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

DISEÑO E IMPLEMENTACIÓN DE UN SISTEMA DE MENSAJERÍA ANÓNIMA BASADO EN DC-NET

TESIS PARA OPTAR AL GRADO DE MAGÍSTER EN CIENCIAS, MENCIÓN COMPUTACIÓN

CAMILO JOSÉ GÓMEZ NÚÑEZ

PROFESOR GUÍA:
ALEJANDRO HEVIA ANGULO

MIEMBROS DE LA COMISIÓN:
TOMÁS BARROS ARANCIBIA
JOSE MIGUEL PIQUER GARDNER
JEROEN VAN DE GRAAF
MARTA BARRÍA MARTINEZ

SANTIAGO DE CHILE
2018

RESUMEN DE LA TESIS

DISEÑO E IMPLEMENTACIÓN DE UN SISTEMA DE MENSAJERÍA ANÓNIMA BASADO EN DC-NET

Recientes reportajes periodísticos han puesto en evidencia que el derecho a la privacidad en Internet ha sido fuertemente menoscabado en los últimos años debido a la existencia de un sinnúmero de operaciones de monitoreo realizadas por organismos estatales de inteligencia de distintas naciones. Con la excusa de detectar actividades terroristas, tales organismos han monitoreado en forma indiscriminada y sin aparente autorización judicial las comunicaciones de millones de usuarios en Internet. Esto ha causado un interés creciente en herramientas que anonimicen las actividades de los usuarios. Tal es el caso del software *TOR*, basado en el protocolo *onion-routing*. Sin embargo, dicho protocolo es *a priori* vulnerable frente a un adversario que posea un poder de monitoreo global de la red – una capacidad probablemente ya existente. Es por esta razón que se hace necesario encontrar un nuevo protocolo para asegurar comunicaciones verdaderamente anónimas.

El protocolo DC-Net (*Dining Cryptographers Network*), es un protocolo que asegura anonimato incondicional, esto es, independiente del poder de cómputo y la capacidad de monitoreo del adversario. Lamentablemente, el protocolo original presenta dos obstáculos: (1) es necesario establecer canales de comunicación entre todos los participantes, lo que dificulta la escalabilidad, y (2) esta diseñado para transmitir solo un único mensaje en forma simultánea, de manera que el envío de más mensajes resulta en servicio degradado (se dice que hay una *colisión* de mensajes). Una DC-Net robusta debe poder manejar dichas colisiones para admitir el envío de varios mensajes en forma simultánea.

En este trabajo se propone una variante del protocolo DC-Net la cual emplea primitivas criptográficas sofisticadas (*commitments* y *zero-knowledge-proofs*) para lidiar con adversarios maliciosos y manejar colisiones de manera eficiente. Si bien ya existen propuestas de protocolos de anonimato que atacan el problema de escalabilidad, lo resuelven sacrificando un poco de seguridad, en particular, agregando servidores que funcionan como tercera parte confiable. La variante propuesta en este trabajo, si bien no alcanza la escalabilidad de otras propuestas, no sacrifica seguridad pues funciona bajo un modelo de amenaza significativamente más fuerte, uno donde el adversario puede monitorear todas las conexiones e incluso infiltrarse como participante del sistema.

A fin de evaluar esta propuesta en la práctica, se realizó una implementación concreta del protocolo. Esta implementación sirvió de base para la construcción de un prototipo de aplicación móvil, el cual efectivamente provee un canal anónimo entre un número potencialmente arbitrario de participantes. Por otro lado, la implementación realizada fue sometida a variados experimentos a fin de evaluar el enfoque en distintos escenarios (todos en una red local). Dichos experimentos muestran que el protocolo se comporta mejor en un escenario de foro anónimo, donde el envío de mensajes se realiza sin esperar una respuesta inmediata, y donde no necesariamente todos los participantes de dicho foro deben enviar un mensaje al mismo tiempo. Si bien los experimentos realizados contemplaron escenarios de hasta 30 participantes, los resultados obtenidos permiten extrapolar el rendimiento del protocolo en situaciones más generales y con un conjunto mayor de emisores y/o participantes, mostrando que para escenarios de baja interactividad el enfoque es suficientemente práctico como para ser una alternativa efectiva de comunicación anónima.

A mi madre, a mi padre y a mi hermano, quienes siempre han estado conmigo.

Agradecimientos

Me gustaría agradecer primero a mi familia, que siempre ha creído en mi y me ha apoyado en todas las decisiones que he tomado durante toda mi vida. Culmina una etapa en mi vida que partió con la decisión de viajar desde Talca a Santiago, con todo lo que ello implica, y mi familia siempre fue pilar fundamental tanto en las buenas con en las malas que he tenido que pasar durante estos años. No habría podido pasarlos sin su ayuda y apoyo incondicional.

Por otro lado me gustaría agradecer a la gente que me apoyó para el desarrollo de esta tesis, desde dentro de la Universidad misma, partiendo por mi profesor guía, Alejandro Hevia, que ya son varios años de trabajar juntos y espero que sigan siendo muchos más, hemos logrado muchas cosas trabajando juntos y aun podemos lograr mucho más. Agradecer también a Javier Bustos, y en su nombre, a toda la gente de Niclabs que me apoyó y trabajamos mano a mano durante los años que se desarrolló este trabajo. Por último agradecer a mis amigos y colegas Boris Romero, Sebastián Blasco y Francisco Cifuentes, que sin su apoyo, sabiduría y constante ayuda este trabajo y mi carrera universitaria no hubiera logrado llegar al exitoso cúlmine que está llegando. Muchas gracias.

Tabla de Contenido

Capítulo 1

Introducción

1.1. Anonimato en Internet

Fruto tanto de las primeras revelaciones del grupo *Wikileaks*[1] en el año 2006, así como de la información revelada por el ex agente de la *CIA*, *Edward Snowden*[2] en el año 2013, muchos usuarios han modificado su comportamiento de navegación por *Internet* simplemente por el hecho de tener la presunción que todos sus movimientos están siendo registrados por organismos de Estado, a pesar de no ser amenaza a la seguridad de ninguna nación.

Por ende, usuarios de todo el mundo han comenzado a reclamar por su derecho a la privacidad[3], por ejemplo, tomando acciones que ocultan las páginas que visitan navegando en Internet de los organismos de Estado, o de cualquier tercero, que estén monitoreando y/o recopilando dichas visitas. Existe una basta comunidad compuesta por activistas de derechos humanos, abogados, ONGs, académicos y profesionales que abogan por que se haga valer el legítimo derecho a la privacidad de los usuarios de Internet, argumentando la importancia del anonimato[4] [5], y motivando su tratamiento desde una perspectiva científica, brindando herramientas y protocolos que puedan asegurar el anonimato en Internet de manera segura, eficaz y eficiente. El trabajo de esta tesis está motivado por dicho objetivo.

Hoy en día navegar de manera anónima en Internet se realiza principalmente usando el software *TOR*[6], el cual en su versión más popular consiste en un explorador web (variante de *Firefox*[7]) que ejecuta el protocolo *onion-routing* (detallado en la siguiente sección), el cual permite dos objetivos: (1) ocultar la identidad del usuario tanto al proveedor del servicio que está consumiendo (página web que está visitando), como de algún observador externo, y (2) acceder a servicios "ocultos" que solo son accesibles por medio de *TOR*. En este trabajo nos concentraremos en el primer objetivo solamente, detallando la manera en que *TOR* logra ocultar la identidad del usuario del servicio. Es

[1] `https://wikileaks.org/`
[2] `http://www.huffingtonpost.com/news/nsa/`
[3] `http://www.livescience.com/37398-right-to-privacy.html`
[4] `https://www.derechosdigitales.org/anonimato/`
[5] `http://www.ted.com/talks/glenn_greenwald_why_privacy_matters`
[6] `https://www.torproject.org/`
[7] `https://www.mozilla.org/en-US/firefox`

importante nombrar que *TOR* no es la única manera de navegar de manera anónima en Internet: protocolos como *mix-networks* y *DC-Nets* (tema central en esta tesis) también logran ocultar la identidad del usuario.

El anonimato, más allá de su atractivo científico, es una problemática socialmente difícil de abordar. Esto se debe a que invoca prejuicios y dilemas éticos que, puestos en un contexto poco informado, pueden llegar a generar aprehensión e incluso rechazo. Parte de las aprehensiones se deben a la posibilidad de realizar acciones ilícitas bajo el paraguas del anonimato, lo cual puede impedir la persecución policial de delitos donde el criminal use herramientas que permitan anonimato efectivo. Entre estas acciones se pueden mencionar: denuncias falsas, compra/venta de drogas, distribución de material pornográfico infantil, fraude, *bullying*, etc. ¿Deberían los científicos crear y promover herramientas que permitan esas acciones? ¿Valen la pena las ventajas del anonimato como para permitir la realización de estas acciones sin la posibilidad de perseguir a sus ejecutores? Estas son algunas preguntas que todo científico debería realizarse antes de generar un trabajo (independiente del tema de investigación) que afecta directamente la sociedad en que está envuelto. Desde el punto de vista del autor, la labor del científico es siempre generar el "mejor conocimiento" posible, expandir la barrera del conocimiento humano, y al mismo tiempo, estar consciente de las posibles implicancias que posea en otras esferas de la sociedad, como por ejemplo en este caso, facilitar acciones ilícitas a través de Internet. Pese a los potenciales usos negativos, es nuestra opinión que los sistemas de anonimato pueden ser herramientas efectivas contra la censura y violación de la privacidad existentes en regímenes autoritarios. También en situaciones de denuncia donde los potenciales informantes arriesgan su vida (denuncias de narcotráfico o espionaje), o bien donde se arriesgue el trabajo o una sanción, como podría ser el caso de denuncias de acoso laboral o sexual, que permitiría garantizar una denuncia anónima pero sabiendo que viene dentro de la misma organización. El problema de limitar el uso socialmente negativo de estas tecnologías es un problema de investigación abierto y extremadamente interesante.

1.2. Protocolos de Anonimato

1.2.1. *Mix-networks*

Una alternativa para anonimizar mensajes lo entregan protocolos de mezcla de mensajes, comúnmente llamados *mixnets*. Una *mixnet* consiste en un proceso iterativo donde un conjunto de mensajes encriptados es mezclado (permutado) en forma secuencial por una colección de servidores o "*mix-servers*" [4]. Este proceso puede verse como una "mezcla no invertible" donde la *mixnet* disocia datos referentes a la emisión del mensaje (identidad, tiempo del envío, etc.) con el mensaje mismo, impidiendo individualizar a un emisor con un mensaje en particular. El principal problema de este protocolo es que el anonimato se preserva gracias a la participación de *mix-servers* honestos, y por ende requiere de una "tercera parte confiable". Para garantizar la privacidad de la mezcla (esto es, el anonimato), es que se utilizan varias etapas (servidores) de mezcla, pues basta que un servidor sea honesto, para que la permutación realmente permanezca oculta. Sin embargo, el depender de varios servidores de mezcla repercute en un proceso más ineficiente, resultando en una alta latencia al momento de enviar un mensaje. Importante es mencionar que la seguridad que, dado que utilizan típicamente encriptación de clave pública, se logra con una *mixnet* es solamente computacional, esto es, en un futuro donde sea posible adquirir un mayor poder de cómputo, el anonimato podría verse comprometido.

1.2.2. *Onion-Routing*

El protocolo más utilizado hoy en día para proveer anonimato en la red es *onion-routing* [16], utilizado principalmente a través del software *TOR*. Su popularidad viene dada principalmente por el carácter abierto que posee el proyecto. Esto le permite recibir contribuciones por parte de voluntarios todos los días[8], arreglando fallas que puedan existir, además de recibir permanentes análisis por parte de instituciones académicas que buscan la mejora, tanto del software como del protocolo subyacente. Si bien el sistema posee múltiples ventajas (llegando a ser recomendado por el mismo Snowden para ocultarse de las intromisiones de organismos de Estado[9]), se sabe que posee varias falencias. Por ejemplo la vulnerabilidad descrita en [18], llegando incluso a poder romper el anonimato que éste provee si se posee la capacidad de monitorear toda la red (como la que poseen los distintos *ISP* que proveen la conexión a Internet a los usuarios de la red). También es importante mencionar que el protocolo *onion-routing* fue pensado bajo la amenaza de un adversario "local", es decir, uno que no puede monitorear toda la red. Este supuesto últimamente se ha ido debilitando, puesto que varias filtraciones muestran que, la *NSA* en particular, efectivamente poseería la capacidad de monitorear todo Internet, haciendo inseguro al protocolo *onion-routing*, y en consecuencia el software *TOR*.

1.3. La Cena de Criptógrafos

David Chaum en el año 1985 propuso un protocolo que permitía el envío de mensajes de manera anónima entre un grupo de participantes [2, 3]. Dicho protocolo fue motivado con una historia llamada "El Problema de la Cena de Criptógrafos" (*Dining Cryptographers Problem*), razón por la cual el protocolo desde entonces se conoce como *DC-Net* (*Dining Cryptographers Network*).

El problema es el siguiente: están tres criptógrafos cenando tranquilamente en su restorán favorito. Al terminar la cena, el mozo se acerca a su mesa y les comunica que su cena ya está pagada y que pueden irse a sus casas. Los criptógrafos, altamente consternados, se miran mutuamente y llegan a la conclusión de que debió haber sucedido una de dos cosas: (1) uno de ellos pagó la cuenta de manera anónima (haciéndose el amable e invitando al resto sin que ellos sepan), ó (2) la cuenta fue pagada por alguien distinto a ellos tres (como la *NSA* por ejemplo), revelando la intromisión del organismo de inteligencia de EE.UU. en la vida de los criptógrafos. En esta situación, se hace necesario poder dilucidar este problema, pero sin comprometer (si es que fue el caso) al criptógrafo que pagó la cuenta de manera anónima. Por lo tanto los criptógrafos necesitan saber si uno de ellos pagó la cuenta (sin saber quién) o si fue alguien distinto.

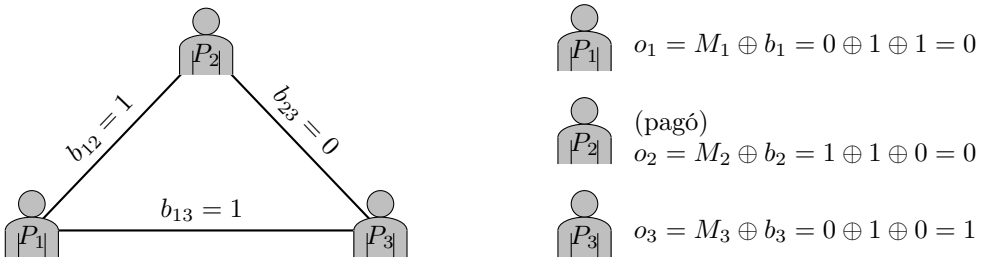
David Chaum generaliza el problema de la siguiente manera: se tienen 3 participantes (los criptógrafos) donde cada uno de ellos quiere comunicar un mensaje (pagó o no pagó la cena). El resultado del protocolo debe entregar, ya sea el mensaje de alguno de ellos (que pagó la cena), sin conocer el emisor de dicho mensaje, o bien que todos los mensajes fueron iguales (revelando que nadie pagó la cena, concluyendo que fue un agente externo). El protocolo debe poder mantener el anonimato del participante que envió el mensaje tanto para el resto de los participantes como para cualquier observador externo que esté monitoreando las conversaciones.

La solución propuesta supone el envío de solo un bit de información (si pagó o no pagó la cena). La solución generalizada consiste en los siguientes pasos:

[8] `https://www.torproject.org/getinvolved/volunteer.html.en`

[9] `https://www.inc.com/larry-kim/5-online-privacy-tips-from-edward-snowden.html`

1. Cada par de participantes (p_i, p_j) escogen un bit al azar compartido b_{ij}. Además, se fija $b_{ij} = -b_{ji}$
2. Cada participante p_i calcula b_i como el `XOR` binario (denotado $\oplus$) entre todos los b_{ij} que posee compartidos. Por ejemplo, en el caso de la cena: $b_1 = b_{12} \oplus b_{13}$, $b_2 = b_{12} \oplus b_{23}$, $b_3 = b_{13} \oplus b_{23}$.
3. Cada participante p_i define su propio M_i, el cual corresponderá al bit que quiere comunicar ($M_i = 1$ si pagó la cena, 0 si no).
4. Cada p_i revelará el valor $o_i = M_i \oplus b_i$.
5. Cualquier p_i u observador externo puede calcular el valor $D = \bigoplus_i o_i$.
6. Si $D = 1$, entonces uno de los participantes envió el mensaje 1 (alguien pagó la cena), si $D = 0$, todos los participantes enviaron el mensaje 0 (la cena la pagó un agente externo a los participantes).



(a) Ilustración del bit al azar b_{ij} compartido entre cada par de participantes.

(b) Cálculo de o_i por cada participante, dependiendo si pagó o no la cena.

Figura 1.1: Ejemplo de Cena de Criptógrafos donde uno de los participantes pagó la cena. En este caso $D = 0 \oplus 0 \oplus 1 = 1$. No es difícil observar que si ningún participante pagó la cena, $D = 0$.

Luego de proponer esta solución, David Chaum realiza el análisis de seguridad del protocolo (para detalles consultar el *paper* original [3]), del cual concluye que el protocolo sugerido entrega anonimato de manera *incondicional*: que incluso un adversario que posea todo el poder de cómputo posible no puede dar con una estrategia que le permita encontrar al emisor del mensaje con mejor probabilidad que $1/n$ (donde n es la cantidad total de participantes).

Posteriormente se propone una manera de generalizar el protocolo para utilizarlo con mensajes más largos (de más de 1 bit), lo cual será visto en la siguiente sección.

1.4. *DC-Net* como protocolo de anonimato

DC-Net (*Dining Cryptographers Network*) es un protocolo que permite enviar un mensaje de manera anónima a un grupo de participantes (llamado *anonymity set*). Este protocolo protege la identidad del emisor del mensaje de manera *incondicional*, es decir, que ningún adversario (independiente del poder de cómputo que posea) puede encontrar a dicho emisor con una probabilidad mayor a $1/n$ (donde n es el tamaño del *anonymity set* o la cantidad total de participantes). Dicho adversario puede ser interno (uno de los participantes) o externo (por ejemplo, un observador externo que esté monitoreando todas las comunicaciones entre los distintos participantes).

A continuación se detallará la generalización del protocolo para permitir el envío de mensajes de largo arbitrario. Supongamos un *anonymity set* de n participantes $\{p_1, p_2, \ldots, p_n\}$. Sin pérdida de generalidad, supondremos que p_1 es el único participante que quiere enviar un mensaje (más adelante se discutirá el caso cuando dos o más participantes quieren enviar un mensaje). En este caso $M_1 = m \in \mathbb{Z}_q$[10], para todo el resto $M_i = 0$ ($\forall i \in \{2, \ldots, n\}$).

El primer paso, al igual que en el protocolo original, es generar valores (o claves) compartidos entre cada par de participantes. Cada par $\{p_i, p_j\}$ comparte un valor único k_{ij} (y se define además que $k_{ij} = -k_{ji}$ y $k_{ii} = 0$).

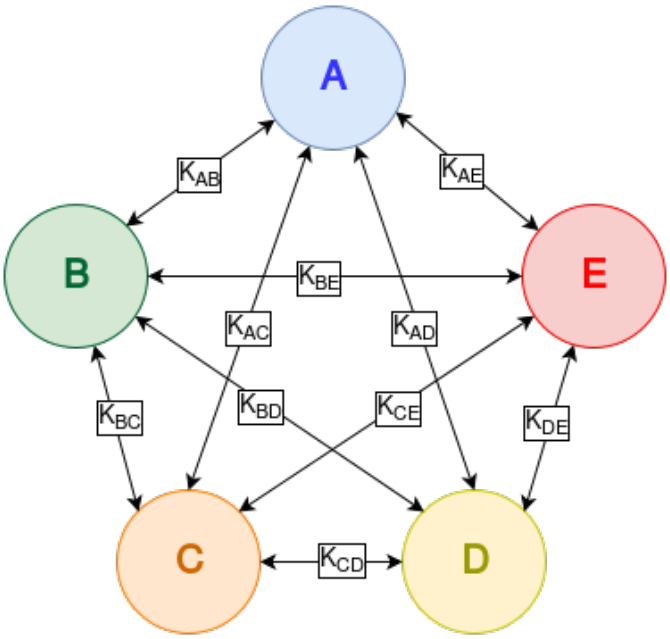


Figura 1.2: Claves compartidas en DC-Net

Con esto, cada participante p_i genera un valor igual a la suma de todas las claves compartidas que posee, esto es $K_i = \sum_{j=1}^{n} k_{ij}$. Luego, genera el valor $O_i = K_i + M_i$ que comunica al resto de los participantes. Este valor es la suma de la clave K_i con el mensaje M_i, (recordemos que solo p_1 posee su mensaje distinto a 0, por lo que para el todo el resto de los participantes $O_i = K_i$). Finalmente, cada participante envía el valor O_i al resto de los participantes (vía *broadcast*, por ejemplo). Con esto, los valores O_i quedan disponibles tanto para el resto de los participantes, como para cualquier observador externo.

Ahora solo queda encontrar el mensaje enviado por p_1. Para ello, se calcula el valor D como la suma de todos los O_i enviados por los participantes:

$$D = \sum_{i=1}^{n} O_i \overset{(1)}{=} \sum_{i=1}^{n} K_i + \sum_{i=1}^{n} M_i \overset{(2)}{=} \sum_{i=1}^{n} K_i + m \overset{(3)}{=} m$$

La primera (1) igualdad se debe a la definición de O_i y la separación de la suma en las dos componentes K_i y M_i. La segunda (2) hace referencia al hecho que $M_i = 0$ $\forall i \geq 2$, por lo que

[10] $\mathbb{Z}_q = \{1, 2, \ldots, q-1\}$

solo "sobrevive" $M_1 = m$. Finalmente la tercera (3) se debe al hecho que las claves compartidas se cancelan mutuamente, debido a que $k_{ij} = -k_{ji}$. Por lo tanto, el valor D corresponde al único mensaje enviado m, el cual es revelado al resto de los participantes, manteniendo en el anonimato a su emisor (en este caso, p_1).

Informalmente, el mensaje m se oculta entre la suma de las claves compartidas. En principio, sin conocer el valor de todas las claves compartidas (a menos que se coludan $n-1$ participantes), no es claro como poder dilucidar si un mensaje m vino de un cierto valor O_i revelado por el participante p_i.

Ahora bien, el protocolo anteriormente descrito sufre de dos problemas: (1) el protocolo funciona sólo cuando un único participante envía un mensaje (de haber dos o más participantes con $M_i \neq 0$, se tendría que $D = \sum M_i$, por lo que sería imposible rescatar los mensajes individuales). Y, (2) no evita que participantes maliciosos envíen mensajes erróneos: por ejemplo, que participantes maliciosos envíen valores de claves compartidas distintos, causando que estas no se cancelen, evitando así la revelación del mensaje m final. En este trabajo se detallará una variante que soluciona estos problema utilizando herramientas criptográficas que serán presentadas en el próximo capítulo.

1.5. Objetivos del Trabajo

1.5.1. Objetivo General

Diseñar e implementar una variante del protocolo *DC-Net* que alcance mejores niveles de seguridad que los ofrecidos por otros protocolos actualmente utilizados para mantener anonimato. Además la variante debe estar optimizada para sistemas completamente distribuidos donde la implementación presente mejores resultados que las alternativas existentes, tomando en cuenta distintos factores como el tipo de seguridad alcanzada y la eficiencia en el envío de los mensajes.

1.5.2. Objetivos Específicos

- Analizar las actuales soluciones (protocolos) que brindan anonimato con el fin de evaluarlos críticamente y mejorarlos o contrastarlos con la solución propuesta en el presente trabajo.
- Analizar otras implementaciones de variantes de *DC-Net*, identificando aspectos a mejorar, a fin de proponer una solución que permita disminuir los supuestos de seguridad en que se basan dichas soluciones.
- Testear la implementación realizada en distintos escenarios, analizando el tiempo de ejecución en cada uno de ellos, con el objetivo de encontrar el mejor contexto en que se desempeñaría la solución final.
- Crear un caso de prueba del sistema implementado, que sirva como motivación a distintos posibles usos que se le pueda dar al código realizado.

1.6. Organización del Documento

En el presente documento se describen los pasos que se siguieron para la propuesta de un diseño y una implementación de un sistema de mensajería que provee anonimato a los emisores basado en el protocolo *DC-Net*. En el Capítulo 2 se describen los antecedentes criptográficos necesarios para comprender la solución propuesta, además de un análisis de soluciones similares propuestas en otros trabajos. Luego en el Capítulo 3 se describe de manera detallada el diseño de la variante propuesta, especificando paso a paso lo que cada uno de los participantes debe realizar para poder contar con el sistema que se propuso como objetivo. Posteriormente en el Capítulo 4 se presentan detalles de la implementación del diseño descrito anteriormente, incluido decisiones de diseño y cómo se fueron tomando para dar forma al sistema finalmente implementado. En este capítulo también se entrega una breve descripción de un caso de uso, una aplicación móvil. En el Capítulo 5 se detallan los experimentos que se llevaron a cabo para testear la solución, incluyendo la evaluación de los tiempos de ejecución bajo distintos escenarios. Los resultados son discutidos en la parte final del capítulo. Luego en el Capítulo 6 se brindan posibles mejoras tanto al protocolo diseñado como a la implementación realizada. Finalmente en el Capítulo 7 se presentan tanto las conclusiones del presente trabajo, como las experiencias que dejaron tanto su diseño como su implementación.

Capítulo 2

Antecedentes

2.1. *Pedersen Commitments*

Un esquema de encriptación es una primitiva criptográfica clásica que tiene como propósito "ocultar" información, es decir, transformar un cierto mensaje en una secuencia que, a primera vista, parece ser totalmente aleatoria, y que solo puede ser "comprendida" (o transformada en el mensaje original) con el uso de una cierta clave secreta.

Por otro lado existe un protocolo que tiene por nombre *commitment scheme* (esquema de vinculación), que se diferencia de un esquema de encriptación clásico, en el hecho que no posee como finalidad comunicar un mensaje a otra persona de manera secreta, sino que más bien enlazar o asociar un cierto participante a un mensaje en particular, dándole la oportunidad que este mensaje pueda permanecer en secreto, pero que *a posteriori* la única manera de aceptar su mensaje es que revele el valor enlazado junto a la clave utilizada.

Existen varias herramientas matemáticas que nos permiten instanciar un esquema de vinculación. En particular en este trabajo se utilizó el protocolo *Pedersen Commitment*[15], el cual se basa en la dificultad de resolver el problema del logaritmo discreto[1].

Pedersen Commitment Scheme [15]: Sea G_q un grupo de orden q, en donde el problema del logaritmo discreto se crea difícil de resolver. Sean g, h generadores de G_q elegidos de manera aleatoria. Sea $s \in \mathbb{Z}_q$ un secreto al cual el participante se vinculará. Además sea $r \in \mathbb{Z}_q$ elegido de manera aleatoria. Se le llama un *Pedersen commitment* sobre s al valor:

$$c := g^s h^r$$

Los *Pedersen commitments* brindan dos propiedades importantes: ocultamiento perfecto (*unconditionally hiding*) y vinculación computacional (*computationally binding*). Esto quiere decir que, si un participante calcula un *commitment* sobre un cierto valor s, (1) la persona se compromete a dicho valor (pero sin revelarlo), ya que, por un lado es imposible para un tercero conocer s a partir de c, y (2) es computacionalmente difícil demostrar que dentro de c existe un valor distinto a s.

[1] `http://www.doc.ic.ac.uk/~mrh/330tutor/ch06s02.html`

2.2. *Zero-Knowledge Proofs of Knowledge*

Una *zero-knowledge proof* permite a una persona poder demostrar el conocimiento de cierto valor α que cumple una propiedad (*statement*), sin revelar ninguna información adicional al hecho que la propiedad se cumple, en particular, sin revelar el valor de α (el testigo) en esa demostración. Se pueden construir demostraciones para diferentes tipos de propiedades (*statements*), como por ejemplo: el conocimiento de un logaritmo discreto; la igualdad de diferentes logaritmos con distintas bases, además de combinaciones con operadores lógicos, entre otros. Además existen maneras para poder demostrar propiedades genéricas sobre logaritmos discretos [1].

Existe una estrecha relación entre *commitment* y *zero-knowledge proofs*, ya que estás últimas generalmente demuestran propiedades que poseen valores "escondidos" dentro de un *commitment*, por lo que se puede convencer a un tercero que un cierto valor que no deseo revelar, satisface una cierta propiedad esperada.

La demostración que se utiliza a lo largo del protocolo descrito en este trabajo, es del tipo *proof-of-knowledge* (o *Zero-Knowledge Proof-of-Knowledge, ZKPoK*), la cual consiste en que el *prover*[2] debe demostrar el conocimiento de un valor α, que hace que la proposición $P(\alpha)$ sea verdadera[3]. Ejemplos de las propiedades para los cuales se utilizan *ZKPoK* en este trabajo son las siguientes:

1. Logaritmo discreto: $PoK\{w : y = g^w\}$
2. Uno de dos logaritmos discretos: $PoK\{x_1, x_2 : h_1 = g^{x_1} \vee h_2 = g^{x_2}\}$
3. Dos logaritmos discretos: $PoK\{x_1, x_2 : h_1 = g^{x_1} \wedge h_2 = g^{x_2}\}$
4. Valores en *Pedersen Commitment*: $PoK\{x, r : c = g^x h^r\}$

El detalle de las *ZKPoK* mencionadas anteriormente se encuentran en la sección 9.1 del Apéndice.

2.3. Implementaciones de *DC-Nets* relacionadas

El estudio de las *DC-Nets* como herramienta para brindar anonimato se ha incrementado en los últimos años, debido a los ataques encontrados a protocolos que ofrecían más rapidez a la hora de entregar mensajes, como *onion-routing*. Según *Wright et al.* [22] se deja claro que el protocolo *DC-Net* entrega mayor seguridad a la hora de asegurar anonimato, comparado con otras alternativas como *mixnets* y *onion-routing*, pero sufre problemas de escalabilidad. Es por ello que el foco de muchos trabajos de investigación ha sido cómo mantener la seguridad que ofrece el protocolo original, pero mejorando los problemas de escalabilidad que posee, y así construir una alternativa viable y eficiente a *TOR*, a la hora de mantenerse anónimo mientras se navega por Internet.

La primera implementación de una variante del protocolo *DC-Net* se encuentra en [13], donde se presenta el sistema *Herbivore*, el cual logra superar el problema de escalabilidad creando pequeños

[2]En el protocolo de *Zero-Knowledge proof* existen dos roles: el *prover* y el *verifier*. El *prover* tiene como misión demostrarle al *verifier* la propiedad deseada.

[3]En este trabajo, una *ZKPoK* específica será denotada como $PoK\{\alpha : P(\alpha)\}$. Los valores en $P(\alpha)$ se asumen públicos, a excepción de α. Una *ZKPoK* garantiza que el *prover* conoce el valor α.

cluster de participantes, reduciendo la cantidad de conexiones necesarias para enviar los mensajes anónimos. El sistema logra escalar hasta unas decenas de participantes. Sin embargo, no posee ninguna manera de asegurar la integridad del sistema, por lo que es vulnerable a ataques realizados por participantes maliciosos, con el objetivo de estropear la ejecución del protocolo.

Luego de *Herbivore*, las implementaciones que siguieron fueron realizadas por el grupo *DEDIS (Decentralized and Distributed Systems Research)*[4] de *Yale University*, quienes presentan tres sistemas: *Dissent*, *D3* y *Verdict* [7, 21, 20, 8]. Estos sistemas tienen como principal objetivo atacar el problema de escalabilidad presente en las *DC-Net* y que *Herbivore* lograba aumentar pero no muy significativamente. La solución apunta a utilizar servidores adicionales que participan en la comunicación entre los participantes, logrando de esta manera que los participantes no se deban conectar entre todos ellos (como una *DC-Net* tradicional), sino que se conectan a servidores intermediarios quienes funcionan como *brokers* y entregan los mensajes a todos los participantes presentes. Los sistemas propuestos fueron evolucionando, llegando a realizarse pruebas con hasta 1000 participantes enviando mensajes. Si bien en *Verdict* logran incluir criptografía para evitar que participantes maliciosos ataquen el sistema, el sistema se basa en que cada participante debe depositar una cuota de confianza en uno de estos servidores intermediarios que utiliza. Si bien esto puede parecer poco (depositar confianza en un servidor externo), en la práctica lleva a muchas dudas y dificultad para llevar a cabo, e incorpora un elemento que no está presente en el protocolo original. En la época actual, la confianza es un elemento que debe ser manejado con cuidado, procurando, desde el punto de vista científico, disminuir, y en mejor medida evitar, que los participantes deban confiar en terceros para poder asegurar su privacidad.

Luego de *Verdict*, las implementaciones que siguieron tenían como objetivo aumentar aun más la escalabilidad del sistema, llegando a aumentar en varios ordenes de magnitud con la adición de técnicas derivadas de *Private Information Retrieval (PIR)* [5]. Estos sistemas son *Riffle* [14] y *Riposte* [6]. Estos sistemas logran escalar hasta millones de usuarios de manera segura, pero aun dependen de servidores externos para poder realizar la comunicación entre los participantes, además de alejarse del protocolo original de *DC-Net*.

Otro sistema a destacar es *Vuvuzela* [19], el cual si bien no utiliza el protocolo *DC-Net*, sino que obtiene anonimato empleando servidores externos que son ocupados como *proxies* para los mensajes, es una solución moderna que oculta información (encripta los mensajes) y anonimiza a los emisores (ocultando metadata), pudiendo convertirse en una alternativa viable para la comunicación de mensajes anónimos.

El sistema propuesto en este trabajo se destaca por los siguientes puntos: (1) no necesita servidores externos ni que los participantes depositen confianza en otras partes del sistema, rescatando así el espíritu original del protocolo *DC-Net*, permitiendo la comunicación entre distintos participantes, desconocidos entre sí, al tiempo que se mantiene la seguridad y privacidad de cada uno de ellos; (2) no necesita fase de asignación de *slots* para cada uno de los participantes como lo hace *Dissent*, ya que el objetivo no es evitar la colisión de mensajes[5], sino más bien aceptar la posibilidad de colisión, y resolverla cuando se presenten, con lo que no se pierde tiempo potencialmente innecesario asignando *slots* de envío, cuando puede que no se presente colisión, o ésta sea

[4] `http://dedis.cs.yale.edu/`

[5] En una *DC-Net*, una colisión de mensajes es el envío simultáneo de dos o más mensajes por parte de distintos participantes durante la misma ronda

más rápida resolverla[6] que evitarla; (3) utiliza técnicas criptográficas que aseguran tanto seguridad incondicional, como la prevención de ataques por parte de participantes maliciosos que pretendan perturbar su correcto desarrollo; por último (4) se tiene asociada una aplicación final enfocada en el usuario, lo cual permite acercar el diseño de un protocolo criptográfico seguro hacia los usuarios finales, aspecto que no ha sido desarrollado por el resto de las propuestas estudiadas. Si bien el sistema propuesto no escala a la cantidad de usuarios que los últimos sistemas discutidos, es, en nuestra opinión, un precio justo a pagar para alcanzar el nivel de seguridad que este sistema ofrece.

[6]Resolver una colisión significa correr un mecanismo que permita el envío individual de todos los mensajes que fueron enviados de manera simultánea.

Capítulo 3

Diseño de la variante *DC-Net*

3.1. Conceptos Básicos

Para explicar el detalle de la variante de *DC-Net* desarrollada e implementada en este trabajo, es necesario aclarar algunos términos y conceptos en que se basa su diseño.

El diseño propuesto permite el envío de mensajes entre un grupo de participantes, quiénes forman el *anonymity set*, o conjunto de personas donde el enviador se "oculta". El envío de estos mensajes se realiza de manera anónima, es decir, no se puede relacionar el mensaje con algún participante que esté tomando parte del protocolo. Cada participante tiene la posibilidad de enviar un mensaje por sesión, la cual finaliza cuando todos los mensajes enviados son recibidos por todos los participantes. Para lograr el envío de estos mensajes, el protocolo se desarrolla en rondas, las cuáles son una serie de pasos (descritos a partir de la próxima sección) que todos los participantes deben desarrollar tanto para mantener el anonimato, como para prevenir ataques o intentos de corrupción del protocolo. Al conjunto de participantes involucrados en una sesión se le llama Sala. Como fue explicado anteriormente, el diseño original de *DC-Net* sólo permite el envío de un solo mensaje. Si se envían más mensajes se produce una colisión de mensajes, donde dos o más mensajes se "mezclan" resultando en un mensaje distinto, posiblemente ininteligible. Esta situación debe ser evitada o resuelta para poder tener un correcto envío de los mensajes en cuestión.

La variante propuesta considera un modelo de amenaza fuerte, donde un adversario puede monitorear todos los canales de comunicación entre los distintos participantes de la sala, y solo se exige que estos canales sean autenticados, es decir, que cada participante firme de manera digital los distintos valores que se están enviando. Además de esto, el protocolo mantiene su seguridad aún si existen hasta $n - 2$ participantes maliciosos (donde n es el tamaño de la sala, o el número total de participantes). Por éstos últimos entenderemos a participantes que tienen como objetivo ya sea de quebrar el anonimato del emisor del mensaje, o bien, corromper el protocolo impidiendo la recepción de los mensajes enviados y, en general, el normal desarrollo de éste.

Un último aspecto importante de mencionar es que, tanto los mensajes que se envían entre los participantes como todos los valores compartidos en la ejecución del protocolo, son públicos. Cualquier observador externo puede saber lo que sucede en el desarrollo del protocolo (qué computador transmite qué datos a qué otro computador), y aun así no podrá generar una relación entre un mensaje enviado y algún participante en particular dentro de la sala.

3.2. Resolución de Colisiones

El protocolo propuesto tiene como principal característica el no evitar las posibles colisiones que puedan producirse (en contraposición con otros protocolos que se enfocan en evitar la colisión de mensajes). De producirse la colisión, se debe resolver corriendo nuevas rondas del protocolo, pero solamente permitiendo que un subconjunto de los mensajes colisionados puedan re-enviarse en cada una de las rondas subsecuentes, produciendo así que en cada nueva ronda que se desarrolle, exista un menor número de mensajes colisionando, resultando así en rondas que se desarrollen sin colisión, enviándose los mensajes de manera solitaria, lo que les permite ser legibles por el resto de los participantes. Esta técnica de resolución de colisiones es conocida como *superposed receiving* [12].

Las rondas que se desarrollan durante el protocolo son de dos tipos: reales y virtuales. En las rondas reales es necesario el envío de múltiples mensajes por parte de los participantes, involucrando el uso de varias herramientas criptográficas con el objetivo de evitar que participantes maliciosos alteren el normal desarrollo del protocolo. En cambio las rondas virtuales son tales que su resultado puede inferirse por rondas reales ejecutadas anteriormente, por lo que no es necesario el envío de ningún mensaje por parte de los participantes (las rondas reales cuestan trabajo por parte de los participantes, y las rondas virtuales no). Si la primera colisión de mensajes es de n mensajes, eso implica que es necesario ejecutar n rondas reales para poder resolver dicha colisión. Esta técnica es conocida como *inference cancellation* [23].

En la Figura 3.1 se muestra un ejemplo de resolución de colisiones, mostrando a grandes rasgos la idea mencionada de no evitar las colisiones sino resolverlas de manera eficiente. En este ejemplo, 5 mensajes colisionan en la primera ronda, lo que significa que en las rondas subsecuentes (segunda y tercera), un subconjunto de esos 5 mensajes son reenviados (una parte de ellos se reenvía en la segunda ronda, y el resto se reenvía en la tercera ronda). Importante notar que el criterio para decidir si un mensaje es reenviado en la ronda 2 o en la ronda 3, tiene relación con el promedio de los mensajes que colisionaron (esta idea es explicada con mayor detalle más adelante). Si en las rondas 2 ó 3 se vuelve a producir una colisión (que es el caso que se produce en el ejemplo), se vuelve a realizar la misma idea, es decir, un subconjunto de los mensajes se reenvían en las rondas subsecuentes (para la colisión de la ronda 2, los mensajes se reenvían en las rondas 4 y 5; para la colisión de la ronda 3, los mensajes se reenvían en las rondas 6 y 7). Como regla general, si la colisión se produce en la ronda k, los mensajes se reenvían en las rondas $2k$ y $2k + 1$. Este protocolo de resolución se realiza hasta que todos los mensajes que colisionaron en la ronda 1, son reenviados en rondas de forma individual, es decir, no colisionaron con ningún otro mensaje (en el ejemplo, esto se produce en las rondas 4, 5, 6, 14 y 15). En la Figura 3.1 se puede observar que la ronda 3 es virtual, ya que su valor puede ser derivado a partir de los valores que se obtuvieron en las rondas 1 y 2. De manera general, las rondas virtuales poseen la forma $2k + 1$, cuyo valor puede ser derivado a partir de los valores generados en las rondas k y $2k$ (en la Figura 3.1, las rondas virtuales están expresadas a partir de líneas punteadas alrededor de los mensajes que son enviados).

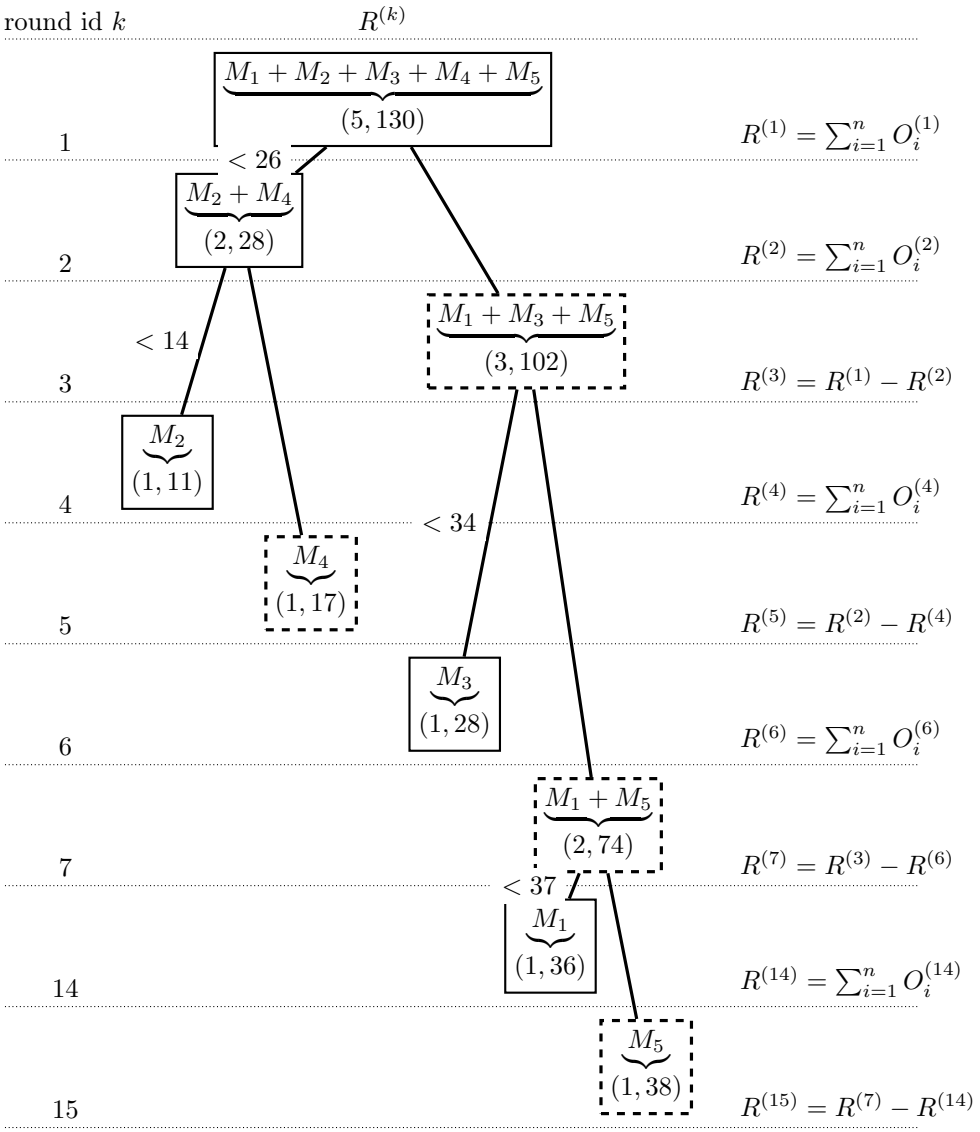


Figura 3.1: Ejemplo de árbol de resolución de colisiones utilizando *superposed receiving*.

3.3. Compartición de Claves

Una ronda real comienza con el proceso de compartición de claves entre cada par de participantes presentes en la sala. Este proceso debe realizarse, tomando un cuenta un posible adversario que esté monitoreando el canal de comunicación existente entre dos participantes cualquiera.

Para solucionar este problema se propone el uso del algoritmo *Diffie-Hellman* [11], el cual permite generar un valor secreto compartido entre dos agentes, cuyo canal de comunicación es monitoreado por un adversario.

Luego que cada par de participantes $\{P_i, P_j\}$ ejecuten el algoritmo *Diffie-Hellman*, obtendrán un valor secreto k_{ij} (y se define que $k_{ij} = -k_{ji}$). Para futuros pasos del protocolo, es necesario que ejecuten este proceso dos veces, ya que es necesario que cuenten con un segundo valor compartido $r(k_{ij})$ (también se define que $r(k_{ij}) = -r(k_{ji})$).

Si bien el proceso descrito (acuerdo *Diffie-Hellman*) no alcanza seguridad incondicional (como sí lo hace el resto del protocolo *DC-Net*), es posible alcanzarla precalculando estas claves en forma *offline* vía un canal privado. Por simplificación, la implementación realizada para este trabajo utiliza *Diffie-Hellman*.

3.3.1. *Commitments* sobre las claves

Posteriormente cada participante P_i debe realizar un *commitment* a cada valor k_{ij} que posee compartido con cada otro participante P_j, utilizando como aleatoriedad el segundo valor compartido $r(k_{ij})$. Así cada participante P_i calcula $n-1$ de los siguientes valores $c_{k_{ij}} = g^{k_{ij}} h^{r(k_{ij})}$.

Luego de esto, cada participante P_i genera dos valores. El primero es la suma de todas las claves compartidas que posee $K_i = \sum k_{ij}$. El segundo valor es un *commitment* sobre dicho valor, calculado utilizando los valores $c_{k_{ij}}$, esto es, $c_{K_i} = \prod c_{k_{ij}}$.

Finalmente cada participante P_i enviará al resto de la sala vía *broadcast* su *commitment* c_{K_i} junto con una *zero-knowledge proof* demostrando que conoce los valores "escondidos" dentro del *commitment* c_{K_i}.

3.3.2. Verificación de los valores enviados

Cada participante, al recibir los *commitments* enviados por el resto de la sala, debe verificar que se cumple la siguiente propiedad: $\prod c_{K_i} = 1$ (esto se debe a la cancelación que se debe producir al sumar las claves). Si es que lo anterior no se produce, es resultado de que uno (o varios) de los participantes envió un valor incorrecto dentro de su *commitment* respectivo, lo que implicaría una no cancelación de las claves compartidas en etapas subsecuentes del protocolo, alterando la entrega de los mensajes.

Es posible encontrar al(a los) participante(s) culpable(s) de enviar información errónea. Para ello cada participante P_j revelará cada valor $c_{k_{ji}}$ calculado anteriormente. Luego de esto, el resto de la sala puede verificar que se deben cumplir dos propiedades: (1) $c_{K_j} = \prod c_{k_{ji}}$, que se debe tener por definición, y (2) $c_{k_{ij}} c_{k_{ji}} = 1$, lo cual se debe a la cancelación de las claves compartidas. Si alguna de las dos propiedades anteriores no se cumplen, el participante P_j es malicioso y deben tomarse las medidas pertinentes que considere la sala, como por ejemplo, expulsión de los participantes maliciosos.

3.4. Formato del Mensaje

Al principio de una sesión, cada participante P_i debe decidir si desea comunicar un mensaje msg al resto de la sala o no. De querer enviar un mensaje, habrán rondas que deberá enviar $m_i = msg$, y otras que enviará $m_i = 0$. De no querer transmitir un mensaje, en todas las rondas enviará $m_i = 0$.

Para que el protocolo funcione correctamente, es necesario que dicho mensaje m_i se concatene con otros valores, permitiendo el correcto manejo de colisiones de mensajes (lo cual es explicado más adelante en el documento). Estos valores auxiliares son los siguientes:

- b_i: bit que indica si el participante está, en la presente ronda, enviando un mensaje distinto a cero o no ($b_i = 1$ *ssi* $m_i \neq 0$).
- pad_i: cadena de bits aleatoria que se añade para prevenir colisión de mensajes iguales, que podría desembocar en que la sesión nunca finalice.

Con estos valores establecidos, en cada ronda, cada participante P_i debe formar el valor M_i descrito en las figuras 3.2 y 3.3. En dichas figuras se muestra el formato cuando el participante envía $m_i \neq 0$, y cuando el participante envía $m_i = 0$ (que significa $M_i = 0$).

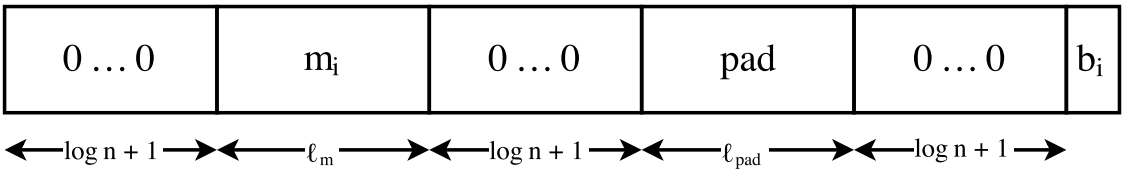


Figura 3.2: Formato de M_i cuando se envía un mensaje

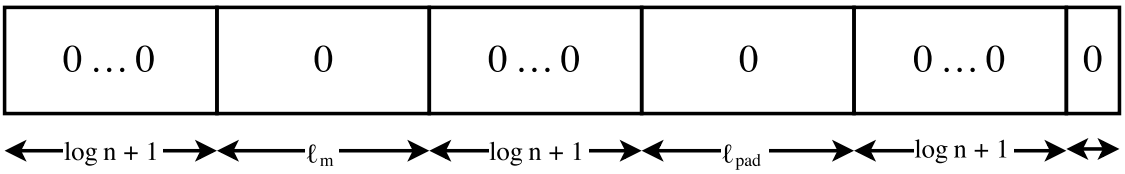


Figura 3.3: Formato de M_i cuando no se envía un mensaje

Las cadenas de bits compuestas por 0s puestas entremedio de los valores, sirven para prevenir *overflows* de los valores, impidiendo que se altere el resultado final al ocurrir una colisión.

3.5. Correctitud del Formato del Mensaje

Para que el protocolo tenga un correcto funcionamiento en el manejo de colisiones, es imperioso que el formato del mensaje M_i se respete por todos los participantes. Para ello se debe satisfacer una de las siguientes restricciones:

- Si $b_i = 0$, entonces $m_i = 0$
- $b_i = 1$

En la primera alternativa, se le fuerza al participante a que si esta diciendo que no envía un mensaje ($b_i = 0$), no lo envíe ($m_i = 0$). En la segunda opción, se le permite cualquier valor de m_i siempre y cuando se cumpla que $b_i = 1$.

3.5.1. *Commitments* sobre los valores individuales

Para poder demostrar que el mensaje M_i posee un correcto formato, cada participante P_i debe primero realizar *commitments* a cada uno de los valores $\{m_i, pad_i, b_i\}$. Para ello, se escogen valores aleatorios $\{r(m_i), r(pad_i), r(b_i)\}$ y se calculan los valores $c_{m_i} = g^{m_i} h^{r(m_i)}; c_{pad_i} = g^{pad_i} h^{r(pad_i)}; c_{b_i} = g^{b_i} h^{r(b_i)}$ (*commitments* a los valores individuales anteriormente descritos).

3.5.2. *Zero-knowledge Proof* sobre formato del mensaje

Ahora es momento que cada participante genere una *zero-knowledge proof* que demuestre que el mensaje M_i esta bien formado. Para ello es necesario establecer primero si en la presente ronda el participante enviará un mensaje ($m_i \neq 0$) o no ($m_i = 0$).

1. Si el participante i-ésimo no envía un mensaje ($m_i = 0$): para demostrar que el formato es correcto cuando no se necesita enviar un mensaje, cada participante debe demostrar la primera restricción que se mostró anteriormente, esto es, que $b_i = 0 \wedge m_i = 0$.

   Es importante notar que cuando sucede lo anterior, los valores de los *commitments* anteriormente descritos quedan de la siguiente manera: $c_{m_i} = h^{r(m_i)}; c_{b_i} = h^{r(b_i)}$.

   Para ello, usando *ZKPs*, el participante debe demostrar que conoce los valores $\{r(m_i), r(b_i)\}$ contenidos en dichos *commitments*.
2. Envío de mensaje ($m_i \neq 0$): para demostrar que el formato es correcto cuando necesita enviar un mensaje, debe demostrar la segunda restricción, esto es solamente que $b_i = 1$.

   Al igual que en el caso anterior el *commitment* asociado queda con un formato particular $c_{b_i} = g h^{r(b_i)}$.

   En este caso se le va a pedir al participante demostrar que conoce el logaritmo discreto $r(b_i)$ de $g^{-1} c_{b_i} = h^{r(b_i)}$ en base h.

Como no se puede saber si el participante enviará o no un mensaje (comprometería su anonimato) se va a solicitar que demuestre cualquiera de las dos condiciones anteriores. La *zero-knowledge proof-of-knowledge* (no interactiva)[1] que necesita demostrar cada participante es la siguiente:

$$\texttt{PoK}_\texttt{i}^{\texttt{format}} = PoK\{(r(m_i), r(b_i)) : (c_{m_i} = h^{r(m_i)} \wedge c_{b_i} = h^{r(b_i)}) \vee (g^{-1} c_{b_i} = h^{r(b_i)})\}$$

[1] El protocolo *ZKP* no interactivo se detalla en el Apéndice A.

3.5.3. Envío de valores a la sala

Después de todos los cálculos anteriores, cada participante P_i debe enviar al resto de la sala vía *broadcast* los siguientes valores: $\{c_{m_i}, c_{pad_i}, c_{b_i}, \mathtt{PoK}_\mathtt{i}^{\mathtt{format}}\}$ (los tres *commitments* calculados y la demostración correspondiente).

3.5.4. Verificación de la demostración

Después de que toda la sala haya recibido todos los conjuntos de valores enviados por cada uno de los participantes, es necesario verificar que la demostración sea correcta, y para ello solamente es necesario los valores de los *commitments* que acompañarán a la *zero-knowledge proof*.

3.6. Cálculo del mensaje a enviar y su *commitment*

Cada participante P_i debe enviar un *commitment* sobre el mensaje M_i construido anteriormente, con el objetivo que más adelante envíe el mensaje que ha estado construyendo durante la presente ronda. Para construir dicho *commitment* utilizará los *commitments* de los valores individuales enviados anteriormente. Para ello calculará $c_{M_i} = f(c_{m_i}, c_{pad_i}, c_{b_i})^2 = g^{M_i} h^{r(M_i)}$ (donde $r(M_i)$ es un valor aleatorio definido por los valores $r(m_i), r(pad_i), r(b_i)$).

Luego de esto se necesitará que cada participante envíe una *zero-knowledge proof* demostrando que conoce los valores $M_i, r(M_i)$ dentro de c_{M_i}:

$$\mathtt{PoK}_\mathtt{i}^\mathtt{M} = PoK\{(M_i, r(M_i)) : c_{M_i} = g^{M_i} h^{r(M_i)}\}$$

.

3.6.1. Verificación de la demostración

Por cada participante P_j que envía la demostración anterior, es necesario primero formar el valor c_{M_j} utilizando los valores de los *commitments* recibidos anteriormente de la siguiente manera: $c_{M_j} = f(c_{m_j}, c_{pad_j}, c_{b_j})$.

Luego de esto, es posible verificar la demostración $\mathtt{PoK}_\mathtt{j}^\mathtt{M}$ utilizando el valor c_{M_j} construido anteriormente.

[2] $f(x, y, z) = x^{2^{|pad|}(n+1)} y^{(n+1)} z$. Esta fórmula se obtiene a partir del formato del mensaje M_i descrito en la Figura 3.2.

3.7. Envío del mensaje de salida

En este punto, cada participante P_i ha demostrado la correctitud de los valores con que se ha comprometido a enviar (hasta ahora, solo ha enviado *commitments* y demostraciones, ningún valor "útil" para la ejecución del protocolo).

Cada participante P_i generará el valor $O_i = K_i + M_i$ (denominado mensaje de salida). Este valor tiene como objetivo "ocultar" al mensaje M_i sumándole el valor secreto K_i. Informalmente, cuando un participante reciba un mensaje O_i, no podrá discernir si este mensaje de salida contiene o no un valor $M_i \neq 0$.

Ahora bien, es necesario realizar una distinción con respecto a lo que cada participante necesita demostrar con respecto a su mensaje de salida O_i. Cuando se desarrolla la ronda 1, solo es necesario demostrar que dicho mensaje O_i se forma a través de los valores "ocultos" en los *commitments* $\{c_{K_i}, c_{M_i}\}$ enviados anteriormente (c_{M_i} no se envió explícitamente, pero se construyó a partir de los *commitments* individuales que sí se enviaron).

En las rondas posteriores, la demostración se vuelve más compleja, ya que se necesita demostrar que el valor M_i que forma parte del mensaje de salida O_i es, ya sea, igual a 0 (no le corresponde enviar un mensaje en dicha ronda), o bien, es el mismo mensaje que el involucrado en la colisión más próxima (los detalles serán analizados más adelante).

3.7.1. Ronda 1

Cada participante P_i necesita generar el siguiente *commitment* para O_i, $c_{O_i} = g^{O_i} h^{r(O_i)}$. Para formar este valor se utilizan los *commitments* enviados anteriormente de la siguiente manera:

$$c_{O_i} = g^{O_i} h^{r(O_i)} = g^{K_i + M_i} h^{r(K_i)+r(M_i)} = g^{K_i} h^{r(K_i)} g^{M_i} h^{r(M_i)} = c_{K_i} c_{M_i}$$

Con lo anterior, se le solicita a cada participante que demuestre en una *zero-knowledge proof-of-knowledge* que conoce el valor $r(O_i) = r(K_i) + r(M_i)$ (suma de las aleatoriedades utilizadas en los *commitments* para la llave K_i y el mensaje M_i) dentro de un *commitment* especial $c_{r(O_i)} = h^{r(O_i)}$. La *zero-knowledge proof-of-knowledge* resulta en $\texttt{PoK}^{\texttt{O}}_{\texttt{i}} = PoK\{r(O_i) : c_{r(O_i)} = h^{r(O_i)}\}$.

Finalmente cada participante enviará el par $\{O_i, \texttt{PoK}^{\texttt{O}}_{\texttt{i}}\}$ al resto de la sala vía *broadcast*.

Verificación de la demostración

Cada participante P_i debe verificar la demostración recibida de cada otro participante P_j. Para ello, primero deberá calcular el valor c_j^O como la multiplicación de los *commitments* recibidos en pasos anteriores de la siguiente manera: $c_j^O = c_j^K \cdot c_j^M$.

Luego de esto, deberá calcular, utilizando el valor O_j recibido recién, el siguiente valor:

$$\beta_j = c_j^O \cdot (g^{O_j})^{-1} = g^{O_j} h^{r(O_j)} (g^{O_j})^{-1} = h^{r(O_j)}$$

.

Finalmente se debe verificar la demostración $\texttt{PoK}^{\texttt{O}}_{\texttt{j}}$ utilizando el valor β_j calculado anteriormente.

3.7.2. Rondas posteriores

En rondas posteriores a la primera, se debe asegurar que los participantes estén enviando un mensaje válido, el cual solo puede tener dos alternativas: o bien no envía ningún mensaje (ya sea porque su mensaje ya fue recibido por todos los participantes en una ronda sin colisión, o no está habilitado para enviar mensaje en esta ronda), o envía exactamente el mismo mensaje que se envió al principio (no ha cambiado el mensaje que quiere comunicar en rondas posteriores).

Para lograr esto, es necesario hacer una distinción con respecto a la ronda exactamente anterior a la ronda actual. Si la ronda anterior fue una ronda real, se verifica que el mensaje enviado en la ronda actual sea el mismo que el enviado en dicha ronda anterior; pero si la ronda anterior es virtual, no hay ningún mensaje con el que comparar (no se envía ningún mensaje en rondas virtuales). A esta ronda anterior se le denominará *ronda padre*.

Caso 1: Ronda padre real

Si actualmente se está desarrollando la ronda k, entonces la ronda padre es la ronda $k/2$. Se va a denotar con un superíndice entre paréntesis cuadrados la ronda en que se envía cada mensaje.

Primero, todo participante P_i necesita calcular el siguiente valor $\alpha_i = (c_{m_i}^{[\mathtt{k}]})^{-1} \cdot (c_{m_i})^{[\mathtt{k}/2]}$. El valor α_i representa la multiplicación del inverso del *commitment* sobre el mensaje m_i enviado en la ronda actual con el mismo *commitment*, pero enviado en la ronda padre.

Notar que el valor α_i se utilizará exclusivamente cuando el participante deba demostrar que envía un mensaje. La otra situación (no enviar un mensaje) se demostrará, igual que en un paso anterior, mostrando que $b_i = 0 \wedge m_i = 0$. Cuando sucede la primera alternativa (enviar un mensaje, y que éste sea igual que al enviado en la ronda padre), α_i toma el siguiente valor:

$$\begin{aligned}
\alpha_i &= (g^{m_i} h^{r_i^m})^{[\mathtt{k}]^{-1}} \cdot (g^{m_i} h^{r_i^m})^{[\mathtt{k}/2]} \\
&= \frac{1}{(\cancel{g^{m_i}} h^{r_i^m})^{[\mathtt{k}]}} \cdot (\cancel{g^{m_i}} h^{r_i^m})^{[\mathtt{k}/2]} \\
&= h^{r_i^{m\,[\mathtt{k}/2]}} \cdot \frac{1}{h^{r_i^{m\,[\mathtt{k}]}}} \\
&= h^{(r_i^{m\,[\mathtt{k}/2]} - r_i^{m\,[\mathtt{k}]})} \\
&= h^{r_i^{\tilde{m}}}
\end{aligned}$$

Con lo anterior, lo que debe demostrar cada participante P_i en este punto es que su mensaje O_i (enviado en la ronda k actual) cumple una de las siguientes alternativas: (1) codifica un mensaje m_i igual al mensaje m_i enviado en la ronda padre (ronda $k/2$), o (2) codifica un mensaje $m_i = 0$, lo cual se traduce en demostrar que $b_i = 0 \wedge m_i = 0$. Con esto, la *zero-knowledge proof-of-knowledge* necesaria es la siguiente:

$$\mathtt{PoK}_\mathtt{i}^{\mathtt{0,real}} = PoK\{(r(b_i), r(m_i), r(\tilde{m}_i)) : (c_{b_i} = h^{r(b_i)} \wedge c_{m_i} = h^{r(m_i)}) \vee (\alpha_i = h^{r(\tilde{m}_i)})\}$$

Luego de calcular la *zero-knowledge proof* anterior, cada participante P_i va a enviar al resto de la sala vía *broadcast* el par $\{O_i, \mathtt{PoK}_\mathtt{i}^{\mathtt{0,real}}\}$.

Para verificar la demostración enviada por cada participante P_j, primero se deben rescatar los *commitments* sobre el mensaje m_j enviados tanto en la ronda k actual como la ronda padre $k/2$. Esto se utiliza para formar el valor $\alpha_j = (c_{m_j}^{[\mathtt{k}]})^{-1} \cdot (c_{m_j})^{[\mathtt{k}/2]}$. Finalmente se verifica $\mathtt{PoK}_\mathtt{j}^{\mathtt{0,real}}$ utilizando α_j y los *commitments* c_{b_j} y c_{m_j} enviados en la ronda actual.

Caso 2: Ronda padre virtual

Cuando la ronda padre $k/2$ es virtual, no es posible ir a rescatar los valores necesarios, ya que en esa ronda $k/2$ no se envió ningún mensaje. Por lo tanto, en vez de comparar los mensajes enviados en la ronda k actual con los de la ronda padre, se comparan con la ronda real más cercana que exista en la línea directa entre la ronda k y la ronda 1.

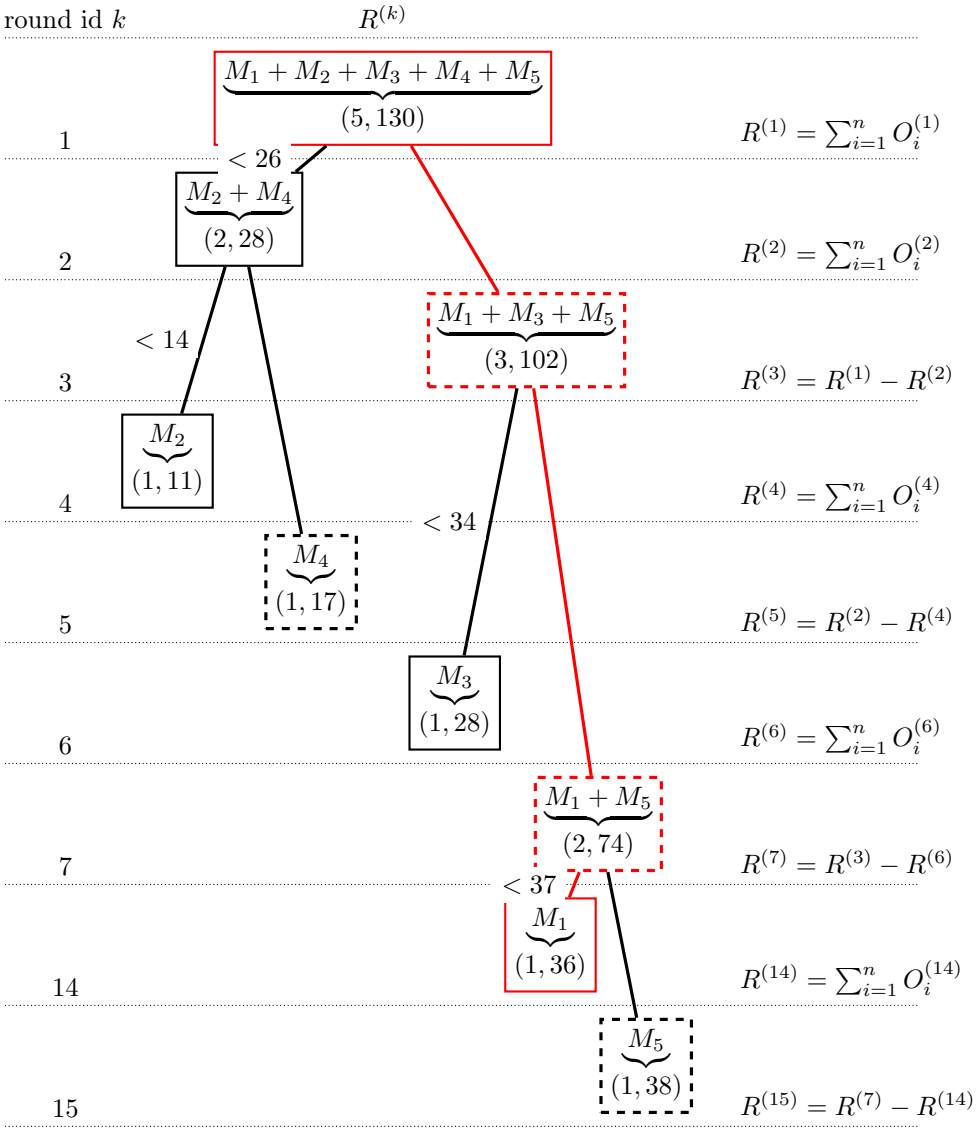


Figura 3.4: Ejemplo de árbol de resolución de colisiones utilizando *superposed receiving*, explicitando el concepto de "línea directa" entre una ronda real (por ejemplo la ronda 14) y la ronda 1.

Las demostraciones necesarias son exactamente iguales que en el caso anterior, pero todo participante P_i, en vez de utilizar la ronda $k/2$, debe usar la ronda `nrst_real` (ronda real más cercana). Tanto el envío como la verificación de las demostraciones son análogas al escenario anterior (ronda padre real).

En la Figura 3.4 se evidencia este concepto por la línea roja que sigue desde la ronda 14 (ronda real cuya ronda padre es virtual), hasta la ronda 1. En este caso la ronda real más cercana en dicho camino (ronda 7 → ronda 3 → ronda 1) es justamente la ronda 1 (peor caso), por lo que en el ejemplo de la figura, cuando $k = 14$, `nrst_real` $= 1$.

3.7.3. Suma de mensajes de salida

Independiente de las demostraciones que se necesitaban, en cada uno de los casos cada participante P_i envió su mensaje de salida O_i. Al juntar todos los mensajes O_i recibidos en la ronda actual k, se produce el valor $R^{[k]}$, denominado mensaje de la ronda k:

$$R^{[k]} = \sum O_i = \sum \cancelto{0}{K_i} + \sum M_i = \sum M_i$$

Dicha cancelación se produce debido a que las claves compartidas K_i están construidas de manera que al sumarse todas juntas se cancelen.

3.8. Ronda Virtual

Cuando la ronda actual es de la forma $k = 2j + 1$, quiere decir que es una ronda virtual. Se denomina de esta manera debido a que no es necesario que se envíen mensajes por parte de los participantes, sino que el mensaje de la ronda se puede deducir utilizando los mensajes de las rondas $2j$ y j de la siguiente manera:

$$R^{[2j+1]} = R^{[j]} - R^{[2j]}$$

3.9. Resolución de la ronda

Independiente si la ronda k actual fue desarrollada como una ronda real o como ronda virtual, en este punto del protocolo se posee un mensaje de ronda $R^{[k]}$. Este mensaje $R^{[k]}$ representa la suma de los mensajes individuales M_i enviados por cada uno de los participantes P_i en la actual ronda. Pueden darse dos situaciones con respecto a $R^{[k]}$: (1) solamente un participante P_i envió un mensaje en la actual ronda, ó (2) varios participantes enviaron su mensaje, resultando en una colisión de mensajes que es necesario resolver. Ambos escenarios son explorados a continuación.

Recordemos, que por el formato que posee cada mensaje M_i, es posible saber con precisión cuántos mensajes colisionaron en cada ronda, debido a como los mensajes se suman bit a bit, se va a tener la suma de cada valor b_i individual, resultando en la cantidad de participantes que enviaron un mensaje en la actual ronda.

Para explicitar el número de mensajes recibidos en la ronda, se denotará $R^{[k]} = (\hat{m}^{[k]}, t^{[k]})$, donde $\hat{m}^{[k]}$ corresponde a la suma de los mensajes m_i enviados por cada participante P_i en la ronda k, mientras que $t^{[k]}$ representa cuantos mensajes están involucrados en dicha suma.

Primera colisión

Es importante hablar de lo que sucede en la primera ronda, debido a que establecerá lo que sucederá a futuro con el protocolo.

Recordemos que $R^{[1]} = (\hat{m}^{[1]}, t^{[1]})$. Lo que importa en este momento es el valor $t^{[1]}$. Este valor referencia el número de mensajes individuales que colisionaron en la primera ronda, es decir, el número de participantes P_i que desearon comunicar un mensaje de manera anónima, pero que no pudieron transmitir dicho mensaje debido a que colisionó con otros mensajes.

Por lo tanto, existen $t^{[1]}$ mensajes que deben ser revelados al resto de los participantes, es decir, ser enviados sin colisionar con otros mensajes. Es por ello que el protocolo seguirá su ejecución hasta que se hayan desarrollado $t^{[1]}$ rondas sin colisión (explicadas a continuación). Además el protocolo es eficiente con respecto a las rondas reales, debido a que si $t^{[1]}$ mensajes colisionaron en la primera ronda, entonces el protocolo tendrá exactamente $t^{[1]}$ rondas reales para revelar los $t^{[1]}$ mensajes involucrados. Se define `session_msgs` $= 0$. Cuando este valor llegue a $t^{[1]}$, el protocolo finaliza.

3.9.1. Ronda sin colisión

Una ronda sin colisión sucede cuando $t^{[k]} = 1$, es decir, solamente un participante envió su mensaje en la actual ronda, por lo que $\hat{m}^{[k]} = m_i$, para algún participante P_i.

Con esto suceden varias cosas interesantes: (1) el mensaje m_i es entregado en forma transparente, es decir, tanto todos los participantes de la sala como cualquier observador externo reciben el mensaje m_i, tal como quiso ser transmitido; (2) el participante P_i queda satisfecho con el protocolo, debido a que pudo transmitir su mensaje m_i de manera anónima, por lo que de aquí en adelante contribuirá con el anonimato enviando $m_i = 0$. Finalmente (3) el número de mensajes enviados de manera satisfactoria `session_msgs` aumenta en uno, acercándose a que todos los mensajes sean enviados sin colisiones.

Cabe destacar que si `session_msgs` llega a ser $t^{[1]}$ después de ejecutarse la ronda, el protocolo finaliza de manera inmediata.

3.9.2. Resolución de colisiones

Una colisión se produce cuando $t^{[k]} > 1$, lo que quiere decir que más de un participante P_i envió su mensaje en la ronda k, por lo que el valor $\hat{m}^{[k]}$ es ininteligible (corresponde a la suma de varios mensajes individuales m_i). Para resolver esto es necesario correr un protocolo de resolución de colisiones, que tiene como objetivo resolver la colisión (que en próximas rondas cada uno de los mensajes individuales m_i involucrados en la colisión se envíen de manera solitaria). Informalmente, lo que se realiza es forzar a los participantes involucrados a enviar los mismos mensajes m_i en rondas distintas, haciendo que en rondas subsecuentes la cantidad de mensajes que se envíen sea cada vez menor, resultando *a posteriori* en rondas donde solamente un mensaje es enviado.

El criterio para "dividir" los mensajes M_i será decidir si el mensaje es menor o mayor que el mensaje promedio $\overline{m}^{[k]} = \hat{m}^{[k]}/t^{[k]}$.

Cada participante P_i deberá calcular cuando (en que ronda) se le permite reenviar su mensaje m_i:

- Si $m_i \leq \overline{m}^{(k)}$, el participante reenviará su mensaje en la ronda $2k$.
- Si $m_i > \overline{m}^{(k)}$, el participante reenviará su mensaje en la ronda $2k+1$ (esta ronda será virtual).

Con este criterio el protocolo asegura que las rondas $2k$ y $2k+1$ tendrán valores $t^{[2k]}, t^{[2k+1]}$ menores a $t^{[k]}$, resultando en que (de manera recursiva) en rondas subsecuentes se envíen los mensajes de forma solitaria (sin colisiones).

Finalmente, luego de establecer cuando los participantes involucrados en la colisión enviarán sus mensajes, la ronda actual k finaliza, dando paso a la ronda $k+1$, que solamente se desarrolla si al menos algún participante está habilitado para enviar un mensaje (se puede saber de manera pública), de no haber participantes habilitados para enviar mensajes en dicha ronda, se salta a la ronda $k+2$ y así sucesivamente. La próxima ronda se desarrolla de manera recursiva tal como fue explicado anteriormente, primero viendo si corresponde a una ronda real o virtual, y luego llevando a cabo los pasos necesarios por todos los participantes en la sala.

Capítulo 4

Detalles de Implementación

La parte esencial de este trabajo esta dada por llevar a cabo una implementación que refleje el diseño propuesto en el capítulo anterior, brindando así una herramienta real a la comunidad que permita la comunicación anónima entre distintas partes interesadas.

Como fue explicado anteriormente, no existen muchas implementaciones de protocolos basados en *DC-Net*, por lo que llevar a cabo este trabajo significó superar muchos obstáculos, los cuales se detallan a continuación.

4.1. Tecnologías Involucradas

1. Lenguaje de Programación: para llevar a cabo este proyecto se escogió realizarlo en *Java*, debido principalmente a que un objetivo era poder realizar una aplicación móvil como prueba de concepto de lo implementado, y el sistema operativo móvil más común hoy en día es *Android*[1], el cual está basado en *Java*. Aparte de esto, no existe una razón de fondo para escoger *Java* como lenguaje de programación, por lo que los mismos resultados se pueden lograr si se utiliza otro lenguaje más común en otras implementaciones de *DC-Net*, como sería *C++*.
2. Capa de comunicación: para la conexión entre las distintas partes se utilizó *ZeroMQ*, el cual es un *framework* de concurrencia, que permite desligarse de lidiar con problemas típicos de mensajería distribuida (desconexiones, pérdida de datos, etc.) y concentrarse únicamente en la lógica de la aplicación. En palabras de sus autores, "*ZeroMQ* son sockets con esteroides". Con *ZeroMQ* se simplifica la implementación de *broadcasting* o de conexiones punto a punto entre los distintos participantes (ambos tipos de conexiones necesarias para el protocolo).

[1] `https://www.idc.com/promo/smartphone-market-share/os`

4.2. Arquitectura del Sistema: Nodo `Directorio` y Nodos `Participantes`

Un desafío importante a considerar en la implementación (y que el diseño del protocolo no se preocupa) es como cada participante descubre la locación del resto de los participantes que formarán parte del *anonymity set*. Para ello se tomó la decisión de contar, además de los nodos `Participantes`, con un nodo `Directorio`. Este nodo funcionará como punto de entrada al *anonymity set* y será el responsable de informar la dirección IP de cada uno de los nodos `Participantes` presentes en la sala. Además de esto, el nodo `Directorio` tiene como responsabilidad establecer los parámetros necesarios para correr el protocolo (número de participantes que admitirá la sala, valores públicos para realizar los *commitments* y largo máximo de los mensajes a enviar, entre otros), por lo que también se vuelve un punto de control dentro del protocolo. Importante mencionar que la incorporación del nodo `Directorio` no modifica la seguridad y privacidad del sistema, debido a que el diseño explicado en el capítulo anterior se empieza a desarrollar *a posteriori* de la labor desempeñada por el nodo `Directorio`.

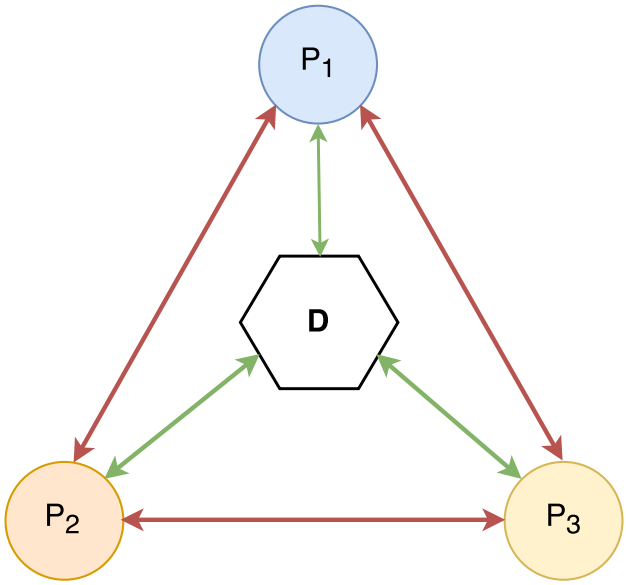


Figura 4.1: Conexión entre nodos `Directorio` y Participantes

La funcionalidad de proveer un punto de entrada a la sala también se podría realizar entre los propios nodos `Participantes`, sin la necesidad de incorporar un nodo `Directorio`. Si bien en esta investigación se priorizó la facilidad de implementar la variante utilizando el nodo adicional, se podría ejecutar alguna variante de *gossip protocol* [10] para informar la identidad de participantes nuevos que vayan entrando a la sala, lo cual podría solucionar el problema. Esta segunda variante además tiene la ventaja de no poseer un punto vulnerable (que sería el nodo `Directorio`), evitando ataques directos al nodo `Directorio`, retrasando (o incluso imposibilitando) la creación del *anonymity set*.

Actualmente el nodo `Directorio` inicia estableciendo los parámetros públicos del protocolo y publica su dirección IP. Luego, cada nodo Participante que se quiera unir se conecta a la dirección pública del `Directorio` y espera que se complete la cuota de participantes establecida en un

comienzo. Cuando se conectan los n participantes necesarios, el `Directorio` informa la dirección IP de cada uno de los participantes a todo el resto, para que posteriormente inicien el protocolo solo enviándose mensajes entre ellos. Esto termina la labor del nodo `Directorio`. En la Figura 4.1 se observa las conexiones que se realizan durante el desarrollo del protocolo, primero conectando cada participante que viene entrando a la sala al nodo `Directorio` (representado por las conexiones verdes), y luego que todos los participantes se hayan conectado al nodo `Directorio`, éste le comunica a toda la sala las direcciones de los nodos `Participantes` presentes, terminando su labor, desconectándose de los nodos `Participantes`, y estos finalmente se conectan mediante los enlaces rojos entre cada uno de ellos. En la Figura 4.2 se observa en mayor detalle la conexión entre cada par de participantes, la cual se realiza a través de *sockets* tipo *requestor-replier*, formando así el *anonymity set* deseado, y continuando con el normal desarrollo del protocolo anteriormente descrito.

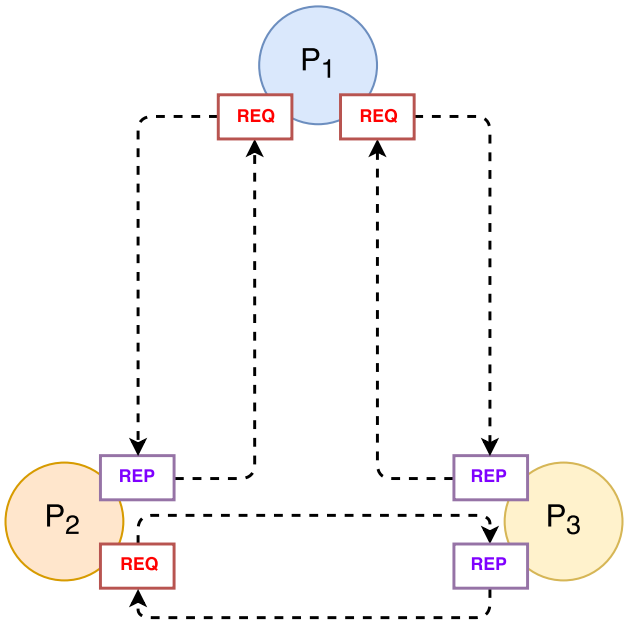


Figura 4.2: Conexión entre nodos `Participantes`

4.3. Primitivas Criptográficas

En la implementación actual, la gran mayoría de las primitivas criptográficas han sido implementadas desde cero, valiéndose principalmente de la biblioteca para manejar números grandes de *Java*, *BigInteger*[2]. Si bien esto no es una práctica recomendada (lo ideal es utilizar bibliotecas criptográficas ya probadas por la comunidad), se decidió por esto luego de no encontrar bibliotecas que implementaran las funcionalidades requeridas por el protocolo (*Pedersen Commitments* y *ZKP* asociadas). Como fue dicho, la implementación de primitivas criptográficas no es recomendado, más bien, se recomienda utilizar implementaciones ya probadas y verificadas por la comunidad debido

[2] `https://docs.oracle.com/javase/7/docs/api/java/math/BigInteger.html`

a la facilidad que existe en agregar vulnerabilidades a la hora de implementar protocolos criptográficos, como lo podría ser Charm-Crypto[3] o Scapi[4]. De todas maneras, la criptografía implementada se desarrolló utilizando interfaces, por lo que de tener una librería que cumpla los requerimientos criptográficos indicados en el Capítulo 3, su implantación podrá ser realizada de manera expedita y transparente al resto de la implementación.

4.4. *API* implementada

Como fue dicho anteriormente, una contribución importante de este trabajo es implementar una *API* disponible para toda la comunidad[5]. Esta API permitirá a la comunidad crear una aplicación que esté basada en el protocolo anteriormente descrito. En 9.2 se destacan algunas secciones del código fuente implementado.

La implementación anterior fue empaquetada como librería *Java*, utilizable por otras aplicaciones al importar el archivo *.jar* generado. Esta librería quedó con los siguientes métodos de manera pública, formando así la *API* disponible:

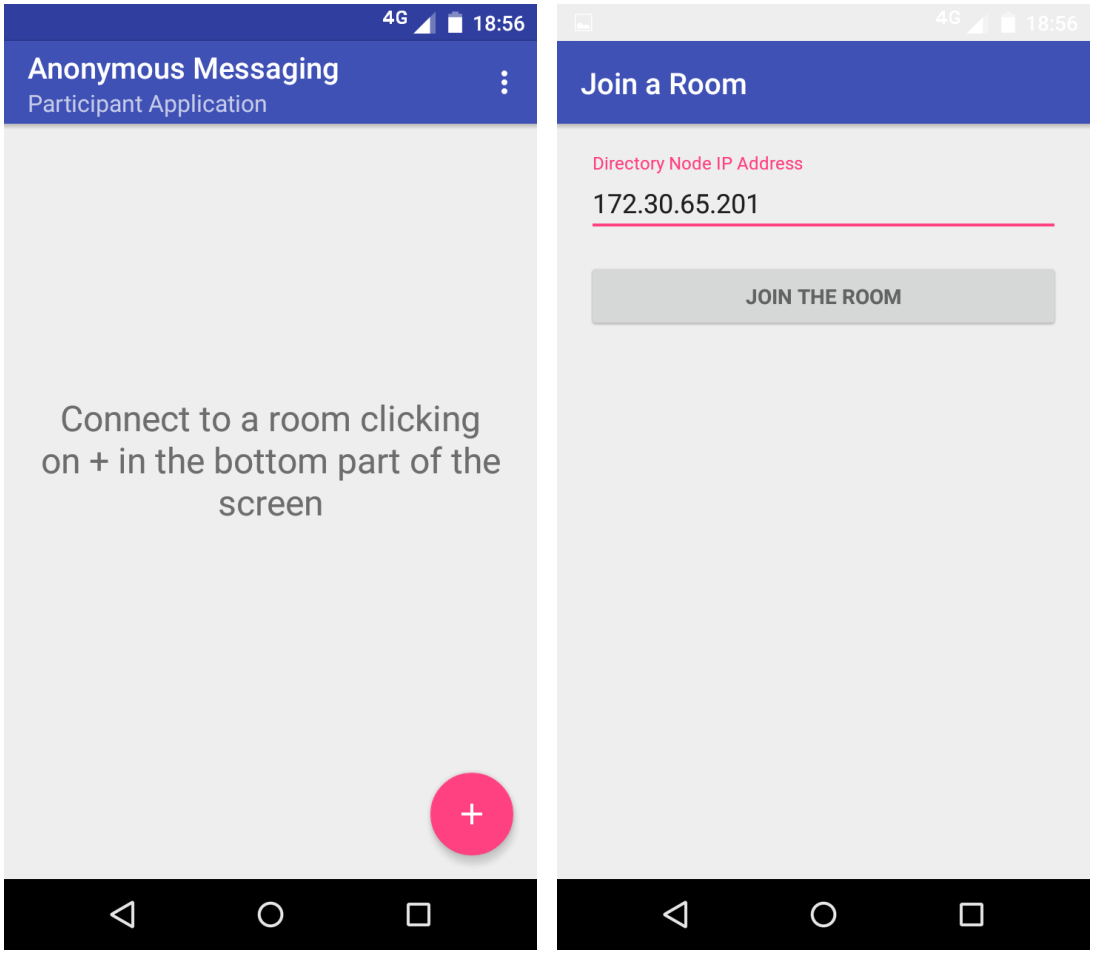
- `DCNETProtocol class:` clase base que entrega la *API* que es necesario instanciar para utilizar los métodos descritos a continuación.
- `boolean connectToDirectory():` luego de setear la dirección IP del nodo `Directorio`, este método realiza la conexión desde el nodo `Participante` hacia el `Directorio`, utilizando las funciones provistas por la librería *ZeroMQ*.
- `void setMessageToSend(String s, boolean b):` este método tiene como finalidad permitir al nodo `Participante` establecer el mensaje que desea comunicar al resto de la sala utilizando el protocolo.
- `void runProtocol(PrintStream p):` corre el protocolo anteriormente descrito de manera automática, calculando los diferentes *commitments* y *zero-knowledge proofs* necesarios, repitiendo las rondas necesarias y finalizando cuando todos los mensajes han sido recibidos por la totalidad de los participantes en cuestión.
- `ObservableParticipantsLeft:` objeto que emplea el patrón de diseño *Observer* para ir informando cuáles participantes se van conectando a la sala a medida que va ocurriendo, y así tener una noción de cuantos participantes faltan para completar la cuota puesta en un principio por el nodo `Directorio`.
- `ObservableMessagesArrived:` también emplea el patrón de diseño *Observer* para informar al participante de los mensajes que van resultando de correr el protocolo, a medida que éste avanza. Importante señalar que los mensajes se despliegan en tiempo real.

[3] `http://charm-crypto.com/index.html`
[4] `https://scapi.readthedocs.io/en/latest/`
[5] `https://github.com/clcert/dcnet-participant`, `https://github.com/clcert/dcnet-directory`

4.5. Aplicación Móvil

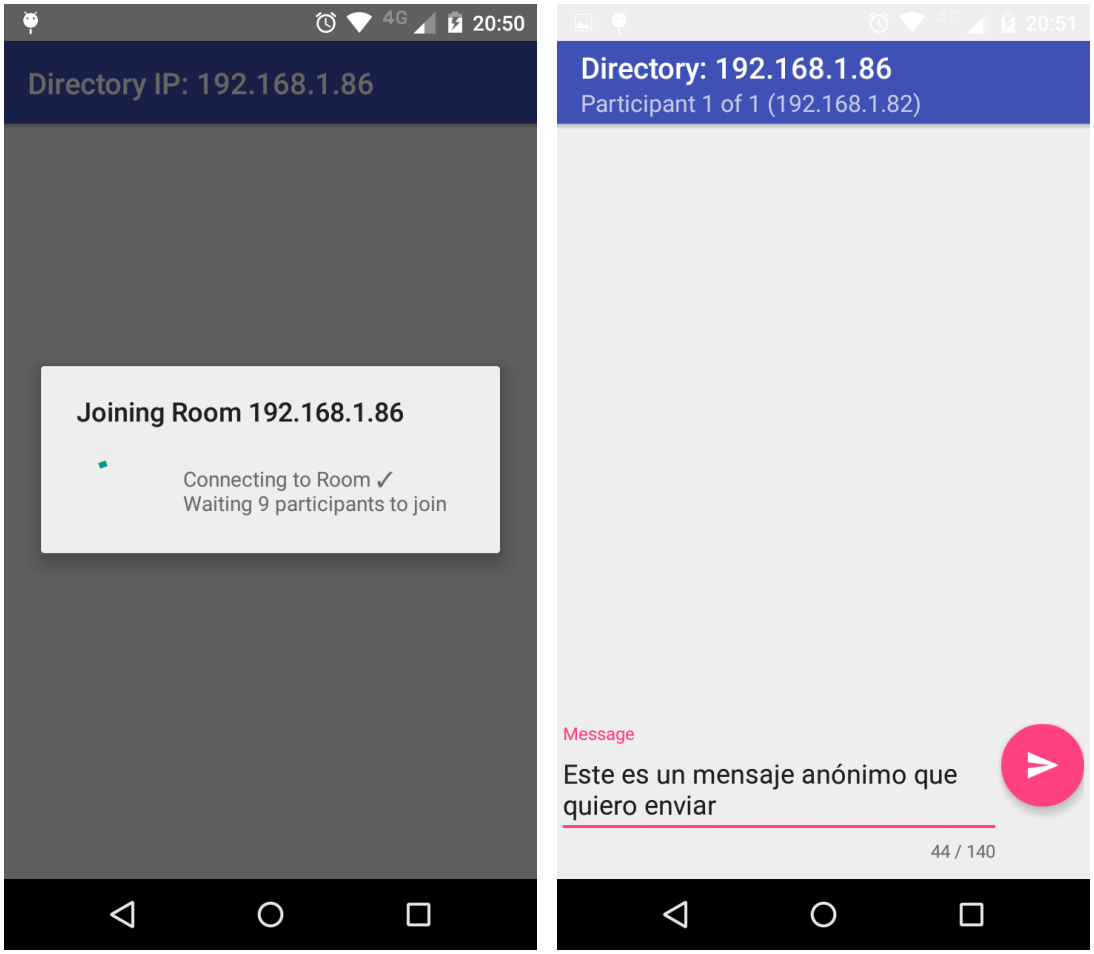
Para poder probar el uso de la *API* implementada, se desarrolló una aplicación móvil prototipo, que tiene como objetivo permitir un uso simplificado del protocolo implementado, que permita a los usuarios montar una conversación que asegure el anonimato de sus participantes.

La aplicación está compuesta de 3 módulos: (1) conexión al nodo `Directorio`, (2) envío del mensaje por parte del participante, y (3) recepción de los mensajes del resto de los participantes de la sala. En las Figuras 4.3 y 4.4 se pueden apreciar *screenshots* de la aplicación implementada, mostrando las distintas fases y vistas que ésta posee: conexión al nodo `Directorio` y el envío del mensaje que se quiere comunicar, respectivamente.



(a) Conexión al Nodo `Directorio` (b) Establecer dirección del `Directorio`

Figura 4.3: Screenshots de Aplicación Móvil (parte 1)



(a) Esperando conexión a Sala (b) Envío del mensaje a la sala

Figura 4.4: Screenshots de Aplicación Móvil (parte 2)

Capítulo 5

Experimentación y Resultados

5.1. Metodología Experimental

Para poner a prueba el protocolo implementado se montaron distintos escenarios de experimentación, cada uno relacionados a distintas situaciones reales donde un sistema de mensajería anónima pueda ser necesario e imperante de utilizar.

La experimentación constó de dos etapas relacionadas a la infraestructura utilizada: (1) simulación de una red a través de un software especializado, y (2) utilización de una red real tanto para confirmar resultados entregados por la simulación, como para tener más precisión en los resultados entregados.

Las pruebas realizadas se centraron en la medición de tiempos reales de ejecución, es decir, el tiempo que un usuario del sistema percibe que tardó su uso en los distintos escenarios propuestos. Los tiempos a medir se centraron en hitos que suceden en toda ejecución del sistema, especificados más adelante en este capítulo.

5.2. Infraestructura utilizada

5.2.1. Simulación de Red

En un principio se hicieron experimentos utilizando software de simulación de redes para poder lograr tamaños de *anonymity set* aceptables. Este software utilizado fue *CORE*[1] el cual permite emular distintos nodos en una red simulada con los parámetros que el usuario estime convenientes. Utilizando este software se simularon nodos conectados a través de una red local, donde cada uno de los nodos se comporta como un participante dentro del protocolo anteriormente descrito.

Además de dicho software, se levantaron instancias de *Docker*[2], pudiendo así formar una red local entre los distintos contenedores corriendo en una misma máquina.

[1] `https://www.nrl.navy.mil/itd/ncs/products/core`
[2] `https://www.docker.com/`

Ambas simulaciones mostraron un pobre rendimiento a la hora de correr el protocolo, mostrando tiempos de un orden de magnitud mayores que los tiempos reales que se lograron *a posteriori*. Es por ello que el autor consideró no presentar dichos resultados en el presente trabajo debido a la posibilidad de malinterpretarse y que el protocolo se viera empobrecido debido a experimentos mal realizados. La razón de los tiempos erróneos entregados por el software de simulación se centrarían principalmente en la inexperiencia en su uso para poder obtener como resultado tiempos más cercanos a los tiempos reales de ejecución. Debido a que se pudieron realizar pruebas en una red real, no se insistió en la mejora de los tiempos de la simulación, aunque queda como un punto para trabajar a futuro y poder probar el sistema en un sinnúmero de escenarios de manera más eficaz.

5.2.2. Uso de red real

Luego de notar los pobres resultados obtenidos en las simulaciones, se pudo obtener acceso a equipos reales conectados a través de una red local. En particular se utilizó el laboratorio *Lorenzo* del Departamento de Ciencias de la Computación de la Universidad de Chile, el cual cuenta con 31 computadores, cantidad razonable para poder realizar las pruebas correspondientes.

5.3. Experimentos Realizados

Las pruebas realizadas fueron dos:

1. Tamaño de sala variable, todos los participantes envían: se varió el tamaño de la sala desde 3 hasta 30 participantes donde, en cada repetición, todos los participantes presentes envían un mensaje de 140 caracteres.
2. Tamaño de sala fijo, algunos participantes envían: se fijó el tamaño de la sala en 30 participantes, y en cada repetición se aumentó el número de mensajes enviados (cada uno de 140 caracteres), desde 1 hasta 30.

En cada una de las dos pruebas realizadas se midieron los siguientes parámetros:

- Tiempos de ejecución: se midieron tres distintos tiempos, (1) tiempo total de la sesión, (2) tiempo que demora en llegar el primer mensaje, y (3) tiempo promedio por ronda.
- *Profiling* de cada etapa del protocolo: se midió que porcentaje del tiempo total se gasta en cada una de las etapas del protocolo, en particular cuanto tiempo se ocupa en etapas de procesamiento, y cuanto tiempo se ocupa en etapas de comunicación.
- *Overhead* resultante: al finalizar cada sesión, se midió el tamaño del mensaje a enviar,y se dividió por la cantidad de tiempo que duró la misma sesión, obteniendo así el *overhead* necesario que añade el protocolo para proporcionar anonimato en el envío de mensajes.

5.4. Resultados y Discusión

5.4.1. Tiempos Totales de Ejecución

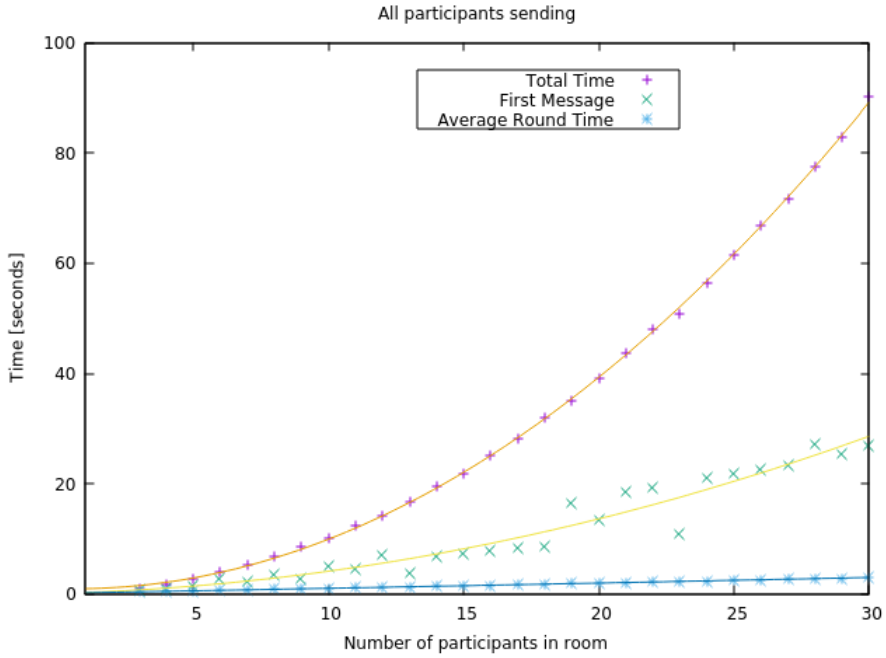


Figura 5.1: Tiempos de Ejecución en Tamaño de sala variable

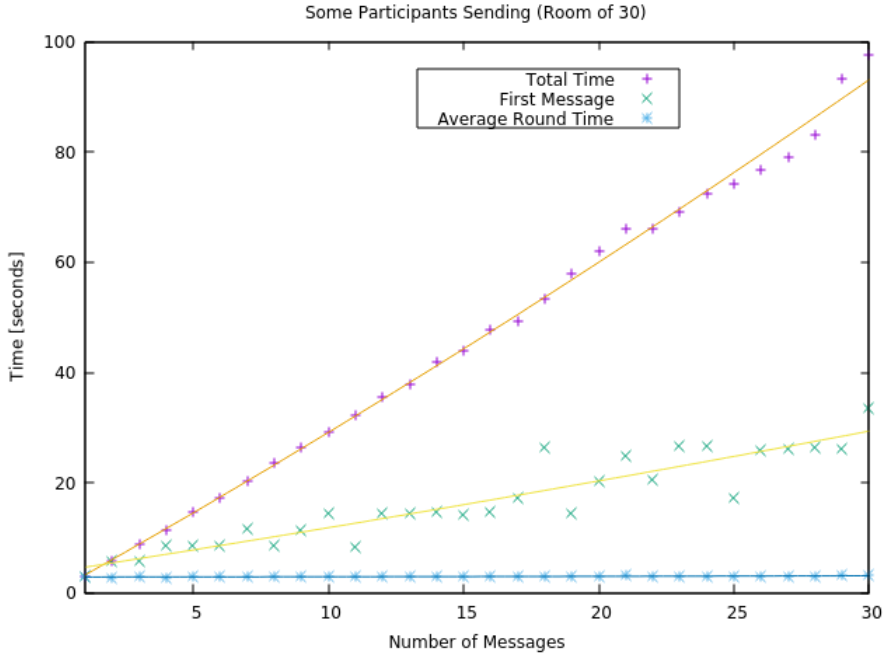


Figura 5.2: Tiempos de Ejecución en Tamaño de sala fijo

Los tiempos totales de ejecución descritos en las Figuras 5.1 y 5.2 muestran resultados alentadores con relación al tiempo necesario que hay que esperar para recibir mensajes de manera anónima. En el peor caso del experimento (30 mensajes colisionando en una sala de 30 participantes), el tiempo total resultó ser de 1 minuto y 30 segundos, tiempo razonable para leer 30 mensajes de 140 caracteres cada uno. Además, importante notar, que los primeros mensajes empiezan a llegar aproximadamente a los 30 segundos de ejecución (en el mismo caso anterior), por lo que hay una retro-alimentación temprana que el protocolo está funcionando y empezando a revelar los mensajes enviados.

Además de lo anterior, los tiempos totales muestran tendencias que eran esperadas. Mientras que en la Figura 5.1 los tiempos totales siguen una tendencia cuadrática, esperada por el hecho que al agregar un participante más a la sala, se elevan al cuadrado la cantidad de conexiones necesarias (grafo totalmente conectado); en cambio en la Figura 5.2 los tiempos totales siguen una tendencia lineal, mostrando que al fijar un tamaño de sala (en este caso, 30 participantes), el tiempo promedio por ronda se mantiene constante, independiente del número de rondas necesarias para resolver la colisión, resultando así en una tendencia lineal a medida que más mensajes colisionan en la primera ronda.

5.4.2. *Profiling* de las etapas

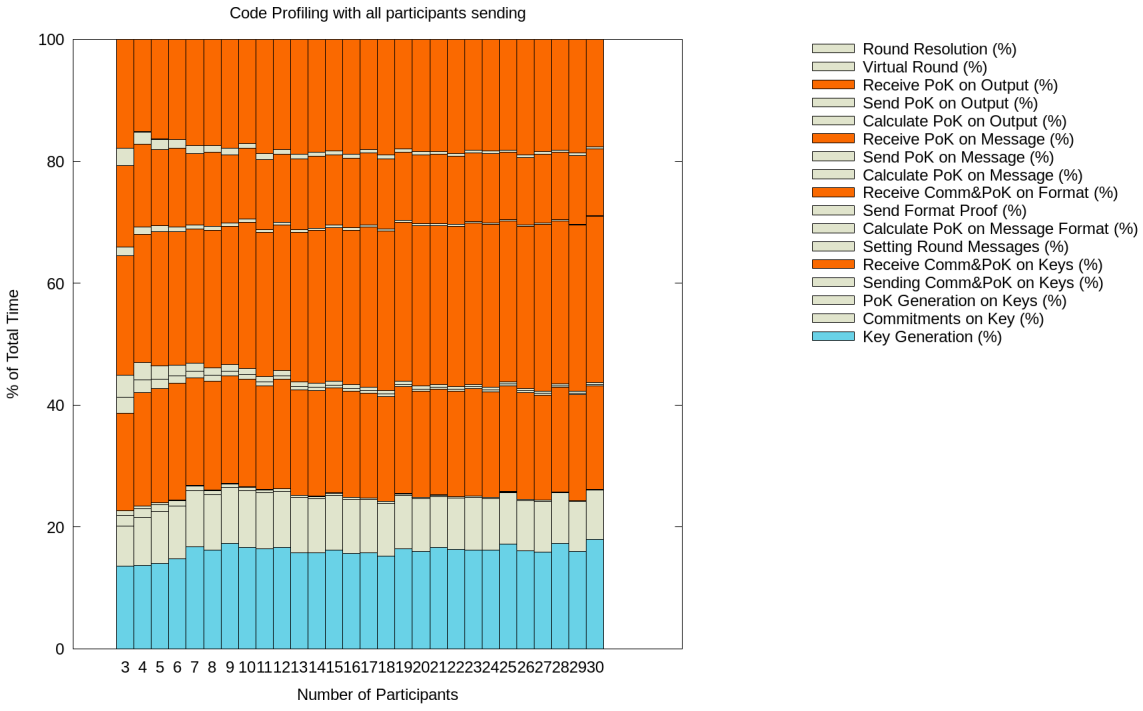


Figura 5.3: *Profiling* de etapas en Tamaño de sala variable

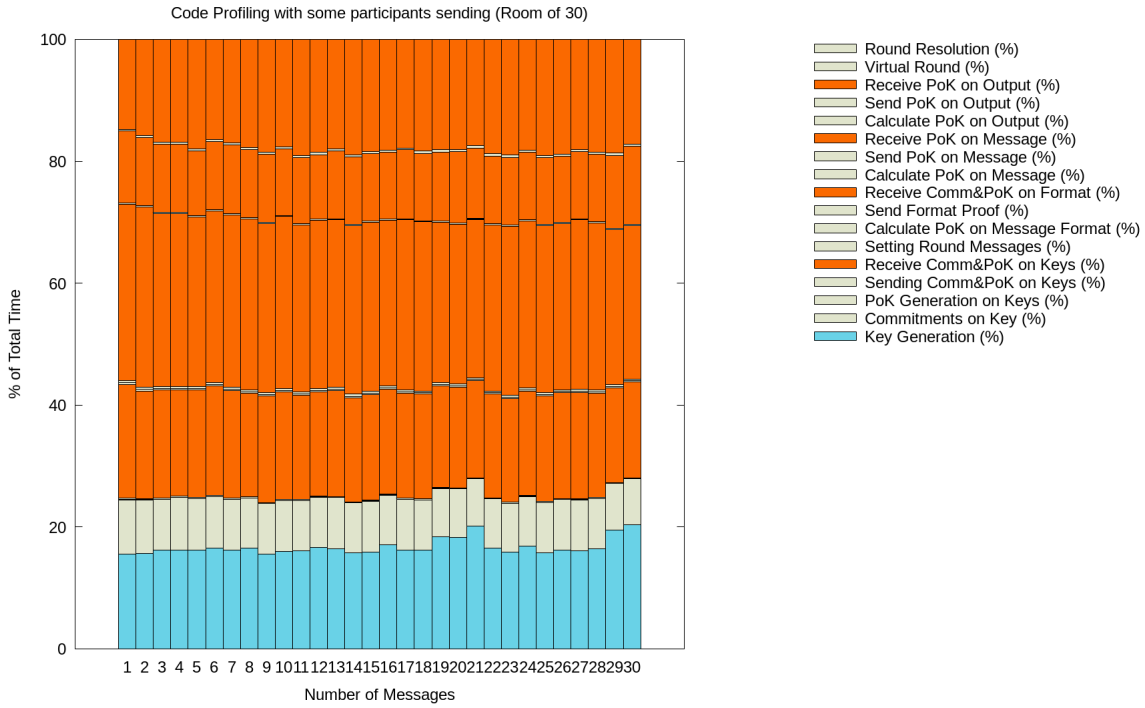


Figura 5.4: *Profiling* de etapas en Tamaño de sala fijo

El *profiling* realizado mostrado en las Figuras 5.3 y 5.4, tanto en el caso de tamaño de sala variable o fijo respectivamente, muestra una predominancia abrumadora de las etapas que implican un alto costo en la capa de comunicación (recepción de *commitments* y *zero-knowledge proofs* enviados por otros participantes). Independiente del tamaño de la sala o el número de mensajes involucrados en la colisión, el tiempo de procesamiento (generación de *commitments* y de *zero-knowledge proofs*, ejecutar una ronda virtual o procesar la resolución de la ronda), es significativamente menor que las etapas de comunicación, por lo que mejoras al protocolo criptográfico en términos de eficiencia (hacer *zero-knowledge proofs* menos complejas, por ejemplo) no significarían un incremento muy importante en el tiempo total de la ejecución del protocolo, por lo que el trabajo futuro debería enfocarse en mejorar el rendimiento de la capa de comunicación, manejando de manera más eficiente el envío y recepción de mensajes en la sala.

Es importante mencionar una propuesta sugerida en el *paper* original de la variante descrita en este trabajo [12], la cual hace referencia a la eliminación de la etapa de compartición de llaves, e intercambiarla por una generación pseudo aleatoria de la llave, compartiendo una semilla de generación entre cada par de participantes. Las pruebas muestran que esta etapa se lleva entre un 10 y un 20 por ciento del tiempo total, por lo que su eliminación mostraría una mejora importante en los tiempos totales de ejecución del protocolo.

5.4.3. *Overhead* del protocolo y Tamaño de los Mensajes

Una observación importante a realizar de la implementación desarrollada es la cantidad de información (medida simplemente como el largo de los mensajes) que se envía en el protocolo. Se debe tomar como base el largo del mensaje original m_i que cada participante debe comunicar. A esto

se le debe agregar todas las *zero-knowledge proofs* que necesita enviar, además de los *commitments*, como también los valores necesarios que debe comunicar para establecer llaves compartidas con todo el resto de la sala y considerar que en realidad lo que se envía en el protocolo es el mensaje M_i. A la suma de todos esos valores enviados le denotaremos S_i. Por último hay que considerar también que entre más participantes colisionen sus mensajes, más veces será necesario reenviar esta cantidad de información, debido a que se debe desarrollar un mayor número de rondas reales.

Para medir este sobrecosto que agrega el protocolo, se realizaron varios experimentos, donde se varió el tamaño del mensaje original que desea comunicar cada participante y se observó cuanta información se enviaba finalmente (por ronda) al resto de la sala. Es importante notar que la cantidad de participantes presentes en la sala hace cambiar la cantidad de información a enviar, éste es un valor fijo (le envío a todos los participantes, la misma información), por lo que los resultados que se muestran en la Tabla 5.1, son por cada participante presente en la sala:

m_i (bytes)	S_i (bytes)	S_i/m_i
5	1633	326.6
10	1933	193.3
20	2624	131.2
40	3987	99.7
60	5288	88.1
80	6276	78.5
100	7459	74.6
120	8848	73.7
140	10281	73.4
200	14008	70.1
300	20200	67.3

Tabla 5.1: Tamaño de mensajes enviados (por ronda)

Cabe destacar, que entre más grande es el mensaje original a enviar, menor es el *overhead* relativo que añade la ejecución del protocolo para asegurar anonimato y seguridad del sistema. Importante mencionar además que el tamaño del mensaje es determinado por el nodo `Directorio`, en el comienzo del protocolo. El nodo directorio determina un largo máximo permitido para un mensaje a enviar durante la sesión actual. Independiente si algún participante envía un mensaje de largo menor a dicho máximo, el *overhead* que presentará es el mismo que alguien que envía un mensaje con el largo máximo establecido por el nodo directorio.

Por último mencionar que alcanzar el nivel de anonimato que ofrece el protocolo significa un sobrecosto importante que cada participante debe pagar, reflejado especialmente en la cantidad de información extra que debe enviarse con respecto al mensaje final que desea comunicar. Estos tamaños se ven reflejados en la Tabla 5.1, lo que muestra un altísimo sobrecosto relativo que experimenta cada participante (alrededor de 70 veces por ronda y por participante presente en la sala). Eso quiere decir que si un participante desea comunicar un mensaje de 140 caracteres en una sala de 30 participantes (y tiene la "mala suerte" que todo el resto de la sala también quiere comunicar un mensaje), tendrá que enviar un total de 8.8 MB de información aproximadamente. Junto con esto, como es descrito en la Figura 5.5, en el peor caso (es decir, cuando su mensaje es el último en ser revelado), experimentará un ancho de banda real de 1.5 bytes/sec (para comunicar su mensaje de 140 bytes, tuvo que esperar 90 segundos). Esto presenta un desafío muy importante

a analizar por parte del diseño del protocolo criptográfico: ¿es posible alcanzar el mismo nivel de anonimato, utilizando menos cantidad de mensajes? ¿pudieran acortarse las rondas o utilizar *zero-knowledge proofs* más "cortas"? También es un desafío el mejorar la implementación y tratar de disminuir la cantidad de información enviada a través de la red, por ejemplo a través de mecanismos de compresión de información, pero procurando que los tiempos necesarios para comprimir y descomprimir la información sean tales que se evidencie una mejora sustancial en los tiempos totales de la ejecución del protocolo.

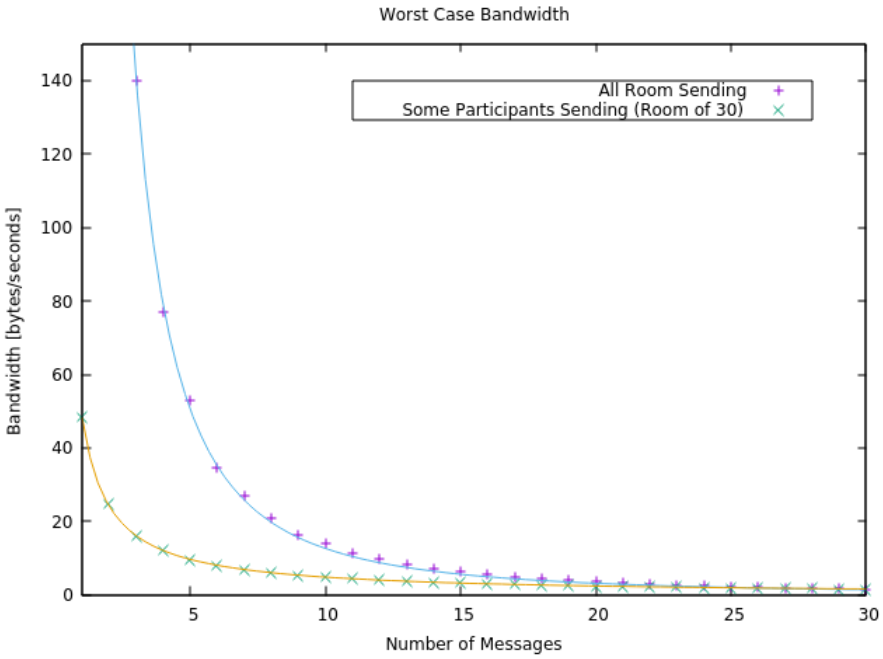


Figura 5.5: *Bandwidth* del peor caso (último mensaje transmitido)

5.4.4. Escenario propuesto de uso

Con los resultados discutidos anteriormente, se propone un escenario que logre proporcionar anonimato a sus participantes, y que esto no suponga pagar un alto costo por un rendimiento pobre de la aplicación desarrollada.

Como el protocolo es altamente sensible a (1) el tamaño de los mensajes, y (2) la cantidad de mensajes involucrados en una colisión, se propone utilizar la aplicación implementada en un contexto de *microblogging*, con una creación de sesiones de manera periódica, otorgando así la posibilidad de mantener un tamaño del *anonymity set* considerable, reducir la posibilidad de colisiones masivas de mensajes, y proveer así un envío de mensajes más eficiente.

Una posible configuración del escenario anterior es el siguiente: creación de una sala de 30 participantes, los cuales participan cada 30 minutos en la ejecución de una sesión, cada uno contribuyendo para el normal desarrollo del protocolo, mientras que una fracción de ellos va a tener la necesidad de enviar un mensaje (de un máximo de 140 caracteres, *á la* Twitter) en la sesión actual. Al tener una repetición periódica de la ejecución de sesiones, la posibilidad de una colisión masiva de mensajes en cada una de ellas es menor, logrando así que en cada una de ellas se pueda

resolver de manera rápida la colisión de los mensajes resultantes. Efectivamente cada participante que necesite enviar un mensaje, puede observar una baja latencia y podrá comunicarse de manera rápida y expedita. La ejecución de sesiones debe hacerse de manera constante, teniendo en cuenta que en muchas de ellas no se enviará ningún mensaje, resultando así en una ejecución rápida y de bajo costo para todos los participantes. Además de esta manera se pueden agregar o expulsar a participantes entre sesiones, para así, ya sea aumentar el anonimato, o bien castigar a participantes que se comportaron de manera maliciosa en sesiones anteriores (la ejecución de las sesiones es totalmente independiente unas de otras).

El escenario anterior podría instanciarse en varios contextos donde se necesita la opción de comunicar mensajes de manera anónima:

- Denuncias de abusos laborales dentro de una corporación privada.
- Foro de denunciantes (*whistleblowers*) de actividades realizadas por alguna institución pública.
- Foro de conversación entre distintos centros periodísticos (al poseer mayor poder de cómputo, se puede permitir el envío de mensajes de mayor longitud).
- Votaciones secretas periódicas entre un grupo de interés (en vez de enviar un mensaje, simplemente se envía un bit, diferenciando entre SI y NO).

5.4.5. Consideraciones sobre escenarios no experimentados

Debido a la falta de posibilidad para usar el software de simulación, existe un sinnúmero de escenarios que no se pudieron experimentar y que sería interesante de observar como se comporta el protocolo en esos casos. En particular, experimentar el comportamiento del protocolo con nodos conectados a través de Internet (y no a través de una red local, como sí fue experimentado). Esto entregaría resultados más cercanos a la realidad y daría la posibilidad de establecer un canal anónimo para usuarios que no necesariamente comparten una cercanía física u organizativa, sino más bien que comparten intereses o simplemente que quieran aportar en fortalecer el anonimato resultante.

Otro escenario interesante de experimentar para un trabajo futuro sería la intermitencia en la conectividad de un subconjunto de los nodos participantes. Esto es que los nodos puedan conectarse/desconectarse en medio de la sesión que se esté ejecutando en dicho momento, y con ello observar tanto el comportamiento que deba tener el protocolo, como la influencia que esto pueda tener en los tiempos finales de ejecución.

Capítulo 6

Trabajo Futuro

Entre más se avanzaba en el trabajo desarrollado, mayor era la posibilidad de transformar una investigación académica-científica en una herramienta disponible para múltiples usuarios en el mundo que deseen transmitir información de manera anónima y simple, solamente utilizando su dispositivo móvil.

Para lograr dicho objetivo aún falta trabajo que realizar, sobre todo en la capa de comunicación entre los distintos nodos partícipes del protocolo. Las tareas que se realizarán a futuro son las siguientes:

1. Autenticar canales: la seguridad del protocolo criptográfico se basa fuertemente en que los canales de comunicación que existan entre los distintos nodos participantes sean autenticados, es decir, que cada usuario debe enviar sus mensajes firmados, para así el receptor de cada mensaje se asegure que el emisor es quien dice ser. Con esto se evitan participantes impostores, es decir, que se hagan pasar por otro participante, lo que podría resultar en bajar la reputación de algún usuario, eliminándolo de la sala.
2. Manejar desconexión de usuarios: cualquier aplicación que pretenda ser utilizada en ambientes "reales" debe manejar la posible desconexión de usuarios en cualquier momento del protocolo. Actualmente esto no es manejado y simplemente la aplicación deja de funcionar. Es importante además, que cualquier medida que se adopte, se verifique que no influye en el anonimato y seguridad de los participantes. Por ejemplo se puede adoptar la medida de "simular" al participante desconectado suponiendo que envía mensajes vacíos, pero hay que establecer si esta medida no afecta tanto la integridad del protocolo (los mensajes que aun no se reciben se van a enviar satisfactoriamente) como el anonimato y seguridad de los participantes que quedan involucrados, o incluso el mismo participante que sufrió la desconexión (se podría saber que el participante que se desconectó envió o no envió alguno de los mensajes publicados anteriormente). Este es un problema abierto.
3. Manejar participantes maliciosos: actualmente el protocolo y la implementación es capaz de encontrar a un participante malicioso, pero más allá de identificarlo no emplea ninguna medida en contra de éste. Podría seguir el protocolo suponiendo la desconexión del participante malicioso, lo que sería análogo al punto anterior, pero tal vez existan medidas más drásticas como suspensión por un cierto tiempo de participar en otras sesiones del protocolo, o derechamente la expulsión del participante para siempre.

4. Optimizar recepción de mensajes: una parte importante a analizar por gente más experta en el área de Redes es la manera en que se están recibiendo los mensajes por parte de los participantes. Actualmente se delegó toda responsabilidad a *ZeroMQ*, el cual emplea una cola para no perder los mensajes entrantes y la implementación actual se queda esperando entre mensajes, sin realizar ninguna operación. Tal vez sea necesario revisar ese protocolo y optimizar la recepción de mensajes, abriendo múltiples *sockets*, o utilizando el tiempo de espera entre mensajes para poder realizar alguna operación criptográfica pendiente, y así no tener tiempo ocioso.
5. Refinar protocolo criptográfico: a medida que se avanzaba en el diseño del protocolo, se iban encontrando más detalles criptográficos necesarios de añadir para asegurar la seguridad e integridad del sistema. Algunos de estas necesarias mejoras alcanzaron a ser implementadas, pero otras aun no (por ejemplo, demostrar que el mensaje enviado en un ronda para resolver colisiones es menor al mensaje promedio de la ronda padre).
6. Mejorar diseño de aplicación móvil: la aplicación móvil implementada fue creada solo como prototipo, para pensar en dejar un producto disponible para la comunidad es muy importante considerar la usabilidad que posea dicha aplicación, requisito que se logra proponiendo un diseño moderno que se ponga a prueba realizando pruebas estándar de usabilidad de aplicaciones (seguir líneas de diseño modernas, pruebas con usuarios reales, etc.).
7. Utilizar aplicación a través de Internet: la implementación actual solo permite establecer comunicación entre nodos conectados en una misma red local. Para masificar su uso es necesario que pueda soportar comunicación nodo-a-nodo a través de Internet. Una posible solución a esto sería replicar el modelo utilizado por el protocolo *Bit Torrent*[1], el cual posee una parte esencial que es la conexión punto-a-punto entre sus nodos (llamados *peers*) para realizar la descarga de algún archivo en particular.
8. Pruebas de seguridad: cualquier aplicación que tenga como objetivo ser utilizada por múltiples usuarios alrededor del mundo, y que además maneje información sensible, es necesario que pase una serie de pruebas de seguridad realizadas por terceras personas a los diseñadores y creadores de la aplicación misma. Con esto se puede tener la certeza que la aplicación ha sido revisada en contra de posibles *leaks* de seguridad, que puedan comprometer la seguridad de sus usuarios, producto de errores en la implementación (y no necesariamente errores en el protocolo criptográfico).

[1] `http://www.bittorrent.org/beps/bep_0003.html`

Capítulo 7

Conclusiones

Las actuales alternativas que existen para proveer anonimato en comunicación entre personas o mientras se navega en Internet están altamente cuestionadas debido a nuevos ataques que vulneran la seguridad de sus usuarios. Es por ello que es necesario explorar nuevas alternativas y proponer nuevos protocolos que no posean las vulnerabilidades de las actuales soluciones y puedan ser usadas por distintas personas, cualesquiera sea su necesidad de anonimato.

Es en esa dirección que el presente trabajo realiza un aporte importante, avanzando en la investigación de crear protocolos basados en el concepto de *DC-Nets*, (un tipo de protocolo que no posee las vulnerabilidades antes mencionadas) que, a la vez, sean eficientes y prácticos para uso masivo. La manera más específica para asegurar la eficiencia del protocolo es probar su desempeño mediante una implementación concreta que se ajuste a todos los detalles criptográficos que el protocolo implica.

Finalmente la implementación realizada muestra varias conclusiones que eran difícil de visualizar solamente con la descripción criptográfica del protocolo:

1. Escenarios ideales: las pruebas realizadas muestran que el protocolo posee una alta dependencia (en términos de tiempo total del protocolo) a dos factores: (1) tamaño de los mensajes, y (2) número de participantes. Por lo anterior, el escenario ideal de uso del protocolo son grupos no muy grandes de participantes, enviando mensajes no muy largos tampoco. Dicho escenario se ajusta muy bien a un protocolo de *microblogging* anónimo (un símil a *Twitter* anónimo).
2. Usabilidad del protocolo: el protocolo propuesto implica muy poca participación del usuario que desea enviar un mensaje anónimo, o que simplemente desea ayudar en proveer mayor anonimato para el resto de los participantes. Esto resulta en que el protocolo puede volverse transparente para el usuario, y simplemente enviar mensajes tal como lo hace con otras aplicaciones más populares como *WhatsApp*[1], *Telegram*[2] o *Signal*[3]. Además de esto, el protocolo no necesita mayor infraestructura adicional (servidores externos o nodos intermedios), sino simplemente un nodo `Directorio` (que puede ser iniciado por uno de los mismos participantes), que igualmente posee una aplicación móvil fácil de utilizar.

[1] `https://www.whatsapp.com/`
[2] `https://telegram.org/`
[3] `https://signal.org//`

3. Requisitos de confianza: la gran diferencia del protocolo propuesto con otras propuestas realizadas en el último tiempo, es la falta del requerimiento de confianza en un servidor externo al protocolo. Para poder escalar de manera importante con el número de participantes, la gran mayoría de las nuevas propuestas han necesitado que los participantes depositen confianza en un servidor externo ajeno a los propios participantes. Si bien el protocolo descrito en este trabajo no posee la escalabilidad de otras propuestas, el hecho de no obligar a los participantes a confiar en un servidor externo, eleva su estándar de seguridad y privacidad, volviendo a la idea original del protocolo *DC-Net*, de compartir un mensaje anónimo, incluso en un contexto de desconfianza hacia el resto de los participantes.
4. Importancia de capa de comunicación: la implementación de un protocolo criptográfico implica un conocimiento, no solo de la matemática y de los conceptos criptográficos del protocolo en sí, sino también de la capa de comunicación entre los distintos participantes, combinando conceptos e implementación de sistemas distribuidos, junto con intercambio de mensajes en redes de manera general. Muchas de las mejoras a la eficiencia de protocolos criptográficos viene de la mano de las mejoras en la capa de comunicación (manejo de colas de mensajes, manejo de desconexiones de usuarios, sincronización entre los distintos participantes, etc).
5. Construcción de aplicación final: un paso muy importante en el estudio de protocolos criptográficos es poder crear aplicaciones finales, permitiendo a los usuarios aprovechar los últimos avances en el estudio de la criptografía. De nada sirven los avances en la teoría, sino se transforman en aplicaciones finales para los usuarios, los cuales pueden mejorar su seguridad y privacidad valiéndose de esos últimos avances.

Capítulo 8

Bibliografía

[1] Jan Camenisch and Markus Stadler. Proof systems for general statements about discrete logarithms. Technical report, Citeseer, 1997.

[2] David Chaum. Security without identification: Transaction systems to make big brother obsolete. *Commun. ACM*, 28(10):1030–1044, 1985.

[3] David Chaum. The dining cryptographers problem: Unconditional sender and recipient untraceability. *Journal of cryptology*, 1(1):65–75, 1988.

[4] David L Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM*, 24(2):84–90, 1981.

[5] Benny Chor, Oded Goldreich, Eyal Kushilevitz, and Madhu Sudan. Private information retrieval. In *Foundations of Computer Science, 1995. Proceedings., 36th Annual Symposium on*, pages 41–50. IEEE, 1995.

[6] Henry Corrigan-Gibbs, Dan Boneh, and David Mazières. Riposte: An anonymous messaging system handling millions of users. In *Security and Privacy (SP), 2015 IEEE Symposium on*, pages 321–338. IEEE, 2015.

[7] Henry Corrigan-Gibbs and Bryan Ford. Dissent: accountable anonymous group messaging. In *Proceedings of the 17th ACM conference on Computer and communications security*, pages 340–350. ACM, 2010.

[8] Henry Corrigan-Gibbs, David Isaac Wolinsky, and Bryan Ford. Proactively accountable anonymous messaging in verdict. *arXiv preprint arXiv:1209.4819*, 2012.

[9] Ronald Cramer, Ivan Damgård, and Berry Schoenmakers. Proofs of partial knowledge and simplified design of witness hiding protocols. In *Advances in Cryptology—CRYPTO'94*, pages 174–187. Springer, 1994.

[10] Alan Demers, Dan Greene, Carl Hauser, Wes Irish, John Larson, Scott Shenker, Howard Sturgis, Dan Swinehart, and Doug Terry. Epidemic algorithms for replicated database maintenance. In *Proceedings of the Sixth Annual ACM Symposium on Principles of Distributed Computing*, PODC '87, pages 1–12, New York, NY, USA, 1987. ACM.

[11] Whitfield Diffie and Martin Hellman. New directions in cryptography. *IEEE transactions on Information Theory*, 22(6):644–654, 1976.

[12] Christian Franck and Jeroen van de Graaf. Dining cryptographers are practical. *arXiv preprint arXiv:1402.2269*, 2014.

[13] Sharad Goel, Mark Robson, Milo Polte, and Emin Sirer. Herbivore: A scalable and efficient protocol for anonymous communication. Technical report, Cornell University, 2003.

[14] Albert Kwon, David Lazar, Srinivas Devadas, and Bryan Ford. Riffle. *Proceedings on Privacy Enhancing Technologies*, 2016(2):115–134, 2016.

[15] Torben Pryds Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *Annual International Cryptology Conference*, pages 129–140. Springer, 1991.

[16] Michael G Reed, Paul F Syverson, and David M Goldschlag. Anonymous connections and onion routing. *Selected Areas in Communications, IEEE Journal on*, 16(4):482–494, 1998.

[17] Claus-Peter Schnorr. Efficient identification and signatures for smart cards. In *Conference on the Theory and Application of Cryptology*, pages 239–252. Springer, 1989.

[18] Paul Syverson, Gene Tsudik, Michael Reed, and Carl Landwehr. Towards an analysis of onion routing security. In *Designing Privacy Enhancing Technologies*, pages 96–114. Springer, 2001.

[19] Jelle Van Den Hooff, David Lazar, Matei Zaharia, and Nickolai Zeldovich. Vuvuzela: Scalable private messaging resistant to traffic analysis. In *Proceedings of the 25th Symposium on Operating Systems Principles*, pages 137–152. ACM, 2015.

[20] David I Wolinsky, Henry Corrigan-Gibbs, Bryan Ford, and Aaron Johnson. Scalable anonymous group communication in the anytrust model. Technical report, DTIC Document, 2012.

[21] David Isaac Wolinsky, Henry Corrigan-Gibbs, Bryan Ford, and Aaron Johnson. Dissent in numbers: Making strong anonymity scale. In *OSDI*, pages 179–182, 2012.

[22] Matthew Wright, Micah Adler, Brian Neil Levine, and Clay Shields. An analysis of the degradation of anonymous protocols. In *NDSS*, volume 2, pages 39–50, 2002.

[23] Yingqun Yu and Georgios B Giannakis. Sicta: a 0.693 contention tree algorithm using successive interference cancellation. In *INFOCOM 2005. 24th Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings IEEE*, volume 3, pages 1908–1916. IEEE, 2005.

Capítulo 9

Apéndice

9.1. Apéndice A: Descripción de *Zero-Knowledge Proofs*

9.1.1. Consideraciones Generales

Las *Zero-Knowledge Proofs* necesitadas para el desarrollo de este trabajo son del tipo *proof-of-knowledge*, las cuales permiten a un cierto *prover* demostrarle a un *verifier* que conoce un cierto valor (o varios valores) que hacen verdad una cierta proposición, sin la necesidad de revelar dicho valor. Por ejemplo, a través de esta técnica criptográfica una persona podría demostrar que conoce los todos los distintos valores que solucionan un cierto tablero de *sudoku*, sin la necesidad de revelar estos valores.

Generalmente estas demostraciones son *interactivas*, es decir, *prover* y *verifier* se comparten distintos valores durante el desarrollo del protocolo, lo cual finalmente convence al *verifier* de la propiedad que se quería demostrar. Eso si, toda demostración interactiva puede transformarse a *no interactiva*, es decir, sin la necesidad de intercambiar valores, el *prover* puede demostrar el conocimiento de cierto valor solamente con el cálculo de algunos valores, suponiendo que existen otros valores que son públicamente conocidos (en particular, conocidos por el *verifier*). Las demostraciones utilizadas en este trabajo y detalladas a continuación son no interactivas.

9.1.2. Logaritmo Discreto

$$PoK\{w : y = g^w\}$$

Sean g, y valores públicos. El *prover* (de identidad pública `id`) debe demostrar que conoce el valor w que hace verdad la siguiente proposición: $y = g^w$. Los pasos que debe seguir son los siguientes:

1. Escoger r aleatorio.
2. Calcular $z = g^r$

3. Calcular $b = H(z \mid\mid g \mid\mid y \mid\mid \texttt{id})$, donde $H(\cdot)$ es una función de hash segura (por ejemplo, `SHA3`).
4. Calcular $a = r + bw$
5. Enviar como demostración la tupla $\mathcal{D} = (g, z, a)$

Finalmente, el *verifier* debe recibir la demostración $\mathcal{D}$ y verificarla siguiendo estos pasos:

1. Calcular $\hat{b} = H(z \mid\mid g \mid\mid y \mid\mid \texttt{id})$ utilizando la misma función de hash usada por el *prover*.
2. Verificar que $g^a = y^{\hat{b}} z$

Este protocolo también es conocido como *Firma de Schnorr* y para más detalles sobre su seguridad y correctitud revisar [17].

9.1.3. Uno de dos logaritmos discretos

$$PoK\{x_1, x_2 : h_1 = g^{x_1} \vee h_2 = g^{x_2}\}$$

Sean g, h_1, h_2 valores públicos. El *prover* (de identidad pública `id`) debe demostrar que conoce al menos uno de los valores (x_1, x_2) que hacen verdad la siguiente proposición: $h_1 = g^{x_1} \vee h_2 = g^{x_2}$. Los pasos que debe seguir son los siguientes (sin pérdida de generalidad, supondremos que conoce el valor x_1):

1. Escoger c, r_1, r_2 aleatorios.
2. Calcular $z_1 = g^{r_1}$
3. Calcular $z_2 = g^{r_2} h_2^{-c}$
4. Calcular $b = H(z_1 \mid\mid z_2 \mid\mid g \mid\mid h_1 \mid\mid h_2 \mid\mid \texttt{id})$, donde $H(\cdot)$ es una función de hash segura (por ejemplo, `SHA3`).
5. Calcular $t = b - c$
6. Calcular $a = r_1 + t x_1$
7. Enviar como demostración la tupla $\mathcal{D} = (t, c, z_1, z_2, a, r_2)$.

Finalmente, el *verifier* debe recibir la demostración $\mathcal{D}$ y verificarla siguiendo estos pasos:

1. Calcular $\hat{b} = H(z_1 \mid\mid z_2 \mid\mid g \mid\mid h_1 \mid\mid h_2 \mid\mid \texttt{id})$ utilizando la misma función de hash usada por el *prover*.
2. Verificar que $\hat{b} = t + c$
3. Verificar que $g^a = h_1^t z_1$
4. Verificar que $g^{r_2} = h_2^c z_2$

El protocolo descrito anteriormente es un caso específico del esquema propuesto en [9], en el cual se describe la manera de demostrar el conocimiento de un subconjunto de valores dentro de un cierto *statement* (en el caso necesario para este trabajo, se demostró el conocimiento de sólo un valor de dos posibles).

9.1.4. Dos logaritmos discretos

$$PoK\{x_1, x_2 : h_1 = g^{x_1} \wedge h_2 = g^{x_2}\}$$

Sean g, h_1, h_2 valores públicos. El *prover* (de identidad pública `id`) debe demostrar que conoce ambos valores (x_1, x_2) que hacen verdad la siguiente proposición: $h_1 = g^{x_1} \wedge h_2 = g^{x_2}$. Los pasos que debe seguir son los siguientes:

1. Escoger r_1, r_2 aleatorios.
2. Calcular $z_1 = g^{r_1}$
3. Calcular $z_2 = g^{r_2}$
4. Calcular $b = H(z_1 \mid\mid z_2 \mid\mid g \mid\mid h_1 \mid\mid h_2 \mid\mid id)$, donde $H(\cdot)$ es una función de hash segura (por ejemplo, `SHA3`).
5. Calcular $a_1 = r_1 + bx_1$
6. Calcular $a_2 = r_2 + bx_2$
7. Enviar como demostración la tupla $\mathcal{D} = (z_1, z_2, a_1, a_2)$.

Finalmente, el *verifier* debe recibir la demostración $\mathcal{D}$ y verificarla siguiendo estos pasos:

1. Calcular $\hat{b} = H(z_1 \mid\mid z_2, \mid\mid g \mid\mid h_1 \mid\mid h_2 \mid\mid \texttt{id})$ utilizando la misma función de hash usada por el *prover*.
2. Verificar que $g^{a_1} = z_1 h_1^{\hat{b}}$
3. Verificar que $g^{a_2} = z_2 h_2^{\hat{b}}$

El protocolo anterior resulta de realizar en paralelo dos demostraciones como la descrita en [17] (conocimiento de logaritmo discreto).

9.1.5. Valores en *Pedersen Commitments*

$$PoK\{x, r : c = g^x h^r\}$$

Sean c, g, h valores públicos. El *prover* (de identidad pública `id`) debe demostrar que conoce ambos valores (x, r) que hacen verdad la siguiente proposición: $c = g^x h^r$. Los pasos que debe seguir son los siguientes:

1. Escoger y, s aleatorios.
2. Calcular $d = g^y h^s$
3. Calcular $e = H(d \mid\mid g \mid\mid h \mid\mid c \mid\mid \texttt{id})$, donde $H(\cdot)$ es una función de hash segura (por ejemplo, `SHA3`).
4. Calcular $u = ex + y$
5. Calcular $v = er + s$
6. Enviar como demostración la tupla $\mathcal{D} = (d, u, v)$.

Finalmente, el *verifier* debe recibir la demostración $\mathcal{D}$ y verificarla siguiendo estos pasos:

1. Calcular $\hat{e} = H(d \mid\mid g \mid\mid h \mid\mid c \mid\mid id)$ utilizando la misma función de hash usada por el *prover*.
2. Verificar que $g^u h^v = c^{\hat{e}d}$

9.2. Apéndice B: Código Fuente

En la presente sección se muestra parte del código fuente implementado para este trabajo, destacando algunas secciones importantes de lo realizado para el desarrollo del sistema descrito anteriormente. Como fue dicho anteriormente, el código está programado en *Java*.

9.2.1. *Zero-Knowledge Proof* de Valores en *Pedersen Commitments*

```
/**
 * Generates Proof of Knowledge that participant knows (x, r) in c = g^x h^r (mód p)
 *
 * @param c commitment s.t. c = g^x h^r (mód p)
 * @param g generator of group G_q
 * @param x value in Z_q
 * @param h generator of group G_q
 * @param r value in Z_q
 * @param q large prime
 * @param p large prime s.t. p = kq + 1
 * @return ProofOfKnowledge s.t. node knows (x, r) in c = g^x h^r (mód p)
 * @throws NoSuchAlgorithmException
 * @throws UnsupportedEncodingException
 */
public ProofOfKnowledgePedersen generateProofOfKnowledgePedersen(BigInteger c,
↪  BigInteger g, BigInteger x, BigInteger h, BigInteger r, BigInteger q,
↪  BigInteger p) throws NoSuchAlgorithmException, UnsupportedEncodingException {
    PedersenCommitment pedersenCommitment = new PedersenCommitment(g, h, q, p);

    // y random value in Z_q
    BigInteger y = pedersenCommitment.generateRandom();
    // s random value in Z_q
    BigInteger s = pedersenCommitment.generateRandom();

    // d = g^y h^s (mód p)
    BigInteger d = pedersenCommitment.calculateCommitment(y, s);

    MessageDigest md = MessageDigest.getInstance("SHA-512");
    String publicValueOnHash = d.toString().concat(
            g.toString()).concat(
            h.toString()).concat(
            c.toString()).concat(
            "" + this.nodeIndex);
    md.update(publicValueOnHash.getBytes("UTF-8"));
    byte[] hashOnPublicValues = md.digest();
    // e = H(d||g||h||c||nodeIndex) (mód q)
    BigInteger e = new BigInteger(hashOnPublicValues).mod(q);
```

```
    // u = ex + y
    BigInteger u = e.multiply(x).add(y);
    // v = er + s
    BigInteger v = e.multiply(r).add(s);

    return new ProofOfKnowledgePedersen(d, u, v, nodeIndex);
}
```

9.2.2. Establecimiento de Llaves Compartidas

```
/* KEY SHARING PART */
// Initialize KeyGeneration
KeyGeneration keyGeneration = new DiffieHellman(room.getRoomSize() - 1,
 ↪  room.getG(), room.getP(), nodeIndex, repliers, requestors, room);

// Generate Participant Node values
keyGeneration.generateParticipantNodeValues();

// Get other participants values (to produce cancellation keys)
keyGeneration.getOtherParticipantNodesValues();

// Generation of the main key round value (operation over the shared key values)
BigInteger keyRoundValue = keyGeneration.getParticipantNodeRoundKeyValue();
```

.....

```
/**
* @return other participant nodes "halves" of the shared key g^b
*/
@Override
public BigInteger[] getOtherParticipantNodesValues() {
    int i = 0;
    BigInteger[] otherNodesKeyHalves = new BigInteger[room.getRoomSize() - 1];
    // The "first" node doesn't have any replier sockets
    if (nodeIndex != 1)
        for (ZMQ.Socket replier : repliers) {
            // The replier wait to receive a key share
            otherNodesKeyHalves[i] = new BigInteger(replier.recvStr());
            // When the replier receives the message, replies with one of their
             ↪  key shares
            replier.send(participantNodeHalves[i].toString());
            i++;
        }
    // The "last" node doesn't have any requestor sockets
    if (nodeIndex != room.getRoomSize())
```

```
        for (ZMQ.Socket requestor : requestors) {
            // The requestor sends a key share
            requestor.send(participantNodeHalves[i].toString());
            // The requestor waits to receive a reply with one of the key shares
            otherNodesKeyHalves[i] = new BigInteger(requestor.recvStr());
            i++;
        }
    this.otherParticipantNodeHalves = otherNodesKeyHalves;

    i = 0;
    this.otherParticipantNodeSharedRandomValueHalves = new
     ↪  BigInteger[room.getRoomSize() - 1];
    // The "first" node doesn't have any replier sockets
    if (nodeIndex != 1)
        for (ZMQ.Socket replier : repliers) {
            // The replier wait to receive a key share
            this.otherParticipantNodeSharedRandomValueHalves[i] = new
             ↪  BigInteger(replier.recvStr());
            // When the replier receives the message, replies with one of their
             ↪  key shares
            replier.send(participantNodeSharedRandomValueHalves[i].toString());
            i++;
        }
    // The "last" node doesn't have any requestor sockets
    if (nodeIndex != room.getRoomSize())
        for (ZMQ.Socket requestor : requestors) {
            // The requestor sends a key share
            requestor.send(participantNodeSharedRandomValueHalves[i].toString());
            // The requestor waits to receive a reply with one of the key shares
            this.otherParticipantNodeSharedRandomValueHalves[i] = new
             ↪  BigInteger(requestor.recvStr());
            i++;
        }

    return otherNodesKeyHalves;
}
```

9.2.3. Recepción y Verificación de *ZKP* on Messages

```
/* RECEIVE COMMITMENTS AND POKs ON MESSAGES */
for (int i = 0; i < room.getRoomSize(); i++) {
    // Wait response from Receiver thread as a String (json)
    String receivedProofOfKnowledgeOnMessageJson = receiverThread.recvStr();

    // Transform String (json) to object ProofOfKnowledgePedersen
```

```
    ProofOfKnowledgePedersen receivedProofOfKnowledgeOnMessage = new
     ↪  Gson().fromJson(
            receivedProofOfKnowledgeOnMessageJson,
             ↪  ProofOfKnowledgePedersen.class);
    int receivedNodeIndex = receivedProofOfKnowledgeOnMessage.getNodeIndex();

    // Verify proof of knowledge
    if (!zkp.verifyProofOfKnowledgePedersen(receivedProofOfKnowledgeOnMessage,
            receivedCommitmentsOnMessageCurrentRound[receivedNodeIndex - 1],
            room.getG(), room.getH(), room.getQ(), room.getP()))
        System.err.println("WRONG PoK on Message. Round: " + currentRound + ",
         ↪  Node: " +
                receivedProofOfKnowledgeOnMessage.getNodeIndex());
}
```

9.2.4. Conexiones al Nodo `Directorio`

```
// Create the PUB socket and bind it to the port 5555
ZMQ.Socket publisher = context.createSocket(ZMQ.PUB);
publisher.bind("tcp://*:6555");

// Create the PULL socket and bind it to the port 5554
ZMQ.Socket pull = context.createSocket(ZMQ.PULL);
pull.bind("tcp://*:6554");

ZMQ.Socket[] participantsConnectedPush = new ZMQ.Socket[roomSize];
// Wait to receive <numberOfNodes> connections from each node that wants to send
 ↪  a message in this room
for (int i = 0; i < roomSize; i++) {
    // Receive a message from the PULL socket, which corresponds to the IP
     ↪  address of this node
    String messageReceived = pull.recvStr();
    // Assign an index to this node and store it in the nodesInTheRoom with his
     ↪  correspondent IP address
    this.infoFromDirectory.nodes[i] = new ParticipantNodeInfoFromDirectory(i+1,
     ↪  messageReceived);
    // Send ACK to participant that just connected
    participantsConnectedPush[i] = context.createSocket(ZMQ.PUSH);
    participantsConnectedPush[i].connect("tcp://" + messageReceived + ":5554");
    participantsConnectedPush[i].send("ack");
    // Notify of change to observer
    participantConnected.setValue(messageReceived);
    // Send message to all connected participants of how many participants left
     ↪  to complete the room
    sendHowManyParticipantsLeft(participantsConnectedPush, i+1);
}
```