PaillierPrivateKey key=KeyGen.PaillierKey(218,122333356);
                esystem.setDecryptEncrypt(key);
                long startT = System.currentTimeMillis();
        /** Inicio calculo de tiempo de Encriptación **/
                long start = System.currentTimeMillis();
                        BigInteger m=null, c=null;
                        m=BigInteger.valueOf(1);
                        c=esystem.encrypt(m);
                long stop = System.currentTimeMillis();
                System.out.println("\tTiempo Encriptacion: "+ ((stop-start)));
        /** Valor del Texto Encriptado **/
                System.out.println("\tValor del texto encriptado: "+c.bitLength());
        /** Inicio calculo de tiempo de Desencriptación**/
                start = System.currentTimeMillis();
                        BigInteger decryption=null;
                        decryption=esystem.decrypt(c);
                stop = System.currentTimeMillis();
                long stopT = System.currentTimeMillis();
        System.out.println("\tTiempo Desencriptacion: "+ ((stop-start)) );
        System.out.println("\tTiempo Prueba: "+ (stopT-startT) + " milisegundos");
}
```

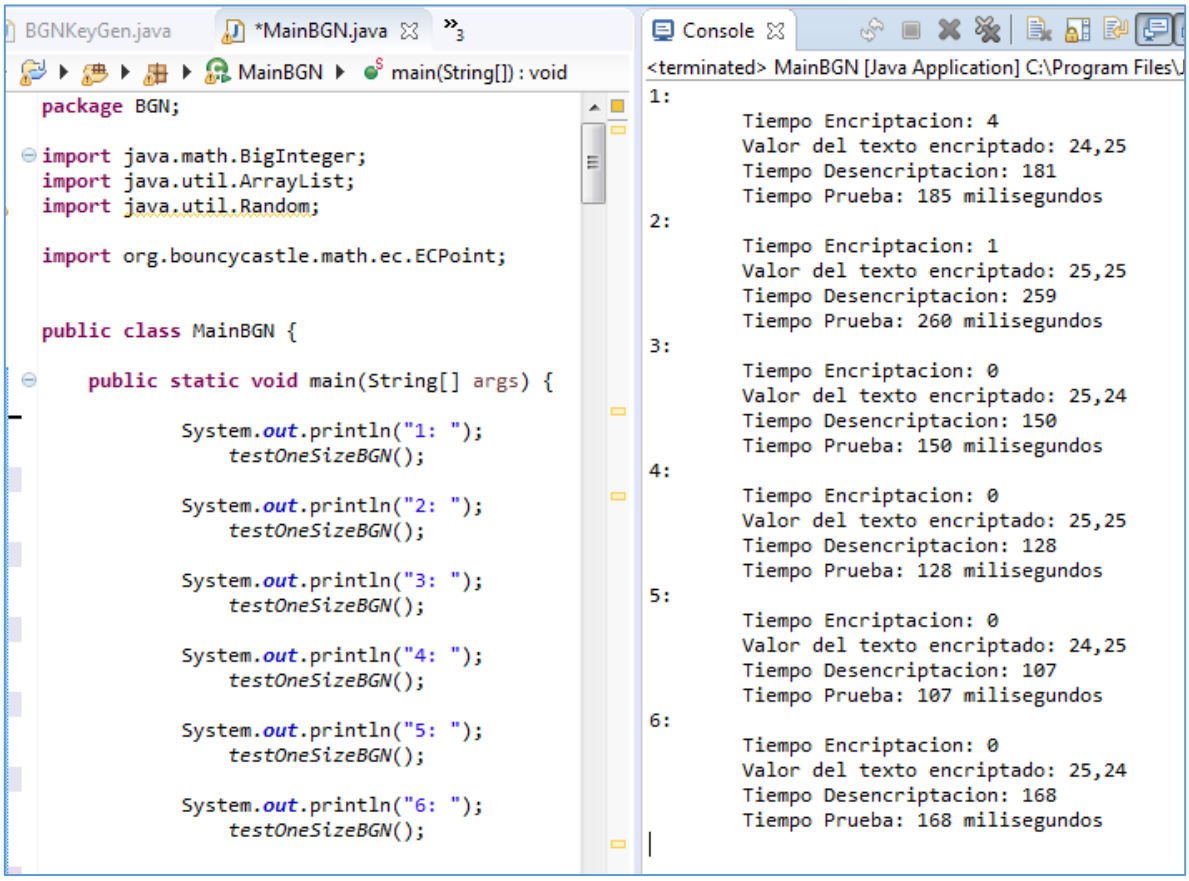
Imagen de Consola:



9.3.1.1.2: Boneh-Goh-Nissim

```
public static void testOneSizeBGN(){
        /** Inicialización de BGN **/
                BGN_p esystem= new BGN_p();
                BGN_privateKey key=BGNKeyGen.BGNKey(24,25165);
                esystem.setDecryptEncrypt(key);
                long startT = System.currentTimeMillis();
        /** Inicio calculo de tiempo de Encriptación **/
                long start = System.currentTimeMillis();
                        BigInteger m=null;
                        ECPoint c;
                        m=BigInteger.valueOf(1);
                        c=esystem.encrypt(m);
                long stop = System.currentTimeMillis();
                System.out.println("\tTiempo Encriptacion: "+ ((stop-start)));
        /** Valor del Texto Encriptado **/
                System.out.println("\tValor del texto encriptado: "
                  +c.getXCoord().toBigInteger().bitLength()+
                  ","+c.getYCoord().toBigInteger().bitLength());
        /** Inicio calculo de tiempo de Desencriptación**/
                start = System.currentTimeMillis();
                        BigInteger decryption=null;
                        decryption=esystem.decrypt_BGN(c);
                stop = System.currentTimeMillis();
                long stopT = System.currentTimeMillis();
                System.out.println("\tTiempo Desencriptacion: "+ ((stop-
start)));
                System.out.println("\tTiempo Prueba: "+(stopT-startT)+"
milisegundos");
}
```

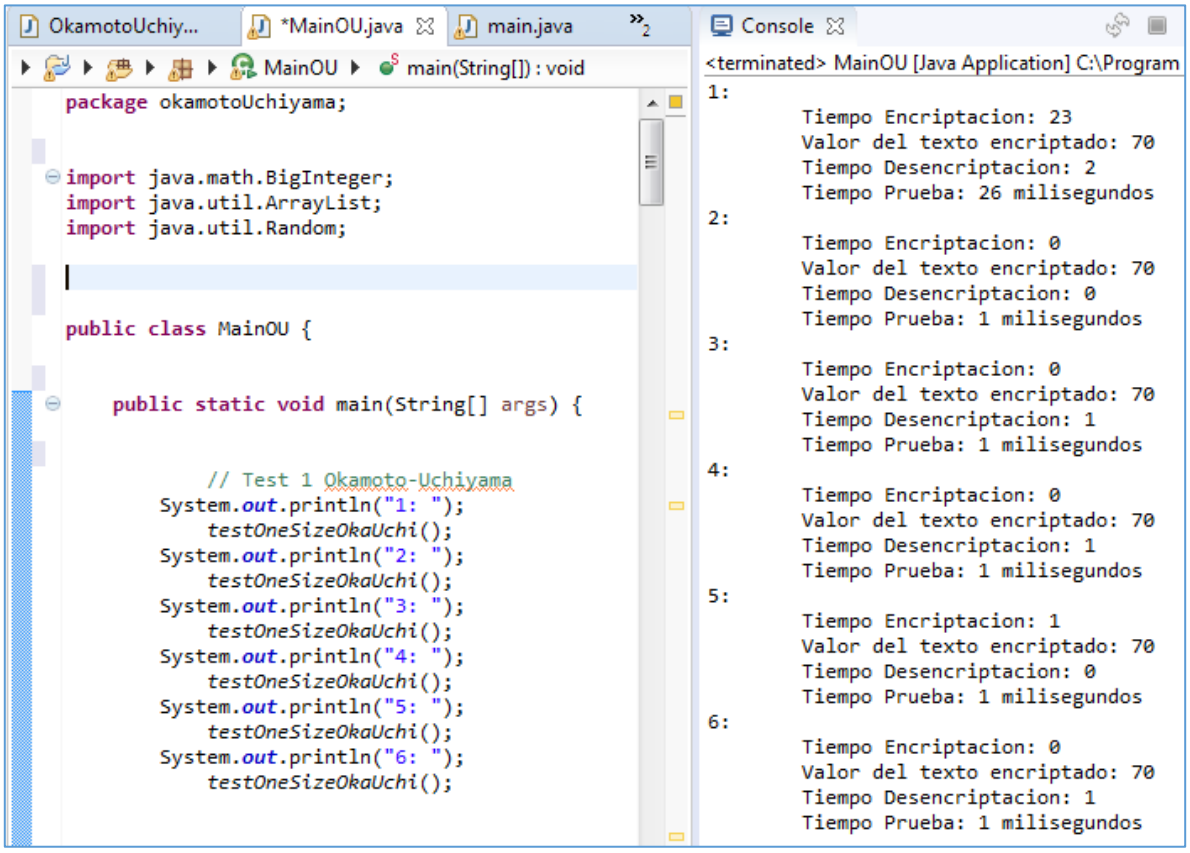
Imagen de Consola:



9.3.1.1.3: Okamoto-Uchiyama

```
public static void testOneSizeOkaUchi(){
        /** Inicialización de Okamoto-Uchiyama **/
                OUKeyGen OUkg = new OUKeyGen();
                OkamotoUchiyama_p esystem= new OkamotoUchiyama_p();
                OkamotoUchiyama_privateKey key=
                        OUkg.OkamotoUchiyamaKey(24,14656546);
                esystem.setDecryptEncrypt(key);
                long startT = System.currentTimeMillis();
        /** Inicio cálculo de tiempo de Encriptación **/
                long start = System.currentTimeMillis();
                        BigInteger m=null, c=null;
                        m=BigInteger.valueOf(1);
                        c=esystem.encrypt_ou(m);
                long stop = System.currentTimeMillis();
                System.out.println("\tTiempo Encriptacion: "+ ((stop-start)));
        /** Valor del Texto Encriptado **/
                System.out.println("\tValor del texto encriptado:
"+c.bitLength());
        /** Inicio calculo de tiempo de Desencriptación**/
                start = System.currentTimeMillis();
                        BigInteger decryption=null;
                        decryption=esystem.decrypt_ou(c);
                stop = System.currentTimeMillis();
                long stopT = System.currentTimeMillis();
                System.out.println("\tTiempo Desencriptacion: "+ ((stop-
start)));
                System.out.println("\tTiempo Prueba: "+ (stopT-startT) + "
milisegundos");
}
```

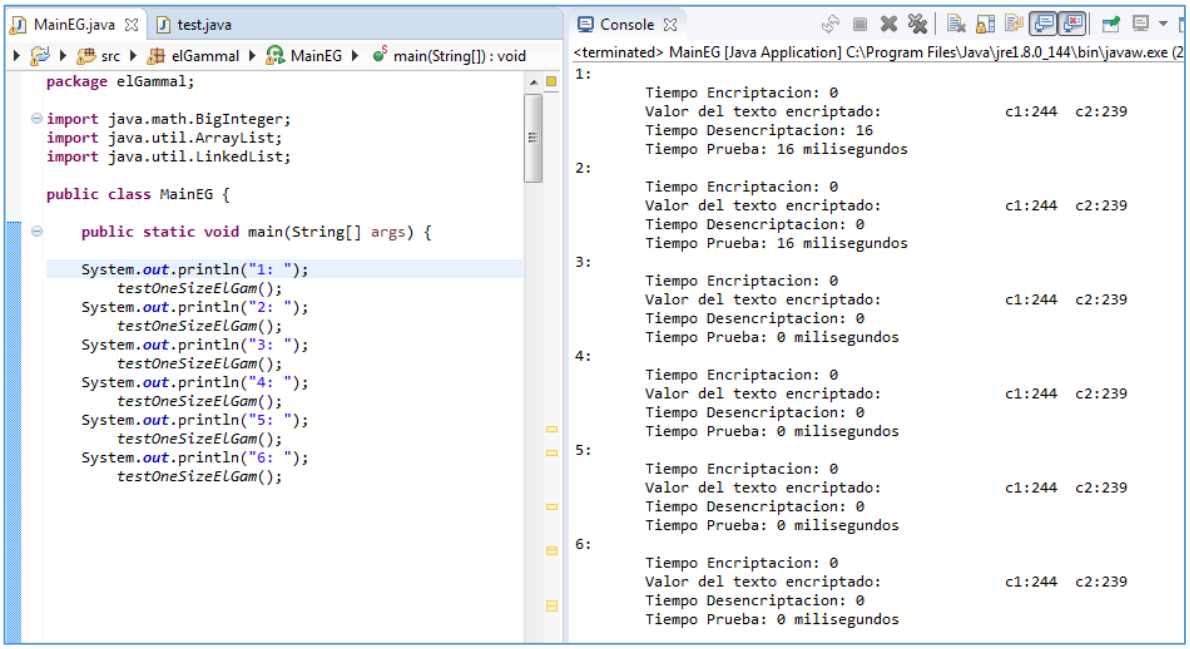
Imagen de Consola:



9.3.1.1.4: ElGammal:

```
public static void testOneSizeElGam(){
        /** Inicialización de ElGammal**/
                EGKeyGen EGkg= new EGKeyGen();
                ElGammal_privateKey key= EGkg.ElGammalKey(245,14656546);
                ElGammal_p esystem = new ElGammal_p();
                esystem.setDecryptEncrypt(key);
                long startT = System.currentTimeMillis();
        /** Inicio cálculo de tiempo de Encriptación**/
                long start = System.currentTimeMillis();
                BigInteger m=null;
                m=BigInteger.valueOf(1);
                LinkedList<BigInteger> c = new LinkedList();
                c= esystem.encrypt_EG(m);
                long stop = System.currentTimeMillis();
                System.out.println("\tTiempo Encriptacion: "+ ((stop-start)));
        /** Valor del Texto Encriptado **/
                System.out.println("\tValor del texto encriptado: \t\t c1:"
                        +c.get(0).bitLength()+"\t c2:"+c.get(1).bitLength());
        /** Inicio cálculo de tiempo de Desencriptación**/
                start = System.currentTimeMillis();
                        BigInteger decryption=null;
                        decryption=esystem.decrypt_EG(c);
                stop = System.currentTimeMillis();
                long stopT = System.currentTimeMillis();
                System.out.println("\tTiempo Desencriptacion: "+ ((stop-
start)));
                System.out.println("\tTiempo Prueba: "+(stopT-startT)+"
milisegundos");
}
```

Imagen de Consola:

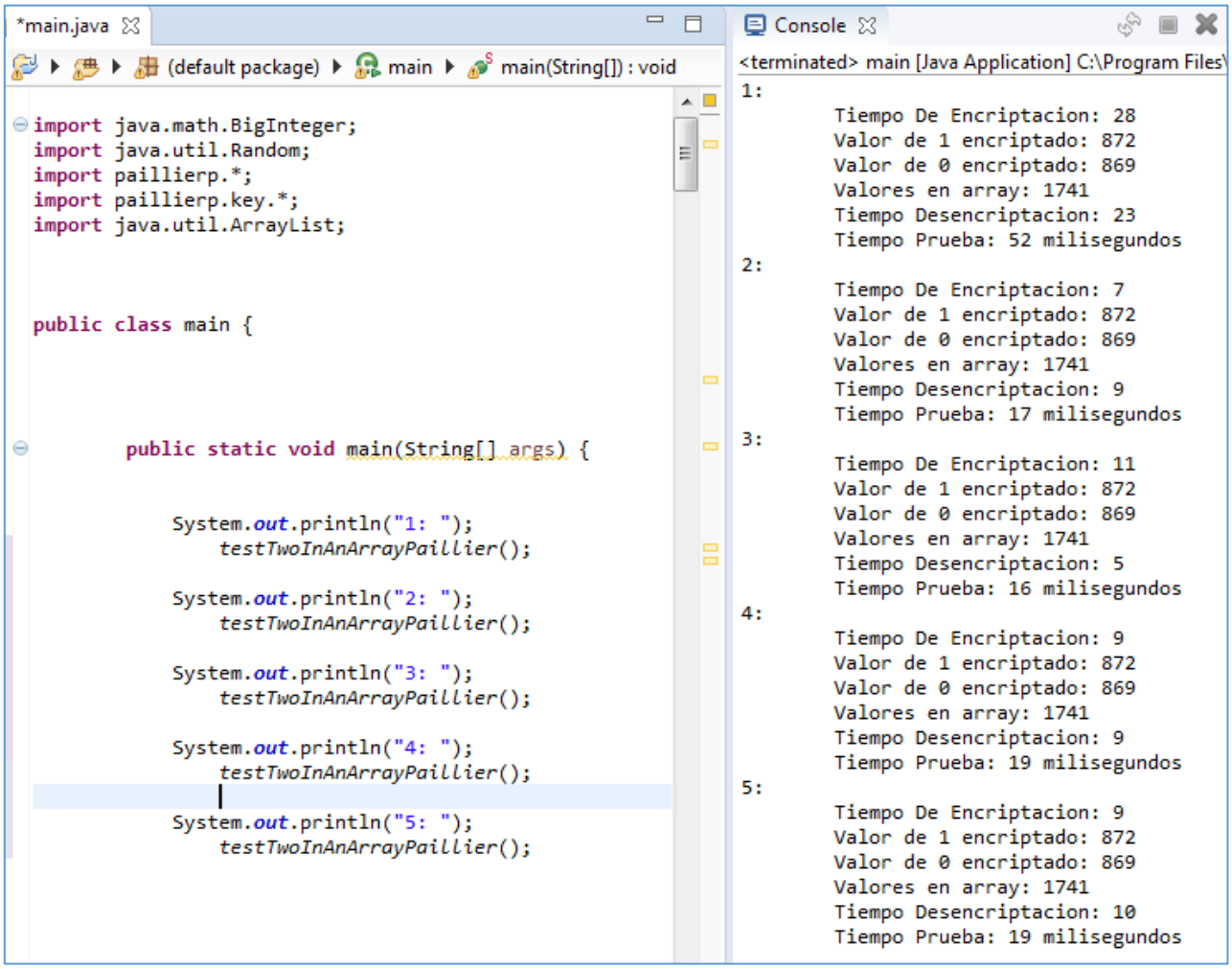


9.3.1.2: Pruebas de Encriptación & Desencriptación a dos valores (1 y 0) almacenados en un arreglo

9.3.1.2.1: Paillier

```
public static void testTwoInAnArrayPaillier(){
        ArrayList<BigInteger> bin= new ArrayList<BigInteger>();
        /** Inicialización de Paillier **/
                Paillier esystem= new Paillier();
                PaillierPrivateKey key=KeyGen.PaillierKey(218,122333356);
                esystem.setDecryptEncrypt(key);
        long startT = System.currentTimeMillis();
        /** Inicio calculo de tiempo de Encriptación **/
                long start = System.currentTimeMillis();
                BigInteger m=null, c=null,m1=null, c1=null;
                m=BigInteger.valueOf(1);
                m1=BigInteger.valueOf(0);
                c=esystem.encrypt(m);
                c1=esystem.encrypt(m1);
                bin.add(c);
                bin.add(c1);
                bin.trimToSize(); //función que limita el tamaño del arraylist
                long stop = System.currentTimeMillis();
                System.out.println("\tTiempo De Encriptacion: "+ ((stop-
start)));
        /** Valor del Texto Encriptado **/
                System.out.println("\tValor de 1 encriptado: " +
c.bitLength());
                System.out.println("\tValor de 0 encriptado: " +
c1.bitLength());
                System.out.println("\tValores en array: " +
(c.bitLength()+c1.bitLength()));
        /** Inicio calculo de tiempo de Desencriptación**/
                start = System.currentTimeMillis();
                BigInteger decryption=null, decryption1=null, t=null, t1=null;
                t= bin.get(0);
                t1=bin.get(1);
                decryption=esystem.decrypt(t);
                decryption1=esystem.decrypt(t1);
                stop = System.currentTimeMillis();
        long stopT = System.currentTimeMillis();
        System.out.println("\tTiempo Desencriptacion: "+ ((stop-start)));
        System.out.println("\tTiempo Prueba: "+ (stopT-startT) + " milisegundos");
}
```

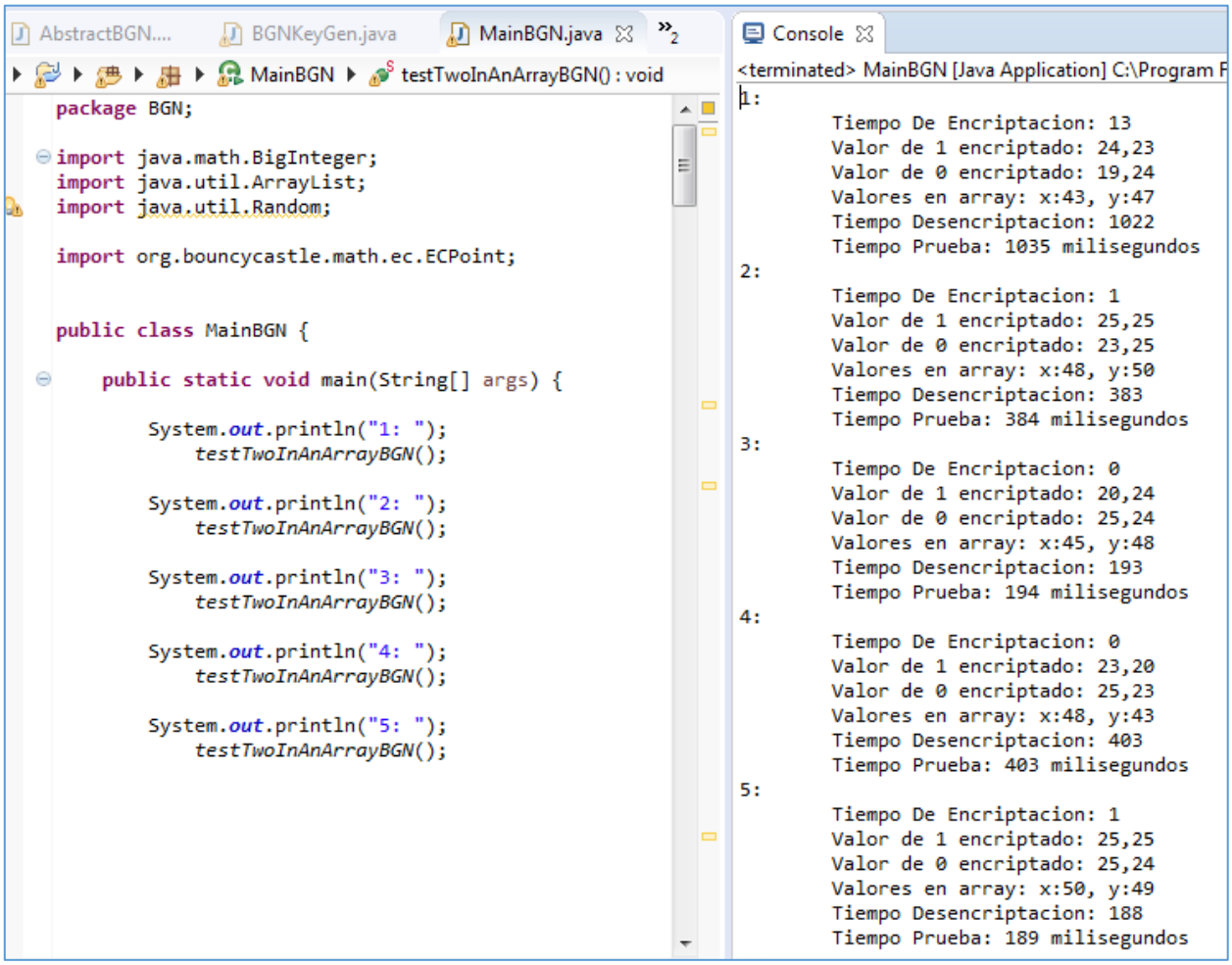
Imagen de Consola:



9.3.1.2.2: Boneh-Goh-Nissim:

```
public static void testTwoInAnArrayBGN(){
        ArrayList<ECPoint> bin= new ArrayList<ECPoint>();
        /** Inicialización de BGN **/
                BGN_p esystem= new BGN_p();
                BGN_privateKey key=BGNKeyGen.BGNKey(24,25165);
                esystem.setDecryptEncrypt(key);
                long startT = System.currentTimeMillis();
        /** Inicio calculo de tiempo de Encriptación **/
                long start = System.currentTimeMillis();
                        BigInteger m=null,m1=null;
                        ECPoint c, c1;
                        m=BigInteger.valueOf(1);
                        m1=BigInteger.valueOf(0);
                        c=esystem.encrypt(m);
                        c1=esystem.encrypt(m1);
                        bin.add(c);
                        bin.add(c1);
                        bin.trimToSize();
                long stop = System.currentTimeMillis();
                System.out.println("\tTiempo De Encriptacion: "+ ((stop-
start)));
        /** Valor del Texto Encriptado **/
            System.out.println("\tValor de 1 encriptado: " +
            c.getXCoord().toBigInteger().bitLength()+","+c.getYCoord().toBigInt
            eger().bitLength());
            System.out.println("\tValor de 0 encriptado: " +
            c1.getXCoord().toBigInteger().bitLength()+","+c1.getYCoord().toBigI
            nteger().bitLength());
            BigInteger txc= c.getXCoord().toBigInteger(),
            txc1=c1.getXCoord().toBigInteger();
            int tx= txc.bitLength() + txc1.bitLength();
            BigInteger tyc= c.getYCoord().toBigInteger(), tyc1=
            c1.getYCoord().toBigInteger();
            int ty= tyc.bitLength() + tyc1.bitLength();
            System.out.println("\tValores en array: x:" + tx+", y:"+ ty);
        /** Inicio calculo de tiempo de Desencriptación**/
                start = System.currentTimeMillis();
                        BigInteger decryption=null, decryption1=null;
                        ECPoint t, t1;
                        t= bin.get(0);
                        t1=bin.get(1);
                        decryption=esystem.decrypt_BGN(t);
                        decryption1=esystem.decrypt_BGN(t1);
                        stop = System.currentTimeMillis();
                long stopT = System.currentTimeMillis();
                System.out.println("\tTiempo Desencriptacion: "+ ((stop-
start)));
                System.out.println("\tTiempo Prueba: "+ (stopT-startT) + "
milisegundos");
}
```

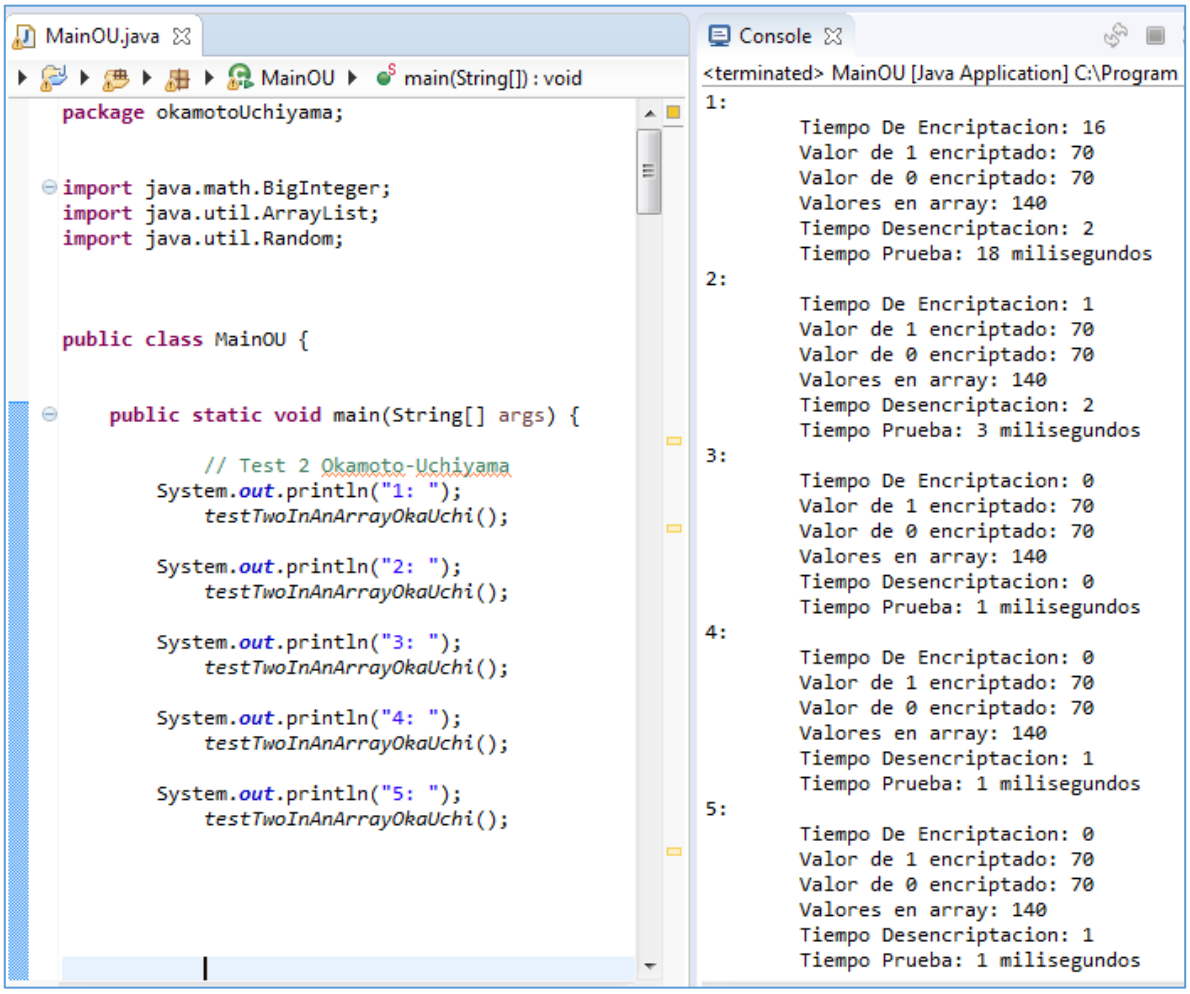
Imagen de Consola:



9.3.1.2.3: Okamoto-Uchiyama:

```
public static void testTwoInAnArrayOkaUchi(){
        ArrayList<BigInteger> bin= new ArrayList<BigInteger>();
        /** Inicialización de Okamoto-Uchiyama **/
                OUKeyGen OUkg = new OUKeyGen();
                OkamotoUchiyama_p esystem= new OkamotoUchiyama_p();
                OkamotoUchiyama_privateKey key=
                        OUkg.OkamotoUchiyamaKey(24,14656546);
                esystem.setDecryptEncrypt(key);
                long startT = System.currentTimeMillis();
        /** Inicio calculo de tiempo de Encriptación **/
                long start = System.currentTimeMillis();
                        BigInteger m=null, c=null,m1=null, c1=null;
                        m=BigInteger.valueOf(1);
                        m1=BigInteger.valueOf(0);
                        c=esystem.encrypt_ou(m);
                        c1=esystem.encrypt_ou(m1);
                        bin.add(c);
                        bin.add(c1);
                        bin.trimToSize();//función que limita el tamaño del
arraylist
                long stop = System.currentTimeMillis();
                System.out.println("\tTiempo De Encriptacion: "+ ((stop-
start)));
        /** Valor del Texto Encriptado **/
                System.out.println("\tValor de 1 encriptado: " +
c.bitLength());
                System.out.println("\tValor de 0 encriptado: " +
c1.bitLength());
                System.out.println("\tValores en array: " +
(c.bitLength()+c1.bitLength()));
        /** Inicio calculo de tiempo de Desencriptación**/
                        start = System.currentTimeMillis();
                        BigInteger decryption=null,
                        decryption1=null, t=null, t1=null;
                        t= bin.get(0);
                        t1=bin.get(1);
                        decryption=esystem.decrypt_ou(t);
                        decryption1=esystem.decrypt_ou(t1);
                stop = System.currentTimeMillis();
                long stopT = System.currentTimeMillis();
                System.out.println("\tTiempo Desencriptacion: "+ ((stop-
start)));
                System.out.println("\tTiempo Prueba: "+ (stopT-startT) + "
milisegundos");
}
```

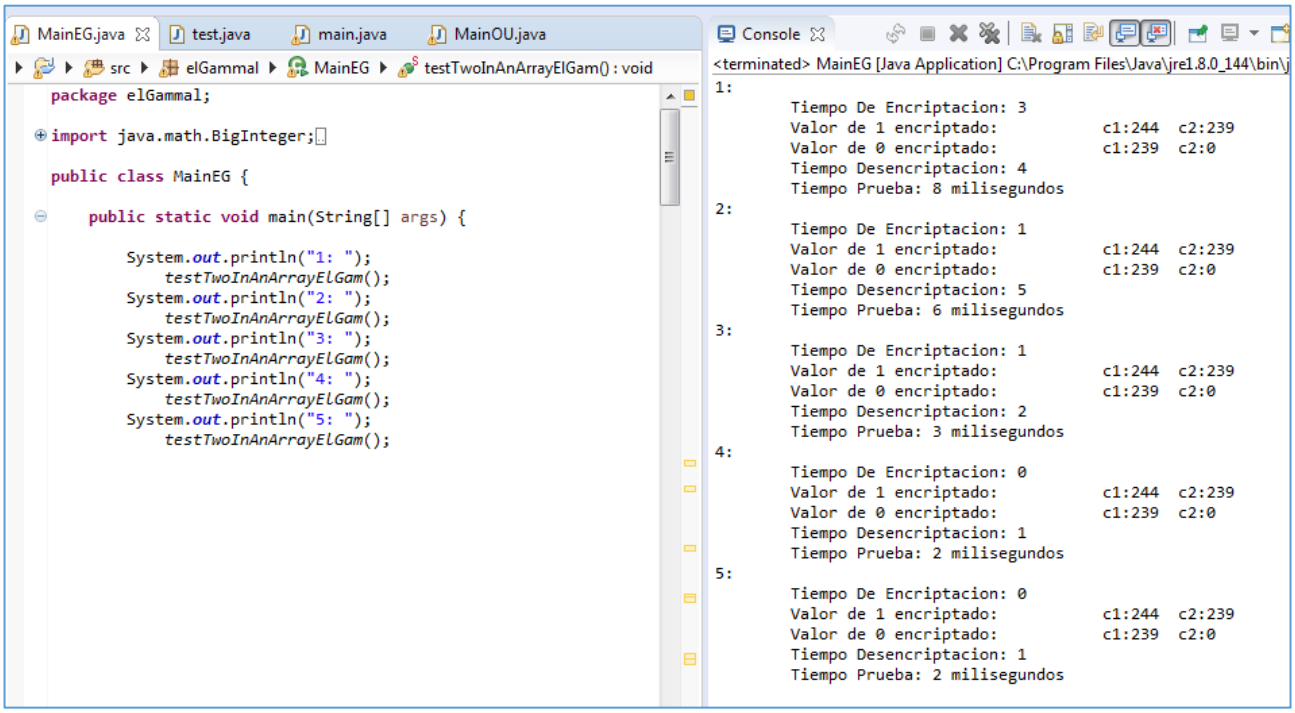
Imagen de Consola:



9.3.1.2.4: ElGammal

```
public static void testTwoInAnArrayElGam(){
        ArrayList<LinkedList<BigInteger>> bin=
                        new ArrayList<LinkedList<BigInteger>>();
        /** Inicialización de ElGammal**/
                EGKeyGen EGkg= new EGKeyGen();
                ElGammal_privateKey key= EGkg.ElGammalKey(245,14656546);
                ElGammal_p esystem = new ElGammal_p();
                esystem.setDecryptEncrypt(key);
        long startT = System.currentTimeMillis();
        /** Inicio calculo de tiempo de Encriptación **/
                long start = System.currentTimeMillis();
                BigInteger m=null, m1=null;
                m=BigInteger.valueOf(1);
                m1=BigInteger.valueOf(0);
                LinkedList<BigInteger> c = new LinkedList(), c1 = new
LinkedList();
                        c=esystem.encrypt_EG(m);
                        c1=esystem.encrypt_EG(m1);
                        bin.add(c);
                        bin.add(c1);
                        bin.trimToSize(); //función que limita el tamaño del
arraylist
                long stop = System.currentTimeMillis();
                System.out.println("\tTiempo De Encriptacion: "+ ((stop-
start)));
        /** Valor del Texto Encriptado **/
                System.out.println("\tValor de 1 encriptado: \t\t c1:"
                        +c.get(0).bitLength()+"\t c2:"+c.get(1).bitLength());
                System.out.println("\tValor de 0 encriptado: \t\t c1:"
                        +c1.get(0).bitLength()+"\t
c2:"+c1.get(1).bitLength());
        /** Inicio cálculo de tiempo de Desencriptación**/
                start = System.currentTimeMillis();
                BigInteger decryption=null, decryption1=null;
                LinkedList<BigInteger> t= new LinkedList(), t1= new
LinkedList();
                t= bin.get(0);
                t1=bin.get(1);
                decryption=esystem.decrypt_EG(t);
                decryption1=esystem.decrypt_EG(t1);
                stop = System.currentTimeMillis();
                long stopT = System.currentTimeMillis();
                System.out.println("\tTiempo Desencriptacion: "+ ((stop-
start)));
                System.out.println("\tTiempo Prueba: "+(stopT-startT)+"
milisegundos");
}
```

Imagen de Consola:

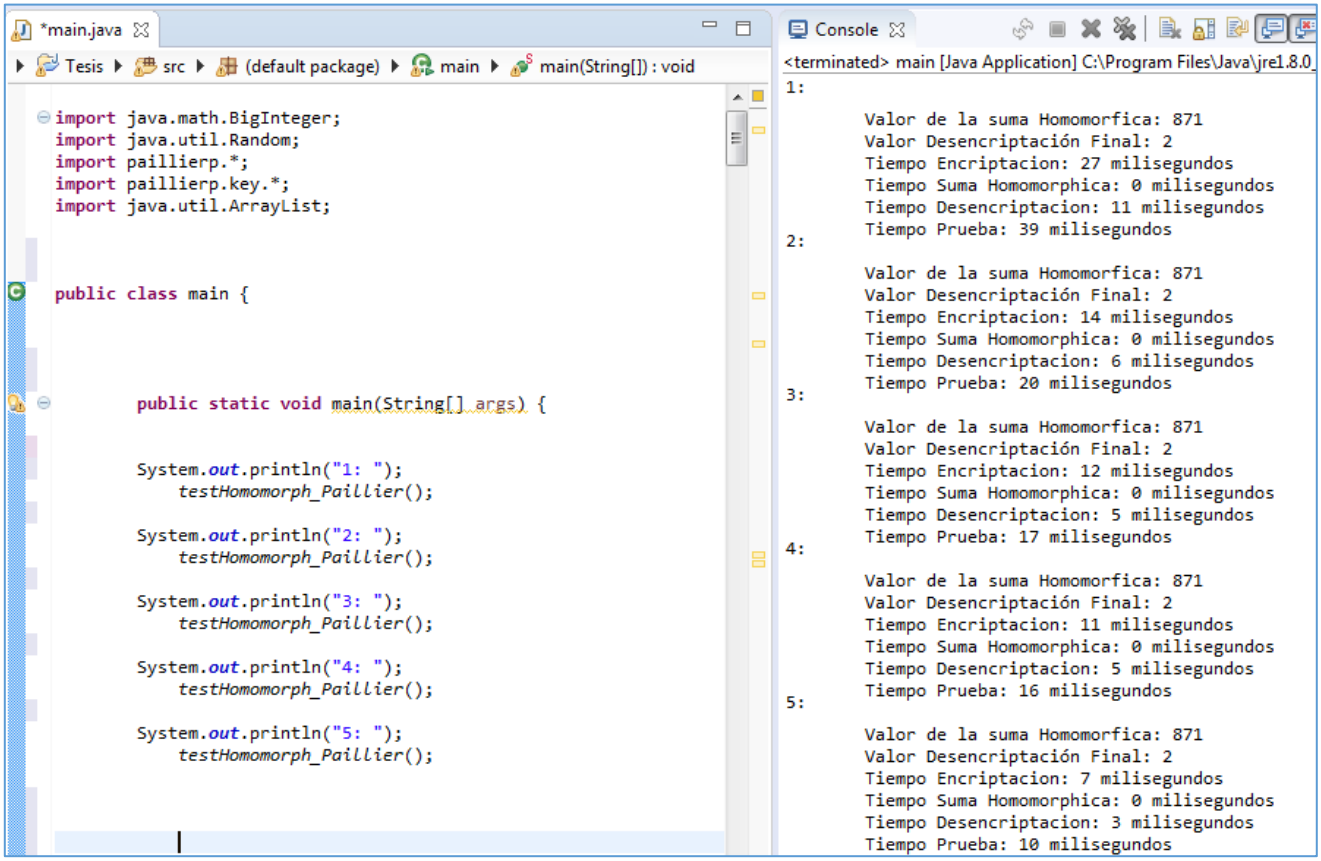


9.3.1.3: Pruebas de Homomorfismo & Desencriptación de dos valores

9.3.1.3.1: Paillier

```
public static void testHomomorph_Paillier(){
        BigInteger decryption=null;
        BigInteger  c=null, c1=null;
        /** Inicialización de Paillier **/
                Paillier esystem= new Paillier();
                PaillierPrivateKey key=KeyGen.PaillierKey(218,122333356);
                esystem.setDecryptEncrypt(key);
        long miliEnc=0, miliDes=0, miliSum=0;
        long startT = System.currentTimeMillis();
        /** Inicio calculo de tiempo de Encriptación **/
                long start = System.currentTimeMillis();
                        c=esystem.encrypt(BigInteger.ONE);
                        c1=esystem.encrypt(BigInteger.ONE);
                long stop = System.currentTimeMillis();
                miliEnc=miliEnc+(stop-start);
        /** Inicio calculo de tiempo de Suma Homomorphica **/
                start = System.currentTimeMillis();
                        BigInteger temp = esystem.add(c,c1);
                stop = System.currentTimeMillis();
                miliSum=miliSum+(stop-start);
                System.out.println("\tValor de la suma Homomórfica:
"+temp.bitLength());
        /** Inicio calculo de tiempo de Desencriptación**/
                start = System.currentTimeMillis();
                        decryption=esystem.decrypt(temp);
                stop = System.currentTimeMillis();
                miliDes=miliDes+(stop-start);
        long stopT = System.currentTimeMillis();
        System.out.println("\tValor Desencriptación Final: "+decryption.toString());
        System.out.println("\tTiempo Encriptacion: "+miliEnc+ " milisegundos");
        System.out.println("\tTiempo Suma Homomorphica: "+miliSum + " milisegundos");
        System.out.println("\tTiempo Desencriptacion: "+miliDes + " milisegundos");
        System.out.println("\tTiempo Prueba: "+ (stopT-startT) + " milisegundos");
}
```

Imagen de Consola:



9.3.1.3.2: Boneh-Goh-Nissim

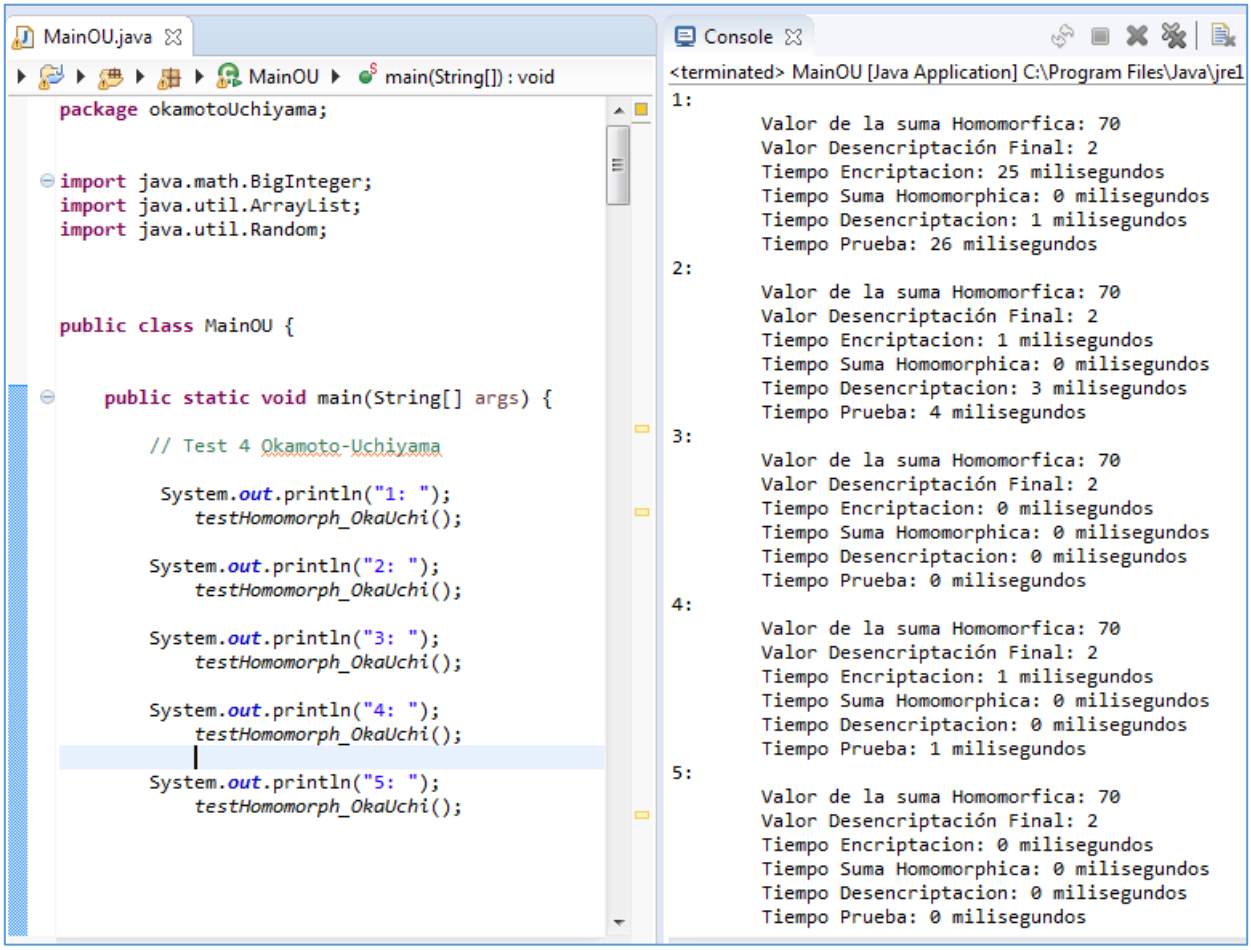
La librería Boneh-Goh-Nissim desarrollada presenta problemas de estabilidad al momento de generarse las llaves públicas “g” y “h”, las cuales al ser puntos válidos de la Curva Elíptica, por razones desconocidas, no aseguran el correcto funcionar de la suma homomórfica ni de la correcta encriptación de diferentes mensajes con un rango $[\ 1, q_2\ ]$.

Por lo cual no fue posible el implementar esta prueba mediante la aplicación de dicha librería.

9.3.1.3.3: Okamoto-Uchiyama

```
public static void testHomomorph_OkaUchi(){
        BigInteger decryption=null;
        BigInteger  c=null, c1=null;
        /** Inicialización de Okamoto-Uchiyama **/
                OUKeyGen OUkg = new OUKeyGen();
                OkamotoUchiyama_p esystem= new OkamotoUchiyama_p();
                OkamotoUchiyama_privateKey key=
                        OUkg.OkamotoUchiyamaKey(24,14656546);
                esystem.setDecryptEncrypt(key);
                long miliEnc=0, miliDes=0, miliSum=0;
                long startT = System.currentTimeMillis();
        /** Inicio calculo de tiempo de Encriptación **/
                long start = System.currentTimeMillis();
                        c=esystem.encrypt_ou(BigInteger.ONE);
                        c1=esystem.encrypt_ou(BigInteger.ONE);
                long stop = System.currentTimeMillis();
                miliEnc=miliEnc+(stop-start);
        /** Inicio calculo de tiempo de Suma Homomorphica **/
                start = System.currentTimeMillis();
                        BigInteger temp = esystem.addOU(c,c1);
                stop = System.currentTimeMillis();
                miliSum=miliSum+(stop-start);
                System.out.println("\tValor de la suma Homomórfica:
"+temp.bitLength());
        /** Inicio calculo de tiempo de Desencriptación**/
                start = System.currentTimeMillis();
                        decryption=esystem.decrypt_ou(temp);
                stop = System.currentTimeMillis();
                miliDes=miliDes+(stop-start);
        long stopT = System.currentTimeMillis();
        System.out.println("\tValor Desencriptación Final: "+decryption.toString());
        System.out.println("\tTiempo Encriptacion: "+miliEnc+ " milisegundos");
        System.out.println("\tTiempo Suma Homomorphica: "+miliSum + " milisegundos");
        System.out.println("\tTiempo Desencriptacion: "+miliDes + " milisegundos");
        System.out.println("\tTiempo Prueba: "+ (stopT-startT) + " milisegundos");
}
```

Imagen de Consola:



9.3.1.3.4: ElGammal

La librería ElGammal desarrollada sólo contempla la multiplicación homomórfica, con lo cual no fue posible el implementar esta prueba mediante la aplicación de dicha librería.

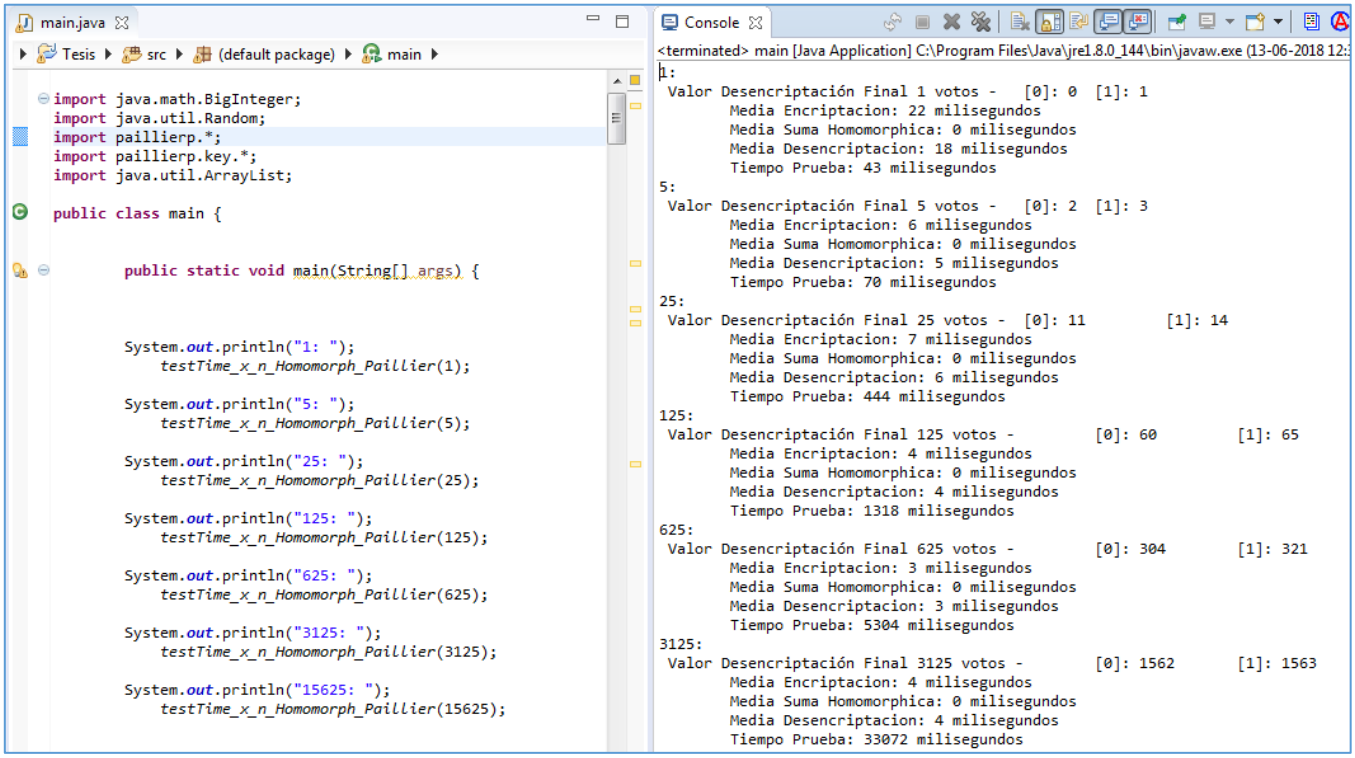
9.3.1.4: Pruebas de Encriptación y Desencriptación de valores almacenados en un vector, contabilizados mediante homomorfismo

9.3.1.4.1: Paillier

```
public static void testTime_x_n_Homomorph_Paillier(int v){
        BigInteger decryption=null, decryption1=null;
        BigInteger m=null, m1=null, c=null, c1=null;
        /** Inicialización de Paillier **/
                Paillier esystem= new Paillier();
                PaillierPrivateKey key=KeyGen.PaillierKey(218,122333356);
                esystem.setDecryptEncrypt(key);
                //Inicialización del arreglo suma
                        ArrayList<BigInteger> suma= new
ArrayList<BigInteger>();
                        m=BigInteger.valueOf(0);
                        m1=BigInteger.valueOf(0);
                        suma.add(esystem.encrypt(m));
                        suma.add(esystem.encrypt(m1));
        long miliEnc=0, miliDes=0, miliSum=0;
        long random=-1;
        long startT = System.currentTimeMillis();
        for (int i=0; i<v;i++){
                ArrayList<BigInteger> bin= new ArrayList<BigInteger>();
                Random rd=new Random();

                /** Inicio calculo de tiempo de Encriptación **/
                        long start = System.currentTimeMillis();
                        random= rd.nextLong();
                        //Generación del arreglo "Voto"
                                if (random%2==0){
                                        m=BigInteger.valueOf(1);
                                        m1=BigInteger.valueOf(0);
                                }else{
                                        m=BigInteger.valueOf(0);
                                        m1=BigInteger.valueOf(1);
                                }
                                c=esystem.encrypt(m);
                                c1=esystem.encrypt(m1);
                                bin.add(c);
                                bin.add(c1);

                        long stop = System.currentTimeMillis();
                        miliEnc=miliEnc+(stop-start);

                /** Inicio calculo de tiempo de Suma Homomorphica **/
                        start = System.currentTimeMillis();
                                BigInteger t0 =bin.get(0);
                                BigInteger t1 =bin.get(1);
                                BigInteger temp0= suma.get(0);
                                BigInteger temp1= suma.get(1);

                                temp0= esystem.add(temp0, t0);
```

```
                                temp1= esystem.add(temp1, t1);
                                suma.set(0, temp0);
                                suma.set(1, temp1);
                        stop = System.currentTimeMillis();
                        miliSum=miliSum+(stop-start);
                        try{
                                Thread.sleep(1);
                        }catch(InterruptedException e){
                                System.out.println("Thread Interrupted");
                        }

                /** Inicio calculo de tiempo de Desencriptación**/
                        start = System.currentTimeMillis();
                        decryption=esystem.decrypt(bin.get(0));
                        decryption1=esystem.decrypt(bin.get(1));

                        miliDes=miliDes+(stop-start);
                        try{
                                Thread.sleep(1);
                        }catch(InterruptedException e){
                                System.out.println("Thread Interrupted");
                        }

                }
        long stopT = System.currentTimeMillis();
        decryption=esystem.decrypt(suma.get(0));
        decryption1=esystem.decrypt(suma.get(1));
        BigInteger total= esystem.decrypt(esystem.add(suma.get(0),suma.get(1)));
        System.out.println(" Valor Desencriptación Final "+total.toString()+
                " votos -\t [0]: "+decryption.toString()+"\t [1]: "+decryption1.toString());
        System.out.println("\tMedia Encriptacion: "+miliEnc/v+ "
milisegundos");
        System.out.println("\tMedia Suma Homomorphica: "+miliSum/v + " milisegundos");
        System.out.println("\tMedia Desencriptacion: "+miliDes/v + " milisegundos");
        System.out.println("\tTiempo Prueba: "+ (stopT-startT) + " milisegundos");
}
```

Imagen de Consola:



main.java

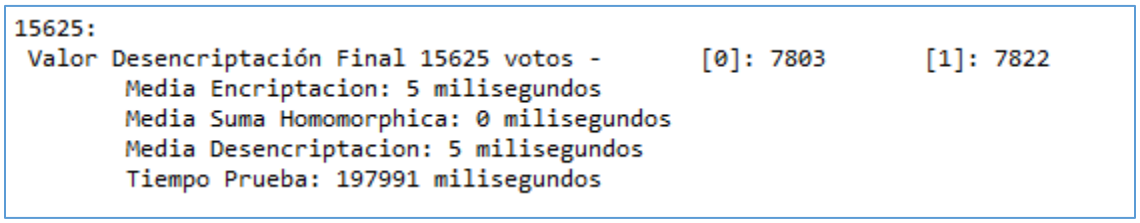
Tesis ▸ src ▸ (default package) ▸ main ▸

```
import java.math.BigInteger;
import java.util.Random;
import paillierp.*;
import paillierp.key.*;
import java.util.ArrayList;

public class main {


        public static void main(String[] args) {



        System.out.println("1: ");
            testTime_x_n_Homomorph_Paillier(1);

        System.out.println("5: ");
            testTime_x_n_Homomorph_Paillier(5);

        System.out.println("25: ");
            testTime_x_n_Homomorph_Paillier(25);

        System.out.println("125: ");
            testTime_x_n_Homomorph_Paillier(125);

        System.out.println("625: ");
            testTime_x_n_Homomorph_Paillier(625);

        System.out.println("3125: ");
            testTime_x_n_Homomorph_Paillier(3125);

        System.out.println("15625: ");
            testTime_x_n_Homomorph_Paillier(15625);
```

Console

<terminated> main [Java Application] C:\Program Files\Java\jre1.8.0_144\bin\javaw.exe (13-06-2018 12:

```
1:
 Valor Desencriptación Final 1 votos -   [0]: 0  [1]: 1
        Media Encriptacion: 22 milisegundos
        Media Suma Homomorphica: 0 milisegundos
        Media Desencriptacion: 18 milisegundos
        Tiempo Prueba: 43 milisegundos
5:
 Valor Desencriptación Final 5 votos -   [0]: 2  [1]: 3
        Media Encriptacion: 6 milisegundos
        Media Suma Homomorphica: 0 milisegundos
        Media Desencriptacion: 5 milisegundos
        Tiempo Prueba: 70 milisegundos
25:
 Valor Desencriptación Final 25 votos -  [0]: 11         [1]: 14
        Media Encriptacion: 7 milisegundos
        Media Suma Homomorphica: 0 milisegundos
        Media Desencriptacion: 6 milisegundos
        Tiempo Prueba: 444 milisegundos
125:
 Valor Desencriptación Final 125 votos -         [0]: 60         [1]: 65
        Media Encriptacion: 4 milisegundos
        Media Suma Homomorphica: 0 milisegundos
        Media Desencriptacion: 4 milisegundos
        Tiempo Prueba: 1318 milisegundos
625:
 Valor Desencriptación Final 625 votos -         [0]: 304        [1]: 321
        Media Encriptacion: 3 milisegundos
        Media Suma Homomorphica: 0 milisegundos
        Media Desencriptacion: 3 milisegundos
        Tiempo Prueba: 5304 milisegundos
3125:
 Valor Desencriptación Final 3125 votos -        [0]: 1562       [1]: 1563
        Media Encriptacion: 4 milisegundos
        Media Suma Homomorphica: 0 milisegundos
        Media Desencriptacion: 4 milisegundos
        Tiempo Prueba: 33072 milisegundos
```



```
15625:
 Valor Desencriptación Final 15625 votos -       [0]: 7803       [1]: 7822
        Media Encriptacion: 5 milisegundos
        Media Suma Homomorphica: 0 milisegundos
        Media Desencriptacion: 5 milisegundos
        Tiempo Prueba: 197991 milisegundos
```

9.3.1.4.2: Boneh-Goh-Nissim

La librería Boneh-Goh-Nissim desarrollada presenta problemas de estabilidad al momento de generarse las llaves públicas “g” y “h”, las cuales al ser puntos válidos de la Curva Elíptica, por razones desconocidas, no aseguran el correcto funcionar de la suma homomórfica ni de la correcta encriptación de diferentes mensajes con un rango $[\ 1, q_2\ ]$.

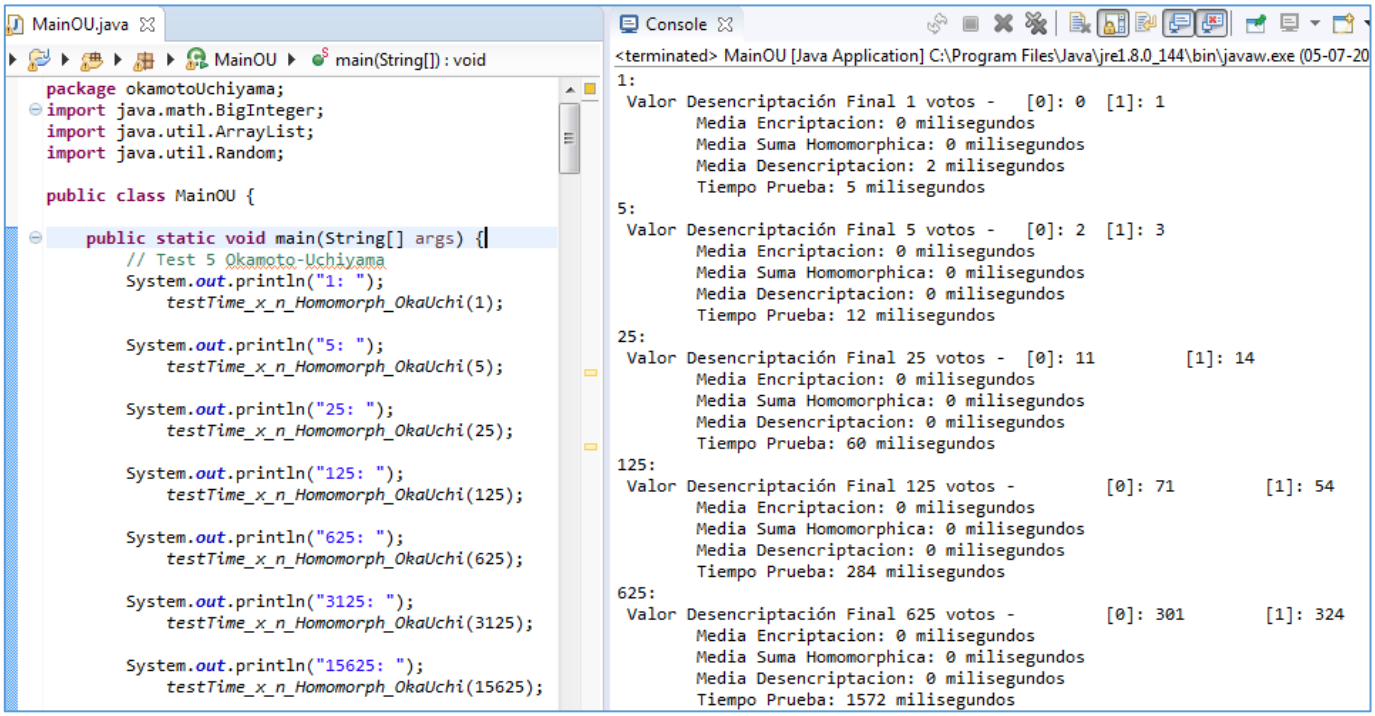
Por lo cual no fue posible el implementar esta prueba mediante la aplicación de dicha librería.

9.3.1.4.3: Okamoto-Uchiyama

```
public static void testTime_x_n_Homomorph_OkaUchi(int v){
                BigInteger decryption=null, decryption1=null;
                BigInteger m=null, m1=null, c=null, c1=null;
        /** Inicialización de Okamoto-Uchiyama **/
                OUKeyGen OUkg = new OUKeyGen();
                OkamotoUchiyama_p esystem= new OkamotoUchiyama_p();
                OkamotoUchiyama_privateKey key=
                OUkg.OkamotoUchiyamaKey(24,14656546);
                esystem.setDecryptEncrypt(key);
        //Inicialización del arreglo suma
                        ArrayList<BigInteger> suma= new
ArrayList<BigInteger>();
                        m=BigInteger.valueOf(0);
                        m1=BigInteger.valueOf(0);
                        suma.add(esystem.encrypt_ou(m));
                        suma.add(esystem.encrypt_ou(m1));
                long miliEnc=0, miliDes=0, miliSum=0;
                long random=-1;
                long startT = System.currentTimeMillis();
                for (int i=0; i<v;i++){
                        ArrayList<BigInteger> bin= new
ArrayList<BigInteger>();
                        Random rd=new Random();

                /** Inicio calculo de tiempo de Encriptación **/
                        long start = System.currentTimeMillis();
                        random= rd.nextLong();
                        //Generación del arreglo "Voto"
                                if (random%2==0){
                                        m=BigInteger.valueOf(1);
                                        m1=BigInteger.valueOf(0);
                                }else{
                                        m=BigInteger.valueOf(0);
                                        m1=BigInteger.valueOf(1);
                                }
                                c=esystem.encrypt_ou(m);
                                c1=esystem.encrypt_ou(m1);
                                bin.add(c);
                                bin.add(c1);
                        long stop = System.currentTimeMillis();
                        miliEnc=miliEnc+(stop-start);
```

```
                /** Inicio calculo de tiempo de Suma Homomorphica **/
                        start = System.currentTimeMillis();
                                BigInteger t0 =bin.get(0);
                                BigInteger t1 =bin.get(1);
                                BigInteger temp0= suma.get(0);
                                BigInteger temp1= suma.get(1);

                                temp0= esystem.addOU(temp0, t0);
                                temp1= esystem.addOU(temp1, t1);
                                suma.set(0, temp0);
                                suma.set(1, temp1);
                        stop = System.currentTimeMillis();
                        miliSum=miliSum+(stop-start);
                        try{
                                Thread.sleep(1);
                        }catch(InterruptedException e){
                                System.out.println("Thread Interrupted");
                        }
        /** Inicio calculo de tiempo de Desencriptación**/
                start = System.currentTimeMillis();
                        decryption=esystem.decrypt_ou(bin.get(0));
                        decryption1=esystem.decrypt_ou(bin.get(1));
                stop = System.currentTimeMillis();
                miliDes=miliDes+(stop-start);
                try{
                        Thread.sleep(1);
                }catch(InterruptedException e){
                        System.out.println("Thread Interrupted");
                }
        }
        long stopT = System.currentTimeMillis();
        decryption=esystem.decrypt_ou(suma.get(0));
        decryption1=esystem.decrypt_ou(suma.get(1));
        BigInteger total=
esystem.decrypt_ou(esystem.addOU(suma.get(0),suma.get(1)));
        System.out.println(" Valor Desencriptación Final "+total.toString()+
                " votos -\t [0]: "+decryption.toString()+"\t [1]: "+decryption1.toString());
        System.out.println("\tMedia Encriptacion: "+miliEnc/v+ "
milisegundos");
        System.out.println("\tMedia Suma Homomorphica: "+miliSum/v + "
milisegundos");
        System.out.println("\tMedia Desencriptacion: "+miliDes/v + " milisegundos");
        System.out.println("\tTiempo Prueba: "+ (stopT-startT) + " milisegundos");
}
```

Imagen de Consola:



MainOU.java

```
package okamotoUchiyama;
import java.math.BigInteger;
import java.util.ArrayList;
import java.util.Random;

public class MainOU {

    public static void main(String[] args) {
        // Test 5 Okamoto-Uchiyama
        System.out.println("1: ");
            testTime_x_n_Homomorph_OkaUchi(1);

        System.out.println("5: ");
            testTime_x_n_Homomorph_OkaUchi(5);

        System.out.println("25: ");
            testTime_x_n_Homomorph_OkaUchi(25);

        System.out.println("125: ");
            testTime_x_n_Homomorph_OkaUchi(125);

        System.out.println("625: ");
            testTime_x_n_Homomorph_OkaUchi(625);

        System.out.println("3125: ");
            testTime_x_n_Homomorph_OkaUchi(3125);

        System.out.println("15625: ");
            testTime_x_n_Homomorph_OkaUchi(15625);
```

Console

```
<terminated> MainOU [Java Application] C:\Program Files\Java\jre1.8.0_144\bin\javaw.exe (05-07-20
1:
 Valor Desencriptación Final 1 votos -   [0]: 0  [1]: 1
        Media Encriptacion: 0 milisegundos
        Media Suma Homomorphica: 0 milisegundos
        Media Desencriptacion: 2 milisegundos
        Tiempo Prueba: 5 milisegundos
5:
 Valor Desencriptación Final 5 votos -   [0]: 2  [1]: 3
        Media Encriptacion: 0 milisegundos
        Media Suma Homomorphica: 0 milisegundos
        Media Desencriptacion: 0 milisegundos
        Tiempo Prueba: 12 milisegundos
25:
 Valor Desencriptación Final 25 votos -  [0]: 11         [1]: 14
        Media Encriptacion: 0 milisegundos
        Media Suma Homomorphica: 0 milisegundos
        Media Desencriptacion: 0 milisegundos
        Tiempo Prueba: 60 milisegundos
125:
 Valor Desencriptación Final 125 votos -         [0]: 71         [1]: 54
        Media Encriptacion: 0 milisegundos
        Media Suma Homomorphica: 0 milisegundos
        Media Desencriptacion: 0 milisegundos
        Tiempo Prueba: 284 milisegundos
625:
 Valor Desencriptación Final 625 votos -         [0]: 301        [1]: 324
        Media Encriptacion: 0 milisegundos
        Media Suma Homomorphica: 0 milisegundos
        Media Desencriptacion: 0 milisegundos
        Tiempo Prueba: 1572 milisegundos
```

```
3125:
 Valor Desencriptación Final 3125 votos -        [0]: 1589       [1]: 1536
        Media Encriptacion: 0 milisegundos
        Media Suma Homomorphica: 0 milisegundos
        Media Desencriptacion: 0 milisegundos
        Tiempo Prueba: 7190 milisegundos
15625:
 Valor Desencriptación Final 15625 votos -       [0]: 7841       [1]: 7784
        Media Encriptacion: 0 milisegundos
        Media Suma Homomorphica: 0 milisegundos
        Media Desencriptacion: 0 milisegundos
        Tiempo Prueba: 34715 milisegundos
```

9.3.1.4.4: ElGammal

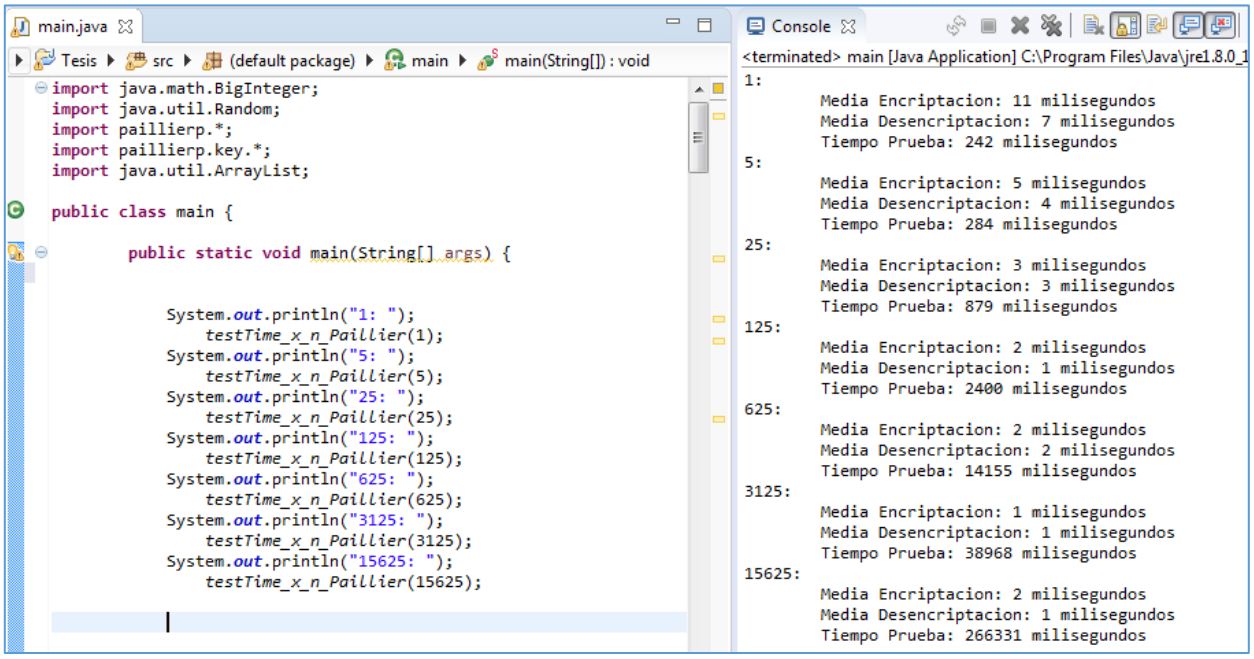
La librería ElGammal desarrollada sólo contempla la multiplicación homomórfica, con lo cual no fue posible el implementar esta prueba mediante la aplicación de dicha librería.

9.3.1.5: Pruebas de Generación de Llaves, Encriptación y Desencriptación del valor 1 ingresando una cantidad de repeticiones de dichos procesos

9.3.1.5.1: Paillier

```
	public static void testTime_x_n_Paillier(int n){
		BigInteger decryption=null;
		BigInteger m=null, c=null;
		long miliEnc=0, miliDes=0;
		long startT = System.currentTimeMillis();
		for (int i=0; i<n;i++){
		/** Inicialización de Paillier **/
			Paillier esystem= new Paillier();
			PaillierPrivateKey
key=KeyGen.PaillierKey(218,122333356);
			m=BigInteger.valueOf(1);
			esystem.setDecryptEncrypt(key);
		/** Inicio calculo de tiempo de Encriptación **/
			long start = System.currentTimeMillis();
			c=esystem.encrypt(m);
			long stop = System.currentTimeMillis();
			miliEnc=miliEnc+(stop-start);
			try{
				Thread.sleep(1);
			}catch(InterruptedException e){
				System.out.println("Thread Interrupted");
			}
		/** Inicio calculo de tiempo de Desencriptación**/
			start = System.currentTimeMillis();
			decryption=esystem.decrypt(c);
			stop = System.currentTimeMillis();
			miliDes=miliDes+(stop-start);
			try{
				Thread.sleep(1);
			}catch(InterruptedException e){
				System.out.println("Thread Interrupted");
			}
		}
		long stopT = System.currentTimeMillis();
		System.out.println("\tMedia de Encriptacion: "+miliEnc/n+ "
milisegundos");
		System.out.println("\tMedia de Desencriptacion: "+miliDes/n + "
milisegundos");
		System.out.println("\tTiempo Prueba: "+ (stopT-startT) + "
milisegundos");

	}
```

Imagen de Consola:

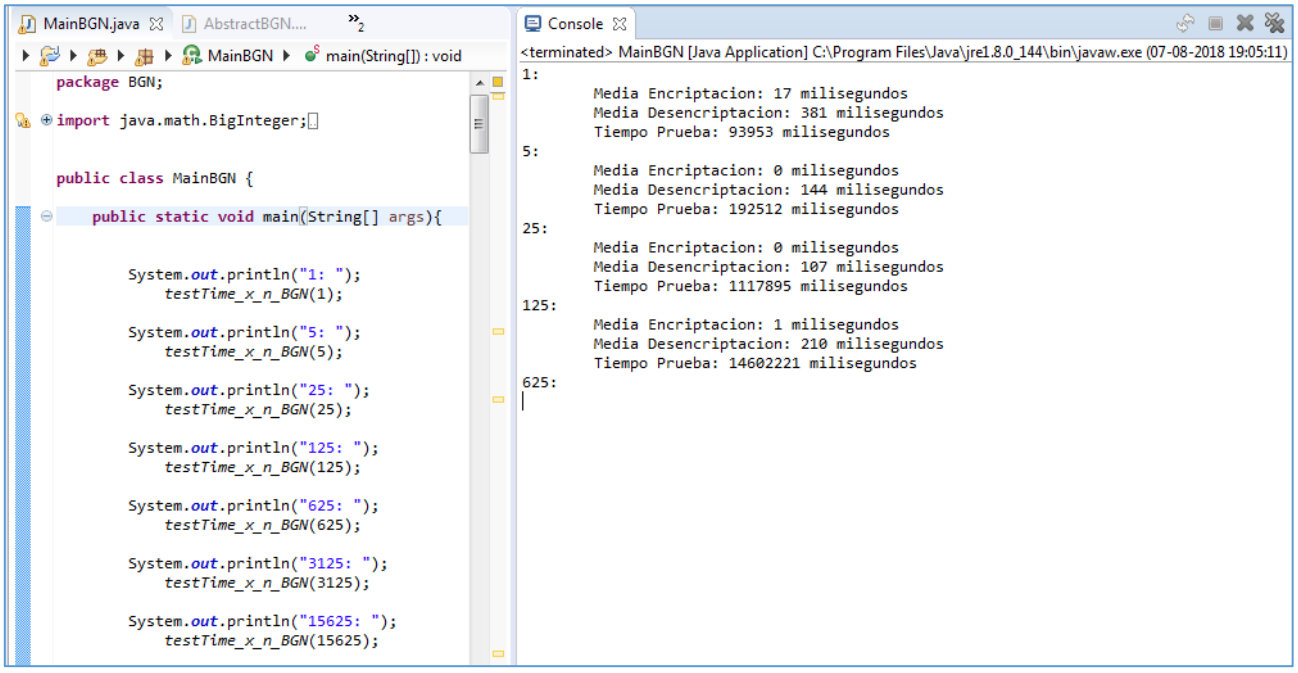


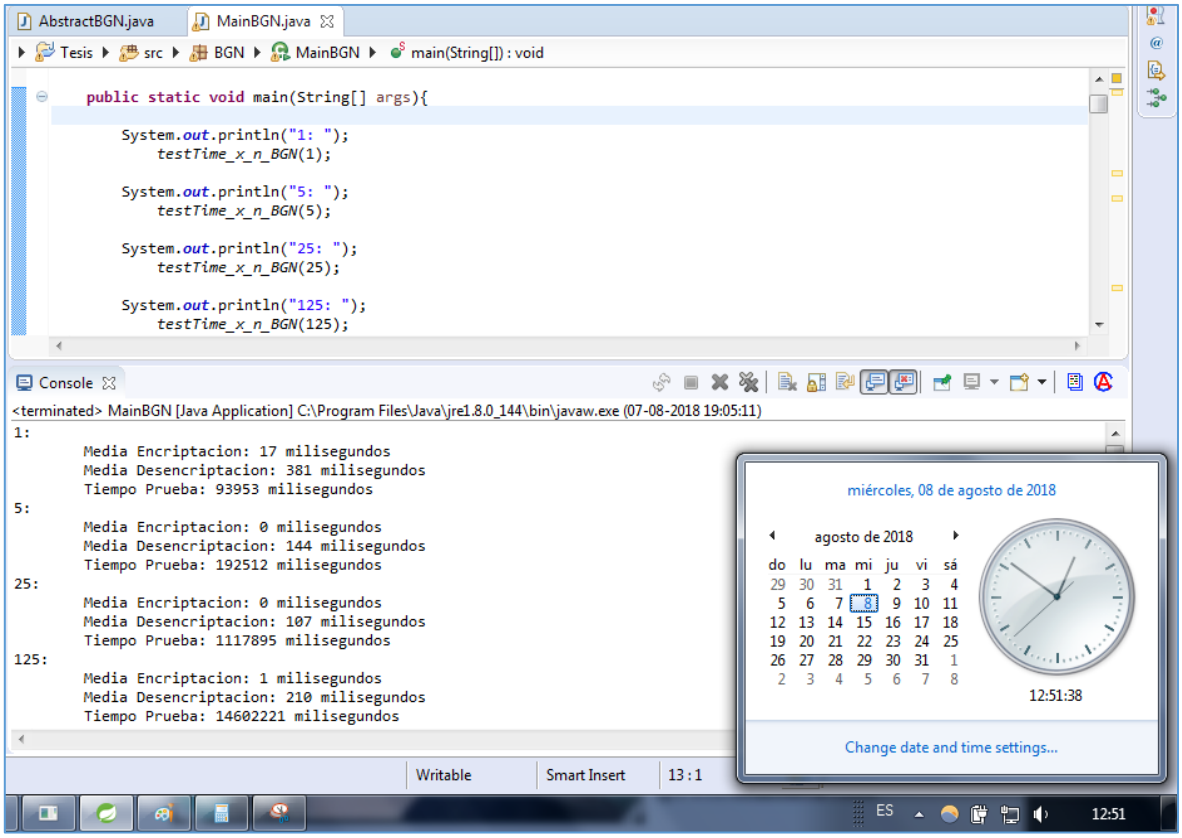
9.3.1.5.2: Boneh-Goh-Nissim

```
public static void testTime_x_n_BGN(int n){
        BigInteger decryption=null;
        BigInteger m=null;
        ECPoint c;
        long miliEnc=0, miliDes=0;
        long startT = System.currentTimeMillis();
        for (int i=0; i<n;i++){
        /** Inicialización de BGN **/
                BGN_p esystem= new BGN_p();
                BGN_privateKey key=BGNKeyGen.BGNKey(24,25165);
                esystem.setDecryptEncrypt(key);
                m=BigInteger.valueOf(1);
        /** Inicio calculo de tiempo de Encriptación **/
                long start = System.currentTimeMillis();
                c=esystem.encrypt(m);
                long stop = System.currentTimeMillis();
                miliEnc=miliEnc+(stop-start);
                try{
                        Thread.sleep(1);
                }catch(InterruptedException e){
                        System.out.println("Thread Interrupted");
                }
        /** Inicio calculo de tiempo de Desencriptación**/
                start = System.currentTimeMillis();
                        decryption=esystem.decrypt_BGN(c);
                stop = System.currentTimeMillis();
                miliDes=miliDes+(stop-start);
                try{
                        Thread.sleep(1);
                }catch(InterruptedException e){
                        System.out.println("Thread Interrupted");
                }
        }
        long stopT = System.currentTimeMillis();
        System.out.println("\tMedia Encriptacion: "+miliEnc/n+ "
milisegundos");
        System.out.println("\tMedia Desencriptacion: "+miliDes/n +"
milisegundos");
        System.out.println("\tTiempo Prueba: "+(stopT-startT)+ "
milisegundos");
}
```

Imagen de Consola:

Debido a la extensa duración de la prueba y, considerando que los resultados obtenidos en el tiempo de duración incrementan conforme el valor de la iteración se ha optado por abortar la prueba tras la consecución de cuatro iteraciones: “1”, “5”, “25” y “125”.





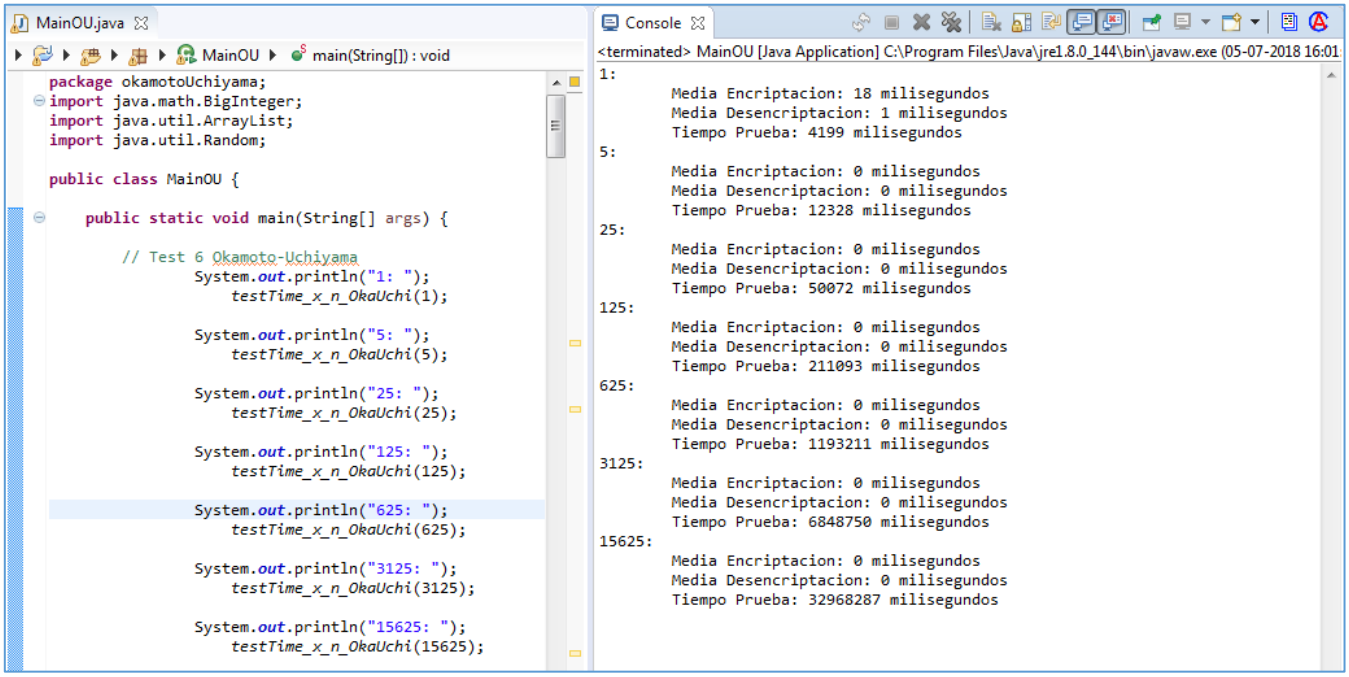
[13]

[13] La prueba inició el 07 de Agosto del 2018 a las 7 de la tarde con cinco minutos. Al momento de la interrumpión del test eran las doce de la tarde con cincuenta y un minutos del día 8 de Agosto del 2018.

9.3.1.5.3: Okamoto-Uchiyama

```
public static void testTime_x_n_OkaUchi(int n){
        BigInteger decryption=null;
        BigInteger m=null, c=null;
        long miliEnc=0, miliDes=0;
        long startT = System.currentTimeMillis();
        for (int i=0; i<n;i++){
        /** Inicialización de Okamoto-Uchiyama **/
                OUKeyGen OUkg = new OUKeyGen();
                OkamotoUchiyama_p esystem= new OkamotoUchiyama_p();
                OkamotoUchiyama_privateKey key=
                        OUkg.OkamotoUchiyamaKey(24,14656546);
                esystem.setDecryptEncrypt(key);
                m=BigInteger.valueOf(1);
        /** Inicio calculo de tiempo de Encriptación **/
                long start = System.currentTimeMillis();
                c=esystem.encrypt_ou(m);
                long stop = System.currentTimeMillis();
                miliEnc=miliEnc+(stop-start);
                try{
                        Thread.sleep(1);
                }catch(InterruptedException e){
                        System.out.println("Thread Interrupted");
                }
        /** Inicio calculo de tiempo de Desencriptación**/
                start = System.currentTimeMillis();
                decryption=esystem.decrypt_ou(c);
                stop = System.currentTimeMillis();
                miliDes=miliDes+(stop-start);
                try{
                        Thread.sleep(1);
                }catch(InterruptedException e){
                        System.out.println("Thread Interrupted");
                }
        }
        long stopT = System.currentTimeMillis();
        System.out.println("\tMedia Encriptacion: "+miliEnc/n+ "
milisegundos");
        System.out.println("\tMedia Desencriptacion: "+miliDes/n + " milisegundos");
        System.out.println("\tTiempo Prueba: "+ (stopT-startT) + " milisegundos");
}
```

Imagen de Consola:



9.3.1.5.4: ElGammal

```
public static void testTime_x_n_ElGam(int n){
        BigInteger decryption=null;
        BigInteger m=null;
        LinkedList<BigInteger> c = new LinkedList();
        long miliEnc=0, miliDes=0;
        long startT = System.currentTimeMillis();
        for (int i=0; i<n;i++){
        /** Inicialización de ElGammal**/
                EGKeyGen EGkg= new EGKeyGen();
                ElGammal_privateKey key= EGkg.ElGammalKey(245,14656546);
                ElGammal_p esystem = new ElGammal_p();
                esystem.setDecryptEncrypt(key);
                m=BigInteger.valueOf(1);
        /** Inicio cálculo de tiempo de Encriptación **/
                long start = System.currentTimeMillis();
                c=esystem.encrypt_EG(m);
                long stop = System.currentTimeMillis();
                miliEnc=miliEnc+(stop-start);
                try{
                        Thread.sleep(1);
                }catch(InterruptedException e){
                        System.out.println("Thread Interrupted");
                }
        /** Inicio calculo de tiempo de Desencriptación**/
                start = System.currentTimeMillis();
                decryption=esystem.decrypt_EG(c);
                stop = System.currentTimeMillis();
                miliDes=miliDes+(stop-start);
                try{
                        Thread.sleep(1);
                }catch(InterruptedException e){
                        System.out.println("Thread Interrupted");
                }
        }
        long stopT = System.currentTimeMillis();
        System.out.println("\tMedia Encriptacion: "+miliEnc/n+ "
milisegundos");
        System.out.println("\tMedia Desencriptacion: "+miliDes/n+"
milisegundos");
        System.out.println("\tTiempo Prueba: "+ (stopT-startT)+"
milisegundos");
}
```

Imagen de Consola: