UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN

IMPLEMENTACIÓN DE MÓDULO DE SELECCIÓN POR *RANKING* PARA EL SISTEMA DE VOTACIÓN ELECTRÓNICA EN LA PLATAFORMA *PARTICIPA UCHILE*

MEMORIA PARA OPTAR AL TÍTULO DE
INGENIERA CIVIL EN COMPUTACIÓN

FERNANDA CATALINA MACÍAS HERRERA

PROFESOR GUÍA:
ALEJANDRO HEVIA ANGULO

MIEMBROS DE LA COMISIÓN:
TOMÁS BARROS ARANCIBIA
GONZALO NAVARRO BADINO

SANTIAGO DE CHILE
2023

Resumen

Existen distintos sistemas para llevar a cabo una elección en la que cierta cantidad de candidatos compite para ganar un lugar en determinada cantidad de cargos equivalentes. En Chile los más utilizados son el sistema de mayorías con posibilidad de segunda vuelta y sistema el de D'Hondt. Sin embargo, dado el contexto social y político en el que se encuentra el país, resulta interesante estudiar otras formas de votación. Una buena opción, reconocida internacionalmente, es el sistema de votación con *ranking* preferencial.

En votación con *ranking* cada votante ordena a los candidatos por preferencia en forma descendente, entregando al sistema una permutación de las opciones. Obtener esta información desde el claustro elector permite lograr mayor representatividad, incluso para aquellos votantes que prefieren candidatos de muy baja popularidad.

Existen distintas formas de calcular los resultados en votación con *ranking*. Dentro de estas destaca *Single Transferable Vote*, también conocido como STV. Este prioriza a aquellos votos que no han conseguido un representante electo, pues pretende lograr una representación proporcional. *Shuffle-Sum* es un algoritmo que permite implementar STV manteniendo la privacidad de los votos y la verificabilidad del proceso, disminuyendo al máximo el riesgo de coerción.

Por otra parte, `Participa UChile` es un servicio en línea de votación electrónica y consultas universitarias implementado en el contexto del *software* `Psifos`. Este proyecto está desarrollo y su equipo está en constante búsqueda de la modernización y extensión de sus funciones, bajo la prioridad de cumplir altos estándares de seguridad, privacidad y transparencia. Con base en lo anterior, se planteó como objetivo de este trabajo de título diseñar e implementar un módulo de votación con *ranking* que cumpla con los atributos de privacidad y verificabilidad y que sea fácilmente integrable en el `Participa UChile`.

El resultado de este trabajo fue un módulo de votación con *ranking* preferencial incorporable en el *back-end* de `Participa UChile`. Este módulo calcula sus resultados mediante *Shuffle-Sum* y utilizando como esquemas de encriptación *El Gamal* y *Damgard Jurik*. Si bien el módulo resultante presenta algunas limitaciones, su diseño es extensible, presentándose como una buena prueba de concepto. En particular, el diseño permite la incorporación de una *mix network*, para realizar procesos de mezcla, y la generación de *zero knowledge proofs* para asegurar la verificabilidad el proceso.

A todas las mujeres ingenieras, presentes y futuras.

Agradecimientos

Quiero expresar mi más profundo agradecimiento a mi madre Mabel, mi padre David, mi hermana Ignacia, mi perro Fidel y a toda mi familia por su amor incondicional y apoyo constante. Gracias por siempre alentarme a confiar en mis capacidades y por ser un pilar en este camino universitario.

No puedo dejar de agradecer a Luisa y Marisol, quienes con su cariño y compañía me ayudaron a sobrellevar los días de estudio en casa. Gracias por estar ahí y ayudarme a hacer este proceso más llevadero.

También quiero agradecer a mis compañeras y compañeros de Beauchef, con quienes compartí la experiencia de aprender, crecer y alcanzar nuestras metas universitarias. Especialmente a Gabriela y Joaquín, por ser amigos y acompañarme en este recorrido.

Quiero agradecer a mi amiga de la infancia, Claudia, por su apoyo incondicional y por siempre celebrar mis logros conmigo. Gracias por ser una fuente de inspiración y motivación.

Mi más sincero agradecimiento a Niñas Pro y todas sus integrantes por brindarme un espacio sororo y de aprendizaje mutuo en un ambiente hostil para las mujeres. Gracias por ayudarme a confiar en mis capacidades.

Finalmente, no puedo dejar de agradecer a mi pareja Franco, quien con su amor, soporte y compañía me inspiró y motivó para dar lo mejor de mí en esta última etapa universitaria. Gracias por tu apoyo incondicional en lo personal, académico y profesional.

Quiero extender mi agradecimiento al CLCERT, especialmente a Camilo Gómez y mi profesor guía Alejando Hevia, por su disposición, conocimiento y simpatía desde el inicio de este trabajo. Sin su ayuda, este logro no hubiera sido posible.

Tabla de Contenido

Índice de Tablas

Índice de Ilustraciones

Capítulo 1

Introducción

1.1. Tipos de votación

Llevar a cabo una elección de representantes es un tema complejo, pues por un lado se deben cumplir ciertos requisitos de correctitud y, a la vez, el proceso debe debe ser confiable para todos los participantes. A lo largo del tiempo, investigadoras e investigadores han intentado definir las características de una buena elección. Por ejemplo, Benaloh et al. señalan que "en elecciones de todo tipo es importante que los votos sean contados correctamente, que los votos individuales sean privados, que los votantes estén libres de coerción, y que todas estas propiedades no solo estén presentes, sino que también sean claramente evidentes para todos" [2]. Todas estas características están directamente relacionadas con el sistema que se utiliza para recibir los votos, calcular el resultado y declarar las opciones ganadoras.

En Chile los sistemas de elección más populares son el "mayoritario con segunda vuelta", utilizado en las elecciones presidenciales, y el sistema *D'Hont*, utilizado en elecciones parlamentarias. Ambos sistemas se basan en un mecanismo presencial que, en general, asegura las características señaladas en el párrafo anterior. Sin embargo, la experiencia señala que estas cualidades no garantizan que el resultado obtenido sea el que represente mejor a la mayoría de los votantes.

Supóngase una elección presidencial donde un votante debe escoger entre varias opciones, dentro de las que hay dos con altas probabilidades de ganar. Sean estas A y B, siendo B la opción que este votante definitivamente no quiere que gane, y sea C la opción preferida del votante. Al votar, esta persona se plantea dos casos. El primero es votar por C, pese a que sus posibilidades de victoria sean bajas, y el segundo es votar por A, para evitar que B gane. Es decir, el votante debe elegir entre emitir un voto a favor de su primera preferencia y uno en contra de su última preferencia.

Este mismo caso se puede replicar en una elección parlamentaria, donde el votante tiene como opciones sumar puntos al partido político con el que simpatiza o sumar puntos al partido que asegurará que sean electos la menor cantidad de candidatos posibles pertenecientes al partido con el que definitivamente no simpatiza.

Finalmente, los votantes se encuentran ante el problema de decidir cuál de las dos opciones es la que mejor representa su opinión. Pivato señala que "quizás el problema es que las reglas de votación no están usando toda la información sobre las preferencias de los votantes, porque solo vemos las primeras opciones, no vemos sus otras preferencias" [1].

La votación con *ranking* preferencial es un tipo de sistema electoral que pretende subsanar esta falta de información. En esta, cada votante ordena a los candidatos por preferencia en forma descendente, entregando al sistema una permutación de los candidatos. Luego, se aplica un algoritmo que calcula el resultado y, finalmente, se publica la o las opciones ganadoras.

1.2. Votación con ranking preferencial

Uno de los algoritmos más destacados para calcular el resultado de una votación con *ranking* es *Single Transferable Vote* (STV). Benaloh et al. señalan que "se puede utilizar para cubrir una o múltiples vacantes, logrando en este último una forma de representación proporcional" [2]. Actualmente STV es utilizado para la elección de representantes en países como Australia, Canadá y Estados Unidos.

Existen distintas estrategias mediante las que se puede utilizar STV para calcular el resultado de una elección. Probablemente la manera más intuitiva sería que cada votante señale su permutación de candidatos en una papeleta, esta se encripte, se mezcle con el resto y, finalmente, se desencripte una a una, para implementar STV a partir de los textos planos de todas las papeletas. Al implementar esta estrategia, no hay manera de conocer quien fue el emisor de un voto. Sin embargo, esto no garantiza la privacidad del voto, ya que existen riesgos asociados a la posibilidad de coerción.

En votación con *ranking*, una estrategia de coerción bastante conocida es el "Ataque Italiano". Este consiste en que un votante marca en su papeleta una permutación de candidatos poco probable dentro del conjunto de votantes, de modo que quien ejerce la coerción pueda identificar que el votante emitió el voto solicitado. Para obtener más información sobre este ataque dirigirse a la sección 3.2. Ante este riesgo surge la necesidad de un algoritmo que implemente STV, pero sin dar demasiada información sobre el contenido de los votos. Además, resulta imperante que este filtro de información no le reste verificabilidad al proceso.

Shuffle-Sum es un algoritmo que pretende implementar STV, haciéndose cargo de este conflicto entre coerción y verificabilidad. La estrategia que emplea para lograr este objetivo es utilizar distintas representaciones de las papeletas a lo largo del proceso, para no mostrar más información que la necesaria en cada paso. Esta característica hace que el procedimiento sea más complejo y demoroso, por lo que solo es viable utilizar *Shuffle-Sum* en votaciones electrónicas.

1.3. Participa UChile

`Participa UChile`[1] es un servicio en línea de votación electrónica y consultas universitarias implementado en el contexto del *software* `Psifos`[2]. Esta herramienta ha sido desarrollada por el Laboratorio de Criptografía y Ciberseguridad de la Universidad de Chile (CLCERT)[3]. Su objetivo es cumplir con altos estándares de seguridad, privacidad y transparencia, para ser utilizada por la universidad en instancias de bajo riesgo.

Resulta importante considerar que `Participa UChile` utiliza como base `Helios`, un servicio de votación electrónica de código abierto. Este ha sido utilizado para la formación de distintos servicios de votación a través del mundo, llegando a ser un estándar para las elecciones en línea que buscan cumplir con los requisitos de privacidad y verificabilidad.

1.4. Objetivos

A raíz del contexto señalado, se propone como objetivo general diseñar e implementar un módulo de votación con *ranking* que cumpla con los atributos de privacidad y verificabilidad y que sea fácilmente integrable en `Participa Uchile`. Más detalladamente, se plantean como objetivos específicos:

1. Diseñar un marco de solución para votación con *ranking* que sea compatible con las características de `Participa Uchile`.
2. Implementar o adoptar una implementación existente del algoritmo STV usando una *mix network*, para mantener el anonimato de los votos.
3. Implementar o adoptar una implementación existente del algoritmo STV usando el protocolo *Shuffle-Sum*.
4. Modularizar el desarrollo del protocolo, para que sea fácilmente integrable a `Participa UChile`.

La utilidad e importancia de esta propuesta radica principalmente en dos puntos. Por una parte, este trabajo significa un aporte y consolida una importante alternativa para llevar a cabo los procesos electorales de la Universidad de Chile. Por otra parte y considerando que en la actualidad no existe una una plataforma consolidada derivada de `Helios` que incorpore votación con *ranking*, este proyecto se configura como un aporte a la comunidad internacional.

[1] `https://participauchile.cl/`
[2] `https://github.com/clcert/psifos-backend-info`
[3] `https://www.clcert.cl/`

Capítulo 2

Herramientas criptográficas y protocolos

2.1. Esquemas de encriptación

En general, se pueden dividir los esquemas de encriptación en dos tipos, los de clave privada y los de clave pública. Estos últimos son los más frecuentes en implementaciones de votación electrónica, por lo que vale la pena estudiarlos más a profundidad.

La criptografía de clave pública se basa en el uso de dos claves matemáticamente relacionadas entre si, una pública y una privada. Su funcionamiento consiste en que el destinatario genera este par de claves, compartiendo la pública y manteniendo en secreto la privada. Luego, un emisor encripta un mensaje utilizando la clave pública y lo envía al destinatario, quien finalmente utiliza la clave privada para desencriptarlo y conocer su contenido. Algunos esquemas de encriptación de clave pública son *Damgard Jurik* y*ElGamal*.

Por un lado, *Damgard Jurik* es una generalización del esquema de encriptación Paillier que calcula la operación de módulo sobre n^{s+1}, siendo n un módulo RSA y s un número natural. Este esquema encripta un mensaje $m \in \mathbb{Z}_{n^s}$, escogiendo un número aleatorio $r \in \mathbb{Z}^*_{n^s}$ tal que el texto cifrado de m es $g^m \cdot r^{n^s} mod\ n^{s+1}$, siendo (n, g) su clave pública y un valor d su clave privada.

Damgard Jurik es aditiviamente homomórfico, esto quiere decir que se puede obtener la encriptación de la suma de dos textos planos a partir de sus textos cifrados, aun sin conocer el valor de estos textos planos. Para más información de este esquema de encriptación dirigirse a *A Generalization of Paillier's Public-Key System with Applications to Electronic Voting*[1].

Por otro lado, *ElGamal* es un esquema que se puede definir sobre cualquier grupo cíclico, pero para este trabajo el enfoque estará en aquellos definidos sobre un grupo multiplicativo de enteros módulo n. Este esquema encripta un mensaje $m \in \mathbb{Z}_p$ utilizando un valor aleatorio $b \in \{2, ..., p-1\}$, de modo que el texto cifrado resulta ser la dupla $(g^b\ mod\ p, K^b m\ mod\ p)$,

[1] `https://www.researchgate.net/publication/225753264_A_generalization_of_Paillier's_public-key_system_with_applications_to_electronic_voting`

donde la clave pública es (g, p, K) y la privada un valor a.

Un aspecto importante de *ElGamal* es que su privacidad depende de la dificultad del atacante para resolver el problema del logaritmo discreto. A diferencia de *Damgard Jurik*, *ElGamal* es multiplicativamente homomórfico, es decir, se puede obtener la encriptación de la multiplicación de dos textos planos a partir de sus textos cifrados. Para más información de este esquema de encriptación dirigirse a *A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms*[2].

Existe una variación de *ElGamal*, llamada *ElGamal Exponencial*. Principalmente, este consiste en encriptar con *ElGamal*, pero considerando como texto plano $g^m mod\ p$, siendo g el generador del grupo y m el mensaje que se desea encriptar. De igual manera, al desencriptar un texto cifrado, se utiliza *ElGamal* y posteriormente se calcula el logaritmo discreto de este valor. A diferencia de su versión original, *ElGamal Exponencial* es aditivamente homomórfico.

Tanto *Damgard Jurik* como *ElGamal* y *ElGamal Exponencial* pueden ser utilizados como esquemas de encriptación umbral. Esto significa que la información puede ser protegida encriptándola y distribuyéndola entre un grupo de computadores tolerantes a fallas. Una forma de implementar esto en votación electrónica es entregar claves privadas a distintas personas para que, a la hora de desencriptar, ellos generen desencriptaciones parciales que en conjunto formen el texto plano.

2.2. Mix Networks

Una *mix network* (MN) es un protocolo que recibe múltiples textos cifrados, los mezcla y devuelve los mismos aunque desencriptados y reordenados aleatoriamente, de modo que ningún agente externo puede relacionar el orden de entrada con el de salida. En general, las MN son verificables, es decir, existe un mecanismo que permite a cualquier agente externo comprobar la correctitud de la operación a partir de la información disponible.

En votación electrónica se suele utilizar un tipo de *mix network* llamado *mix network de reencriptación* (RMN). Ésta tiene como característica que el conjunto de salida no es descifrado, sino sólo reencriptado. Por lo tanto, los textos cifrados de salida parecen cifrados nuevos, pero corresponden al mismo conjunto de entrada permutado. Para más información sobre las RMN dirigirse a *Networks Without User Observability - Design Option*[3].

Una de las implementaciones más conocidas y mejor ponderadas de RMN es la `Verificatum Mix-Net`[4], esta se caracteriza por trabajar con *ElGamal* como esquema de encriptación. Pese a la existencia de modelos teóricos, no se conocen implementaciones eficientes de RMN que utilicen *Damgard Jurik*.

[2] `https://people.csail.mit.edu/alinush/6.857-spring-2015/papers/elgamal.pdf`
[3] `https://link.springer.com/content/pdf/10.1007/3-540-39805-8_29.pdf`
[4] `https://www.verificatum.org/html/product_vmn.html`

2.3. Zero knowledge proof

Una *Zero knowledge proof* (ZKP) es un método mediante el que una entidad puede probar a otra que una declaración determinada es verdadera, evitando transmitir más detalle que la veracidad de la declaración misma. Una ZKP está compuesta por dos algoritmos: uno para producir la evidencia (denominada la *proof*) a partir de los valores privados, y otro para verificar la *proof*, sin utilizar los valores privados. En votación electrónica las ZKP tienen distintos usos. Por ejemplo, se pueden utilizar para demostrar la validez de un voto sin revelar su contenido. Es decir, se pueden utilizar como garantía de que el voto emitido cumple con los requerimientos de cantidad y forma de las alternativas marcadas, cantidad y representación de los candidatos, etc. Otra caso de uso en votación electrónica es en la verificación de la correctitud de procesos aleatorizados, esto es, que el proceso descrito fue calculado correctamente sin revelar claves ni valores usados en la aleatoriedad. Por ejemplo, al usar una *mix network* se utiliza una *zero knowledge proof* para demostrar que los textos planos detrás de los cifrados dados como *input* a la *mix network*, tienen una correspondencia con los textos planos en los textos cifrados entregados como salida de la *mix network*, sin revelar la correspondencia concreta, esto es, la permutación.

Capítulo 3

Votación con ranking preferencial

Actualmente existen distintos sistemas de votación con *ranking*, dentro de los que destacan tres. El primero es *Instant-runoff voting*, el que está pensado para utilizarse en elecciones de las que, existiendo más de dos postulantes, resulta un único ganador o ganadora. Aquí, el procedimiento para obtener la opción ganadora es recursivo. En cada iteración se elimina al candidato que aparece como primera preferencia en la menor cantidad de papeletas, y su caso base es cuando solo queda una opción, la que es declarada como ganadora. Algunos países donde ha sido utilizado para la elección de cargos gubernamentales son Australia, Estados Unidos e Irlanda.

El segundo sistema es *Borda Count*, que permite escoger una o más opciones ganadoras dentro de varias opciones postulantes. Lippman et al. [3] señalan que este sistema "a veces se describe como un sistema de votación basado en el consenso, ya que a veces puede ser elegida una opción más ampliamente aceptable que la que cuenta con el apoyo de la mayoría". Es decir, una opción que contiene la mayoría de votos de opción favorita en primer lugar podría no resultar electa. A esto se le llama que no cumple el criterio de mayoría. Actualmente, el *Borda Count* es utilizado en Nauru, nación de las islas del Pacífico.

Por último, el tercer destacado es *Single Transferable Vote* (STV), que permite llenar una o múltiples variables indistinguibles entre sí. Benaloh et al. [2] mencionan que "intenta reducir la cantidad de votos desechados". Si bien cumple con el criterio de mayoría, lamentablemente posee la cualidad de ser especialmente susceptible a coerción. Actualmente, STV es utilizado de manera no electrónica en países como Australia, a nivel senatorial; Canadá, a nivel municipal, y Estados Unidos, principalmente a nivel universitario. Dado que STV permite seleccionar a más de un ganador o ganadora (al tener uno o más puestos disponibles) y que cumple con el criterio de mayoría, se considera la mejor opción para implementar el módulo de selección por *ranking*.

3.1. Single Transferable Vote

STV tiene como entrada una papeleta por elector, donde cada papeleta está compuesta por un arreglo de votos, es decir, una secuencia de candidatos ordenada por preferencia. Para más claridad considérese como ejemplo de papeleta la expresión 3.1, donde el candidato 5 es *rankeado* primero, luego el candidato 2, luego el 1, el 3 y finalmente, el 4. Además, a cada papeleta se le asigna un peso, que inicialmente es 1 para todas.

$$5, 2, 1, 3, 4 \tag{3.1}$$

Una vez recibida la entrada se calcula el número mínimo de votos en primer lugar que debe recibir un candidato para ser electo. Esta cifra es una constante llamada cuota y se simboliza con la letra q. Considerando n como el número de votos válidamente emitidos y s como el número de puestos a los que se están postulando, se define la cuota como se muestra en 3.2.

$$q \leftarrow \left\lfloor \frac{n}{s+1} \right\rfloor + 1 \tag{3.2}$$

La cuota se calcula una única vez y se mantienen constante para todo el proceso.

El cálculo de los resultados en STV se realiza mediante un algoritmo iterativo que posee dos casos base. El primero es cuando todos los puestos fueron llenados y el segundo es cuando el número de candidatos restantes es igual al número de puestos restantes. Cada iteración se lleva a cabo en cuatro pasos y concluye indicando qué candidatos son elegidos, es decir, asignados a los cupos disponibles. A continuación se explican los pasos de los que se compone cada iteración.

1. **Computar la primera preferencia**
   Para cada candidato se suma el peso de las papeletas que contengan a ese candidato como primera opción.

2. **Elegir o eliminar candidatos**
   Todo candidato que alcanzó la cuota calculada (3.2) en el peso de sus votos de primera preferencia, esto es, el cálculo realizado en el paso anterior, es declarado electo. Si ningún candidato alcanza la cuota, se elimina al candidato con la menor cantidad de votos de primera preferencia.

3. **Reponderar votos**
   Para cada candidato electo, se reponderan las papeletas en las que este aparece como primera preferencia, es decir, se multiplica su peso por el valor de transferencia.
   Para un candidato electo el valor de transferencia se define como lo muestra 3.3.

   $$t \leftarrow \frac{m-q}{m} \tag{3.3}$$

   Donde m es el valor calculado para dicho candidato en el cómputo de la primera preferencia (1). Para los candidatos que no fueron electos se considera que el valor de transferencia es 1.

4. **Eliminar candidatos**
   Los candidatos que fueron eliminados o electos en la ronda se eliminan de todas las papeletas.

De cada iteración resulta cada papeleta con su permutación sin los candidatos eliminados en dicha iteración y en las anteriores, y con un peso igual o diferente, dependiendo de si debió ser reponderada o no. Estas papeletas y la misma cuota calculada inicialmente son utilizadas para llevar a cabo la siguiente iteración. Se puede observar el pseudocódigo del proceso explicado en el Apéndice A y un ejemplo del mismo en el Apéndice B.

3.2. Implementaciones Single Transferable Vote

A partir de la definición de la misma, surge como primera opción implementar STV utilizando una *mix network* (protocolo presentado en 2.2). Este proceso consiste en recibir los votos, mezclarlos utilizando la *mix network*, desencriptarlos y obtener el contenido de las papeletas para aplicar el algoritmo, manteniendo el anonimato del emisor de cada voto.

En general, las implementaciones conocidas para STV, como la mencionada en el párrafo anterior, funcionan correctamente. Sin embargo, el análisis de estas ha mostrado que existe un permanente conflicto entre verificabilidad y privacidad. Esto es porque normalmente lograr verificabilidad implica hacer pública información contenida en las papeletas, abriendo la posibilidad de que algún atacante que ejerza coerción pueda verificar la efectividad de su presión en la información publicada. A su vez, mantener privada la información de cada papeleta dificulta convencer a los participantes de que el proceso fue realizado honestamente.

Benaloh et al. [2] señalan que "si todos los votos individuales son finalmente revelados, entonces esto produce un problema de coerción, a veces llamado *ataque italiano*". Este consiste en incitar a un votante a ingresar como preferencia determinada permutación de los candidatos. En ésta, el primer lugar corresponde al candidato al que se desea favorecer, seguido de una permutación de candidatos poco probable dado el contexto de las elecciones. Así, el adversario lograría identificar esta anomalía dentro de la información revelada y, por ende, el cumplimiento de su objetivo, esto es, confirmar que el votante ha seguido sus instrucciones.

Dependiendo del tipo y la cantidad de información que se revele de cada papeleta, el *ataque italiano* puede presentar variaciones. En general, las variaciones consisten en agregar determinada cantidad de candidatos que con muy baja probabilidad serán electos antes de la permutación especificada en el párrafo anterior.

Hoy por hoy, existen modelos teóricos de STV que pretenden resolver el conflicto entre verificabilidad y privacidad, intentando disminuir la probabilidad de coerción. Uno de ellos es *Shuffle-Sum*, protocolo que implementa STV calculando el resultado de manera verificable y sin revelar la información entregada por los votantes. La estrategia utilizada para lograr este objetivo es representar las permutaciones entregadas por lo votantes de distintas maneras en las distintas etapas del proceso.

3.3. Shuffle-Sum

3.3.1. Explicación del algoritmo

Para llevar a cabo el conteo de una elección con *Shuffle-Sum* las papeletas se deben poder representar de las siguiente cuatro formas.

1. **Representación por orden de candidatos:** Consiste en una lista ordenada por el identificador de los candidatos, cada uno de ellos acompañado de la encriptación del número de preferencia en la que se sitúa dicho candidato. En esta papeleta su peso se representa encriptado una sola vez. Se puede observar un ejemplo de esta representación en la Tabla 3.1.

2. **Representación por orden de preferencia:** Consiste en una lista ordenada por las preferencias de los candidatos, cada una de ellas acompañada de la encriptación del número de candidato al que se sitúa en dicha preferencia. En esta papeleta su peso se representa encriptado una sola vez. Se puede observar un ejemplo de esta representación en la Tabla 3.2.

3. **Representación de primera preferencia:** Se basa en la estructura definida en la representación por orden de candidatos, la única diferencia es que, en ésta, el peso no se representa sólo una vez. Aquí a cada candidato se le asigna un peso. Aquel candidato que ocupe el primer lugar de preferencia tendrá asignada la encriptación del peso de la papeleta, mientras que el resto tendrá asignada la encriptación de 0. Se puede observar un ejemplo de esta representación en la Tabla 3.3.

4. **Representación de la eliminación de los candidatos en C:** Esta representación se basa en la representación por orden de preferencia. La primera diferencia con esta última es que aquí las preferencias en que se sitúa cada candidato están encriptadas. Siendo C un conjunto de candidatos a eliminar, la segunda diferencia es que cada candidato tiene asignado un valor encriptado que indica si pertenece al conjunto C o no. Este valor es la encriptación de 1 si debe ser eliminado y la encriptación de 0 en el caso contrario. Se puede observar un ejemplo de esta representación en la Tabla 3.4.

Candidato	1	2	...	m
Preferencia	$E(\sigma_v(1))$	$E(\sigma_v(2))$	...	$E(\sigma_v(m))$
Peso	$E(w_v)$			

Tabla 3.1: Representación por orden de candidatos, donde $E(x)$ corresponde al elemento x encriptado, v es la papeleta sobre la que se está trabajando, $\sigma_v(x)$ corresponde a la preferencia en la que se sitúa el elemento x en dicha papeleta y w_v es el peso de la papeleta.

Para llegar a cabo *Shuffle-Sum*, se requiere que las distintas representaciones de las papeletas se realicen utilizando un esquema de encriptación umbral. Es decir, debe ser de clave

Candidato	$E(\sigma_v^{-1}(1))$	$E(\sigma_v^{-1}(2))$	...	$E(\sigma_v^{-1}(m))$
Preferencia	1	2	...	m
Peso	$E(w_v)$			

Tabla 3.2: Representación por orden de preferencia, donde $E(x)$ corresponde al elemento x encriptado, v es la papeleta sobre la que se está trabajando, $\sigma_v^{-1}(x)$ corresponde al candidato que se sitúa en la preferencia x y w_v es el peso de la papeleta.

Candidato	1	...	i	...	m
Preferencia	$E(\sigma_v(1))$	...	$E(\sigma_v(i))$	...	$E(\sigma_v(m))$
Peso	$E(0)$	...	$E(w_v)$	...	$E(0)$

Tabla 3.3: Representación de primera preferencia, donde $E(x)$ corresponde al elemento x encriptado, v es la papeleta sobre la que se está trabajando, $\sigma_v(x)$ corresponde a la preferencia en la que se sitúa el elemento x en dicha papeleta y w_v es el peso de la papeleta.

pública y la información se debe proteger distribuyéndola entre un grupo de computadores tolerantes a fallas. Además, este debe ser aditivamente homomórfico. Esto significa que se debe poder obtener la encriptación de la suma de valores a partir de la encriptación de estos valores y sin conocer los valores originales (sin encriptar).

Tal como se mencionó en la sección anterior, cada iteración de STV se compone de cuatro pasos. Cumpliendo con los requisitos descritos, estos se pueden llevar a cabo computacionalmente implementando *Shuffle-Sum*, como se explica a continuación.

1. **Computar la primera preferencia:** Desde las papeletas en su representación de primera preferencia se puede calcular directamente el conteo de primera preferencia para un candidato, sumando homomórficamente los valores encriptados para ese candidato en todas las boletas en primera preferencia.

2. **Elegir o eliminar candidatos:** Se descifra el conteo de la primera preferencia. A partir de lo anterior, se puede definir el conjunto de candidatos electos y/o el conjunto C de candidatos a eliminar.

3. **Reponderar votos:** El valor de transferencia de cada candidato se calcula públicamente utilizando los conteos reveladas de primera preferencia. Luego, el peso de cada voto de primera preferencia para ese candidato debe ser multiplicado por el valor de transferencia. Para no revelar qué votos son reponderados, se multiplica cada uno de los pesos por el valor de transferencia del candidato y homomórficamente se suman los pesos. Como en esta representación de la papeleta todos los pesos son 0 a excepción del situado en la columna de la primera preferencia, el resultado de esta suma es el peso real de la papeleta.

4. **Eliminar candidatos:** Se comienza con la representación de las papeletas en su forma por orden de los candidatos. Esto entrega un indicador encriptado para cada candidato que indica los candidatos eliminados (1 eliminados, 0 otros). Luego, se encriptan los valores de preferencia, pasando a la representación de eliminación de candidatos en C. A partir de esto y del valor de preferencia encriptado de cada candidato se puede

Candidato	$E(\sigma_n^{-1}(1))$	...	$E(\sigma_n^{-1}(i))$	...	$E(\sigma_n^{-1}(m))$
Preferencia	$E(1)$	...	$E(i)$	...	$E(m)$
Eliminado	$E(0)$	...	$E(1)$	...	$E(0)$
Peso	$E(w_v)$				

Tabla 3.4: Representación de eliminación de candidatos en C, donde $E(x)$ corresponde al elemento x encriptado, v es la papeleta sobre la que se está trabajando, $\sigma_v^{-1}(x)$ corresponde al candidato que se sitúa en la preferencia x y w_v es el peso de la papeleta.

obtener homomórficamente el número de candidatos eliminados que le preceden en el *ranking*, para calcular el nuevo valor de preferencia de cada candidato. Por último, se pasan las preferencias encriptadas a su representación por orden de los candidatos y se eliminan los candidatos en el conjunto C y sus valores de preferencia.

La implementación de STV mediante *Shuffle-Sum* explicada se puede entender graficamente observando las figuras 3.1 y 3.2.

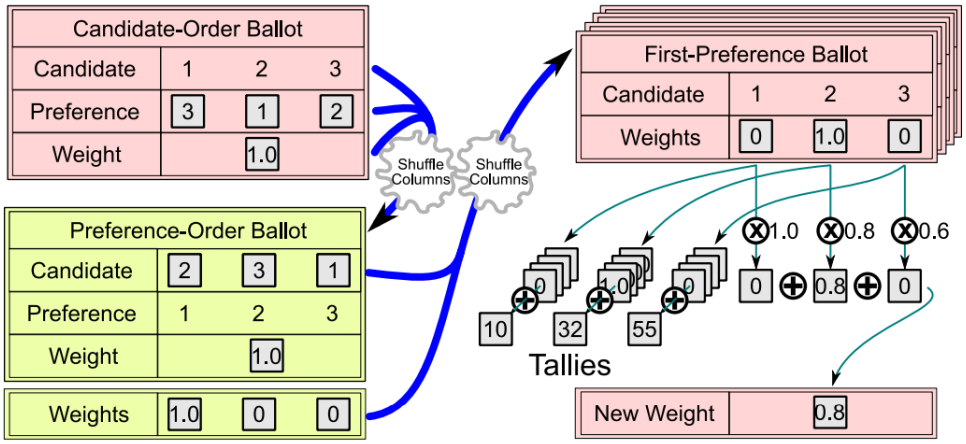


Figura 3.1: Ejemplo de conteo y reponderación en *Shuffle-Sum* (imagen obtenida del paper *Shuffle-Sum: Coercion-Resistant Verifiable Tallying for STV Voting* [2]).

Se puede observar el pseudocódigo del proceso explicado en el apéndice C y un ejemplo de este mismo en el apéndice D.

3.3.2. Posible optimización

Un aspecto importante a considerar al trabajar con grandes cantidades de papeletas es que el proceso de *Shuffle-Sum* puede llegar a ser muy costoso en tiempo. Esto se debe a que en cada iteración, cada papeleta debe pasar repetidamente de una representación a otra. En este sentido y analizando el proceso completo, se puede observar que la parte más costosa del esquema es la eliminación de candidatos. Sin embargo, esta se puede hacer más eficiente utilizando una estructura de datos llamada *tabla de comparaciones*. Esta tiene como ventaja

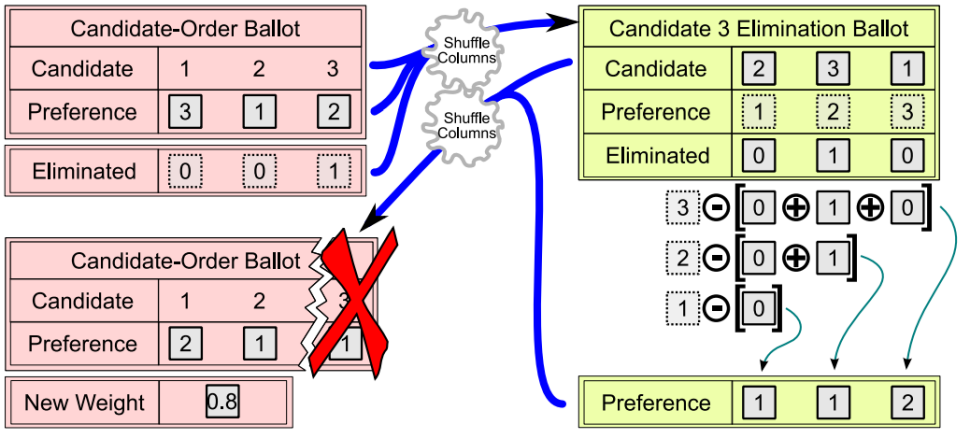


Figura 3.2: Ejemplo de la eliminación de un candidato en *Shuffle-Sum* (imagen obtenida del paper *Shuffle-Sum: Coercion-Resistant Verifiable Tallying for STV Voting* [2]).

que permite una construcción muy eficiente de una papeleta, pero utilizando mucho más espacio que la estrategia original ($O(m^2)$, donde m es el número de candidatos).

La *tabla de comparaciones* consiste en un peso W y una matriz V de tamaño $m \times m$. En esta, cada celda está independientemente encriptada, la diagonal de la matriz es insignificante y el resto de los valores se define como se muestra en la expresión 3.4. En esta se considera $E(x)$ como el valor encriptado de x.

$$V_{i,j} = \begin{cases} E(-1) & \textit{si el candidato } i \textit{ es preferido por sobre } j \\ E(0) & \textit{en el caso contrario} \end{cases} \tag{3.4}$$

La manera de eliminar los candidatos presentes en un conjunto C utilizando la *tabla de comparaciones* consiste en ignorar las filas y columnas correspondientes a los candidatos eliminados. Así, la actualización de las preferencias de una papeleta en su representación por orden de candidatos consiste en trabajar la *tabla de comparaciones*, de modo que para cada candidato que sigue en competencia, j, se suman homomórficamente los valores V_{ij} para todos los candidatos i que siguen en competencia. A esto se le suma 1, para que el *ranking* parta desde 1. Se puede observar un ejemplo de la etapa de eliminación de candidatos utilizando *tabla de comparaciones* en el apéndice E.

La ventaja de utilizar esta estructura por sobre la formula original es que, la *tabla de comparaciones* permite recuperar de manera eficiente la suma homomórfica de sus elementos. Así, una vez eliminados los candidatos correspondientes, la redistribución de los candidatos en las preferencias se puede llevar a cabo sin revelar qué candidatos están siendo redistribuidos y sin mezclarlos. Además, se utiliza la misma estructura durante el todo el proceso, sin necesidad de recalcular sus valores en cada iteración.

Cabe destacar que, al igual que en la implementación original, la utilización de esta estructura permite que los usuarios ingresen permutaciones incompletas en sus papeletas. Es decir, pueden escoger incluir ene el *ranking* a algunos candidatos y a otros no. En dicho caso un candidato que no fue incluido simplemente no es electo ni eliminado en ninguna parte del

proceso.

3.3.3. Ajustes de implementación

Un aspecto importante a considerar es que los pesos de todas las papeletas inicialmente son 1, y que sus valores van variando a medida que son multiplicados por distintos valores de transferencia. Por consecuente, la forma en que varía un peso depende directamente de los valores de transferencia con los que se ha operado hasta el momento. Al mismo tiempo, resulta importante considerar que para mantener el secreto de los votos, es necesario que los pesos individuales se mantengan encriptados, ya que de lo contrario un atacante podría obtener información sobre la cantidad de papeletas cuyas primeras opciones ya fueron electas o descartadas. Por el contrario, los valores de transferencia son públicos en todo momento.

De lo mencionado resulta que, los valores de transferencia corresponden a número racionales y, en muchas ocasiones, decimales. Esto significa, que los pesos asociados a las papeletas con alta probabilidad pueden llegar a ser valores decimales. El problema producido por esto es que la encriptación de números decimales y la operación de textos cifrados con valores decimales no resulta trivial. La solución existente para mantener oculto el valor de los pesos y poder llevar a cabo el proceso correctamente es aplicar una variación en la manera en que se operan los pesos con los valores de transferencia, para que los pesos correspondan a la encriptación de valores enteros a lo largo de todo el algoritmo.

La lógica detrás de esta solución es que cada valor de transferencia (racional) se puede representar como la división de dos enteros, a/b, de modo que la reponderación de las papeletas asociadas a dicho valor de transferencia puede consistir en dos pasos:

1. Multiplicar por a el peso de todas las papeletas donde el candidato, dueño del valor de transferencia, fue electo como primera opción.
2. Multiplicar por b el peso de todas las papeletas donde el candidato no fue electo como primera opción.

La manera de implementar este proceso para todos los valores de transferencia, $s_1, ..., s_N$, con sus respectivas representaciones $\frac{a_1}{d_1}, ..., \frac{a_N}{d_N}$, es calcular un común denominador $d' = lcm(d_1, ..., d_N)$ y, a partir de él, calcular nuevas representaciones para cada valor de transferencia, $\frac{a'_1}{d'}, ..., \frac{a'_N}{d'}$. Así, para cada papeleta, si su candidato i en primera preferencia fue electo, su peso se multiplica por a'_i, mientras que, en el caso contrario, se multiplica por d'. Esto último porque el valor de transferencia de los candidatos que no fueron electos se puede ignorar o considerar como 1. Finalmente, la cuota q se multiplica por d'.

La modificación explicada es paralelizable y permite llevar a cabo el proceso de reponderación mediante la operación de textos planos enteros, manteniendo el valor de los textos planos de los pesos dentro del conjunto de los enteros.

Capítulo 4

Contexto del desarrollo

4.1. Herramientas previas

4.1.1. Psifos y Participa UChile

Tal como se mencionó en el capítulo 1, la propuesta de este trabajo consiste en crear un módulo de votación con *ranking* que sea fácilmente integrable en `Participa UChile`. Para esto resulta importante comprender las herramientas con las que se cuenta y las restricciones que esto presenta. En primer lugar, resulta importante considerar que `Participa UChile` realiza elecciones que tienen como objetivo obtener un ganador dentro de varias opciones candidatas, utilizando el tradicional sistema de mayorías.

Para llevar a cabo este tipo de elecciones se cuenta con la implementación de *ElGamal* como esquema de encriptación, esto significa que se dispone de mecanismos de generación de claves, encriptación y desencriptación. Sin embargo, el criptosistema utilizado es *ElGamal Exponencial*, esto tiene como finalidad aprovechar su propiedad de suma homomórfica. La manera en que se utiliza *ElGamal Exponencial* a partir de *ElGamal* es que, el servidor de `Participa UChile` recibe de cada votante la encriptación bajo *ElGamal* de g^m, siendo g el generador y m el voto emitido, para luego desencriptar los valores que sean necesarios utilizando *ElGamal* y calcular el logaritmo discreto.

Otro aspecto importante es que cada proceso de elección cuenta con tres custodios de clave. Estos son tres representantes encargados de generar un par de claves, almacenando la privada, para luego utilizarla en la generación de desencriptaciones parciales. Estas 3 desencriptaciones posteriormente se mezclan, generando una desencriptación total. Esta es la manera en que *Participa UChile* implementa criptografía umbral, término explicado en la sección 2.1. Con base en todo lo anterior, se utiliza el siguiente mecanismo para llevar a cabo la elección:

1. Cada votante selecciona su opción de preferencia.
2. La opción se encripta de manera local y se envía al servidor de `Participa UChile`.

3. Una vez reunidos todos los votos, con ayuda de los custodios, se obtiene el resultado final.

Actualmente existen dos maneras de obtener el resultado final, ya contando con los votos cifrados. La primera consiste en aprovechar la propiedad homomórfica de *ElGamal Exponencial*. Para esto, se suman homomórficamente los votos recibidos para cada candidato y posteriormente se desencripta el total de votos recibidos por cada uno de ellos y se determina la opción ganadora. La segunda consiste en aprovechar las propiedades de una *RMN*. Esto consiste en mezclar y reencriptar los votos obtenidos, para posteriormente desencriptarlos y calcular públicamente los resultados. Esta opción sigue la lógica utilizada en una elección presencial, donde los votos son doblados y mezclados en una urna, para posteriormente ser revisados uno a uno.

Para llevar a cabo el proceso señalado en la segunda opción del párrafo anterior, `Participa UChile` cuenta con una *RMN*, instancia de `Verificatum`. Esta funciona como una `API`, que recibe consultas con los cifrados que se quiere trabajar y retorna una respuesta con los mismos valores mezclados y reencriptados.

Finalmente, se puede mencionar que la arquitectura física de `Participa UChile` se formula como lo presenta la imagen 4.1.

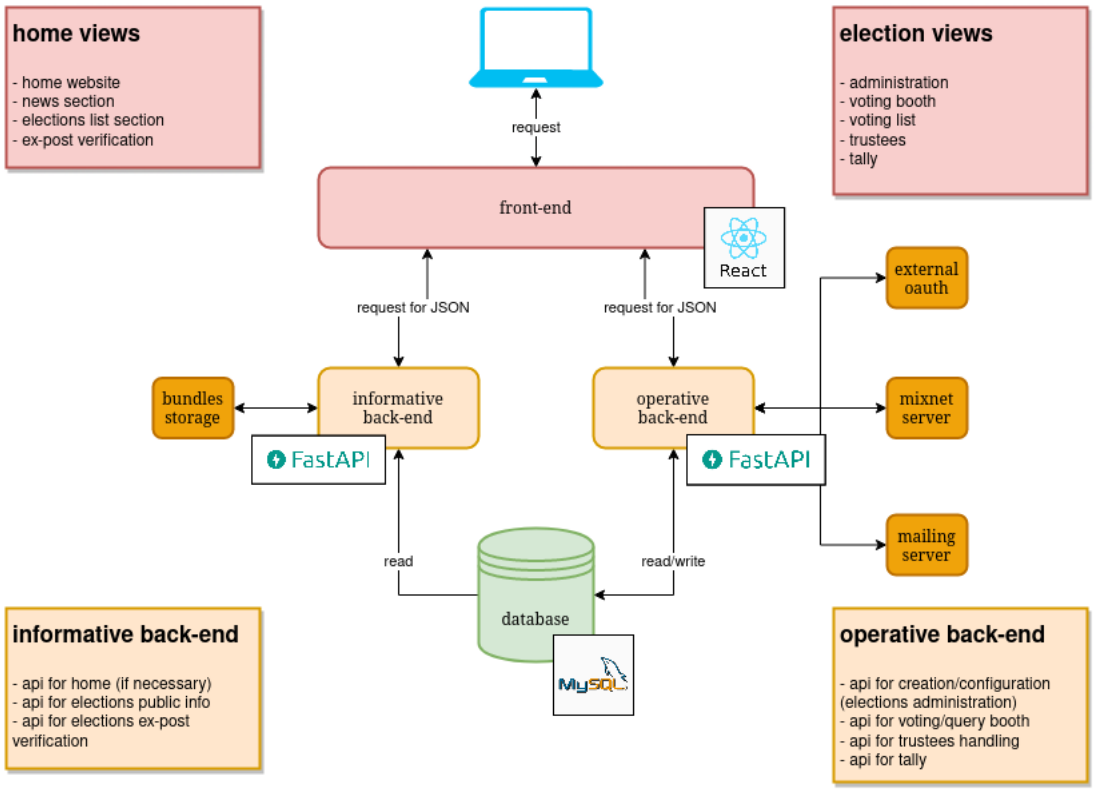


Figura 4.1: Arquitectura física `Participa UChile`.

4.1.2. Trabajos relacionados

Nicholas Boucher, Luka Govedič, Pasapol Saowakon y Kyle Swanson son los creadores de una implementación en `Python`[1] de *Shuffle-Sum* bajo la licencia MIT[2]. Esta implementación se caracteriza por los siguientes aspectos:

- La eliminación de candidatos se realiza mediante el esquema original, es decir, no utiliza *tablas de comparaciones*.
- Posee clases de `Python` que representan los distintos tipos de papeletas.
- Posee distintas funciones que implementan los distintos protocolos necesarios para llevar a cabo una elección. Estos protocolos son los especificados en el apéndice C.
- Su entrada consiste en un archivo con la información general de la elección (cantidad y nombre de los candidatos) y otro archivo con el texto plano de cada papeleta.
- Utiliza como esquema de encriptación *Damgard Jurik*.
- Realiza los procesos de mezcla utilizando el método `randbelow` de la librería `secrets`[3] de `Python`. La estrategia utilizada para mezclar consiste en ir secuencialmente escogiendo elementos al azar e ir construyendo una nueva permutación.
- No genera *ZKP* para encriptaciones, desencriptaciones ni mezclas.

Se considera que esta implementación es un buen punto de partida para alcanzar los objetivos propuesto, pues implementa de manera clara los complejos protocolos que en conjunto conforman *Shuffle-Sum*. Por lo tanto, se propone trabajar en base a esta implementación y aplicar las siguientes modificaciones:

- Modificar el sistema de recepción de papeletas para que estas se reciban y encripten de manera individual, con la finalidad de mantener la privacidad de las preferencias de cada votante.
- Incorporar el concepto de custodio, definido en la sección 4.1.1, para que el trabajo realizado sea compatible con el sistema de encriptación umbral utilizado en Participa UChile.
- Utilizar una *RMN* para realizar la mezclas de elementos dentro de una papeleta, con la finalidad de reducir la posibilidad de ataques en los cambios de representación de cada papeleta.
- Agregar la funcionalidad de permitir la verificabilidad, por medio de ZKP, de las encriptaciones, desencriptaciones y procesos de mezcla.
- Cambiar el esquema de encriptación a *ElGamal Exponencial*, para que sea compatible con `Participa UChile`.

[1] `https://github.com/cryptovoting/shuffle-sum`
[2] `https://mit-license.org/`
[3] `https://docs.python.org/3/library/secrets.html?highlight=secrets#secrets.randbelow`

Se considera que tras la realización de estos cambios, el código resultante sería incorporable a la lógica por el lado del servidor de `Participa UChile`.

4.2. Análisis de costos

Anteriormente se comentó que existen dos posibles maneras de implementar la etapa de eliminación de candidatos en *Shuffle-Sum*. La primera corresponde al esquema original, presentado en la sección 3.3.1, y la segunda es la que la lleva a cabo utilizando la estructura llamada *tabla de comparaciones*, presentada en la sección 3.3.2. Como se mencionó anteriormente, esta última presenta como ventaja que disminuye la cantidad de veces que se debe pasar de una representación de papeleta a otra.

En el artículo en el que se define *Shuffle-Sum* [2] se presenta un análisis de costos en memoria que comparan las dos posibles implementaciones. En este, Benaloh et al. concluyen que "el uso exclusivo de papeletas por orden de candidatos es más eficiente. Sin embargo, es posible que, para algunos métodos de ingreso de votos, para elecciones con aproximadamente tantas rondas como candidatos, valga la pena introducir la estructura de datos de papeletas en su forma de *tabla de comparaciones*" [2]. Sin embargo, queda pendiente el análisis de costos en tiempo. Para este, se observa que los procesos que se llevan a cabo en ambas versiones son principalmente de dos tipos. El primero es el computo de operaciones homomórficas, mientras que el segundo es el paso de papeletas desde una representación a otra.

En cuanto al computo de operaciones homomórficas, se puede observar que este es determinista y verificable en si mismo. Es decir, el resultado siempre será el mismo y, por ende, la manera de verificarlo será revisar el cálculo realizado. Por otro lado, el costo en tiempo será lineal en la cantidad de elementos operados. Ahora bien, el caso del paso de una papeleta desde una representación a otra no es tan directo de analizar. Pues, tal como se puede observar en el apéndice C, cada uno de estos procesos está compuesto por varios subprocesos. A grandes rasgos, estos últimos pueden ser de encriptación, desencriptación, mezcla de las columnas de la papeleta, orden de las columnas de la papeleta y computo de operaciones homomórficas.

Por un lado, los subprocesos de encriptación y desencriptación dependen directamente del criptosistema que se utilice. Sin embargo, se puede mencionar que su tiempo de ejecución está dentro de $O(n)$. Por otro lado, el subproceso de ordenar las columnas dentro de una papeleta constituye una tarea que se puede realizar bajo algún algoritmo de ordenamiento, por lo que su tiempo de ejecución se estima dentro de $O(n \cdot log(n))$. Por último, se observa que el subproceso de mezclar las columnas de una papeleta es un poco más delicado. Pues, realizarlo directamente como la reordenación de una permutación podría hacer pública información sobre las preferencias marcadas por algún votante, posibilitando la ocurrencia de algún ataque.

Para evitar posibles ataques y problemas de coerción, se propone como medida implementar la mezcla de columnas dentro de una papeleta utilizando una *mix network*. Esto se considera una buena opción, dado que garantiza aleatoriedad y verificabilidad. Sin embargo, la utilización de esta herramienta podría significar un costo importante en cuanto a tiempo.

En vista de esto y de un análisis global del algoritmo se obtiene que, dentro de los subprocesos definidos, los más significativos en cuanto a tiempo y ocurrencias serían los de desencriptación y mezcla de las columnas de la papeleta.

Para estudiar estos subprocesos se hace necesario definir las variables con las que se trabajará. Aquí, resulta importante considerar la experiencia de `Participa UChile` en las elecciones que han llevado a cabo hasta el momento. Tras un proceso de recaudación de datos se estima que la cantidad de votantes por elección varía entre las 10 y las 50.000 personas. A su vez, se estima que el número de candidatos en las distintas elecciones varía entre 1 y 30 personas.

Con base en lo anterior y a la posibilidad de dividir a los votantes en distintas mesas, tal como se hace en los sistemas presenciales, se estable como caso de estudio un conjunto de votantes que varía entre 1 y 1000 personas. A su vez, considerando que existe una barrera de usabilidad para *rankear* a demasiados candidatos, se decide que se estudiará el caso en que hay a lo más 8 candidatos. Por último, en vista del peor caso, se estima que el número de iteraciones que se realizará en cada elección será igual al número de candidatos.

Así, se analizó la cantidad de veces que se mezclan las columnas de una papeleta (número de accesos a la *mix network*) y la cantidad de desencriptaciones. Esto se hizo contando las veces que se realiza cada acción durante todo el proceso para la implementación completa de *Shuffle-Sum*, contrastando aquella que realiza la eliminación de candidatos original con aquella que la realiza por medio de una *tabla de comparaciones*. Los resultados obtenidos se muestran en la tabla 4.1.

	Accesos a la MN	Desencriptaciones
Esquema original	$4nm + n$	$(4n + 1)m(m + 1)/2 + nm$
Esquema con TC	$2nm + n$	$(2n + 1)m(m + 1)/2 + 2nm(m - 1)$

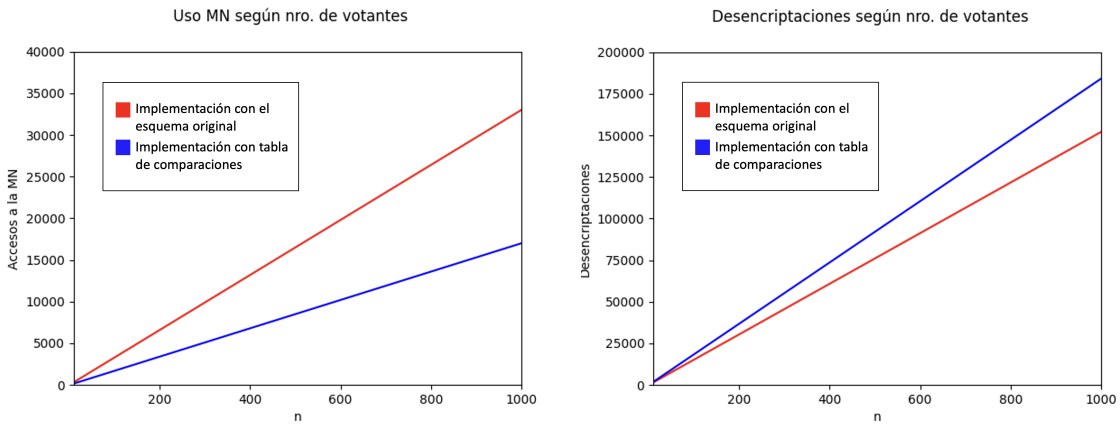
Tabla 4.1: Número de veces que se realiza cada acción cuando hay n votantes y m candidatos, teniendo tantas iteraciones como candidatos.

Para analizar de manera más clara la situación, se graficaron los resultados considerando la existencia de 8 candidatos. El resultado de este proceso se puede observar en la figura 4.2.

Los resultados obtenidos muestran que la implementación con *tabla de comparaciones* se comporta mejor en cuanto a accesos a la *mix network*, mientras que el esquema original se comporta mejor en cuando a número de desencriptaciones. Como el resultado obtenido no mostró mayor eficiencia de una implementación por sobre la otra se hizo necesario llevar a cabo un análisis cuantitativo de la situación.

Para esto se realizaron experimentos que miden el tiempo que demora en trabajar una *mix network* y el tiempo que demora desencriptar valores, utilizando las herramientas proporcionadas por `Participa UChile`. A partir de lo anterior, se estimó el tiempo que demora cada una de las implementaciones en obtener los resultados de una elección. Este resultado se puede observar en la figura 4.3 y en la tabla 4.2.

Los resultados obtenidos muestran que la implementación que utiliza *tabla de comparación* es más eficiente en tiempo. Considerando esto se propone modificar el código base para que



(a) Análisis de accesos a la *mix network*. (b) Análisis del nro. de desencriptaciones.

Figura 4.2: Cantidad de acciones en *Shuffle-Sum* vs número de votantes.

n	**Tiempo esquema original**	**Tiempo con TC**
10	19 min 47 seg	10 min 22 seg
50	1 hrs 38 min 53 seg	51 min 49 seg
100	3 hrs 17 min 36 seg	1 hrs 43 min 47 seg
500	16 hrs 28 min 55 seg	8 hrs 38 min 15 seg
800	1 día 2 hrs 22 min 16 seg	13 hrs 49 min 12 seg
1000	1 día 8 hrs 57 min 50 seg	17 hrs 16 min 30 seg

Tabla 4.2: Número de veces que se realiza cada acción cuando hay n votantes y 8 candidatos, teniendo tantas iteraciones como candidatos.

utilice el esquema con *tabla de comparaciones*.

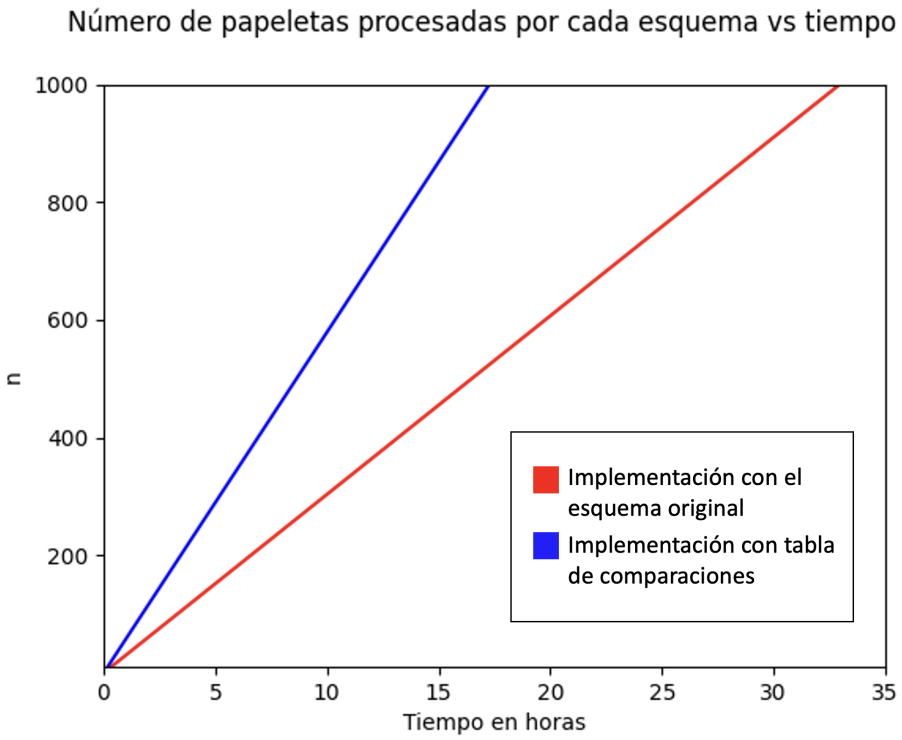


Figura 4.3: Gráfico que estima el tiempo que demora cada implementación en actuar en función del número de votantes, donde la línea roja corresponde al que utiliza la implementación original y la línea azul corresponde a la que utiliza *tabla de comparaciones*.

Capítulo 5

Implementación módulo de votación con ranking

5.1. Implementación esquema con tabla de comparaciones

Tal como se explica en la sección 3.3.2, una *tabla de comparaciones* se puede resumir como una matriz rellena con la encriptación de 0 y la encriptación de -1. Esto significa que, al implementar esta estructura, se debe encriptar valores negativos y desencriptar sumas entre valores negativos, es decir, otros valores negativos. Pese a que existe la manera de encriptar valores negativos utilizando *ElGamal Exponencial*, se considera que este proceso se podría volver confuso si se quisiera utilizar un criptosistema diferente. Con el objetivo de crear un módulo de votación con *ranking* preferencial extensible y que funcione independiente de su esquema de encriptación, se propone definir la *tabla de comparaciones* como se muestra en la expresión 5.1.

$$V_{i,j} = \begin{cases} E(1) & si\ el\ candidato\ i\ es\ preferido\ por\ sobre\ j \\ \\ E(0) & en\ el\ caso\ contrario \end{cases} \tag{5.1}$$

Al utilizar esta estructura modificada se evita la encriptación y operación de valores negativos para calcular las preferencias correspondientes a cada candidato. Finalmente, el código base fue modificado para que incorpore y utilice como representación de la papeleta la *tabla de comparaciones* modificada.

5.2. Subdivisión del módulo

Un objetivo implícito de este trabajo fue garantizar el desarrollo de una implementación segura, más allá del algoritmo mismo. En el tutorial *The Protection of Information in Com-*

puter Systems[1] se especifican 8 principios para proteger información del uso indebido o la modificación no autorizada. Dentro de estos, hubo dos principios que guiaron el cumplimiento de este objetivo.

El primero, fue el *Principio de Menor Cantidad de Mecanismos Comunes*. Este recomienda no compartir estados o variables entre muchas partes del sistema. Saltzer et al. explican que "Cada mecanismo compartido (especialmente uno que involucre variables compartidas) representa una ruta de información potencial entre los usuarios y debe diseñarse con mucho cuidado para asegurarse de que no comprometa la seguridad de forma involuntaria" [4].

El segundo fue el *Principio de economía de mecanismos*. Este recomienda mantener el sistema lo más pequeño y simple posible. Para esto, Saltzer et al. argumentan que "los errores de diseño e implementación que resultan en rutas de acceso no deseadas no se notarán durante el uso normal (ya que el uso normal generalmente no incluye intentos de ejercer vías de acceso indebidas). Como resultado, son necesarias técnicas como la inspección línea por línea del software y el examen físico del hardware que implementa mecanismos de protección. Para que tales técnicas tengan éxito, es esencial un diseño pequeño y simple." [4].

De acuerdo con los principios presentados, se diseñó un módulo de votación con *ranking* simple y seccionado. Para este diseño también se consideró que `Participa UChile` es un sistema que está en desarrollo y que, por ende, está en constante evolución. Es por esta razón que las secciones del módulo de votación se diseñaron de tal forma que fueran lo suficientemente claras, concisas e intuitivas como para que alguien sin conocimientos acabados de *Shuffle-Sum* pueda realizar cambios.

5.2.1. Procedimiento seccionado

El procedimiento que lleva a cabo `Participa UChile` en una elección de cualquier tipo se puede resumir en los siguientes tres pasos.

1. *Registro de la elección:* Se crea la elección en el sistema, especificando sus características principales, y los custodios generan las claves, almacenando la privada de manera local (en su computador o en un pendrive, por ejemplo) y compartiendo la pública con el servidor.
2. *Recepción de los votos:* Se abre la elección y los votantes ingresan sus preferencias.
3. *Obtención de los resultados:* Se cierra la elección, se corre el algoritmo que calcula los resultados con la participación de los custodios y se publican.

Como se mencionó anteriormente, el módulo de votación con *ranking* pretende formar parte de la lógica por lado del servidor. Con base en esto, se definieron 5 agentes que pretenden llevar a cabo los tres pasos recién mencionados, desde la perspectiva del *back-end*, para *ranking* preferencial. Estos son:

[1] https://www.cl.cam.ac.uk/teaching/1011/R01/75-protection.pdf

1. *Run:* Sección que dirige el proceso de elección de inicio a fin. Orquesta la interacción de los distintos agentes en la ejecución del proceso.
2. *Trustees:* Sección encargada de manejar procesos en los que se involucra a los custodios. Es decir, la generación de claves y la desencriptación de elementos.
3. *Crypto:* Sección encargada de dirigir las operaciones criptográficas. Este se define según el esquema de encriptación que se utilice.
4. *Election data:* Sección que maneja los *input* de la elección. Es decir, su información general (candidatos y puestos disponibles) y las papeletas emitidas por los votantes.
5. *Shuffle-Sum:* Sección encargada de calcular el resultado implementando el algoritmo *Shuffle-Sum*.

La manera en que los agentes interactúan entre sí se muestra en el diagrama de secuencias de la Figura 5.1. El detalle del diagrama se irá explicando a lo largo de las siguientes secciones.

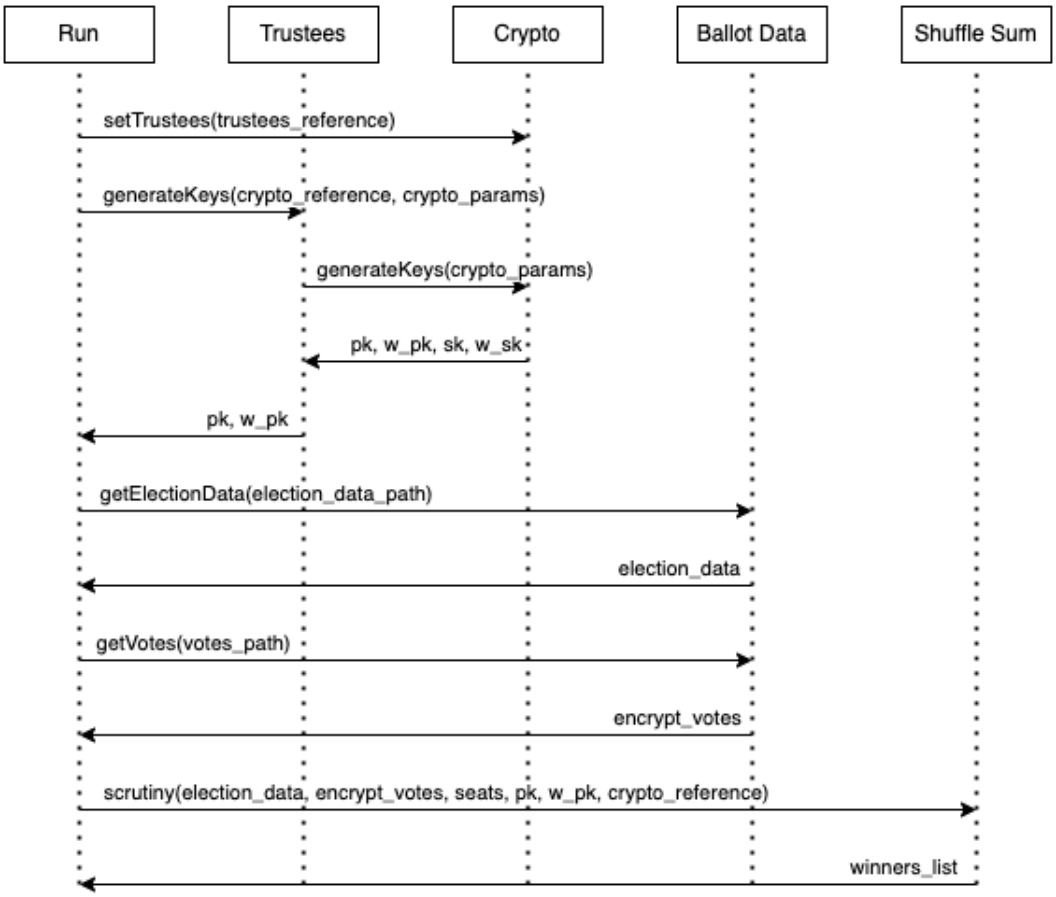


Figura 5.1: Diagrama de secuencias que detalla el procedimiento definido para el módulo de votación con *ranking*.

5.2.2. Estructura seccionada

La manera de implementar la lógica explicada fue mediante un código también seccionado. Esto se tradujo en una estructura bien definida que implementa y ejecuta de manera explícita a los distintos agentes. Este diseño se muestra en la Figura 5.2.

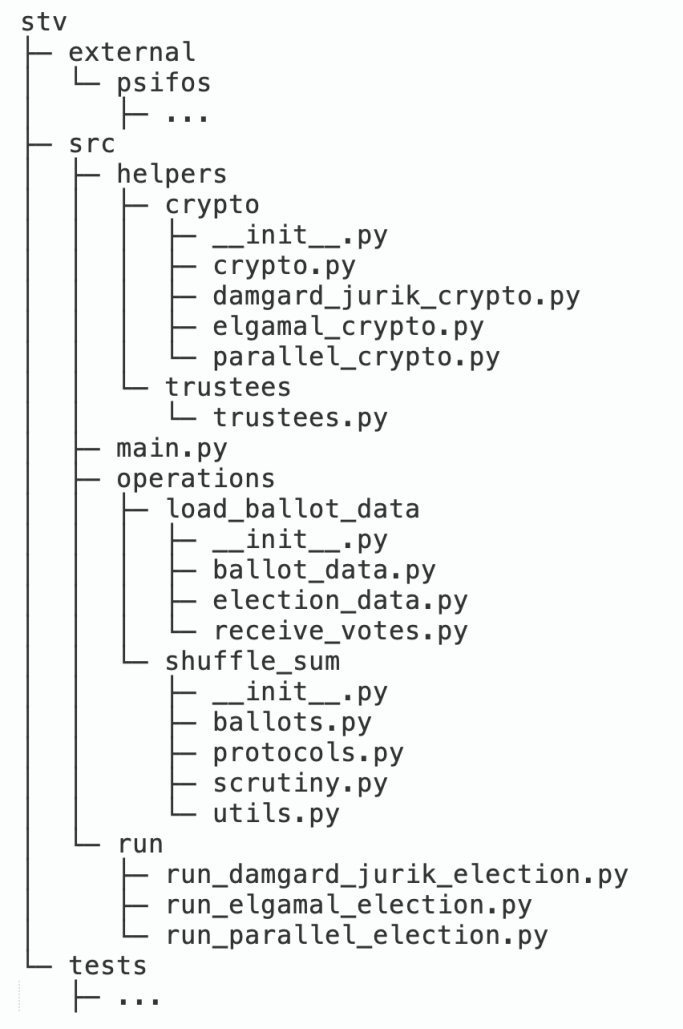


Figura 5.2: Diagrama de secuencias que detalla el procedimiento definido para el módulo de votación con *ranking*.

A grandes rasgos, el código *STV* se divide en tres grandes grupos. El primero es *external*, donde se encuentran las herramientas externas al proyecto, en este caso la implementación de *ElGamal* extraída de *Psifos*. El segundo es *src*, que contiene las estructuras y protocolos que permiten llevar a cabo una elección con *ranking*. El tercero es *tests*, que contiene elecciones de prueba que se pueden ejecutar para verificar la correctitud de los ganadores.

A su vez, *src* se divide en varias secciones. Por un lado está *helpers*, sección en que se definen estructuras a las que es necesario acudir durante la ejecución del algoritmo, como lo son custodios y criptosistemas. Luego, está *operations*, donde se encuentran estructutas y protocolos que permiten la obtención de la data de una elección, su procesamiento y el cálculo del respectivo resultado mediante *Shuffle-Sum*. Por último está *tests*, sección for-

mada por ejecutables que permiten calcular el resultado de una elección bajo determinado criptosistema.

5.3. Generalización esquema de encriptación

La propuesta presentada contempla la implementación de *ElGamal* para utilizar *ElGamal Exponencial* como criptosistema. Sin embargo, existe la posibilidad de que en el futuro se prefiera un criptosistema diferente. Para abordar este caso, se utilizó el patrón de diseño *Double Dispatch* para implementar *Crypto* (agente 3).

Así, se creó una clase de `Python` *Crypto*, que sirve como interfaz de todos los posibles agentes *Crypto*, señalando las operaciones con las que debe contar para poder implementar *Shuffle-Sum*. De esta clase hereda *ElGamalCrypto*, que implementa las diferentes operaciones según corresponde al criptosistema. Luego, se crearon distintos agentes *Run* (agente 1), que corren el proceso con los parámetros correspondientes al esquema de encriptación que se esté utilizando.

En resumen, y tal como se explica en la figura 5.1, el agente *Run* entrega una referencia del esquema de encriptación que se está utilizando a los custodios y al algoritmo de obtención de resultados. Finalmente, existe la opción de ejecutar *run_elgamal_election* (para utilizar *ElGamalCrypto*) o *run_damgard_jurik_election* (para utilizar *DamgardJurikCrypto*). Para entender la utilidad de *run_parallel_election* y *ParallelCrypto*) hace falta más contexto, por lo que su importancia se explica en el capítulo 6.

5.4. Simulación criptografía umbral

Como se mencionó anteriormente, `Participa UChile` cuenta con encriptación umbral. Esta característica se ha incorporado de tal forma que, existe una cantidad determinada de custodios o *trustees* encargados de generar de manera independiente un par de claves, de modo que las claves públicas se unen para llevar a cabo encriptaciones, mientras que las claves privadas se almacenan por separado. Así, la desencriptación de un elemento consiste en obtener la desencriptación parcial por parte de cada custodio y generar la desencriptación total, uniendo todas las parciales.

En `Participa UChile` los custodios corresponden a usuarios que generan las claves y las utilizan por medio de la aplicación web a la hora de hacer la desencriptación parcial. Tal como se explicó anteriormente, si se quisiera utilizar *Shuffle Sum*, sería necesario realizar varias desencriptaciones consecutivas, por lo que cada custodio debería ingresar la clave una cantidad inviable de veces.

Para una implementación final, resulta imperante idear una estrategia de desencriptación que traiga consigo una buena experiencia de usuario. Sin embargo, para esta prueba de concepto se decide simular la presencia de custodios, de modo que estos tengan la capacidad de realizar desencriptaciones parciales todas las veces que sea necesario.

La forma en que esto se llevó a cabo fue mediante la incorporación de un agente *Trustees* (agente 2), quien representa al conjunto de custodios y, por lo tanto, tiene la capacidad de generar claves y desencriptar textos cifrados. Para mayor simplicidad se consideró que el conjunto de custodios se conforma por un único usuario. Esto fue posible ya que, bajo este supuesto, el agente *Trustees* contiene las mismas entradas y salidas que contando con más de un custodio.

Se puede analizar el comportamiento de este agente observando el diagrama de la figura 5.1. Aquí, un primer aspecto a destacar es que *Trustees* recibe una solicitud de generación de claves, que desencadena la generación según corresponda al criptosistema utilizado. Luego, *Trustees* almacena la(s) clave(s) privada(s) y comparte la pública. Otro aspecto a destacar es que el agente *Crypto* recibe una referencia al agente *Trustees* y, a la vez, el agente *Shuffle-Sum* recibe una referencia al agente *Crypto*. Esto se traduce en que *Shuffle-Sum* contiene una referencia a *Trustees*, para poder solicitar la desencriptación de elementos cuando sea necesario. Todo esto fue logrado mediante la utilización de una clase de `Python` *Trustees*, presente en el archivo `trustees.py`.

La forma en la que se plantea la futura incorporación de los procesos descritos en el párrafo anterior en *Participa UChile* se resume en los siguientes dos puntos:

1. El agente *Trustees* debe ser representado tal como lo han los procesos actuales de `Participa UChile`.
2. Una solicitud de generación de claves o de desencriptación debe desencadenar en una notificación al *front-end* o directamente al usuario.

5.5. Simulación shuffle

Como se mencionó anteriormente, `Participa UChile` cuenta con una RMN, creada con `Verificatum` que recibe solicitudes y las responde mediante una `API REST`. Resulta importante recordar es que esta RMN funciona únicamente para el esquema de encriptación *ElGamal*, por lo que su incorporación en el sistema debe ser específicamente para el agente *Crypto* relacionado a *ElGamal*, es decir, el método *shuffle* de la clase *ElGamalCrypto*.

Para asegurar la confiabilidad del resultado incluso a pesar de que exista un servidor malicioso, la *mix network* se compone de tres servidores, que realizan tres mezclas consecutivas. Estos servidores se sincronizan por medio de sesiones. Esto significa que, para mezclar un conjunto de tuplas, se debe inicializar cada servidor con la información del conjunto. Una vez hecho esto la RMN realiza su labor y devuelve la permutación reencriptada de las tuplas.

Inicialmente la RMN de `Participa UChile` solo era capaz de procesar una sesión. De modo que, para iniciar una segunda sesión y mezclar un segundo conjunto de tuplas era necesario reiniciar la RMN. Sin embargo, *Shuffle-Sum* requiere de la realización de muchas mezclas, por lo que no era viable reiniciar la `API` antes de cada una de ellas. Por esto, se solicitó al equipo de *Participa UChile* modificar la RMN para poder iniciar varias sesiones secuencialmente.

Una vez modificada la RMN se procedió a incorporarla en el módulo de votación con *ranking*. Sin embargo, surgió un problema asociado a que se solicitaba el inicio de una sesión de la *mix network* durante el transcurso de otra. Este problema se produjo debido a que las llamadas a la RMN se realizan en paralelo. Aquí, es importante recordar que los procesos de *shuffle* se realizan en el contexto de la transformación de una paleta desde una representación a otra y que estas son independientes y, por ende, paralelizables.

Una posible solución a este conflicto sería secuencializar las transformaciones de papeletas, pero esto aumentaría considerablemente el tiempo de ejecución. Finalmente, se decide solicitar al equipo de *Participa UChile* la modificación de la *mix network* para que soporte procesos paralelizables. Como esta solución es a largo plazo, se decide simular las llamadas a la *mix network*, pensando en realizar la incorporación efectiva en el futuro.

5.6. Simulación Zero Knowledge Proof

Actualmente `Participa UChile` genera *Zero Knowledge Proof* para cada elección realizada por el sistema. Sin embargo, se conoce que el método de conteo de dichas elecciones difiere diametralmente al método de conteo propuesto por el módulo de votación con *ranking*. Adecuar la generación de ZKPs desde un método a otro no es un trabajo directo, motivo por el cual se estima que el tiempo de desarrollo del módulo no alcanzaría para aterrizar este punto. Sin embargo, se propone como solución simular la generación de ZKPs de modo que, una vez realizado este trabajo, sea fácilmente incorporable al módulo.

Como los únicos procesos no deterministas en el desarrollo *Shuffle-Sum* son los de encriptación, desencriptación y mezcla, se decidió simular ZKPs en ellos. La metodología seguida fue incorporar en el agente *Crypto*, más específicamente en la clase *Crypto*, tres métodos que simulan la generación de ZKP para encriptación, desencriptación y mezcla, respectivamente. Estos son llamados cada vez que se realiza su proceso respectivo durante la ejecución del algoritmo. De este modo, al tener definida la generación de ZKP de alguno de estos procesos, su incorporación consistiría explicitar el algoritmo en el método correspondiente.

Capítulo 6

Manejo de pesos

6.1. Problema

La representación entera de los pesos, definida en la sección 3.3.3, tiene como consecuencia que se vuelve muy difícil acotar el máximo valor que puede llegar a alcanzar el peso de una papeleta. Sin embargo, se puede estudiar la manera en la que estos valores se comportan.

Para esto, es importante recordar que al utilizar la representación entera de un peso, el valor de transferencia asociado a un candidato se puede escribir como la división de dos valores enteros, $s_i = \frac{a_i}{d_i}$. Luego, con base en la totalidad de los valores de transferencia, $s_1, ..., s_N$, con sus respectivas representaciones, se puede calcular un común denominador $d' = lcm(d_1, ..., d_N)$ y, a partir de él, calcular nuevas representaciones para cada valor de transferencia, $\frac{a'_1}{d'}, ..., \frac{a'_N}{d'}$. Así, para cada papeleta, si su candidato i en primera preferencia fue electo, su peso se multiplica por a'_i, mientras que, en el caso contrario, su peso se multiplica por d'. A su vez, la cuota q se multiplica por d'.

De esta manera, se tiene que al iniciar la j-ésima iteración el peso de determinada papeleta será $w = 1 \cdot v_1 \cdot v_2 \cdot ... \cdot v_{j-1}$, donde v_x representa el factor por el que se multiplicó el peso en la iteración x. A partir de lo descrito en el párrafo anterior se tiene que v_x puede ser el numerador del valor de transferencia a'_i correspondiente o el denominador común d' correspondiente.

En consideración de la definición del valor de transferencia asociado a determinado candidato en determinada ronda (expresión 3.3), resulta directo definir $a_i = m_i - q$ y $d_i = m_i$, siendo q la cuota y m_i la suma de los pesos de las papeletas en las que el candidato i fue electo como primera preferencia, es decir, su conteo. De esto resulta que el valor v_x, definido en el párrafo anterior, podría crecer rápidamente respecto de su equivalente en la ronda previa, v_{x-1}. Por ende, se estima que el peso de alguna papeleta podría llegar a ser muy alto.

Considerando el valor del conteo de un candidato al computar la primera preferencia (paso 1 de *Shuffle-Sum*), resulta que en las últimas iteraciones del algoritmo, al trabajar con pesos grandes se obtendrán conteos aun mayores. Esto podría llegar a ser un problema al implementar *Shuffle Sum* con *ElGamal Exponencial*. Ya que, tal como se mencionó en la sección 3.3, en el paso 2 se debe descifrar el conteo de los candidatos para poder compararlos

y seleccionar la primera preferencia. Esto se traduce en que en algunos casos se podría tener que calcular el logaritmo discreto de $g^m \ mod \ p$, para valores de m muy grandes.

La afirmación anterior se comprobó experimentalmente en un proceso que consistió en obtener el máximo conteo asociado a un candidato en una elección. El resultado para determinada elección, que contempla 30 papeletas, arrojó que resolver uno de los cálculos de logaritmo discreto tardaría aproximadamente 6 días. Considerando que el módulo de votación con *ranking* en desarrollo debe ser utilizable por `Participa UChile` y que el cálculo de las elecciones debe ocurrir en determinado margen de tiempo, no es viable contar con operaciones que duren días en realizarse.

La solución inmediata a este problema sería utilizar otro criptosistema. Sin embargo, no se conocen implementaciones buenas y eficientes de *mix networks* que utilicen un esquema de encriptación diferente a *ElGamal*. Además, no es posible utilizar *ElGamal* en vez de *ElGamal Exponencial* para trabajar los pesos, ya que este no es aditivamente homomórfico y, tal como se mencionó anteriormente, esta propiedad es necesaria para la implementación de *Shuffle-Sum*. En vista de lo anterior, se analizan las opciones expuestas en la sección 6.2 para sortear la dificultad expuesta.

6.2. Opciones

6.2.1. Precómputo

Aun conociendo la dificultad que implica el cálculo del logaritmo discreto para un valor alto, la primera opción intuitiva para abordar este desafío es buscar una manera eficiente de calcularlo. Para esto, se planteó la opción de contar con un pre-cómputo al que acudir durante la ejecución del algoritmo, con la finalidad de agilizar el proceso. Sin embargo, la incorporación del módulo de votación con *ranking* en `Participa UChile` pone una restricción fuerte en el sentido de que deben ser perfectamente distinguibles las elecciones en las es posible calcular el resultado, independientemente de la magnitud de conteos a la que se pueda llegar. Por eso, es necesario contar con una cota superior para el valor de los conteos en todas las iteraciones.

Con vista en lo anterior, se realizó un análisis de los márgenes entre los que permanecen los valores de los pesos en cada iteración, para posteriormente establecer márgenes que contengan a los posibles conteos. De este análisis, presente en el apéndice F, se concluye que un conteo m, arbitrario, siempre va a cumplir con lo señalado por la expresión 6.1, siendo n el número de votos válidamente emitidos.

$$m \leq n^{128} \tag{6.1}$$

Por lo tanto, se propone como solución precomputar todos los posibles valores en $A = \{g^x mod \ p : x \in \{0, 1, ..., n^{128}\}\}$ y almacenarlos en un archivo. De modo que, el cálculo del logaritmo discreto de un valor cualquiera, consiste en buscar dicho valor en el archivo creado.

Un aspecto importante de esta solución es que, al estar trabajando con multiplicaciones

módulo p, el tiempo en calcular secuencialmente las potencias de g crece linealmente con el valor de la potencia alcanzada. En consideración de esto, se puede realizar una estimación del tiempo que demoraría en calcular los elementos del conjunto A para distintos valores de n. Realizando experimentos de manera local, se determinó que calcular las 10^5 primeras potencias de g demora aproximadamente 1,6 segundos utilizando una tabla programación dinámica. Luego, si se consideran 100 votos ($n = 100$), resulta que el tiempo de creación del archivo sería de siglos.

Una solución alternativa para lograr el cálculo del logaritmo discreto es la utilización del algoritmo *Baby Step Giant Step*, también conocido como *Algoritmo de Shank*, creado para calcular logaritmos discretos de manera más eficiente. Para más información de este algoritmo dirigirse a *Cryptography: theory and practice*[1].

De esta opción es necesario considerar que, para cualquier valor de n, la implementación de *Baby Steps Giant Steps* depende de la capacidad de poder calcular y/o acceder a los valores en $B = \{(g^m)^x mod\ p : x \in \{0, 1, ..., m-1\}\}$, donde $m = \sqrt{p-1}$ y p es el valor módulo de *ElGamal*. Suponiendo que se conoce el valor de $g^m mod\ p$, se puede estimar el tiempo que tardaría calcular los valores del conjunto B, tal como se hizo para el conjunto A. De esto resulta que, considerando el valor de p utilizado por `Participa UChile` (donde $10^{616} < p < 10^{617}$), el cálculo del conjunto B también tardaría siglos.

Como opción alternativa al cálculo secuencial de las potencias se consideró utilizar GPU, transformado el problema del cálculo de muchas potencias en un problema de multiplicación de vectores. Sin embargo, los números utilizados son demasiado grandes, por lo que no se encontró una estructura de datos que permitiera representar los valores y realizar operaciones entre ellos.

6.2.2. Representar los pesos como decimales de punto flotante

Se propone almacenar los pesos como decimales en su representación de punto flotante, con 5 dígitos de precisión. Se plantea considerar cada decimal como un polinomio que se puede escribir como lo muestra la expresión 6.2, a la que se le denomina representación polinómica.

$$e_n \cdot x^n + ... + e_1 \cdot x^1 + e_0 \tag{6.2}$$

Al utilizar esta representación, los coeficientes de cada polinomio se almacenan en un arreglo ordenado. Se puede observar un ejemplo de esta representación en la figura 6.1

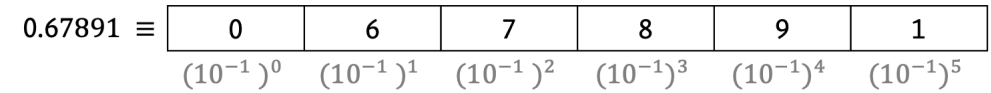


Figura 6.1: Cifra decimal 0,67891 en su representación polinómica con base 10^{-1}.

[1] `https://archive.org/details/cryptographytheo00stin_826`

Para poder utilizar esta representación en *Shuffle-Sum* es necesario recordar que este algoritmo contempla dos instancias relacionadas con el manejo de pesos de papeletas. En primer lugar está la obtención del conteo obtenido por cada candidato en competencia, en el paso 1 de STV. Esto consiste en sumar homomórficamente los pesos encriptados de determinadas papeletas. Para realizar esta operación utilizando la representación polinómica basta con sumar homomórficamente los *items* correspondientes de cada arreglo. Este proceso se ilustra en la figura 6.2.

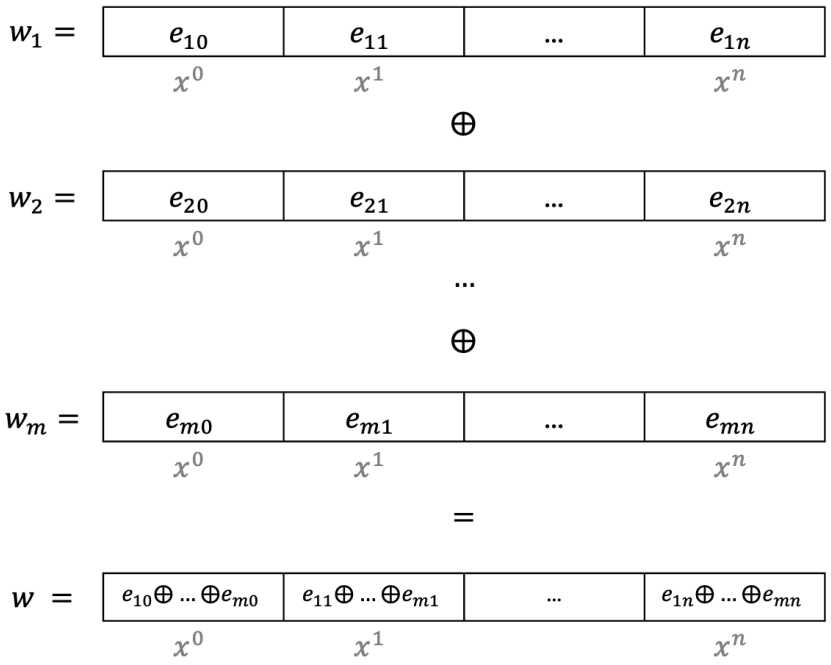


Figura 6.2: Suma homomórfica de m pesos $(w_1, w_2, ...w_m)$ en su representación polinómica, y su resultado w.

En segundo lugar se encuentra la reponderación de un peso. Esto ocurre en el paso 3 de STV, donde se multiplica el texto cifrado correspondiente a determinado peso por el valor de transferencia correspondiente, siendo este último un texto plano. La utilización de la representación polinómica para los valores de transferencia implica su utilización para los pesos de las papeletas. Esto significa que, tanto los valores de transferencia como los pesos se representarán polinómicamente, solo que en el caso de los pesos los valores almacenados en el arreglo estarán encriptados y en el caso de los valores de transferencia no. Luego, la multiplicación entre un peso y un texto plano consistiría en aplicar la fórmula matemática correspondiente a la multiplicación entre dos polinomios, para posteriormente aplicar la operación correspondiente entre un texto plano y un texto cifrado. Este proceso se ilustra en la figura 6.3.

Bajo esta lógica, cada vez que un peso se multiplica por un valor de transferencia de tamaño l, el arreglo que lo representa aumenta su tamaño en l. Es decir, considerando base 10^{-1} y cinco dígitos de precisión tanto para el peso como para los valores de transferencia, en la x-ésima iteración un peso tendría un arreglo de tamaño $6 \cdot x$. Luego, considerando que el número de iteraciones al correr el algoritmo es menor o igual a la cantidad de candidatos, y que la cantidad de candidatos definida para este módulo de votación con *ranking* varía

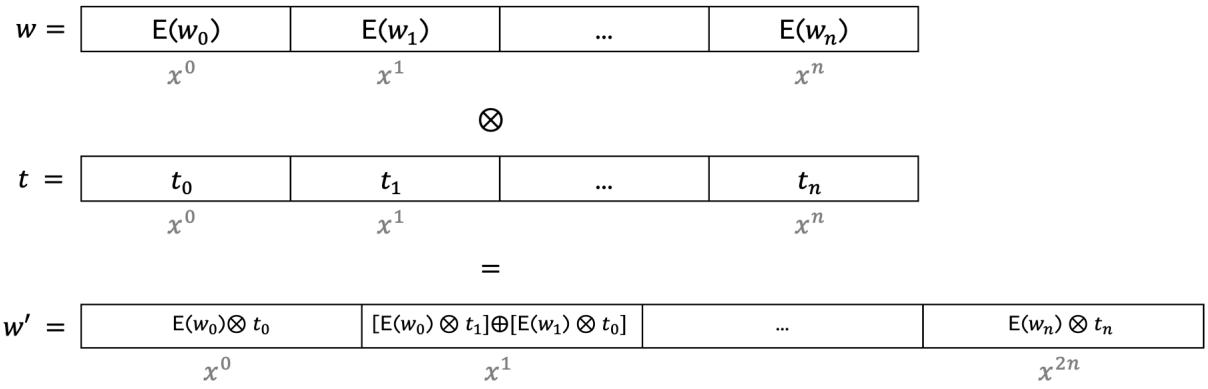


Figura 6.3: Multiplicación homomórfica de un peso encriptado w y un valor de transferencia sin encriptar t, ambos en su representación polinómica de tamaño n, y su resultado w'.

entre 1 y 8, pero que en la última ronda no hay reponderación, se concluye que el tamaño del arreglo asociado a un peso podría llegar a ser $6 \cdot 7 = 42$. El trabajar con pesos de mayor tamaño implica tener que ingresar estructuras más complejas en la *mix network* y considerando que se realizan muchas llamadas secuenciales a esta, se terminaría superando su capacidad, produciendo una denegación del servicio.

Una posible solución a este problema sería escoger una base grande, de modo que el tamaño del arreglo para cierta cantidad de cifras significativas sea menor, por ende, su tamaño en la última iteración también lo sea. Sin embargo, este aumento en la base significaría el aumento de los valores almacenados en cada espacio del arreglo. Por lo que, se tendrían que desencriptar valores mayores y, por lo tanto, se estaría incurriendo en el problema inicial.

Una segunda solución sería truncar la estructura que compone cada peso a medida que estos son reponderados. Para esto, es necesario considerar que los valores almacenados para cada índice del arreglo pueden ser mayores a la base x. Esto se traduce en que, al omitir los índices correspondientes a los exponentes más altos, en algunos casos se estaría desechando un *carry* que podría cambiar el resultado de la elección.

Para sortear este último problema se propuso desencriptar uno o más de los valores asociados a los exponentes mayores del arreglo, de modo que, al momento de truncar se pudiese rescatar el *carry* correspondiente y desechar las cifras menos significativas. Sin embargo, se descartó esta opción, ya que, tal como se muestra en la figura 6.3, el valor correspondiente al mayor exponente en el arreglo siempre será $E(w_n) \otimes t_n$. Como el valor de transferencia es calculado públicamente, revelar este valor permitiría a un atacante calcular el texto plano de uno de los coeficientes asociados al peso, revelando información sensible de una o más papeletas.

6.3. Solución final

Finalmente, se tiene que ninguna de las opciones propuestas es suficientemente sólida. Por eso, se decide implementar una representación de los pesos que, aunque no cumpla con los estándares de seguridad, sea fácil de modificar en el futuro. La estrategia utilizada para esto fue utilizar como criptosistema para pesos *Damgard Jurik* y para el resto de los elementos *ElGamal Exponencial*.

Esta solución tiene como consecuencia que los procesos de mezcla que incluyen pesos en los vectores reordenados no se pueden realizar utilizando la RMN. De manera concisa, el paso desde una papeleta en su representación *por orden de candidatos* a *por primera preferencia* que se explica en el algoritmo 2, utilizaría la RMN para su primer proceso de mezcla, pero no para el segundo. En este sentido, una mezcla se realiza tal como se define en la implementación base de *Shuffle-Sum*. Para más detalle sobre los procesos de mezcla leer la sección 5.5.

Para implementar la forma de encriptación propuesta se definió *ParallelCrypto*, clase de `Python` que hereda de *ElGamalCrypto* y que almacena un atributo w_crypto que define el criptosistema con el que manejar los pesos, en este caso *DamgardJurikCrypto*. Utilizando estas estructuras se determina que las operaciones criptográficas en pesos se realizan con un esquema y el resto con otro. Para implementar esta idea es necesario contar con dos pares de claves, una para cada esquema. En la figura 5.1 el par de claves asociadas al manejo de pesos es (w_pk, w_sk) y el par asociado al resto de elementos es (pk, sk).

Por último, cabe destacar que, por la manera en que se desarrolló el módulo de votación con *ranking*, al ejecuta el proceso se puede escoger entre utilizar *ElGamalCrypto*, *Damgard-JurikCrypto* o *ParallelCrypto*. A pesar de ello y en consideración del análisis realizado, se establece *ParallelCrypto* como el sistema de encriptación oficial para votación con *ranking* en `Participa Uchile`.

Capítulo 7

Validación

La manera de validar el módulo de votación con *ranking* resultante fue mediante diferentes elecciones. Cada una de ellas tiene características que ponen a prueba la correctitud de los resultados obtenidos ante los distintos conjuntos de entrada. Estas elecciones fueron creadas desde un inicio y se utilizaron de manera local a lo largo de todo el proceso de desarrollo, bajo los distintos criptosistemas que soporta el módulo de votación. El listado de elecciones creadas se puede resumir como se señala a continuación.

7.1. Caso base

Esta elección contempla 5 candidatos (enumerados del 1 al 5) y 5 votantes, quienes definen sus papeletas con una misma permutación. Se probó este caso considerando 5, 4, 3, 2 y 1 puestos disponibles.

7.2. Muchos votantes, muchos candidatos y con permutaciones variadas

Esta elección contempla 7 candidatos (enumerados del 1 al 7) y 100 votantes. Además, se definieron 4 permutaciones de los candidatos, siendo todas muy diferentes entre sí. Se marcaron 30 papeletas con la primera, segunda y tercera permutación, respectivamente, y se marcaron 10 con la cuarta. Por último, se definieron los conjuntos de ganadores considerando 7, 6, 5, 4, 3, 2 y 1 puestos disponibles. Cada uno de ellos correspondiente a una prueba distinta.

7.3. Muchos candidatos para pocos votantes

Esta elección contempla 4 opciones candidatas (enumeradas del 1 al 4) y 5 votantes. La distribución de permutaciones en las papeletas se dio tal que tres personas votaron diferente y las dos restantes votaron igual. Para este caso se definieron los conjuntos ganadores considerando 4, 3, 2 y 1 puestos disponibles.

7.4. Máxima cantidad de candidatos en competencia

Esta elección contempla 8 opciones (enumeradas del 1 al 8) y 50 votantes. Las papeletas se definieron tal que grupos de 11, 24, 5 y 10 personas declararon la misma preferencia, respectivamente. Con base en lo anterior, se definieron los conjuntos ganadores para 8, 7, 6, 5, 4, 3, 2 y 1 puestos disponibles.

7.5. Máxima cantidad de votantes

Esta elección contempla 5 opciones candidatas (enumeradas del 1 al 5) y 1000 votantes. Las papeletas se definieron tal que grupos de 270, 80, 110, 260, 50, 110 y 120 personas declararon la misma preferencia, respectivamente. Por último. Se definieron los conjuntos ganadores para 5, 4, 3, 2 y 1 puestos disponibles.

Capítulo 8

Conclusión

El producto resultante de este trabajo de título es un módulo de votación con *ranking* compatible con la lógica por el lado del servidor del sistema `Participa UChile`. Este se caracteriza por permitir elecciones de hasta 1000 votantes a partir de un grupo de entre dos y ocho candidatas y/o candidatos, calculando el resultado con el algoritmo *Single Transferable Vote* mediante la ejecución de *Shuffle-Sum*.

Del análisis presentado se concluye que uno de los criptosistemas más propicios para implementar *Single Transferable Vote* es *Damgard Jurik*. Sus características matemáticas hacen que sea favorable incluso para la implementación de *Shuffle-Sum*, siempre y cuando esta tolere vulnerabilidades al pasar desde la representación de una papeleta a otra. Al contrario, se considera poco recomendable utilizar *ElGamal Exponencial* como esquema de encriptación por sí solo, pues limita fuertemente la magnitud de las elecciones para las que se puede utilizar el algoritmo. Sin embargo, este último permite la utilización de una RMN, otorgando privacidad y verificabilidad al paso de la representación de una papeleta a otra. Finalmente, se concluye que, aun cuando *Shuffle-Sum* representa una buena solución a la implementación de *Single Transferable Vote*, al no contar con un criptosistema que asegure una implementación robusta, se puede afirmar que el algoritmo es perfectible.

Con base en lo expuesto en el párrafo anterior, se considera una buena opción utilizar esquemas de encriptación que funcionen en paralelo. Luego, se concluye que la estrategia implementada para el manejo de pesos es una buena opción para esta prueba de concepto. Sin embargo, resulta importante destacar que la omisión de la RMN para determinados procesos de mezcla hace que la implementación sea vulnerable a ataques. De manera concisa, la estrategia de no utilizar la RMN para mezclar las columnas de la representación *por primera preferencia* de una papeleta podría exponer información del candidato en la primera preferencia y del peso de la papeleta. En particular, la información expuesta podría facilitar la ejecución de un *ataque italiano*.

En cuanto al estudio de *Shuffle-Sum* presentado, se concluye que una de las mayores limitaciones del algoritmo es su baja capacidad de paralelización, ralentizando el proceso. Este aspecto se ve potenciado por el actual funcionamiento secuencial de la RMN utilizada.

Respecto del análisis cuantitativo de costos del producto resultante, se considera que este

no es comparable con el análisis presentado en la sección 4.2, pues la exclusión de la RMN en el proceso significa una variación importante respecto del costo real, siendo la mezcla uno de los procesos más significativos en tiempo y memoria.

Finalmente, se puede mencionar que aun cuando algunos aspectos importantes para la privacidad y la verificabilidad fueron simulados y no implementados, el producto resultante se considera un buen punto de inicio para la automatización del cálculo de resultados en procesos de votación con *ranking* preferencial. Luego, se concluye que se logró en gran medida el objetivo propuesto, puesto que se diseñó e implementó un módulo de votación con *ranking* que, en general, cumple con los atributos de privacidad y verificabilidad, y que es compatible con `Participa UChile`.

Bibliografía

[1] *Una mejor democracia es posible, Charlas del futuro*, 2020.

[2] J. Benaloh, T. Moran, L. Naish, K. Ramchen, and V. Teague. Shuffle-sum: Coercion-resistant verifiable tallying for stv voting. *IEEE Transactions on Information Forensics and Security*, 7(1):28–33, 1942.

[3] Pierce College David Lippman. *Math in Society*. David Lippman, 2.5 edition, 2017.

[4] J.H. Saltzer and M.D. Schroeder. The protection of information in computer systems. *Proceedings of the IEEE*, 63(9):1278–1308, 1975.

Anexo A

El Protocolo 1 corresponde a un pseudocódigo que muestra el procedimiento de conteo para votación con *ranking* utilizando STV.

Protocolo 1: Proceso de conteo con STV

```
1  Calcular la cuota de la elección. // Con la fórmula descrita en 3.2.
2  mientras quedan puestos restantes y el número de candidatos en competencia es
     mayor al número de puestos restantes hacer
       // 1. Computar la primera preferencia
3      para cada candidato hacer
4          Sumar el peso de las papeletas que lo contengan como primera preferencia. A
             este valor se le llama conteo.
       // 2. Elegir o eliminar candidatos
5      para cada candidato hacer
6          si al menos un candidato alcanzó la cuota entonces
7              para todo candidato que alcanzó la cuota hacer
8                  Declararlo electo.
9          en otro caso
10             Eliminar al candidato con el menor conteo.
       // 3. Reponderar votos
11     si al menos un candidato fue electo entonces
12         para cada candidato electo hacer
13             Calcular su valor de transferencia. // Con la fórmula descrita en
                 3.3).
14             para cada voto en que fue primera preferencia hacer
15                 Multiplicar el peso por el valor de transferencia del candidato electo.
       // 4. Eliminar candidatos
16     Eliminar de las papeletas a todos los candidatos que fueron eliminados o electos.
```

Anexo B

A continuación se muestra un ejemplo del procedimiento de conteo para votación con *ranking* utilizando STV.

Sea la Figura B.1 una representación de las papeletas emitidas en un proceso de elección con *ranking* que busca escoger tres representantes. En este participan cinco personas, los votantes I, II, III, IV y V, respectivamente. Estos emiten sus papeletas *rankeando* 4 opciones, los candidatos A, B, C y D, respectivamente.

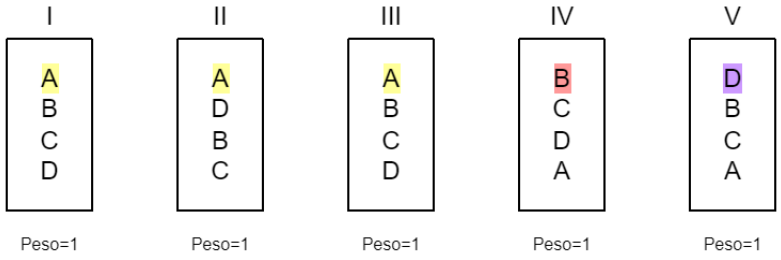


Figura B.1: Conjunto inicial de papeletas, destacando sus respectivas primeras preferencias. Cada papeleta tiene peso inicial 1.

La cuota q correspondiente a esta elección se calcula tal como se muestra en 3.2. Esto considerando que el número de votantes, n, es 5 y el número de puestos a cubrir, s, es 3.

$$\begin{aligned} q &= \left\lfloor \tfrac{n}{s+1} \right\rfloor + 1 \\ &= \left\lfloor \tfrac{5}{3+1} \right\rfloor + 1 \\ &= 2 \end{aligned} \tag{B.1}$$

Una vez señalados los datos iniciales se comienzan las iteraciones para determinar a las y los ganadores.

Primera iteración

Se inicia con el proceso de *computar la primera preferencia*. En este, se realiza el conteo para cada uno de los candidatos. Es decir, se suman los pesos de las papeletas en las que fueron seleccionados como primera preferencia. Los resultados de los conteos en la primera iteración son:

- Conteo A: $1 + 1 + 1 = 3$
- Conteo B: 1
- Conteo C: 0
- Conteo D: 1

Se continua con el proceso de *elegir o eliminar candidatos y candidatas*. Aquí, resultarán electos aquellos candidatos o candidatas cuyo conteo supere la cuota q. Tal como se observa en la Figura B.2, solamente el candidato A logró superar la cuota, por lo que solo él es electo para esta iteración.

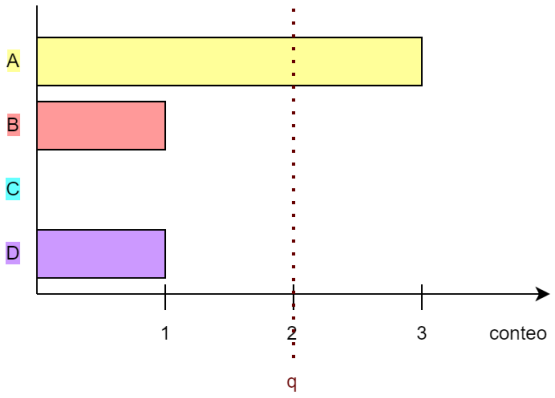


Figura B.2: Representación gráfica de los conteos obtenidos para las y los candidatos en la primera iteración.

Con el resultado anterior se procede a realizar el proceso de *reponderar votos*. Aquí, se procede a calcular el valor de transferencia y reponderar los votos donde el candidato electo A es primera preferencia, es decir, los votos I, II y III. El valor de transferencia para A, t_A, se calcula tal como se muestra en 3.3, donde m_A corresponde al conteo obtenido para A.

$$\begin{aligned} t_A &= \frac{m_A - q}{m_A} \\ &= \frac{3-2}{3} \\ &= \frac{1}{3} \end{aligned} \tag{B.2}$$

El peso inicial de las papeletas I, II y III era 1, al reponderarlas queda que su nuevo peso es $1 \times t_A = t_A = 1/3$. El resto de las papeletas mantiene su peso.

Por último, se realiza el proceso de *eliminar candidatos y candidatas*. De la primera iteración solo fue electo el candidato A y ninguno fue descartado. Por eso, solo corresponde remover al candidato A de todas las papeletas. El resultado de esto se observa en la Figura B.3.

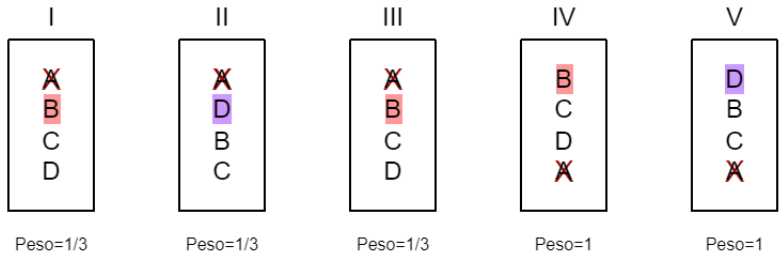


Figura B.3: Conjunto de papeletas al terminar la primera iteración, destacando sus respectivas primeras preferencias.

Segunda iteración

Se inicia con el proceso de *computar la primera preferencia*. Los resultados de los conteos de la segunda iteración son:

- Conteo B: $1/3 + 1/3 + 1 = 5/3$
- Conteo C: 0
- Conteo D: $1/3 + 1 = 4/3$

Se prosigue con el proceso de *elegir o eliminar candidatas y candidatos*. Tal como se observa en la Figura B.4, para esta iteración ningún candidato alcanzó la cuota. Por este motivo resulta eliminado aquel candidato que obtuvo el menor conteo, es decir, el candidato C.

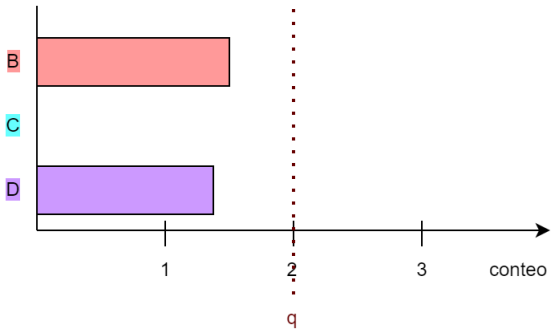


Figura B.4: Representación gráfica de los conteos obtenidos para las y los candidatos en la segunda iteración.

En vista del resultado anterior se realiza el proceso de *reponderar votos*. Para esto es importante considerar que, al no resultar ningún candidato electo se puede tomar el valor de transferencia de todos los candidatos igual a 1, mantenido el peso de todas las papeletas.

Por último, se realiza el proceso de *eliminar candidatos y candidatas*. Tal como ya se dijo, no hay candidatos electos y el candidato C fue descartado. Por esto, solo corresponde remover al candidato C de todas las papeletas. El resultado de esto se observa en la Figura B.5.

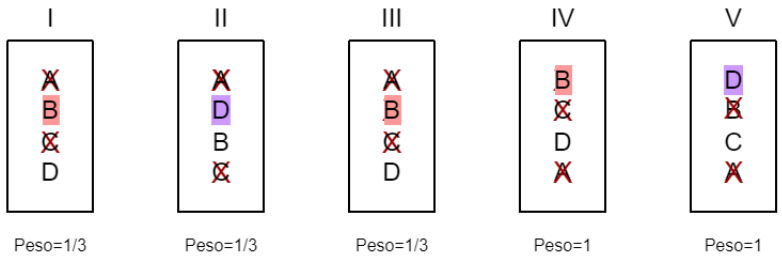


Figura B.5: Conjunto de papeletas al terminar la segunda iteración, destacando sus respectivas primeras preferencias.

Fin del algoritmo

En este punto resulta importante observar que quedan dos puestos disponibles y dos candidatos en competencia. Se ha llegado al caso base, por lo que los candidatos restantes son declarados electos. Por último, el resultado de la elección es que los tres puestos disponibles serán ocupados por los candidatos A, B y D.

Anexo C

Los algoritmos presentados a continuación corresponden a pseudocódigos que muestran el procedimiento de conteo para votación con *ranking* utilizando STV mediante *Shuffle-Sum*.

Algoritmo 1a: Conteo STV

$q \leftarrow \left\lfloor \frac{\#votos\ validos}{\#puestos+1} \right\rfloor + 1$ `// Calcula` q`, la cuota requerida para una elección.`
mientras *el número de candidatos restantes es mayor al número de puestos disponibles* **hacer**
    Computar los conteos de primeras preferencias (Algoritmo 1b).
    si *al menos un candidatos fue electo (su conteo fue mayor a* q*)* **entonces**
        $S \leftarrow$ conjunto de candidatos cuyo conteo fue mayor a q.
        Anunciar que cada candidato en S fue electo.
        Reponderar los votos de los candidatos electos en S y cuota q (Algoritmo 1c).
        Eliminar todos de las papeletas a los candidatos en S (Algoritmo 1d).
    en otro caso `// Ningún candidato fue electo`
        $i \leftarrow$ el candidato cuyo conteo fue el menor.
        Anunciar que i fue eliminado.
        Eliminar $\{i\}$ de las papeletas (Algoritmo 1d).
Anunciar que los candidatos restantes fueron electos.

Algoritmo 1b: Calcular los conteos de primeras preferencias

input: Todas las papeletas en su representación por *orden de candidato* (Tabla 3.1).
para *cada votante* $v \in \{1, ..., n\}$ **hacer**
    Convertir la papeleta de v a su representación de *primera preferencia* (Tabla 3.3, Algoritmo 2).
para *cada candidato* $i \in \{1, ..., m\}$ **hacer**
    $T_i \leftarrow \bigoplus_{v=1}^{n} W'_{v,i}$ `// Sumar homomorficamente los pesos encriptados correspondientes al candidato` i`.`
    Desencriptación umbral del conteo $t_i \leftarrow D(T_i)$.

Algoritmo 1c: Reponderar los votos para los candidatos electos en S con cuota q

input: Todas las papeletas en su representación de *primera preferencia* (Tabla 3.3).

para *cada candidato* $i \in S$ **hacer**
  $s_i \leftarrow 1 - \frac{q}{t_i}$ `// Computar el valor de transferencia para` i.
  Escoger $a_i/d_i \approx s_i$ `// Aproximar` s_i.
$d' \leftarrow lcm(d_i : i \in S)$ `// Computar el común denominador`
para *cada votante* $v \in \{1, ..., n\}$ **hacer**
  para *cada candidato* $i \in S$ **hacer**
    $W'_{v,i} \leftarrow a_i \frac{d'}{d_i} \otimes W'_{v,i}$ `// Multiplicar homomorficamente el` i`-ésimo peso encriptado por el numerador del correspondiente factor de escala.`
  para *cada candidato* $j \notin S$ **hacer**
    $W'_{v,j} \leftarrow d' \otimes W'_{v,j}$ `// Multiplicar homomorficamente el` j`-ésimo peso encriptado por el común denominador de los factores de escala para los candidatos electos.`
  $W_v \leftarrow \bigoplus_{i=1}^{m} W'_{v,i}$ `// Sumar homomorficamente los pesos encriptados de todos los candidatos para obtener el nuevo peso encriptado para el voto` v `(notar que a lo más uno va a ser distinto a cero).`

Algoritmo 1d: Eliminar a los candidatos en el conjunto C

input: Todas las papeletas en su representación de *orden por candidatos* (Tabla 3.1)

para *cada votante* $v \in \{1, ..., n\}$ **hacer**
  Convertir la papeleta de v a su representación de *eliminación de candidatos en* C (Tabla 3.4, Algoritmo 3).
  para *cada candidato* $l \in \{1, ..., m\}$ **hacer**
    $P_{v,l} \leftarrow P_{v,l} - \bigoplus_{i=1}^{l} X_{v,i}$ `// Actualizar homomorficamente la lista de preferencias para eliminar a los candidatos etiquetados (`$X_{v,i} = E(1)$ `para todo` $i \in C$`).`
  Convertir la papeleta de v a su representación *por orden de candidatos* modificada. (Tabla 3.1, Algoritmo 4). `// Es modificado porque la lista de preferencias no es más una permutación.`
  Remover de la papeleta cada candidato en C (y su correspondiente preferencia encriptada).

Algoritmo 2: Convertir una papeleta desde su representación *por orden de candiatos* a *por primera preferencia*

1 Encriptar la fila de candidatos.
2 Mezclar las columnas de la tabla.
3 Desencriptación umbral de la fila de preferencias.
4 Ordenar las columnas por orden de preferencias.
5 Añadir una fila *pesos*, tal que $W_{v,1} \leftarrow W_v$ y $\forall i > 1 : W_{v,i} \leftarrow E(0)$.
6 Encriptar la fila de preferencias.
7 Mezclar las columnas de la tabla.
8 Desencriptación umbral de la fila de candidatos.
9 Ordenar las columnas por orden de candidatos.

Algoritmo 3: Convertir una papeleta desde su representación *por orden de candidatos* a *eliminación de candidatos en C*

input: La papeleta del votante v en su representación *por orden de candidatos* (Tabla 1)
1 Agregar una fila *eliminado* a la papeleta en su representación *por orden de candidatos*, de modo que el valor en cada columna, $X_{v,i}$, sea $E(1)$ si $i \in C$ y en el caso contrario.
2 Encriptar la fila de candidatos.
3 Mezclar las columnas de la tabla.
4 Desencriptación umbral de la fila de preferencias.
5 Ordenar las columnas de la tabla por preferencia.
6 Encriptar la fila de preferencias.

Algoritmo 4: Convertir una papeleta desde su representación de *eliminación de candidatos en C* a su representación *por orden de candidatos* (modificada).

1 Mezclar las columnas de la tabla.
2 Desencriptación umbral de la fila de candidatos.
3 Ordenar las columnas de la tabla por orden de candidatos.

Algoritmo 5: Convertir una papeleta desde su *tabla de comparaciones* a su representación *por orden de candidatos*.

input: Las *tablas de comparación* de todas las papeletas.
1 **para** *cada votante* $v \in \{1, ..., n\}$ **hacer**
2 **para** *cada candidato que continúa en competencia* $j \in \{1, ..., m\}$ **hacer**
3 $P_{v,j} \leftarrow E(1) - \bigoplus_{i \neq j} V_{i,j}$ `// Sumar homomorficamente la columna de comparaciones para ese candidato.`

Algoritmo 6: Eliminar al candidato i usando *tabla de comparaciones*.

input: La *tabla de comparación* de cada papeleta y su respectiva representación *por orden de candidatos* (Tabla 1).

```
1 para cada votante v ∈ {1, ..., n} hacer
2     para cada candidato que continúa en competencia j ∈ {1, ..., m} hacer
3         P_{v,j} ← P_{v,j} ⊕ V_{i,j} // Actualizar homomorficamente la lista de
              preferencias para eliminar al candidato i.
```

Anexo D

A continuación se muestra un ejemplo del procedimiento de conteo para votación con *ranking* utilizando STV mediante Shuffle-Sum.

Sea la Figura D.1 una representación de los votos emitidos en un proceso de elección con *ranking* que busca escoger a tres representantes. En este participan cinco personas, los votantes I, II, III, IV y V, respectivamente. Estos emiten sus papeletas *rankeando* 4 opciones, los candidatos 1, 2, 3 y 4, respectivamente.

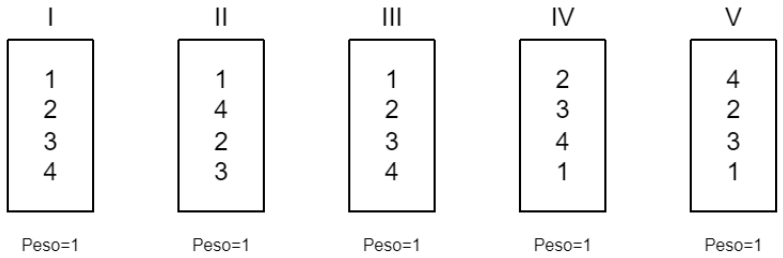


Figura D.1: Conjunto inicial de papeletas, cada una tiene peso inicial 1.

La cuota q correspondiente a esta elección se calcula tal como se muestra en la Expresión D.1. Esto considerando que el número de votantes n es 5 y el número de puestos a cubrir s es 3.

$$\begin{aligned} q &= \left\lfloor \tfrac{n}{s+1} \right\rfloor + 1 \\ &= \left\lfloor \tfrac{5}{3+1} \right\rfloor + 1 \\ &= 2 \end{aligned} \tag{D.1}$$

Una vez señalados los datos iniciales se comienzan las iteraciones para determinar a las y los ganadores.

Primera iteración

Se inicia con el proceso de *computar la primera preferencia*. Para esto, se pasan todas las papeletas a su representación *por orden de primera preferencia*, quedando como se muestran

en la Figura D.2.

Candidato	1	2	3	4
Preferencia	**1**	**2**	**3**	**4**
Peso	**1**	**0**	**0**	**0**

(a) Papeleta I

Candidato	1	2	3	4
Preferencia	**1**	**3**	**4**	**2**
Peso	**1**	**0**	**0**	**0**

(b) Papeleta II

Candidato	1	2	3	4
Preferencia	**1**	**2**	**3**	**4**
Peso	**1**	**0**	**0**	**0**

(c) Papeleta III

Candidato	1	2	3	4
Preferencia	**4**	**1**	**2**	**3**
Peso	**0**	**1**	**0**	**0**

(d) Papeleta IV

Candidato	1	2	3	4
Preferencia	**4**	**2**	**3**	**1**
Peso	**0**	**0**	**0**	**1**

(e) Papeleta V

Figura D.2: Conjunto inicial de papeletas en su representación *por orden de primera preferencia*, donde cada valor en negrita representa la encriptación de dicho valor.

Para calcular el conteo alcanzado por cada candidato y candidata se suma homomórficamente el peso que se le asigna en cada papeleta. Se puede observar la manera en la que se lleva a cabo este proceso en la Tabla D.1.

Candidato	1	2	3	4
Peso papeleta I	**1**	**0**	**0**	**0**
Peso papeleta II	**1**	**0**	**0**	**0**
Peso papeleta III	**1**	**0**	**0**	**0**
Peso papeleta IV	**0**	**1**	**0**	**0**
Peso papeleta V	**0**	**0**	**0**	**1**
CONTEO	**3**	**1**	**0**	**1**

Tabla D.1: Suma homomórfica del peso de las papeletas para obtener el conteo de cada candidato o candidata. Cada valor en negrita representa la encriptación de dicho valor.

Al desencriptar las cuotas obtenidas se obtienen los siguientes resultados:

- Conteo 1: 3
- Conteo 2: 1
- Conteo 3: 0
- Conteo 4: 1

Se obtiene que A es el único candidato que alcanzó la cuota y que, por ende, es electo. Con esto concluye el proceso de *elegir o eliminar candidatos*, dando paso al proceso de *reponderar los votos*. Aquí, se calculan los valores de transferencia, obteniendo:

- $t_1 = (3 - 2)/3 = 1/3$
- $t_2 = 1$
- $t_3 = 1$
- $t_4 = 1$

En base a esto se actualizan los pesos contenidos para cada candidato en todas las papeletas en su representación *por orden de primera preferencia*, multiplicando el peso asignado a cada candidato por su valor de transferencia correspondiente. En resumen, de este proceso quedan los valores presentados en la Figura D.3.

Candidato	1	2	3	4
Peso	**1/3**	**0**	**0**	**0**

(a) Papeleta I

Candidato	1	2	3	4
Peso	**1/3**	**0**	**0**	**0**

(b) Papeleta II

Candidato	1	2	3	4
Peso	**1/3**	**0**	**0**	**0**

(c) Papeleta III

Candidato	1	2	3	4
Peso	**0**	**1**	**0**	**0**

(d) Papeleta IV

Candidato	1	2	3	4
Peso	**0**	**0**	**0**	**1**

(e) Papeleta V

Figura D.3: Conjunto de papeletas en su representación *por orden de primera preferencia*, donde solo se muestran las filas de candidatos y pesos. Cada valor en negrita representa la encriptación de dicho valor.

Por último, se lleva a cabo el proceso de *eliminar candidatas y candidatos*. Para este se utilizan las papeletas en su representación de *eliminación de candidatos en C*. Para esta primera iteración se considera que el conjunto de candidatos a remover desde las papeletas es $\{1\}$, por lo que las papeletas quedan como se muestran en la Figura D.4.

La lógica utilizada para llevar a cabo este proceso es que el número en que disminuirá (mejorará) la preferencia de cada candidato después de la eliminación, será igual a la cantidad de candidatos eliminados que poseían menor (mejor) preferencia que este. En base a esto, se realiza el cálculo de las nueva preferencia correspondiente a cada candidato en cada papeleta. Esto se puede resumir en la actualización de la papeleta I, pues el proceso es análogo para el resto de las papeletas.

Para la papeleta I resulta:

- $\mathbf{4} \ominus (\mathbf{1} \oplus \mathbf{0} \oplus \mathbf{0} \oplus \mathbf{0}) = \mathbf{3}$
  La preferencia 4 pasa a ser la 3.
- $\mathbf{3} \ominus (\mathbf{1} \oplus \mathbf{0} \oplus \mathbf{0}) = \mathbf{2}$
  La preferencia 3 pasa a ser la 2.

Candidato	**1**	**2**	**3**	**4**
Preferencia	**1**	**2**	**3**	**4**
Eliminado	**1**	**0**	**0**	**0**
Peso	**1/3**			

(a) Papeleta I

Candidato	**1**	**4**	**2**	**3**
Preferencia	**1**	**2**	**3**	**4**
Eliminado	**1**	**0**	**0**	**0**
Peso	**1/3**			

(b) Papeleta II

Candidato	**1**	**2**	**3**	**4**
Preferencia	**1**	**2**	**3**	**4**
Eliminado	**1**	**0**	**0**	**0**
Peso	**1/3**			

(c) Papeleta III

Candidato	**2**	**3**	**4**	**1**
Preferencia	**1**	**2**	**3**	**4**
Eliminado	**1**	**0**	**0**	**0**
Peso	**1/3**			

(d) Papeleta IV

Candidato	**4**	**2**	**3**	**1**
Preferencia	**1**	**2**	**3**	**4**
Eliminado	**1**	**0**	**0**	**0**
Peso	**1/3**			

(e) Papeleta V

Figura D.4: Conjunto de papeletas en su representación *de eliminación de candidatos en C*, donde C es el conjunto {1} y cada valor en negrita representa la encriptación de dicho valor.

- $\mathbf{2} \ominus (\mathbf{1} \oplus \mathbf{0}) = \mathbf{1}$
  La preferencia 2 pasa a ser la 1.
- $\mathbf{1} \ominus (\mathbf{1}) = \mathbf{0}$
  La preferencia 1 pasa a ser la 0. Este es justamente el que se desea eliminar.

Finalmente, se obtiene que las papeletas resultantes para la primera iteración son las presentadas en la Figura D.5.

Habiendo entendido la primera iteración y el ejemplo dado en el Anexo B, se puede observar que el resto de las iteraciones corresponden a procesos totalmente análogos, pues no dependen de condiciones y su principal dificultad está en lograr el paso desde un tipo de papeleta a otra.

Candidato	**1**	**2**	**3**	**4**
Preferencia	**1**	**2**	**3**	**4**
Eliminado	**1**	**0**	**0**	**0**
Peso	**1/3**			

(a) Papeleta I

Candidato	**1**	**4**	**2**	**3**
Preferencia	**1**	**2**	**3**	**4**
Eliminado	**1**	**0**	**0**	**0**
Peso	**1/3**			

(b) Papeleta II

Candidato	**1**	**2**	**3**	**4**
Preferencia	**1**	**2**	**3**	**4**
Eliminado	**1**	**0**	**0**	**0**
Peso	**1/3**			

(c) Papeleta III

Candidato	**2**	**3**	**4**	**1**
Preferencia	**1**	**2**	**3**	**4**
Eliminado	**1**	**0**	**0**	**0**
Peso	**1**			

(d) Papeleta IV

Candidato	**4**	**2**	**3**	**1**
Preferencia	**1**	**2**	**3**	**4**
Eliminado	**1**	**0**	**0**	**0**
Peso	**1**			

(e) Papeleta V

Figura D.5: Conjunto de papeletas en su representación *de eliminación de candidatos en C*, donde C es el conjunto {1} y cada valor en negrita representa la encriptación de dicho valor.

Candidato	2	3	4
Preferencia	**1**	**2**	**3**
Peso	**1/3**		

(a) Papeleta I

Candidato	2	3	4
Preferencia	**2**	**3**	**1**
Peso	**1/3**		

(b) Papeleta II

Candidato	2	3	4
Preferencia	**1**	**2**	**3**
Peso	**1/3**		

(c) Papeleta III

Candidato	2	3	4
Preferencia	**1**	**2**	**3**
Peso	**1**		

(d) Papeleta IV

Candidato	2	3	4
Preferencia	**2**	**3**	**1**
Peso	**1**		

(e) Papeleta V

Figura D.6: Conjunto de papeletas resultantes de la primera iteración en su representación *por orden de candidatos*, donde cada valor en negrita representa la encriptación de dicho valor.

Anexo E

A continuación se muestra un ejemplo del proceso de *Eliminar candidatas y candidatos* en *Shuffle-Sum* utilizando *tabla de comparaciones*.

Sea V la papeleta emitida por un votante en una elección con 5 candidatos, siendo el candidato número 4 el que se desea eliminar. En V el orden de candidatos de mejor a peor preferencia es 3, 5, 4, 1 y 2. La representación en *tabla de comparaciones* de V se puede observar en la Tabla 10.

	1	2	3	4	5
1	**x**	**-1**	**0**	**0**	**0**
2	**0**	**x**	**0**	**0**	**0**
3	**-1**	**-1**	**x**	**-1**	**-1**
4	**-1**	**-1**	**0**	**x**	**0**
5	**-1**	**-1**	**0**	**-1**	**x**

Tabla E.1: Representación en *tabla de comparaciones* de la papeleta V, donde el contenido en (i, j) será la encriptación de -1 si el candidato i es preferido por sobre j y la encriptación de 0 en el caso contrario.

El primer paso para llevar a cabo este proceso es sumar homomórficamene los valores en cada columna y sumar homomórficamente 1 a su valor absoluto. El resultado de esto se observa en la Tabla 11.

Candidato	1	2	3	4	5
s	**4**	**5**	**1**	**3**	**2**

Tabla E.2: Preferencia de cada candidato antes de remover al candidato 4 de la papeleta.

Para eliminar al candidato 4 es necesario actualizar la fila s sumando homomórficamente y de manera secuencial los valores en la fila correspondiente al candidato 4, tal como se muestra en la Tabla 12.

Por último, la tabla de comparaciones queda igual, pero con la información de 4 descartada, y la representación de la papeleta en su forma *por orden de candidatos* actualiza eliminando la columna del candidato 4 e intercambiando la preferencia de cada candidato por las calculadas en s.

Candidato	1	2	3	4	5
s	**4** $\oplus$ **-1**	**5** $\oplus$ **-1**	**1** $\oplus$ **0**	**x**	**2** $\oplus$ **0**

(a) Suma homórfica de la fila 4 para actualizar s.

Candidato	1	2	3	4	5
s	**3**	**4**	**1**	**x**	**2**

(b) Resultado actualización s.

Tabla E.3: Actualización de preferencias al eliminar al candidato 4.

Anexo F

La explicación presentada a continuación pretende definir una cota superior para cualquier conteo, obtenido en un proceso de cómputo de primera preferencia.

Para el cálculo de esta cota se considera que, tal como dicta el algoritmo, el peso inicial de cada papeleta es 1, con un total de n papeletas y 8 candidatos. Luego, tal como se explicó anteriormente, al actualizar el valor de un peso en el proceso de reponderación, el peor caso sería multiplicar el peso por el denominador del valor de transferencia, es decir, el conteo m obtenido por el candidato en esa ronda. Como el conteo corresponde a la suma del peso de las papeletas en las que el candidato es primera preferencia, el caso más extremo sería aquel en el que todas las papeletas contienen a un mismo candidato como primera preferencia. Esta afirmación se define algebraicamente en la Expresión F.1.

$$\forall i \in \{1, ..., 8\}, m_i \leq \sum_{j=1}^{n} w_j \tag{F.1}$$

Luego, estudiando los peores casos para una papeleta j, se puede observar el siguiente avance:

Primera iteración

La cota para el conteo alcanzado por un candidato i_1 sería la mostrada en la Expresión F.2.

$$m_{i_1} \leq \sum_{j=1}^{n} w_j \leq \sum_{j=1}^{n} 1 = n \tag{F.2}$$

Reponderar el peso de la papeleta j bajo dicho conteo resulta en lo obtenido en la Expresión F.3.

$$w_j = w_j \cdot m_{i_1} \leq 1 \cdot n = n \tag{F.3}$$

Segunda iteración

La cota para el conteo alcanzado por un candidato i_2 sería la mostrada en la Expresión F.4.

$$m_{i_2} \leq \sum_{j=1}^{n} w_j \leq \sum_{j=1}^{n} n = n^2 \tag{F.4}$$

Reponderar el peso de la papeleta j bajo dicho conteo resulta en lo obtenido en la Expresión F.5.

$$w_j = w_j \cdot m_{i_2} \leq n \cdot n^2 = n^3 \tag{F.5}$$

Tercera iteración

La cota para el conteo alcanzado por un candidato i_3 sería la mostrada en la Expresión F.6.

$$m_{i_3} \leq \sum_{j=1}^{n} w_j \leq \sum_{j=1}^{n} n^3 = n^4 \tag{F.6}$$

Reponderar el peso de la papeleta j bajo dicho conteo resulta en lo obtenido en la Expresión F.7.

$$w_j = w_j \cdot m_{i_3} \leq n^3 \cdot n^4 = n^7 \tag{F.7}$$

Séptima iteración

Según la metodología utilizada para calcular los resultados de las primeras iteraciones, se obtiene que al finalizar la séptima iteración el peso asociado a la papeleta j se comporta como muestra la Expresión F.8.

$$w_i \leq n^{127} \tag{F.8}$$

Octava iteración

La cota para el conteo alcanzado por un candidato i_x sería la mostrada en la Expresión F.9.

$$m_{i_x} \leq \sum_{j=1}^{n} w_j \leq \sum_{j=1}^{n} n^{127} = n^{128} \tag{F.9}$$