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

DISEÑO E IMPLEMENTACIÓN DE PLATAFORMA DE AUTENTIFICACIÓN ANÓNIMA Y SU USO PARA DENUNCIAS EN LA UNIVERSIDAD DE CHILE

MEMORIA PARA OPTAR AL TÍTULO DE INGENIERO CIVIL EN COMPUTACIÓN

CLEMENTE ENRIQUE PAREDES GÓMEZ

PROFESOR GUÍA:
ALEJANDRO HEVIA ANGULO

PROFESOR CO-GUÍA:
CAMILO GÓMEZ NÚÑEZ

MIEMBROS DE LA COMISIÓN:
CLAUDIO GUTIÉRREZ GALLARDO
TOMÁS BARROS ARANCIBIA

SANTIAGO DE CHILE
2022

RESUMEN DE LA MEMORIA PARA OPTAR
AL TÍTULO DE INGENIERO CIVIL EN COMPUTACIÓN
POR: CLEMENTE ENRIQUE PAREDES GÓMEZ
FECHA: 2022
PROF. GUÍA: ALEJANDRO HEVIA

DISEÑO E IMPLEMENTACIÓN DE PLATAFORMA DE AUTENTIFICACIÓN ANÓNIMA Y SU USO PARA DENUNCIAS EN LA UNIVERSIDAD DE CHILE

Las firmas de grupo distribuidas, cómo la descrita por Kuykendall, Krawczyk y Rabin (Private Enhancing Technologies 2019), le permiten a miembros individuales de un grupo firmar mensajes en nombre del grupo de tal forma que la identidad del firmante se mantenga oculta, esto es, sólo revelan que un miembro del grupo ha firmado el documento, con la excepción de que las autoridades del grupo pueden colaborar para individualizarlo de ser necesario.

Este trabajo presenta una aplicación práctica de firmas de grupo para un sistema de autentificación anónima para permitir, por ejemplo, denuncias anónimas de corrupción o acoso, para la Universidad de Chile.

Concretamente, se implementó un prototipo basado en el esquema de firmas de grupo de Boneh, Boyen, Shacham (Advances in Cryptology, Crypto 2004) extendido para permitir autoridades distribuidas y el recambio seguro de estas.

Junto al prototipo se entregan las indicaciones y consideraciones para su desarrollo completo como sistema de autentificación anónima el cual pueda ser utilizado en el futuro por la Universidad.

A mis padres, mi pareja y mis amigos

Agradecimientos

Agradezco a mis padres, abuelos y padrinos por haberme apoyado en todas las decisiones que me han llevado a ser quién soy ahora y proveerme de todo lo que necesité, tanto emocional como material.

A Catalina por haberme acompañado a lo largo de toda mi carrera universitaria dándome la motivación para dar lo mejor de mí.

A mis amigos, tanto del colegio como de la facultad, por haber estado siempre dispuestos a ayudarme.

Y por último, a mis profesores y auxiliares, en especial a los que fueron fundamentales para este trabajo: Alejandro Hevia, Camilo Gómez e Ilana Mergudich.

Tabla de Contenido

Índice de Ilustraciones

Lista de Algoritmos

Capítulo 1

Introducción

1.1. Contexto

Un mecanismo para denunciar las malas prácticas que pueden darse al interior de una institución es esencial para su correcto funcionamiento. Esto porque, al no existir este mecanismo, un actor malicioso puede pasar desapercibido con mayor facilidad.

En relación a ello, se realizó un estudio para el Consejo para la Transparencia durante el año 2020[1]. Dentro de los encuestados pertenecientes a la demografía "Educación técnica y universitaria incompleta", un $43,8\,\%$ declaró enterarse mediante terceros o haber sido testigo de un caso de corrupción en el sector público, un $74\,\%$ de ellos desconoce como denunciar y un $36\,\%$ está en desacuerdo con el enunciado "En nuestra sociedad, existen las condiciones institucionales para que las personas denuncien hechos de corrupción con seguridad".

Dentro de la demografía "Educación universitaria completa", un $7\,\%$ afirma haber pagado un soborno, dado un regalo o hecho un favor a un funcionario público, o conocer a un tercero que lo haya hecho, $3\,\%$ por encima del total de encuestados.

En la Universidad de Chile, existe un protocolo de actuación ante denuncias que considera a integrantes de la comunidad universitaria o personas vinculadas de cualquier forma a las actividades de la institución. Pueden ser denunciados hechos que hayan sido cometidos dentro o fuera de sus recintos. Las denuncias pueden ser presentadas de manera escrita o de manera verbal (acta escrita), por medios electrónicos o presenciales. Las denuncias escritas se realizan mediante correo electrónico, estas deben contener principalmente la identificación de la persona afectada y del denunciado, un relato de los hechos en que se fundamenta la denuncia, señalando fecha y lugar en que ocurrieron, además de indicar si existen pruebas de la conducta denunciada.

Desafortunadamente, el protocolo, tanto de manera escrita como verbal, expone la identidad del denunciante, lo que resulta intimidante y puede ser un desincentivo para las denuncias legítimas. Por ello, toma importancia garantizar el anonimato del denunciante para así incentivar el uso del protocolo de denuncia. Sin embargo, se debe considerar un mecanismo de autentificación, de lo contrario el sistema será vulnerable a denuncias falsas y/o la saturación del canal.

Lograr autentificación sin sacrificar el anonimato es un problema no trivial y ha sido abordado en la literatura mediante esquemas relativos al concepto de firmas digitales [1]. Uno

[1] Este informe puede ser encontrado en el Estudio Nacional de Transparencia 2020 https://www.consejotransparencia.cl/wp-content/uploads/estudios/2021/07/Estudio-Nacional-Transparencia-2020.pdf

de estos esquemas es el de firmas de grupo [2]. En este, un grupo compuesto por posibles firmantes y un administrador operan como sigue:

- Cada miembro del grupo puede firmar sus mensajes a nombre del grupo sin revelar su identidad.
- Sólo los miembros del grupo pueden generar una firma a nombre del grupo.
- Sólo el administrador es capaz de determinar la identidad de un firmante, siempre y cuando pertenezca al grupo.

Siendo el denunciante un miembro del grupo y una autoridad de la Universidad de Chile tomando el rol de administrador, se puede ver que basta un sistema con estas 3 propiedades para resolver el presente problema. Un denunciante no debe revelar su identidad al denunciar, denuncias por un agente externo no son permitidas y un denunciante malicioso puede perder su anonimato en caso de mal comportamiento.

La gran mayoría de esquemas de firma de grupo requieren que cada miembro del grupo cuente con un par de claves, pública y privada. Actualmente la Universidad de Chile no provee dichas claves a sus miembros, por lo que es necesario afrontar este problema como primer obstáculo en el desarrollo de esta solución.

Contar con una plataforma de generación de claves puede ser beneficioso, no solo para el desarrollo de un sistema de denuncias anónimas, sino también para otros servicios digitales que la Universidad puede implementar para facilitar acciones que puedan realizar sus miembros remotamente.

La librería presentada en este trabajo está basada en la implementación WhoTooPlus[2] [3] de WhoToo [4] desarrollada por Ilana Mergudich.

1.2. Motivación

Para analizar los requerimientos mínimos de un esquema que solucione el problema es útil considerar el caso de un miembro de la universidad que desea realizar una denuncia, pero que naturalmente no quiere que su identidad sea revelada, y el de la universidad, que tiene un tiempo limitado para atender denuncias, por lo que le interesa minimizar la cantidad de denuncias falsas posibles.

El sistema ideal para el denunciante sería uno sin autenticación, en el que simplemente puede entrar y dejar una denuncia sin ninguna validación, incluso puede usar algún método para ocultar su dirección IP del sistema.

Mientras tanto, el sistema ideal para la universidad es uno al que sólo pueden acceder miembros de la comunidad, ya que alguien externo está fuera del público objetivo para estas denuncias. También le gustaría que existiera alguna forma de penalizar a quienes sean miembros de la comunidad y hagan mal del sistema, pero para esto es necesario saber la identidad de los usuarios problemáticos.

Viendo los requisitos de estos sistemas, parece ser que se ha llegado a una contradicción. No se puede asegurar anonimato al mismo tiempo que se aseguran consecuencias para actores insinceros una vez detectados. Para reconciliar los intereses de estas dos partes se pueden usar **firmas de grupo**. Ellas aseguran que los comportamientos maliciosos puedan ser penalizados, pero limitan el tiempo del anonimato al criterio de su administrador.

[2] El código de esta implementación puede ser encontrado en https://github.com/ilanamt/WhoTooPlus/tree/parallel

Si el denunciante confía en que la universidad no le quitará el anonimato a menos que haga mal uso del sistema, entonces bastaría con esta solución. Sin embargo, si el objeto de la denuncia es el administrador o alguien muy cercano a este, es imposible lograr esta confianza.

Para minimizar la ocurrencia de este tipo de situaciones es útil **distribuir** el rol del administrador, esto es, considerar un escenario donde el administrador es reemplazado por un grupo de administradores, los cuales deben cooperar para realizar acciones como desanonimizar a firmantes. De esta manera, la situación de desconfianza sólo se dará cuando la mayoría de los administradores sean malintencionados.

Llegado este punto se tiene un sistema que cumple las necesidades inmediatas de la universidad y la mayoría de los denunciantes legítimos. Sin embargo, existe otro problema que no se ha considerado: si el sistema está en funcionamiento de manera indefinida eventualmente se volverá necesario reemplazar a los administradores, pero ¿Qué pasaría si un grupo de ex-administradores conspirara con un grupo minoritario de administradores actuales? Serían capaces quitarle el anonimato a usuarios sin el consenso adecuado de los administradores. Para evitar una situación así, el sistema debería ser capaz de inutilizar la clave de un ex-administrador, a esto se le llamará "**proactividad**" y será definida formalmente más adelante en la sección 2.3.3.

1.3. Observaciones Generales

Es importante notar que al momento de la investigación para este proyecto no existe una solución que cumpla los requerimientos mínimos expuestos anteriormente:

- Software como el navegador TOR es capaz de ocultar la identidad del denunciante, pero no habría forma de validar que el denunciante sea un miembro de la institución, mucho menos de individualizarlo en caso de que sea necesario.
- Un *login* tradicional le permitiría a un administrador malicioso determinar la identidad del denunciante sin que se cumplan las condiciones apropiadas.
- Implementar un sistema de *login* con librerías existentes de firmas de grupo, cómo por ejemplo libgroupsig[3], estaría limitado a un único administrador. Por lo tanto, ese administrador sería el punto de fallo del sistema.
- La librería propuesta en [4] e implementada en WhoTooPlus[3], cumple con la distribución de los administradores, pero carece de proactividad. Por lo que administradores expulsados del sistema podría coludirse para identificar al denunciante.

1.4. Objetivos

1.4.1. Objetivo General

Creación de una librería criptográfica capaz de ofrecer las funcionalidades requeridas para un sistema autenticación con firmas de grupo y demostrar su uso mediante un prototipo, o prueba de concepto. Este sistema de autentificación pseudo-anónima debe contar con distribución de administrador para emitir claves y desanonimizar firmas además de proactividad.

[3] El código de esta implementación puede ser encontrado en https://github.com/IBM/libgroupsig/

1.4.2. Objetivos Específicos

1. Sintetizar el estado del arte relativo a implementaciones de firmas de grupo con autoridad distribuida.
2. Desarrollar una librería de autentificación anónima a partir de la implementación desarrollada por Ilana Mergudich [3].
3. Incorporar proactividad a la implementación distribuida.
4. Utilizar la librería de firma de grupo distribuido proactivo para implementar un prototipo de sistema de autenticación anónima (*script*).

1.5. Organización del Documento

En este trabajo se describe la propuesta de un protocolo de autenticación condicionalmente anónimo, mostrando tanto su diseño, como su implementación realizada y propuesta. En el Capítulo 2 se exponen los conceptos matemáticos y trabajos relacionados necesarios para entender la solución presentada en este proyecto. Luego en el Capítulo 3 se especifica el alcance del problema y los requisitos que debería cumplir una solución. Posteriormente en el Capítulo 4 se describe el diseño de la solución y se menciona qué partes de este sistema ya se encuentran implementados. En el Capítulo 5 se discuten las características de la solución que no han sido completamente integradas, aspectos del trabajo que pueden ser revisados a futuro y una breve mención a la investigación que no fue usada en este trabajo. Finalmente en el Capítulo 6 se evalúa el cumplimiento de las propiedades de la solución y los requisitos para llevar este trabajo a producción.

Capítulo 2

Marco Teórico

En este capítulo, se describen los conceptos y herramientas matemáticas (criptográficas) necesarias para entender el presente trabajo.

2.1. Encriptación

Encriptar un mensaje m significa modificarlo de tal manera que, a cualquier observador distinto del receptor objetivo B no lo sea factible determinar información sobre su contenido. Llamamos a este observador el adversario Adv. Luego, para recuperar (desencriptar) el mensaje original es necesario utilizar una clave preestablecida.

Antes de explicar como llevar a cabo este objetivo, es bueno aclarar los conceptos a utilizar:

- Mensaje m: Contenido que se quiere transmitir, pero que no sea posible leer o modificar por un agente distinto al receptor.
- Texto cifrado c: Producto de la encriptación del mensaje, necesario para poder recuperarlo.
- Emisor A: Agente que produce el mensaje y posteriormente el texto cifrado.
- Receptor B: Agente que recibe el texto cifrado y recupera el mensaje.
- Adversario Adv: Cualquier agente que no sea ni el emisor ni el receptor.
- Generar claves $\mathcal{K}$: Crear las claves necesarias para encriptar y desencriptar un mensaje.
- Encriptar $\mathcal{E}$: Modificar un mensaje, en un texto cifrado, de tal forma que sea ininteligible para cualquier adversario.
- Desencriptar $\mathcal{D}$: Modificar un texto cifrado de tal manera que resulte en el mensaje original generado por el emisor.
- Esquema de encriptación $\mathcal{S}$: Tupla de algoritmos $\mathcal{S} = (\mathcal{K}, \mathcal{E}, \mathcal{D})$, que definen como crear las claves, encriptar mensajes y desencriptar textos cifrados, respectivamente. Un esquema de encriptación es correcto cuando el resultado de encriptar y desencriptar un mensaje resulta en el mensaje mismo, es decir, $m = \mathcal{D}(\mathcal{E}(m))$.

Los esquemas de ecriptación se diferencian en 2 categorías por la manera de usar sus claves:

Un esquema es simétrico si la clave para encriptar y desencriptar son iguales o se puede determinar trivialmente una a partir de la otra.

Para ofrecer comunicación entre n participantes de un sistema, cada par de participantes debe acordar una clave, es decir se requieren $\binom{n}{2} = \frac{n(n-1)}{2}$ claves diferentes, con cada participante almacenando $n-1$ claves en todo momento.

Este tipo de esquema no es utilizado en el presente trabajo, por lo que no se ahondará más al respecto [5][6].

Por el contrario, un esquema es asimétrico cuando la clave para encriptar es distinta a la clave para desencriptar.

2.1.1. Encriptación Asimétrica

La encriptación asimétrica, también llamada de clave pública, utiliza dos claves generadas por el receptor B:

- Clave pública pk: Esta clave es utilizada para poder encriptar un mensaje m. Como lo indica su nombre, esta clave es pública, por lo que cualquiera puede encriptar mensajes para B.
- Clave privada sk: Luego de recibir un texto cifrado (encriptado bajo pk), B debe desencriptarlo utilizando la clave privada sk asociada a pk. Esta clave debe mantenerse en secreto, ya que de lo contrario cualquiera (en particular el adversario) puede conocer el contenido de los mensajes dirigidos a B.

Para ofrecer comunicación entre n participantes basta con que cada uno de ellos genere un par de claves (pk, sk), es decir, se necesitan $2n$ claves y ningún participante requiere almacenar más de 3 claves en un momento dado, estas son, el par de claves propia del participante y la clave pública del participante al que se pretende enviar un mensaje cifrado [5].

En resumen un esquema asimétrico $(\mathcal{K}, \mathcal{E}, \mathcal{D})$ puede visualizarse de la siguiente manera:

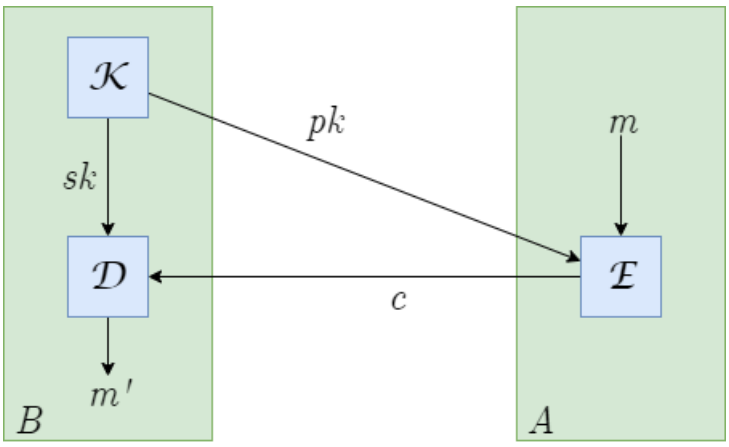


Figura 2.1: Esquema de encriptación asimétrica.

2.1.2. Encriptación ElGamal

El esquema de encriptación ElGamal es una encriptación de clave pública [7] definido de la siguiente manera:

Sea $\mathbb{Z}_p$ el conjunto de los enteros módulo primo p y $\mathbb{G}_1$ un grupo de enteros [3].

Se utilizará la notación $a \in_R S$ para referirse a un elemento a escogido de un conjunto S de manera aleatoria con distribución uniforme.

Algoritmo 1 $\mathcal{K}$: ElGamal.KeyGen()

Autores: Kuykendall et al.[4]

Salida:

pk: clave pública.

sk: clave privada.

1: Escoger $x \in_R \mathbb{Z}_p$

2: Escoger $h \in_R \mathbb{G}_1$

3: Definir $g \leftarrow h^{1/x}$

4: Retornar $pk \leftarrow (g, h)$

5: Retornar $sk \leftarrow x$

Algoritmo 2 $\mathcal{E}$: ElGamal.Enc(pk, m)

Autores: Kuykendall et al.[4]

Entrada:

pk: clave pública.

m: mensaje a encriptar.

Salida:

(c_1, c_2): cifrado de m.

1: Escoger $a \in_R \mathbb{Z}_p$

2: Retornar $(c_1, c_2) \leftarrow (g^a, h^a m)$

Algoritmo 3 $\mathcal{D}$: ElGamal.Dec$(sk, c1, c2)$

Autores: Kuykendall et al.[4]

Entrada:

sk: clave privada.

(c_1, c_2): texto cifrado.

Salida:

m': mensaje desencriptado.

1: Definir $x \leftarrow sk$

2: Retornar $m' \leftarrow c_2 / c_1^x$

Se puede ver que esta desencriptación es correcta:

$$m' = \frac{c_2}{c_1^x} = \frac{h^a m}{(g^a)^x} = \frac{h^a m}{(h^{1/x})^{ax}} = \frac{h^a m}{h^a} = m \tag{2.1}$$

2.2. Firma Electrónica

Mientras la encriptación permite transmitir un mensaje sin que su contenido sea revelado al adversario, las firmas permiten asegurar que un mensaje proviene de un emisor específico y no de un adversario pretendiendo ser el emisor. Esta propiedad se llama "autenticidad".

Un esquema de firma electrónica es una tupla de algoritmos $\mathcal{DS} = (\mathcal{K}, \mathcal{S}, \mathcal{V})$, que definen como crear las claves, firmar mensajes y verificar firmas, respectivamente [1].

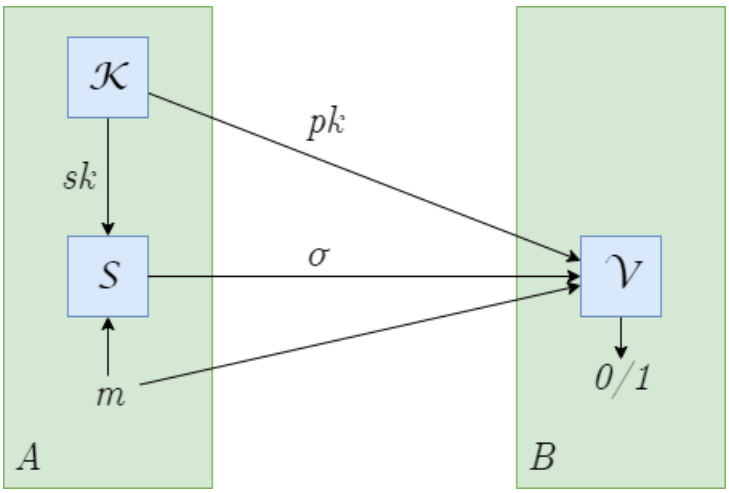


Figura 2.2: Esquema de firma digital.

En general para un firmante A y un verificador B se tiene:

1. A crea un par (sk, pk) y publica pk.
2. Al enviar un mensaje m, A genera una firma σ a partir del mensaje y sk.
3. Al recibir el mensaje m y la firma σ, B puede usar pk para verificar que el remitente del mensaje es A [5].

2.2.1. Firma de Grupo

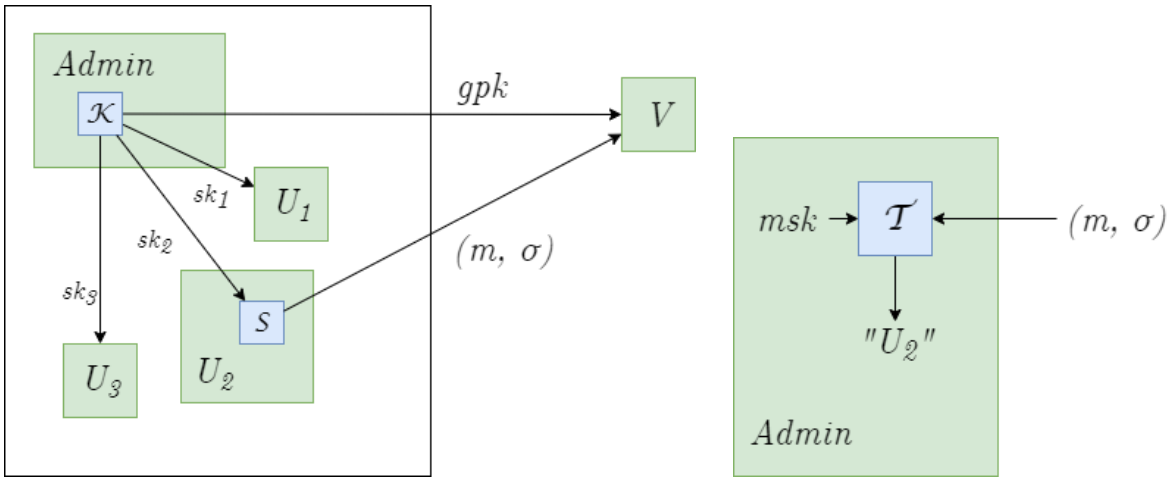
Un esquema de firma de grupo busca darle, a un conjunto de usuarios autorizados, la capacidad de generar firmas electrónicas asociadas al grupo, pero no a la identidad individual del firmante.

En este tipo de esquemas participan n usuarios y un administrador; y se cumplen las propiedades:

- Anonimato: Dado en mensaje m y dos usuarios U_1 y U_2 con sus claves privadas. Si uno de los usuarios crea una firma σ de m un adversario es incapaz de determinar si el firmante fue U_1 o U_2.
- Trazabilidad: Dado un mensaje m y un subconjunto de usuarios coludidos, estos no pueden colaborar de ninguna manera para producir una firma σ de m que pueda simultáneamente asociada al grupo y no ser trazable a alguno de estos usuarios por el administrador.

Para este fin, el grupo tiene una única clave pública gpk, cada usuario tiene su propia clave privada usk_i que le permite generar una firma σ relativa a gpk y el administrador tiene una clave privada msk tal que, dada una firma, le permite al saber la identidad de quién la generó.

Este esquema puede expresarse como una tupla $\mathcal{DGS} = (\mathcal{K}, \mathcal{S}, \mathcal{V}, \mathcal{T})$ de los algoritmos que definen como generar las claves, firmar un mensaje, verificar una firma y trazar la identidad de un firmante [2].



(a) Firma (b) Apertura

Figura 2.3: Esquema de firma de grupo.

2.2.2. Firma Boneh Boyen Shacham (BBS)

El esquema BBS es una implementación de firma de grupo que hace uso de pares bilineales [8], los cuales se definen de la siguiente manera:

Sean $\mathbb{G}_1$, $\mathbb{G}_2$ y $\mathbb{G}_T$ grupos cíclicos de orden primo para el operador $\cdot$. Se dice que $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ es un emparejamiento bilineal si satisface:

i) e es bilineal, es decir, $e(g \cdot g', h) = e(g, h) \cdot e(g', h)$ y $e(h, g \cdot g') = e(h, g) \cdot e(h, g')$.

ii) e es no degenerado, es decir, $\exists g_1 \neq 1 \in \mathbb{G}_1$ y $\exists g_2 \neq 1 \in \mathbb{G}_2$ tales que $e(g_1, g_2) = 1$

iii) Existe un algoritmo que computa e en tiempo polinomial.

Existen 3 tipos de pares bilineales

Tipo I: $\mathbb{G}_1 = \mathbb{G}_2$

Tipo II: $\mathbb{G}_1 \neq \mathbb{G}_2$ y existe un isomorfismo conocido $\psi : \mathbb{G}_1 \to \mathbb{G}_2$ que se puede computar eficientemente.

Tipo III: $\mathbb{G}_1 \neq \mathbb{G}_2$ y no existe un isomorfismo conocido $\psi : \mathbb{G}_1 \to \mathbb{G}_2$ que se pueda computar eficientemente.

Para BBS y para este trabajo se utilizan pares bilineales de tipo III [3].

Luego, definiendo g_1 y g_2 como los generadores de $\mathbb{G}_1$ y $\mathbb{G}_2$ respectivamente y $\mathcal{H} : \{0,1\}^* \to \mathbb{Z}_p$ como una función de hash criptográfica pública:

Algoritmo 4 $\mathcal{K}$: BBS.KeyGen(n)

Autores: Boneh et al.[8]

Entrada:

$n \in \mathbb{N}$: número de participantes en el grupo.

Salida:

gpk: clave pública del grupo.

msk: clave privada del adminitrador.

$\{usk_i\}_i$: conjunto de pares de claves de los miembros del grupo.

1: Escoger $h \in_R \mathbb{G}_1 \backslash \{1\}$
2: Escoger $\xi_1, \xi_2 \in_R \mathbb{Z}_p^*$
3: Encontrar $u, v \in \mathbb{G}_1$ tales que $u^{\xi_1} = v^{\xi_2}$
4: Definir $h \leftarrow u^{\xi_1}$
5: Escoger $\gamma \in_R \mathbb{Z}_p^*$
6: Definir $w \leftarrow g_2^{\gamma}$
7: **for** $i \leftarrow 1...n$ **do**
8: Escoger $x_i \in_R \mathbb{Z}_p^*$
9: Definir $A_i \leftarrow g_1^{1/(\gamma + x_i)}$
10: Definir $usk_i \leftarrow (A_i, x_i)$
11: Enviar usk_i a U_i
12: **end for**
13: Definir $gpk \leftarrow (g_1, g_2, h, u, v, w)$
14: Definir $msk \leftarrow (\xi_1, \xi_2)$

Algoritmo 5 $\mathcal{S}$: BBS.Sign(gpk, usk_i, m)

Autores: Boneh et al.[8]

Entrada:

gpk: clave pública del grupo.
usk_i: pares de claves del miembro U_i.
m: mensaje a firmar.

Salida:

σ: firma de m.

1: Definir $(g_1, g_2, h, u, v, w) \leftarrow gpk$
2: Definir $(A, x) \leftarrow usk_i$
3: Escoger $\alpha, \beta \in_R \mathbb{Z}_p$
4: Definir $T_1 \leftarrow u^{\alpha}$
5: Definir $T_2 \leftarrow v^{\beta}$
6: Definir $T_3 \leftarrow Ah^{\alpha+\beta}$
7: Definir $\delta_1 \leftarrow x\alpha$
8: Definir $\delta_2 \leftarrow x\beta$
9: Escoger $r_{\alpha}, r_{\beta}, r_x, r_{\delta_1}, r_{\delta_2} \in_R \mathbb{Z}_p$
10: Definir $R_1 \leftarrow u^{r_{\alpha}}$
11: Definir $R_2 \leftarrow v^{r_{\beta}}$
12: Definir $R_3 \leftarrow e(T_3, g_2)^{r_x} \cdot e(h, w)^{-r_{\alpha}-r_{\beta}} \cdot e(h, g_2)^{-r_{\delta_1}-r_{\delta_2}}$
13: Definir $R_4 \leftarrow T_1^{r_x} \cdot u^{-r_{\delta_1}}$
14: Definir $R_5 \leftarrow T_2^{r_x} \cdot v^{-r_{\delta_2}}$
15: Definir $c \leftarrow \mathcal{H}(m, T_1, T_2, T_3, R_1, R_2, R_3, R_4, R_5)$
16: Definir $s_{\alpha} \leftarrow r_{\alpha} + c\alpha$
17: Definir $s_{\beta} \leftarrow r_{\beta} + c\beta$
18: Definir $s_x \leftarrow r_x + cx$
19: Definir $s_{\delta_1} \leftarrow r_{\delta_1} + c\delta_1$
20: Definir $s_{\delta_2} \leftarrow r_{\delta_2} + c\delta_2$
21: Definir $\sigma \leftarrow (T_1, T_2, T_3, c, s_{\alpha}, s_{\beta}, s_x, s_{\delta_1}, s_{\delta_2})$

Algoritmo 6 $\mathcal{V}$: BBS.Verify(gpk, m, σ)

Autores: Boneh et al.[8]

Entrada:

gpk: clave pública del grupo.
m: mensaje firmado.
σ: firma del mensaje.

Salida:

True si la firma fue producida por un miembro del grupo.

1: Definir $(g_1, g_2, h, u, v, w) \leftarrow gpk$
2: Definir $(T_1, T_2, T_3, c, s_\alpha, s_\beta, s_x, s_{\delta_1}, s_{\delta_2}) \leftarrow \sigma$
3: Definir $\tilde{R}_1 \leftarrow \dfrac{u^{s_\alpha}}{T_1^c}$
4: Definir $\tilde{R}_2 \leftarrow \dfrac{v^{s_\beta}}{T_2^c}$
5: Definir $\tilde{R}_3 \leftarrow e(T_3, g_2)^{s_x} \cdot e(h, w)^{-s_\alpha - s_\beta} \cdot e(h, g_2)^{-s_{\delta_1} - s_{\delta_2}} \cdot \left(\dfrac{e(T_3, w)}{e(g_1, g_2)}\right)^c$
6: Definir $\tilde{R}_4 \leftarrow \dfrac{T_1^{s_x}}{u^{s_{\delta_1}}}$
7: Definir $\tilde{R}_5 \leftarrow \dfrac{T_2^{s_x}}{v^{s_{\delta_2}}}$
8: **if** $c = \mathcal{H}(m, T_1, T_2, T_3, \tilde{R}_1, \tilde{R}_2, \tilde{R}_3, \tilde{R}_4, \tilde{R}_5)$ **then**
9: Retornar **True**
10: **else**
11: Retornar **False**
12: **end if**

Es posible ver que cada $\tilde{R}$ es igual a su correspondiente R:

$$\tilde{R}_1 = \frac{u^{s_\alpha}}{T_1^c} = \frac{u^{r_\alpha + c\alpha}}{(u^\alpha)^c} = u^{r_\alpha} = R_1 \tag{2.2}$$

$$\tilde{R}_2 = \frac{v^{s_\beta}}{T_2^c} = \frac{v^{r_\beta + c\beta}}{(v^\beta)^c} = v^{r_\beta} = R_2 \tag{2.3}$$

$$\tilde{R}_3 = e(T_3, g_2)^{s_x} \cdot e(h, w)^{-s_\alpha - s_\beta} \cdot e(h, g_2)^{-s_{\delta_1} - s_{\delta_2}} \cdot \left(\frac{e(T_3, w)}{e(g_1, g_2)}\right)^c \tag{2.4}$$

$$= \left(\frac{e(T_3, g_2)^{r_x + cx}}{e(h, w)^{r_\alpha + c\alpha + r_\beta + c\beta} \cdot e(h, g_2)^{r_{\delta_1} + c\delta_1 + r_{\delta_2} + c\delta_2}}\right) \cdot \left(\frac{e(T_3, w)}{e(g_1, g_2)}\right)^c \tag{2.5}$$

$$= R_3 \left(\frac{e(T_3, g_2)^{cx} \cdot e(T_3, w)^c}{e(h, w)^{c\alpha + c\beta} \cdot e(h, g_2)^{cx\alpha + cx\beta} \cdot e(g_1, g_2)^c}\right) \tag{2.6}$$

$$= R_3 \left(\frac{e(A, g_2)^{cx} \cdot e(h, g_2)^{cx(\alpha+\beta)} \cdot e(A, w)^c \cdot e(h, w)^{c(\alpha+\beta)}}{e(h, w)^{c(\alpha+\beta)} \cdot e(h, g_2)^{cx(\alpha+\beta)} \cdot e(g_1, g_2)^c}\right) \tag{2.7}$$

$$= R_3 \left(\frac{e(A, g_2)^{cx} \cdot e(A, w)^c}{e(g_1, g_2)^c}\right) \tag{2.8}$$

$$= R_3 \left(\frac{e(g_1, g_2)^{x/(\gamma+x)} \cdot e(g_1, g_2)^{\gamma/(\gamma+x)}}{e(g_1, g_2)}\right)^c \tag{2.9}$$

$$= R_3 \left(\frac{e(g_1, g_2)}{e(g_1, g_2)}\right)^c \tag{2.10}$$

$$= R_3 \tag{2.11}$$

$$\tilde{R}_4 = \frac{T_1^{s_x}}{u^{s_{\delta_1}}} = \frac{T_1^{r_x + cx}}{u^{r_{\delta_1} + c\delta_1}} = \frac{T_1^{r_x} (u^\alpha)^{cx}}{u^{r_{\delta_1}} u^{c(x\alpha)}} = T_1^{r_x} \cdot u^{-r_{\delta_1}} = R_4 \tag{2.12}$$

$$\tilde{R}_5 = \frac{T_2^{s_x}}{v^{s_{\delta_2}}} = \frac{T_2^{r_x + cx}}{v^{r_{\delta_2} + c\delta_2}} = \frac{T_2^{r_x} (v^\beta)^{cx}}{v^{r_{\delta_2}} v^{c(x\beta)}} = T_2^{r_x} \cdot v^{-r_{\delta_2}} = R_5 \tag{2.13}$$

Algoritmo 7 $\mathcal{T}$: BBS.Trace(gpk, msk, m, σ)

Autores: Boneh et al.[8]

Entrada:

gpk: clave pública del grupo.
m: mensaje firmado.
σ: firma del mensaje.

Salida:

A: clave pública del firmante.

1: **if** $BBS.Verify(gpk, m, \sigma)$ **then**
2: Definir $(\xi_1, \xi_2) \leftarrow msk$
3: Definir $(T_1, T_2, T_3, c, s_\alpha, s_\beta, s_x, s_{\delta_1}, s_{\delta_2}) \leftarrow \sigma$
4: Definir $A \leftarrow \dfrac{T_3}{T_1^{\xi_1} T_2^{\xi_2}}$
5: Retornar A
6: **else**
7: Retornar $\perp$
8: **end if**

Es posible ver que A es la identidad del firmante i:

$$A = \frac{T_3}{T_1^{\xi_1} T_2^{\xi_2}} = \frac{A_i h^{\alpha+\beta}}{(u^\alpha)^{\xi_1}(v^\beta)^{\xi_2}} = \frac{A_i h^\alpha h^\beta}{h^\alpha h^\beta} = A_i \tag{2.14}$$

2.3. Criptografía de Umbral

Un protocolo de umbral consta de n participantes, de los cuales, a lo menos $k+1$ deben estar de acuerdo para realizar una acción determinada. Todos los protocolos de este tipo están basados en el protocolo de compartición de secretos verificable (VVS) [9] que a su vez está basado en el protocolo de compartición de secretos de Shamir [10].

2.3.1. Compartición de Secretos

Para compartir un valor secreto $s \in \mathbb{Z}_p$, se escoge un polinomio P de grado $k \leq n$ tal que $P(0) = s$ y cada "trozo" $s_i = P(i)$ es entregado al participante U_i.

Luego, usando el método de multiplicadores de Lagrange es posible determinar el valor $P(0)$ sólo si se conocen $k+1$ o más trozos s_i.

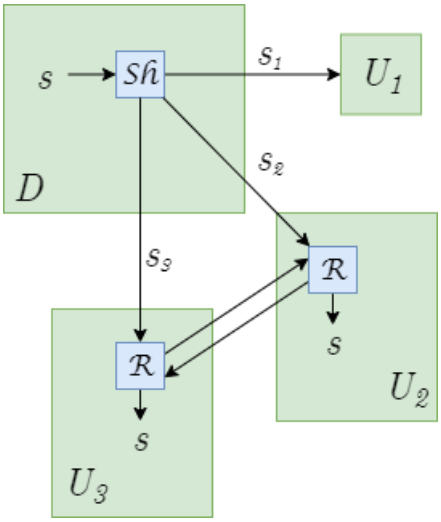


Figura 2.4: Esquema de compartición de secretos.

2.3.2. Compartición de Secretos Verificable

VVS es una extensión de SSS que permite verificar la validez de los trozos. Para esto es necesario definir públicamente un grupo $\mathbb{G}$ con generadores g y h.

Para compartir un secreto $s \in \mathbb{Z}_p$, se escoge un valor aleatorio $r \in_R \mathbb{Z}_p$ y dos polinomios P y Q de grado $k \leq n$ tales que $P(0) = s$ y $Q(0) = r$, luego se generan k valores de verificación $v_j = g^{a_j} h^{b_j}$ donde a_j y b_j son los coeficientes de P y Q respectivamente.

Se le entrega a cada participante U_i sus trozos s_i y r_i junto a los valores de verificación $\{v_1 ... v_k\}$, cuya validez pueden confirmar evaluando si $g^{s_i} h^{r_i} = \Pi_{j=0}^{k} v_j^{ij}$ [3].

Asumiendo g y h conocidos, estos algoritmos se pueden definir de la siguiente manera:

Algoritmo 8 $\mathcal{SH}$: SecShare.Share(s)

Autor: Pedersen[9]

Entrada:

$s \in \mathbb{Z}_p$: secreto.

Salida:

(s_i, r_i, v, e_0, r): trozo para U_i.

1: Escoger $r \in_R \mathbb{Z}_p$
2: Generar P tal que $P(0) = s$
3: Generar Q tal que $Q(0) = r$
4: Definir $e_0 \leftarrow g^r$
5: **for** $j \leftarrow 0...k$ **do**
6: Definir $v[j] \leftarrow g^{a_j} h^{b_j}$
7: **end for**
8: **for** $i \leftarrow 1...n$ **do**
9: $s_i \leftarrow P(i)$; $r_i \leftarrow Q(i)$
10: Enviar (s_i, r_i, v, e_0, r) a U_i
11: **end for**

Algoritmo 9 $\mathcal{V}$: SecShare.Verify(s_i, r_i, v)

Autor: Pedersen[9]

Entrada:

s_i: trozo del secreto correspondiente a U_i.

r_i: trozo del coeficiente aleatorio correspondiente a U_i.

v: compromisos de los coeficientes.

Salida:

True si los valores son consistentes con el protocolo.

1: **if** $g^{s_i} h^{r_i} = \Pi_{j=0}^{k} v_j^{ij}$ **then**
2: Retornar **True**
3: **else**
4: Retornar **False**
5: **end if**

Como g y h son generadores del mismo grupo, podemos definir $h = g^d$ para algún $d \in \mathbb{Z}_p$, por lo tanto se puede redefinir $v[j] = g^{a_j} h^{b_j} = g^{a_j + db_j}$, luego existe un polinomio $R(x) = (P + dQ)(x)$ de grado k con coeficientes $c_j = a_j + db_j$ que codifica el secreto $t = R(0) = s + dr$

$$\Pi_{j=0}^{k} v_j^{ij} = \Pi_{j=0}^{k} (g^{c_j})^{ij} = g^{\Sigma_{j=0}^{k} c_j i^j} = g^{t_i} = g^{s_i + dr_i} = g^{s_i} h^{r_i} \tag{2.15}$$

Algoritmo 10 $\mathcal{R}$: SecShare.Reconstruct($\{s_i\}_{k+1}$)

Autor: Pedersen[9]

Entrada:

$\{s_i\}_{k+1}$: trozos del secreto de $k+1$ participantes.

Salida:

s': reconstrucción del secreto compartido.

1: Definir $indices \leftarrow \{s_i\}_{k+1}.keys()$
2: **for** $i \in indices$ **do**
3: Definir $cf[i] \leftarrow \Pi_{j \in indices \setminus \{i\}} \frac{0-j}{i-j}$
4: **end for**
5: Modificar $s' \leftarrow \Sigma_{i \in indices}\ cf[i]s_i$
6: Retornar s'

Adicionalmente, se pueden definir sobre este esquema otras primitivas útiles para este trabajo:

- Obtener los trozos de la suma entre dos secretos:

Algoritmo 11 SecShare.Add(x_i, y_i)

Autores: Kuykendall et al.[4]

Entrada:

x_i: trozo del primer secreto.

y_i: trozo del segundo secreto.

Salida:

z_i: trozo de la suma de ambos secretos.

1: **for** Cada participante U_i **do**
2: Define $z_i \leftarrow x_i + y_i$
3: **end for**

Esto se sustenta en la propiedad de la suma de polinomios $\forall x \in \mathbb{R} : f(x) + g(x) = (f+g)(x)$

- Crear trozos de un secreto en el grupo $\mathbb{G}$ sin revelarlo *a priori*:

Algoritmo 12 SecShare.Gen()

Autores: Kuykendall et al.[4]

Salida:

s_i: trozo de un secreto desconocido.

1: **for** Cada participante U_i **do**
2: Escoge $x_i \in_R \mathbb{G}$
3: Comparte $SecShare.Share(x_i)$ al resto de los participantes
4: **end for**
5: **for** Cada participante U_i **do**
6: Recibe un trozo $x_{j,i}$ de cada U_j
7: Define $s_i \leftarrow \Sigma_{j=1}^{n} x_{j,i}$
8: **end for**

El secreto s codificado por los trozos s_i corresponde a la suma $\Sigma_{i=1}^{n} x_i$, como cada x_i es aleatorio y conocido solamente por el participante que lo generó, s es aleatorio y desconocido para un grupo de participantes menor a $k+1$.

- Obtener los trozos del producto entre dos secretos:

Algoritmo 13 SecShare.Mult(x_i, y_i)

Autores: Kuykendall et al.[4]

Entrada:

x_i: trozo del primer secreto.

y_i: trozo del segundo secreto.

Salida:

z_i: trozo del producto entre ambos secretos.

1: Escoger $a, b \in_R \mathbb{Z}_p$
2: Definir $c \leftarrow ab$
3: Compartir $SecShare.Share(a)$ a todos los participantes
4: Compartir $SecShare.Share(b)$ a todos los participantes
5: Compartir $SecShare.Share(c)$ a todos los participantes
6: **for** Cada participante U_i **do**
7: Define $d_i \leftarrow x_i - a_i$
8: Define $e_i \leftarrow y_i - b_i$
9: Publica (d_i, e_i)
10: **end for**
11: **for** Cada participante U_i **do**
12: Reconstruye $d \leftarrow SecShare.Reconstruct(\{d_j\}_n)$
13: Reconstruye $e \leftarrow SecShare.Reconstruct(\{e_j\}_n)$
14: Define $z_i \leftarrow c_i + x_i e + y_i d - ed$
15: **end for**

Como se vio anteriormente, la suma de polinomios permite definir el polinomio R asociado a los z_i como $R(x) = C(x) + eP(x) + dQ(x) - ed$, luego al reconstruir se tiene:

$$\begin{aligned} z = R(0) &= C(0) + eP(0) + dQ(0) - ed && (2.16)\\ &= ab + (y-b)x + (x-a)y - (y-b)(x-a) && (2.17)\\ &= ab + xy - bx + xy - ay - xy + ay + bx - ab && (2.18)\\ &= (ab - ab) + xy + (xy - xy) + (ay - ay) + (bx - bx) && (2.19)\\ &= xy && (2.20) \end{aligned}$$

- Obtener los trozos del inverso de un secreto:

Algoritmo 14 SecShare.Invert(s_i)

Autores: Kuykendall et al.[4]

Entrada:

s_i: trozo del secreto.

Salida:

z_i: trozo del inverso del secreto.

1: Escoger $r \in_R \mathbb{Z}_p$
2: Compartir $SecShare.Share(r)$ a todos los participantes
3: **for** Cada participante U_i **do**
4: Define $w_i \leftarrow SecShare.Mult(s_i, r_i)$
5: Publica w_i
6: **end for**
7: **for** Cada participante U_i **do**
8: Reconstruye $w \leftarrow SecShare.Reconstruct(\{w_j\}_n)$
9: Define $z_i \leftarrow r_i w^{-1}$
10: **end for**

Utilizando los algoritmos $SecShare.Mult$ y $SecShare.Reconstruct$ es posible definir $w = sr$ sin revelar ni s ni r, luego basta con escalar el polinomio Q que codifica $r = Q(0)$ por w^{-1} y al reconstruir se tiene:

$$w^{-1}Q(0) = w^{-1}r = s^{-1}r^{-1}r = s^{-1} \tag{2.21}$$

- Obtener el resultado de una base elevada a un secreto:

Algoritmo 15 SecShare.Exp(b, s_i)

Autores: Kuykendall et al.[4]

Entrada:

$b \in \mathbb{G}_1 \bigcup \mathbb{G}_2$: base.

s_i: trozo del secreto correspondiente a U_i.

Salida:

z: resultado de b elevado al secreto.

for Cada participante U_i **do**
    Define $b_{si} \leftarrow b^{s_i}$
    Define $g_{si} \leftarrow g_1^{s_i}$
    Escoge $r \leftarrow \mathbb{Z}_p$
    Define $a_1 \leftarrow g_1^r$
    Define $a_2 \leftarrow b^r$
    Define $a \leftarrow (a_1, a_2)$
    Define $c \leftarrow \mathcal{H}(g_1, g_{si}, a_1, a_2)$
    Define $t \leftarrow r + cs_i$
    Define $proof \leftarrow (a, c, t)$
    Publica $(b_{si}, g_{si}, proof)$
end for
for Cada participante U_i **do**
    for $j \in \{1...n\} \backslash \{i\}$ **do**
        Define $(a, c, t) \leftarrow proof_j$
        Define $(a_1, a_2) \leftarrow a$
        Define $v_1 \leftarrow g_1^t = a_1 g_{sj}^c$
        Define $v_2 \leftarrow b^t = a_2 b_{sj}^c$
        if not $(v1 \wedge v2)$ **then**
            Reporta a U_j
        end if
    end for
end for
for $i \leftarrow \{1...n\}$ **do**
    Definir $cf[i] \leftarrow \Pi_{j \in \{1...n\} \backslash \{i\}} \dfrac{0 - j}{i - j}$
end for
Definir $z \leftarrow \Pi_{i=1}^n b_{si}^{cf[i]}$

La validez de este resultado se puede ver de manera similar a *SecShare.Reconstruct*()

$$z = \Pi_{i=1}^n b_{si}^{cf[i]} = \Pi_{i=1}^n (b^{s_i})^{cf[i]} = b^{\Sigma_{i=1}^n cf[i] s_i} = b^s \tag{2.22}$$

2.3.3. Proactividad

Un ataque a compartición de secretos estudiado en [11] es el de un adversario capaz de acceder a la clave privada de un usuario en un determinado paso de tiempo.

Dadas las condiciones de este ataque, para un esquema de compartición de secretos como los vistos anteriormente, basta con que el adversario corrompa un usuario distinto durante $k+1$ pasos de tiempo para que sea capaz de reconstruir el secreto.

A partir de esta noción se define que un esquema de compartición de secretos es "proactivo" si el adversario debe corromper por lo menos a $k+1$ miembros del esquema en el mismo paso de tiempo para poder aprender algo del secreto.

La forma en que [11] propone lograr esta propiedad es mediante dos algoritmos (*Update* y *Recover*) que definen como los usuarios pueden computar nuevos trozos sin modificar el secreto original y como $k+1$ usuarios pueden asistir a otro usuario a recuperar su trozo del secreto, respectivamente.

Para estos algoritmos es necesario añadir una noción de tiempo a los trozos del secreto, por tanto, se define $s_i^{(t)}$ como el i-ésimo trozo del secreto s en el periodo t.

También es necesario destacar que los polinomios δ_i son de grado k y se asume cada participante dispone de claves de encriptación asimétrica (pk_i, sk_i).

Algoritmo 16 SecShare.Update()

Autores: Herzberg et al.[11]

Entrada:
$s_i^{(t-1)}$: trozo del secreto correspondiente a U_i en el periodo $(t-1)$.
Salida:
$s_i^{(t)}$: trozo del secreto correspondiente a U_i en el periodo (t).

1: **for** Cada participante U_i **do**
2: Genera δ_i tal que $\delta_i(0) = 0$
3: **for** $\forall \delta_{i,m}$ coeficiente de δ_i **do**
4: $\varepsilon_{i,m} \leftarrow g^{\delta_{i,m}}$
5: **end for**
6: **for** $j \leftarrow 1...n$ **do**
7: Define $u_{i,j} \leftarrow \delta_i(j)$
8: Define $e_{i,j} \leftarrow Enc(pk_j, u_{i,j})$
9: Define $VSS_i^{(t)} \leftarrow (i, t, \{\varepsilon_{i,m}\}_{m\in\{1...k\}}, \{e_{i,j}\}_{j\in\{1...n\}\setminus\{i\}})$
10: Define $\sigma_i^{(t)} \leftarrow Sign(sk_i, VSS_i^{(t)})$
11: Publica $(VSS_i^{(t)}, \sigma_i^{(t)})$
12: **end for**
13: **end for**
14: **for** Cada participante U_i **do**
15: **for** $j \leftarrow 1...n$ **do**
16: Define $u_{j,i} \leftarrow Dec(sk_i, e_{j,i})$
17: **if not** $g^{u_{j,i}} = \Pi_{m=1}^{k} \varepsilon_{j,m}^{i^m}$ **then**
18: Reporta a U_j
19: **end if**
20: **if not** $Verify(pk_j, VSS_j^{(t)}, \sigma_j^{(t)})$ **then**
21: Reporta a U_j
22: **end if**
23: **end for**
24: Define $s_i^{(t)} \leftarrow s_i^{(t-1)} + \Sigma_{j=1}^{n} u_{j,i}$
25: **end for**

Este algoritmo aprovecha la propiedad de suma de polinomios para crear el polinomio $P(x)^{(t)} = P(x)^{(t-1)} + \Sigma_{i=1}^{n}\delta_i(x)$, como cada δ_i codifica el secreto $\delta_i(0) = 0$, entonces:

$$s^{(t)} = P(0)^{(t)} = s^{(t-1)} + \Sigma_{i=1}^{n}0 = s^{(t-1)} \tag{2.23}$$

Como cada δ_i tiene coeficientes aleatorios, $P^{(t)}$ tiene coeficientes aleatorios distintos a $P^{(t-1)}$ inutilizando la información que el adversario haya recolectado en el periodo $t-1$.

Por otro lado, para asistir al participante U_r a recuperar su trozo del secreto:

Algoritmo 17 SecShare.Recover(r)

Autores: Herzberg et al.[11]

Entrada:

r: índice para el cuál se recuperará el trozo del secreto.

Salida:

s_r: trozo del secreto recuperado.

for Cada participante distinto de U_r **do**
    Genera δ_i tal que $\delta_i(r) = 0$
    for $j \in \{1...n\}\backslash\{r\}$ **do**
        Define $u_{i,j} \leftarrow \delta_i(j)$
        Define $e_{i,j} \leftarrow Enc(pk_j, u_{i,j})$
        Publica $e_{i,j}$
    end for
    for $j \in \{1...n\}\backslash\{r\}$ **do**
        Define $u_{j,i} \leftarrow Dec(sk_i, e_{j,i})$
        Define $s'_i \leftarrow s_i + \Sigma_{j=1}^{n}u_{j,i}$
        Define $x_i \leftarrow Enc(pk_r, s'_i)$
        Publica x_i
    end for
end for

`El participante` U_r:
for $j \in \{1...n\}\backslash\{r\}$ **do**
    Define $s'_j \leftarrow Dec(sk_r, x_j)$
end for
Define $s_r \leftarrow \Sigma_{j\in\{1...n\}\backslash\{r\}}s'_j$

De manera muy similar a *SecShare.Update* este proceso define un polinomio auxiliar $R(x) = P(x) + \Sigma_{i\in\{1...n\}\backslash\{r\}}\delta_i(x)$ que tiene la propiedad de ser aleatorio y que al ser evauado en r:

$$R(r) = P(r) + \Sigma_{i\in\{1...n\}\backslash\{r\}}\delta_i(r) = s_r + \Sigma_{i\in\{1...n\}\backslash\{r\}}0 = s_r \tag{2.24}$$

2.4. BBS Distribuido

El esquema WhoToo, descrito en [4], es un prototipo para el manejo de denuncias anónimas que hace uso de una extensión de BBS basada en [12]. Esta extensión utiliza la clave privada del administrador de BBS (msk) como un secreto distribuido y de esta forma logra distribuir

el rol del administrador.

Para este trabajo sólo es relevante la versión distribuida de los algoritmos de BBS descritos a continuación:

- Dados g_1 y g_2 generadores de $\mathbb{G}_1$ y $\mathbb{G}_2$, n número total de administradores y $k+1$ quórum de administradores:

Algoritmo 18 DistBBS.Init(n, k)

Autores: Kuykendall et al.[4]

Entrada:

$n \in \mathbb{N}$: número total de administradores.

$k \in \mathbb{N}$: número de administradores necesario para realizar una acción.

Salida:

gpk: clave pública del grupo.

$\{msk_i\}_i$: conjunto de claves privadas de los administradores.

1: **for** Cada administrador M_i **do**
2: Participa en $x_i \leftarrow SecShare.Gen()$
3: **for** $j \in \{1...n\} \backslash \{i\}$ **do**
4: **if not** $SecShare.Verify(x_{i,j}, r_{i,j}, v_j)$ **then**
5: Reporta M_j
6: **end if**
7: **end for**
8: Define $h \leftarrow \Pi_{j \in \{1...n\} \backslash \{i\}} v_j[0]$
9: Participa en $\gamma_i \leftarrow SecShare.Gen()$
10: **for** $j \in \{1...n\} \backslash \{i\}$ **do**
11: **if not** $SecShare.Verify(\gamma_{i,j}, r_{i,j}, v_j)$ **then**
12: Reporta M_j
13: **end if**
14: **end for**
15: Define $w \leftarrow SecShare.Exp(g_2, \gamma_i)$
16: Define $gpk \leftarrow (g_1, g_2, h, w)$
17: Define $msk_i \leftarrow (\gamma_i, x_i)$
18: **end for**

- Emisión de una clave de firma para un usuario U:

Algoritmo 19 DistBBS.Issue($gpk, \{msk_i\}_{k+1}$)

Autores: Kuykendall et al.[4]

Entrada:

gpk: clave pública del grupo.

$\{msk_i\}_{k+1}$: claves privadas de $k+1$ adminitradores.

Salida:

usk: par de claves del usuario U.

for Cada administrador M_i **do**
    Participa en $a_i \leftarrow SecShare.Gen()$
    for $j \in \{1...n\} \backslash \{i\}$ **do**
        if not $SecShare.Verify(a_{i,j}, r_{i,j}, v_j)$ **then**
            Reporta M_j
        end if
    end for
    Define $(g_1, g_2, h, w) \leftarrow gpk$
    Define $(\gamma_i, x_i) \leftarrow msk_i$
    Define $\alpha_i \leftarrow a_i - \gamma_i$
    Invierte $a_i' \leftarrow SecShare.Invert(a_i)$
    Participa en $R \leftarrow SecShare.Exp(g_1, a_i')$
    Envía (R, α_i) al usuario U
end for

`El usuario` U:
Recibe $k+1$ paquetes de la forma (R_i, α_i)
for $i \in emisores$ **do**
    if not $R_1 = R_i$ **then**
        Reporta a M_i
    end if
end for
Define $R \leftarrow R_1$
Define $\alpha \leftarrow SecShare.Reconstruct(\{\alpha_i\}_{k+1})$
Define $usk \leftarrow (R, \alpha)$

- Firma de un mensaje m por un usuario autorizado U:

Algoritmo 20 DistBBS.Sign(gpk, usk, m)

Autores: Kuykendall et al.[4]

Entrada:

gpk: clave pública del grupo.
usk: par de claves del usuario U.
m: mensaje a firmar.

Salida:

(c, σ): firma del mensaje m.

1: Definir $(g_1, g_2, h, w) \leftarrow gpk$
2: Definir $(R, \alpha) \leftarrow usk$
3: Definir $pk \leftarrow (g_1, h)$
4: Escoger $a \in_R \mathbb{Z}_p$
5: Definir $c \leftarrow ElGamal.Enc(pk, R, a = a)$
6: Definir $(c_1, c_2) \leftarrow c$
7: Escoger $r_1, r_2, r_3 \in_R Z_p$
8: Definir $T_1 \leftarrow g_1^{r_1}$
9: Definir $T_2 \leftarrow c_1^{r_2} g_1^{-r_3}$
10: Definir $T_3 \leftarrow e(c_2, g_2)^{r_2} \cdot e(h, w)^{-r_1} \cdot e(h, g_2)^{-r_3}$
11: Definir $z \leftarrow \mathcal{H}(m, c, T_1, T_2, T_3)$
12: Definir $t \leftarrow (za, z\alpha, za\alpha)$
13: Definir $s \leftarrow (r_1, r_2, r_3) + t$
14: Definir $\sigma \leftarrow (z, s)$
15: Retornar (c, σ)

- Verificar una firma (c, σ) creada por un miembro del grupo a partir del mensaje m:

Algoritmo 21 DistBBS.Verify(gpk, m, c, σ)

Autores: Kuykendall et al.[4]

Entrada:

gpk: clave pública del grupo.
m: mensaje firmado.
(c, σ): firma del mensaje m.

Salida:

True: si la firma corresponde al mensaje y al grupo.

1: Definir $(g_1, g_2, h, w) \leftarrow gpk$
2: Definir $(c_1, c_2) \leftarrow c$
3: Definir $(z, s) \leftarrow \sigma$
4: Definir $(s_1, s_2, s_3) \leftarrow s$
5: Definir $\tilde{T}_1 \leftarrow g_1^{s_1}$
6: Definir $\tilde{T}_2 \leftarrow c_1^{s_2} g_1^{-s_3}$
7: Definir $\tilde{T}_3 \leftarrow e(c_2, g_2)^{s_2} \cdot e(h, w)^{-s_1} \cdot e(h, g_2)^{-s_3}$
8: Definir $\tilde{t}_1 \leftarrow c_1^z$
9: Definir $\tilde{t}_2 \leftarrow 1$
10: Definir $\tilde{t}_3 \leftarrow \left(\frac{e(g_1, g_2)}{e(c_2, w)}\right)^z$
11: Definir $\tilde{r} \leftarrow \left(\frac{\tilde{T}_1}{\tilde{t}_1}, \frac{\tilde{T}_2}{\tilde{t}_2}, \frac{\tilde{T}_3}{\tilde{t}_3}\right)$
12: Definir $H \leftarrow \mathcal{H}(m, c, \tilde{r})$
13: **if** $z = H$ **then**
14: Retornar **True**
15: **else**
16: Retornar **False**
17: **end if**

Para ver que esta verificación es correcta basta con demostrar la igualdad entre los argumentos de la función de hash $\mathcal{H}$, es decir, que $(m, c, \tilde{r}) = (m, c, T_1, T_2, T_3)$

$$\frac{\tilde{T}_1}{\tilde{t}_1} = \frac{g_1^{s_1}}{c_1^z} = \frac{g_1^{r_1 + za}}{(g_1^a)^z} = \frac{g_1^{r_1} g_1^{za}}{g_1^{za}} = g_1^{r_1} = T_1 \tag{2.25}$$

$$\frac{\tilde{T}_2}{\tilde{t}_2} = \frac{c_1^{s_2}}{g_1^{s_3}} = \frac{c_1^{r_2 + z\alpha}}{g_1^{r_3 + za\alpha}} = \frac{c_1^{r_2} (g_1^a)^{z\alpha}}{g_1^{r_3} g_1^{za\alpha}} = \frac{c_1^{r_2}}{g_1^{r_3}} = T_2 \tag{2.26}$$

$$\frac{\tilde{T_3}}{\tilde{t_3}} = \frac{e(c_2, g_2)^{s_2}}{e(h, w)^{s_1} \cdot e(h, g_2)^{s_3}} \left(\frac{e(c_2, w)}{e(g_1, g_2)} \right)^z \tag{2.27}$$

$$= \frac{e(c_2, g_2)^{r_2 + z\alpha}}{e(h, w)^{r_1 + za} \cdot e(h, g_2)^{r_3 + za\alpha}} \left(\frac{e(c_2, w)}{e(g_1, g_2)} \right)^z \tag{2.28}$$

$$= T_3 \frac{e(c_2, g_2)^{z\alpha}}{e(h, w)^{za} \cdot e(h, g_2)^{za\alpha}} \left(\frac{e(c_2, w)}{e(g_1, g_2)} \right)^z \tag{2.29}$$

$$= T_3 \frac{e(h^a R, g_2)^{z\alpha} \cdot e(h^a R, w)^z}{e(h, w)^{za} \cdot e(h, g_2)^{za\alpha} \cdot e(g_1, g_2)^z} \tag{2.30}$$

$$= T_3 \frac{e(h, g_2)^{za\alpha} \cdot e(R, g_2)^{z\alpha} \cdot e(h, w)^{za} \cdot e(R, w)^z}{e(h, w)^{za} \cdot e(h, g_2)^{za\alpha} \cdot e(g_1, g_2)^z} \tag{2.31}$$

$$= T_3 \frac{e(g_1^{1/a}, g_2)^{z\alpha} \cdot e(g_1^{1/a}, g_2^{\gamma})^z}{e(g_1, g_2)^z} \tag{2.32}$$

$$= T_3 \frac{e(g_1, g_2)^{z\alpha/a + z\gamma/a}}{e(g_1, g_2)^z} \tag{2.33}$$

$$= T_3 \frac{e(g_1, g_2)^{z(\alpha+\gamma)/(\alpha+\gamma)}}{e(g_1, g_2)^z} \tag{2.34}$$

$$= T_3 \tag{2.35}$$

- Trazar la identidad de un firmante entre los usuarios del grupo

---

Algoritmo 22 DistBBS.Trace($gpk, \{msk_i\}_{k+1}, m, c, \sigma$)

Autores: Kuykendall et al.[4]

Entrada:

gpk: clave pública del grupo.
$\{msk_i\}_{k+1}$: claves privadas de $k+1$ administradores.
m: mensaje firmado.
(c, σ): firma del mensaje m.

Salida:

R: clave pública del firmante.

1: **for** Cada administrador M_i **do**
2: **if not** $DistBBS.Verify(gpk, m, c, \sigma)$ **then**
3: Retorna $\perp$
4: **end if**
5: Define $(\gamma_i, x_i) \leftarrow msk_i$
6: Define $(c_1, c_2) \leftarrow c$
7: Participa en $d \leftarrow SecShare.Exp(c_1, x_i)$
8: Define $R \leftarrow \frac{c_2}{d}$
9: Retorna R
10: **end for**

El procedimiento posterior a la validación es el algoritmo *ElGamal.Dec* adaptado a este

contexto distribuido:

$$\frac{c_2}{d} = \frac{h^a R}{c_1^x} = \frac{(g_1^x)^a R}{g_1^a x} = R \tag{2.36}$$

Capítulo 3

Problema

3.1. Descripción del Problema

Antes de describir el problema es necesario dar ciertas definiciones dado el contexto:

- Grupo: Conjunto de miembros registrados de la comunidad de la Universidad de Chile.
- Administrador: Miembro del grupo con privilegios de administrador, es decir, puede participar en el registro de miembros, asignación de privilegios y remoción de anonimato.
- Usuario: Miembro del grupo sin privilegios de administrador, es decir, sólo puede entrar al sistema y generar denuncias pseudo-anónimas.
- Autoridad: Subconjunto de administradores necesario para realizar una acción privilegiada.

El problema entonces es autentificar a un usuario y que este realice acciones en su sesión. De tal forma que sólo los administradores actuales, como conjunto, sean capaces de asociar a este usuario con dicha sesión.

3.2. Requisitos de la Solución

3.2.1. Autenticación

Solamente miembros del grupo pueden iniciar sesión en el sistema. La forma más sencilla de lograr esto es restringir la solicitud de claves para acceder al sistema a miembros de la comunidad que ya estén autenticados por un medio externo como "Pasaporte UChile".

3.2.2. Pseudo-Anonimato

Una vez que un usuario inicia sesión en el sistema, su sesión y toda actividad que realice en ella no debe poder ser asociada directamente a su identidad. Excepto por quién el sistema designa como autoridad.

3.2.3. Trazabilidad

La autoridad del sistema siempre debe ser capaz de trazar una sesión al usuario que la creó. En particular, un usuario autorizado no puede suplantar la identidad de otro.

3.2.4. Distribución

La autoridad del sistema no puede ser una entidad individual, sino que debe estar distribuida en múltiples personas. De lo contrario el sistema tendría un único punto de fallo, ya sea por un ataque informático o por corrupción.

3.2.5. Proactividad

Naturalmente un sistema que debe ser operado activa e indefinidamente por seres humanos debe cambiar su personal. Es por esto que el sistema debe contar con un mecanismo que permita reemplazar a las personas que componen la autoridad del sistema (administradores) sin comprometer el funcionamiento ni la seguridad del sistema.

Formalmente, dados $k+1$ administradores necesarios para tomar una acción en el sistema y periodos de tiempo t: La única forma de que un adversario vulnere el sistema es corrompiendo a $k+1$ en el mismo periodo de tiempo.

Capítulo 4

Solución

4.1. Descripción General de la Solución

La solución consiste en un sistema de autenticación anónima que consta de 3 partes principales: un cliente para gestionar la asignación de claves del esquema, un módulo que permita crear una sesión a partir de una clave del esquema y un servidor que gestione la comunicación entre los clientes.

Para esta solución se consideran dos tipos de agentes: administradores y usuarios.

- Los administradores como conjunto tienen la capacidad de inicializar el esquema, emitir claves de firma anónima para los usuarios, trazar la identidad de un usuario a partir de su firma y reemplazar a uno de sus miembros.
- Los usuarios pueden solicitar una clave de firma anónima y firmar.

La función del servidor es ser una entidad central que almacena la información pública del esquema y direccionar la comunicación entre los distintos agentes, anteriormente mencionados.

Todo dato enviado a través del servidor está encriptado de tal forma que sólo su receptor objetivo sea capaz de leerlo. Así se logra evitar que el servidor sea un único punto de fallo como lo sería tener un único administrador.

Tanto los dos tipos de agentes como el servidor tienen la capacidad de validar individualmente que una firma sea producto de un miembro del grupo.

4.2. Algoritmos

Cabe destacar que para los algoritmos presentes en la implementación de [3] se realizó un proceso de refactorización, ya que el prototipo de aquel trabajo dependía de una tercera parte confiable para las operaciones distribuidas en lugar de que cada administrador realice el computo internamente sin revelar información secreta.

El prototipo de este trabajo está pensado para ser usado en una implementación con clientes distribuidos reales, por lo que el único supuesto que hace, es que existe un servidor central encargado de direccionar paquetes encriptados entre los administradores y almacenar la información pública del esquema.

4.2.1. Inicialización

La inicialización del esquema es una versión más acotada de la propuesta de [4] y la implementación de [3], en esta etapa se registran los administradores y colaboran en la inicialización de los esquemas $DistBBS$ y $ElGamal$. Como resultado se tienen n administradores con sus claves privadas $msk_i^{(0)} = (\gamma_i^{(0)}, x_i^{(0)})$ y la clave pública del grupo gpk.

Se eliminó de esta etapa la generación de triples de Beaver, usadas para $SecShare.Mult()$, la inicialización de estructuras usadas para el manejo de acusaciones y la emisión de claves de usuarios.

Algoritmo 23 WhoTooPSS.Init(n, k)

Modificado a partir de [3]

Entrada:

$n \in \mathbb{N}$: número total de administradores.

$k \in \mathbb{N}$: número de administradores necesarios para realizar una acción.

Salida:

gpk: clave pública del grupo.

$\{msk_i^{(0)}\}_i$ claves privadas de los administradores para el periodo 0.

1: **for** Cada administrador M_i **do**
2: Se registra en el esquema
3: Participa en $(gpk, msk_i^{(0)}) \leftarrow DistBBS.Init(n, k)$
4: Publica gpk
5: **end for**

En la práctica se verifica que los valores de gpk ofrecidos por los administradores coincidan. En el caso que alguno difiera, se reporta al administrador y se detiene el proceso. En el caso que haya consenso, se guarda en la base de datos y queda disponible para cualquiera que desee consultarlo.

4.2.2. Emisión

Este algoritmo se desprende del algoritmo de inicialización de [3]. En este algoritmo los administradores y un usuario colaboran para otorgarle un par de claves al usuario asociando su clave pública a su identidad.

Como se ve en la definición de $DistBBS.Issue$ en 2.4, esta etapa requiere el uso de $SecShare.Invert$ y a su vez $SecShare.Mult$ que utiliza triples de Beaver. Para un funcionamiento más eficiente de este esquema, se mantiene una pila con triples de Beaver que debe ser rellenada en esta etapa si se encuentra vacía.

Al terminar este proceso el usuario U tiene su clave privada α y su clave pública R es conocida por todos los administradores.

Algoritmo 24 WhoTooPSS.Issue($gpk, \{msk_i^{(t)}\}_{k+1}$)

Modificado a partir de [3]

Entrada:

gpk: clave pública del grupo.

$\{msk_i^{(t)}\}_{k+1}$: claves privadas de $k+1$ administradores en el periodo actual.

Salida:

usk: par de claves del usuario U.

1: **if** Pila de triples de Beaver $= \varnothing$ **then**
2: **for** Cada administrador M_i **do**
3: Escoge $a, b \in_R \mathbb{Z}_p$
4: Define $c \leftarrow ab$
5: Comparte $SecShare.Share(a)$
6: Comparte $SecShare.Share(b)$
7: Comparte $SecShare.Share(c)$
8: **end for**
9: **end if**
10: **for** Cada administrador M_i **do**
11: Participa en $(R, \alpha_i) \leftarrow DistBBS.Issue(gpk, \{msk_i\}_{k+1})$ (parte de administradores)
12: Envía (R, α_i) al usuario U
13: Publica R como identidad de U
14: **end for**

15: `El usuario` U:
16: Participa en $usk \leftarrow DisBBS.Issue()$ (parte de usuario)

En la práctica una vez que se tiene consenso en el valor de R se almacena en la base de datos como un atributo de la entidad asociada al usuario U.

4.2.3. Firma

El algoritmo de firma, al no ser distribuido es idéntico al presentado por [3]. Este algoritmo le permite al usuario U generar una firma (c, σ) que puede ser verificada como proveniente del grupo por cualquiera y como proveniente de U sólo por los administradores.

Algoritmo 25 WhoTooPSS.Sign(gpk, usk, m)

Autora: Mergudich[3]

Entrada:

gpk: clave pública del grupo.

usk: par de claves del usuario U.

m: mensaje a firmar.

Salida:

(c, σ): firma del mensaje m.

1: Definir $(c, \sigma) \leftarrow DistBBS.Sign()$

4.2.4. Verificación

Al igual que la firma, la verificación no es distribuida, por lo que no sufre ningún cambio. Este algoritmo le permite a cualquiera verificar que una firma fue generada por un miembro del grupo.

Algoritmo 26 WhoTooPSS.Verify(gpk, m, c, σ)

Autora: Mergudich[3]

Entrada:

gpk: clave pública del grupo.
m: mensaje firmado.
(c, σ): firma del mensaje m.

Salida:

True si la firma corresponde al mensaje y al grupo.

1: Retornar $DistBBS.Verify(gpk, m, c, \sigma)$

4.2.5. Rastreo

El algoritmo de rastreo (en inglés *tracing*) es una versión distribuida y simplificada de $WhoToo.OpenAccusations$ implementado en [3], que es conceptualmente idéntico a como se describió $DistBBS.Trace$ en la sección 2.4.

Algoritmo 27 WhoTooPSS.Trace($gpk, \{msk_i^{(t)}\}_{k+1}, m, c, \sigma$)

Modificado a partir de [3]

Entrada:

gpk: clave pública del grupo.
$\{msk_i^{(t)}\}_{k+1}$: claves privadas de $k+1$ administradores en el periodo actual.
m: mensaje firmado.
(c, σ): firma del mensaje m.

Salida:

R: clave pública del firmante.

1: **for** Cada administrador M_i **do**
2: Participa en $R \leftarrow DistBBS.Trace(gpk, \{msk_i\}_{k+1}, m, c, \sigma)$
3: **end for**

4.2.6. Recuperación

La implementación del algoritmo de recuperación está basada en la propuesta de [11] y le permite a los administradores asistir al administrador M_r a recuperar el trozo de msk asociado al índice r.

Algoritmo 28 WhoTooPSS.Recover(r)

Modificado a partir de [11]

Entrada:
r: índice para el cual se busca recuperar la clave privada.
Salida:
(γ_r, x_r): clave privada recuperada.
1: **for** Cada administrador M_i distinto de M_r **do**
2: Participa en $\gamma_i' \leftarrow SecShare.Recover(r)$ respecto a γ (primera parte)
3: Participa en $x_i' \leftarrow SecShare.Recover(r)$ respecto a x (primera parte)
4: Envía (γ_i', x_i') a M_r
5: **end for**

6: `El administrador` M_r:
7: Reconstruye (γ_r, x_r) como en $SecShare.Recover$

4.2.7. Renovación

Al igual que $WhoTooPSS.Recover$, esta implementación está basada en [11]. Este algoritmo le permite a los administradores cambiar sus claves privadas sin modificar el secreto msk y por tanto sin modificar el resto del esquema.

Algoritmo 29 WhoTooPSS.Update()

Modificado a partir de [11]

Entrada:
$msk_i^{(t-1)}$: clave privada del administrador en el periodo actual.
Salida:
$msk_i^{(t)}$: clave privada del administrador en el periodo siguiente.
1: **for** Cada administrador M_i **do**
2: Participa en $\gamma_i^{(t)} \leftarrow SecShare.Update()$ respecto a γ
3: Participa en $x_i^{(t)} \leftarrow SecShare.Update()$ respecto a x
4: Define $msk_i^{(t)} \leftarrow (\gamma_i^{(t)}, x_i^{(t)})$
5: **end for**

4.2.8. Casos de Uso

4.2.8.1. Inicio de sesión

Para iniciar sesión de manera anónima el servidor debe crear un desafío aleatorio (un número al azar) m y el usuario debe firmarlo. Si la firma producida por el usuario es verificable, entonces el usuario es un miembro del grupo, por lo que se le debe permitir acceso.

El servidor debe almacenar tanto el desafío como la firma del mismo, asociados a la sesión, para permitir trazabilidad cuando sea necesario.

Algoritmo 30 WhoTooPSS.Login()

El usuario U:
Solicita iniciar sesión

El sistema:
Genera la sesión asociada a $m \leftarrow RandStr()$
Envía m al usuario U

El usuario U:
Define $(c, \sigma) \leftarrow WhoTooPSS.Sign(gpk, usk, m)$
Envía (c, σ) como respuesta

El sistema:
if $WhoTooPSS.Verify(gpk, m, c, \sigma)$ **then**
    Permite a U ingresar a la sesión m
    Guarda (m, c, σ)
else
    Rechaza el inicio de sesión y reporta a U
end if

4.2.8.2. Rastreo de sesión

Cuando $k+1$ administradores están de acuerdo en que se debe rastrear al responsable de una sesión en particular, les basta con rastrear usando los atributos (m, c, σ) de la sesión.

Algoritmo 31 WhoTooPSS.TraceSession(m, c, σ)

Entrada:
m: desafío aleatorio asociado a la sesión.
(c, σ): firma del desafío.
for Cada administrador M_i **do**
    Participa en $R \leftarrow WhoTooPSS.Trace(gpk, m, c, \sigma)$
    Busca usuario asociado a R
end for

4.2.8.3. Remplazo de administrador

Para remplazar al administrador asociado al índice r de msk, por un administrador entrante M_e, basta con modificar el algoritmo de recuperación, de tal forma que a pesar de recuperar el trozo de M_r el proceso se realice con M_e.

Luego para evitar que el administrador reemplazado M_r sea capaz de interactuar con el sistema como administrador, se lo comienza a considerar como un adversario que conoce un trozo del secreto, por tanto, se deben renovar los trozos al paso de tiempo siguiente.

Algoritmo 32 WhoTooPSS.Replace(r)

Entrada:

r: índice del administrador que será reemplazado.

1: **for** Cada administrador M_i distinto de M_r y M_e **do**
2: Participa en $(\gamma_i', x_i') \leftarrow WhoTooPSS.Recover(r)$
3: Envía $(\gamma_i' x_i')$ a M_e
4: **end for**

5: `El administrador entrante` M_e:
6: Reconstruye $msk_r^{(t-1)}$ como en $WhoTooPSS.Recover$
7: Define $msk_e^{(t-1)} \leftarrow msk_r^{(t-1)}$

8: **for** Cada administrador M_i distinto de M_r **do**
9: Participa en $msk_i^{(t)} \leftarrow WhoTooPSS.Update()$
10: **end for**

4.3. Implementación

4.3.1. Optimización

Como se mencionó en la descripción de los algoritmos 4.2, para esta implementación de la librería fue necesario refactorizar el código de tal forma que los protocolos distribuidos pudieran ser ejecutados por cada cliente del sistema de manera distribuida.

Adicionalmente a la librería, se inició el desarrollo de lo implementación en la forma de una aplicación web en `Flask`, para esto fue necesario la creación de una base de datos, una API REST, una aplicación del lado del cliente y definir cómo codificar los tipos de dato usados por la librería (`paring.element`) en un formato compatible con `json` y `SQL` (`Base64 string`) y vice versa.

Los algoritmos distribuidos están divididos en métodos internos de los clientes y comunicaciones con el servidor. Es importante recalcar que para evitar que el servidor sea una tercera parte confiable todos los paquetes enviados a través del servidor están encriptados con la clave pública de encriptación del destinatario objetivo, y las claves públicas de encriptación están almacenadas en la base de datos, representadas como `mgr_pk` a continuación:

- En la inicialización los administradores definen las claves públicas de ElGamal (h) y BBS (w) en la base datos y sus trozos de las claves privadas (x_i, γ_i) internamente.

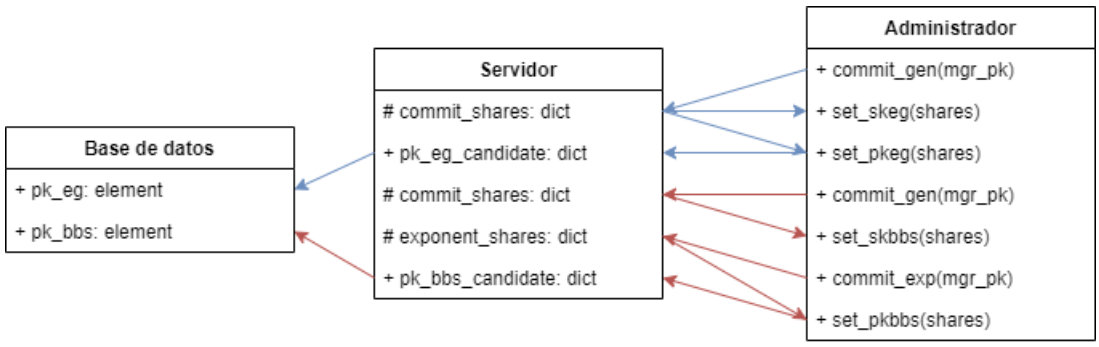


Figura 4.1: Inicialización.

((h, x) de ElGamal en azul, (w, γ) de BBS en rojo)

- En la emisión los administradores definen la identidad pública del usuario (R) en la base de datos y le entregan al usuario los trozos para determinar su clave privada (α)

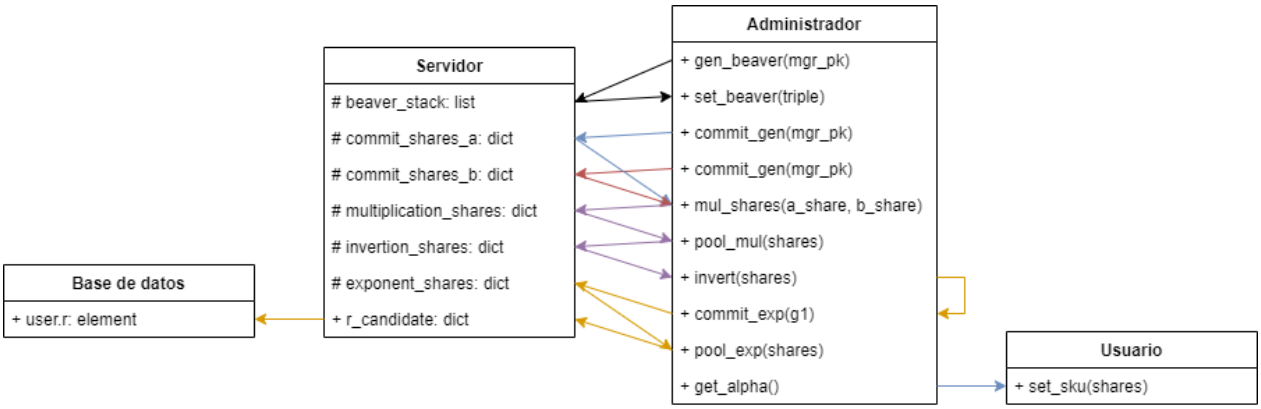


Figura 4.2: Emisión.

(a en azul, b en rojo, ab en morado, R en amarillo)

- En el rastreo los administradores colaboran para calcular c_1^x, luego pueden calcular independientemente $R = c_2/c_1^x$ y consultar al servidor qué usuario está asociado a R

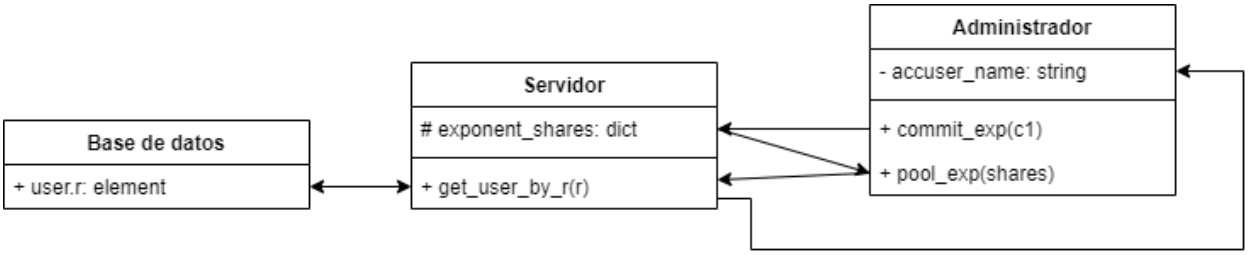


Figura 4.3: Rastreo.

- En la recuperación los administradores colaboran para recuperar el trozo de msk asociado a un índice r.

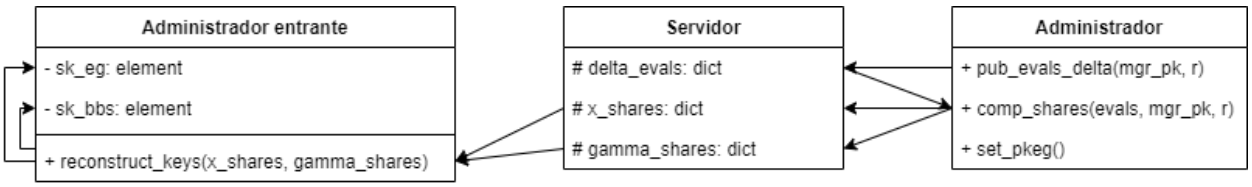


Figura 4.4: Recuperación.

- En la renovación los administradores colaboran para cambiar sus trozos de msk sin alterar el secreto.

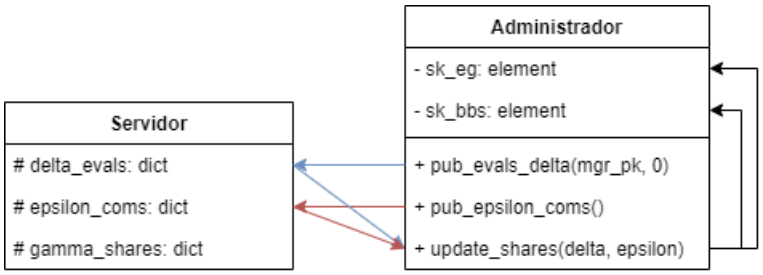


Figura 4.5: Renovación.

4.3.2. Arquitectura

El sistema consta de un servidor y un cliente.

El servidor se encarga de almacenar toda la información pública del esquema y los datos de inicio de sesión de los distintos clientes, además de ser un intermediario en los protocolos distribuidos.

Por otro lado, el cliente se encarga de el cómputo descrito en los algoritmos de la librería y de almacenar en un archivo local la información privada del agente, ya sea un administrador o un usuario común.

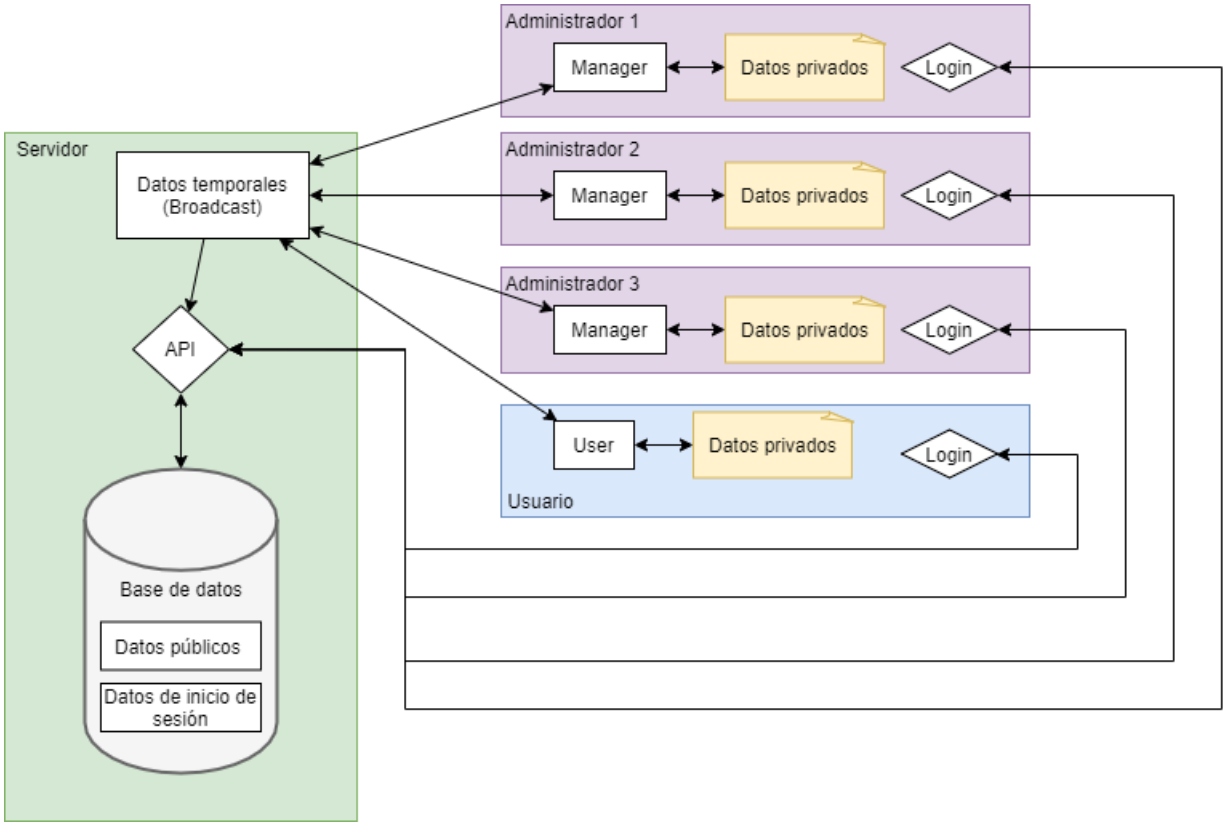


Figura 4.6: Arquitectura.

El modelo de datos considera un tablero con los datos públicos del sistema y agentes con sus datos públicos, rol e información de inicio de sesión.

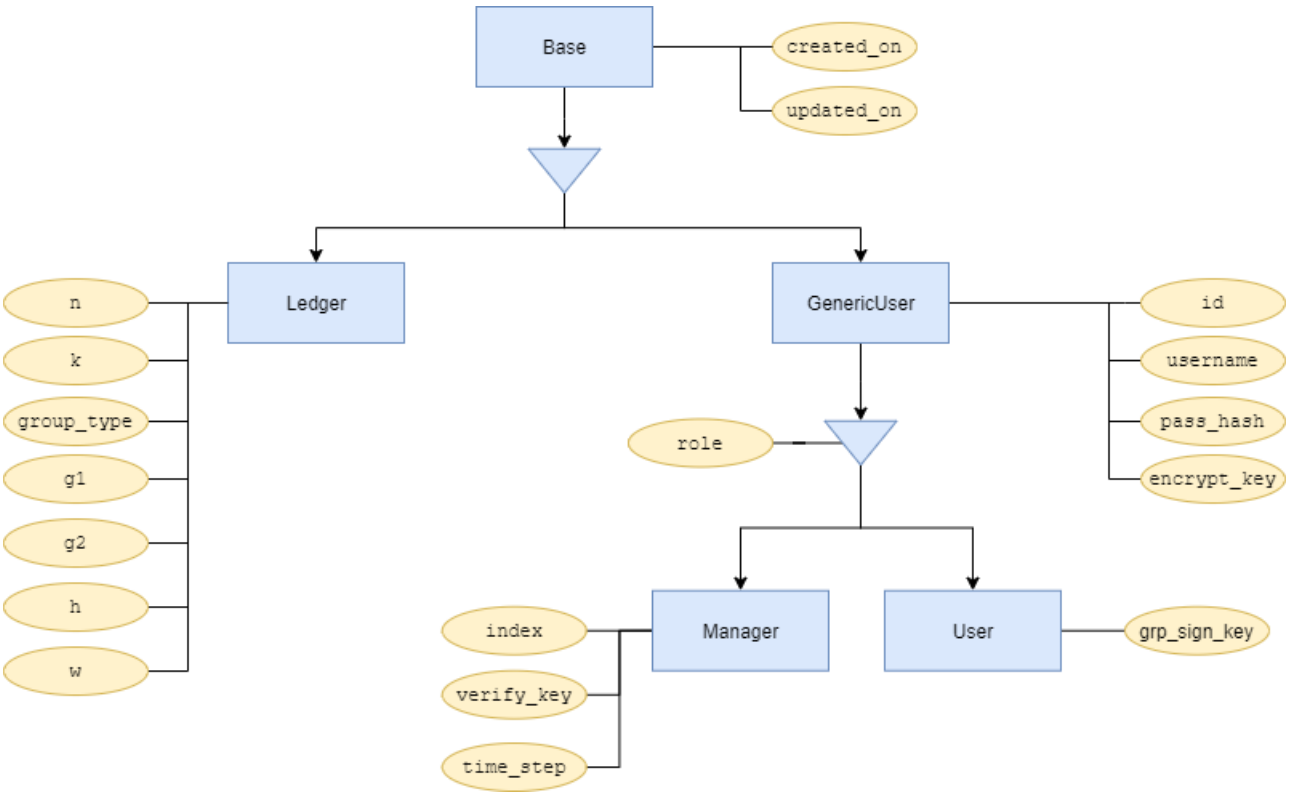


Figura 4.7: Modelo de datos.

4.3.3. Interfaces

El cliente cuenta con 4 interfaces:

- Registro: Permite crear una cuenta como administrador o usuario en el sistema de generación de claves.

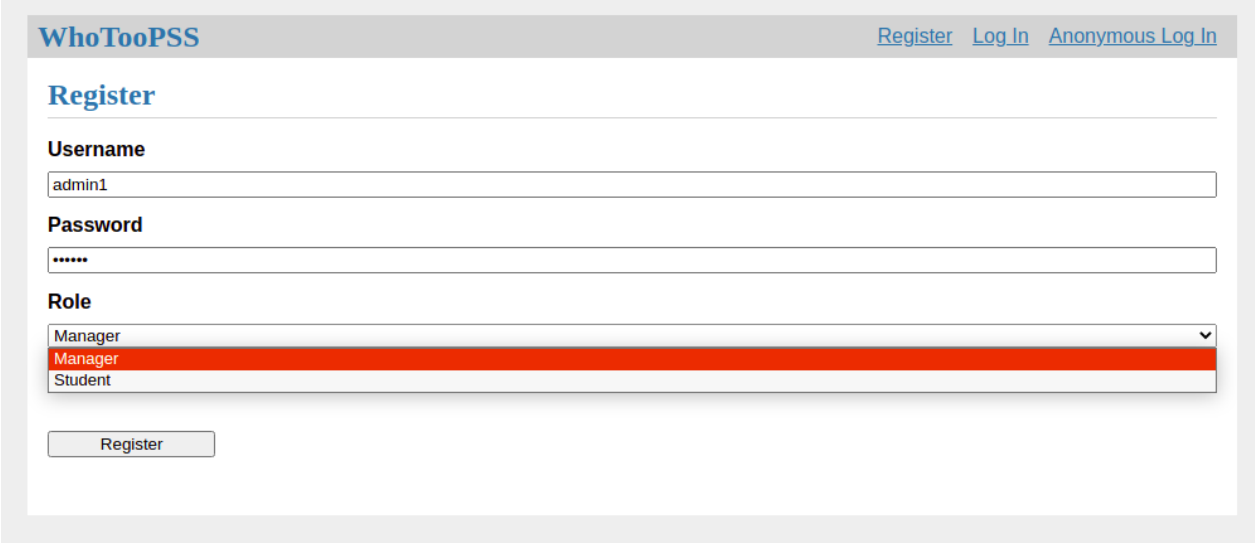


Figura 4.8: Registro.

- Inicio de sesión tradicional: Permite iniciar sesión en el sistema de generación de claves

Figura 4.9: Inicio de sesión.

- Acciones: Permite ejecutar las distintas etapas requeridas para los procesos distribuidos. Cuenta con vistas distintas, una para administradores y una para usuarios, ya que estos pueden realizar acciones distintas.

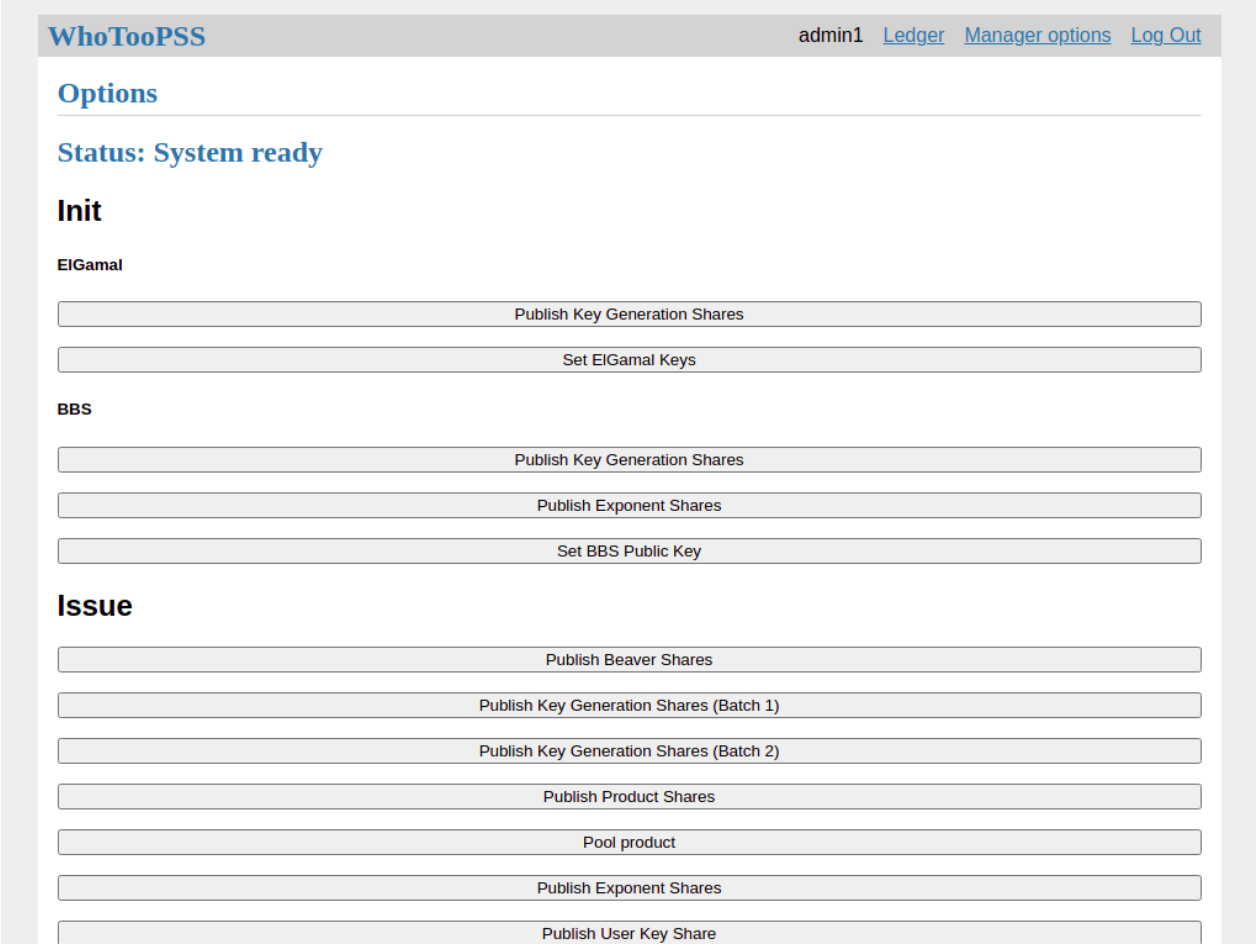


Figura 4.10: Acciones de administrador.

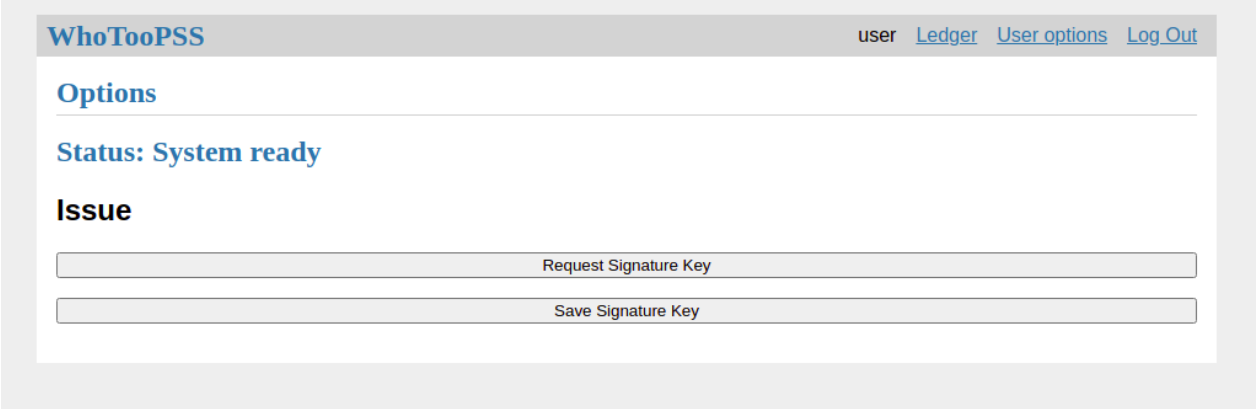


Figura 4.11: Acciones de usuario.

- Tablero de información pública: Le permite tanto a administradores como usuarios ver la información pública del esquema y de todos los administradores. En estricto rigor esto no es necesario ya que esto se maneja internamente, pero se agregó por transparencia.

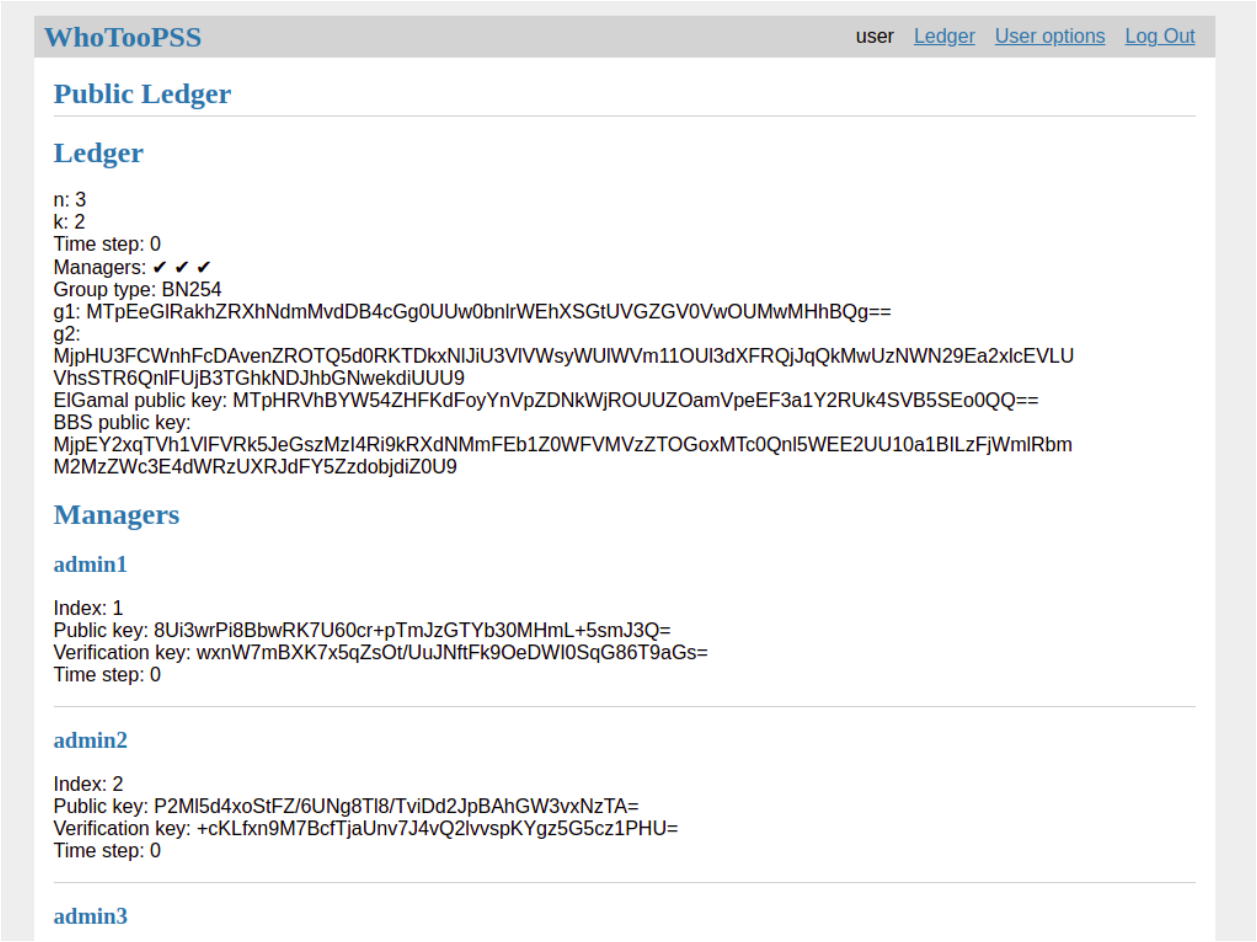


Figura 4.12: Información pública.

Capítulo 5

Discusión

5.1. Trabajo Propuesto

5.1.1. Desarrollo de la Aplicación Web

En la propuesta de este trabajo se mencionó la implementación de un prototipo. Si bien se logró implementar la librería con los algoritmos descritos en el capítulo anterior y llevarlos a un *script* que cumple con las propiedades necesarias, no sé logró concretar el desarrollo de una aplicación web, que hubiera sido un prototipo más sofisticado y cercano al uso real.

Dado el tiempo que fue necesario invertir en la librería, el tema central de este trabajo, y las complejidades propias de diseñar y gestionar apropiadamente un protocolo con clientes distribuidos, esta versión del prototipo sólo se logró implementar hasta el registro de administradores, creación de clave pública del grupo y creación de claves privada de cada administrador.

Se propone como trabajo a futuro extender la funcionalidad del prototipo para ofrecer las características del sistema completo:

- Asignar firmas a los usuarios.
- Permitir a los usuarios iniciar sesión mediante la firma de un desafío aleatorio.
- Que los administradores sean capaces de rastrear una sesión una vez que se alcance el quórum $k + 1$.
- Integrar la autenticación para generación y solicitud de claves con el sistema Mi.Uchile.

Además sería deseable llevar estas claves a un uso práctico cómo el de denuncias anónimas discutido en este trabajo.

5.1.2. Evaluaciones para Usabilidad

En suma a los aspectos funcionales del sistema, una puesta en producción requiere evaluar y experimentar respecto a las siguientes interrogantes:

- ¿Cuántos recursos se necesitarían para escalar la operación a toda la comunidad universitaria?
- ¿Qué miembros de la comunidad serían elegibles para ser administradores en el esquema?

- ¿Qué modificaciones a la interfaz serían necesarias para que un usuario no experimentado pueda utilizar el sistema?

5.1.3. Ataques a Proactividad

En la literatura se ha analizado la seguridad de compartición de secretos proactiva, en [13] se definen tres tipos de ataque sobre un esquema de compartición de secretos verificable proactivo y luego se concluye que es imposible modificar el esquema de tal forma que sea resistente a más de dos de ellos.

Es necesario evaluar cual de estos ataques es el menos costoso para el sistema en la práctica y tomar los resguardos necesarios para evitar los otros dos de manera criptográfica y el primero de mitigarlo lo más posible.

Estos ataques a grandes rasgos consisten en:

1. Si el adversario conoce u $(1 \leq u \leq k)$ trozos del secreto, correspondientes al subgrupo de miembros B, en el periodo $(t-1)$ y durante la etapa de renovación $SecShare.Update()$ es capaz de corromper k miembros del esquema fuera de B, entonces es capaz de renovar los u trozos que ya conoce. Esto es posible porque, a pesar de que cada δ es de orden k, es sabido que $\delta(0) = 0$, lo que le permite al adversario reconstruir todos los δ y los trozos de los u miembros que corrompió en $(t-1)$.

   Luego, no se tiene proactividad, porque al adversario le bastó con corromper $u < (k+1)$ miembros en $(t-1)$ y $k < (k+1)$ en t para obtener $(k+u) \geq (k+1)$ trozos que le permiten reconstruir el secreto.

   Para evitar este ataque se propone usar el esquema VSS proactivo con seguridad incondicional de Stinson y Wei [14] (SW). Este codifica el secreto mediante un polinomio simétrico bidimensional $f^{(t)}(x, y)$ en que el i-ésimo trozo es el polinomio $h_i^{(t)}(x) = f^{(t)}(x, i)$ y el secreto es $f^{(t)}(0, 0) = s$. Similarmente reemplaza los polinomios unidimensionales δ por polinomios simétricos bidimensionales tales que $\delta(0, 0) = 0$.

2. Para renovar los trozos, el esquema SW cada miembro U_i le envía el polinomio $\delta_{i,j}(x) = \delta_i(x, j)$ a casa miembro U_j y publica $\delta_{i,0}(x)$ como compromiso.

   Si el adversario conoce el trozo correspondiente al miembro U_j en el periodo $(t-1)$. Puede usar los compromisos $\delta_{i,0}(x)$ para renovar parcialmente el trozo de la siguiente manera:

   $\delta_{i,0}(j) = \delta_{i,j}(0)$, luego $h_j^{(t)}(0) = h_j^{(t-1)}(0) + \Sigma_i \delta_{i,j}(0)$

   Si bien esto no le permite renovar todo el polinomio $h_j(x)$ correspondiente al trozo, si le permite renovar $h_j(0) = f(0, j)$ que es todo lo que necesitaría para recuperar el secreto $s = f(0, 0)$ si llega a conseguir los trozos de otros k miembros. Por lo tanto al adversario le basta con corromper un miembro por paso de tiempo en $k+1$ pasos de tiempo para reconstruir el secreto y nuevamente no se tiene proactividad.

   Para evitar este ataque se propone usar el esquema VSS proactivo con seguridad incondicional de D'Arco y Stinson [15] (DS) que reemplaza $f^{(t)}(x, y)$ por un polinomio asimétrico en que el i-ésimo trozo corresponde a los polinomios $h_i^{(t)}(x) = f^{(t)}(x, i)$ y $g_i^{(t)}(y) = f^{(t)}(i, y)$.

3. Similar a SW, para renovar trozos en el esquema DS cada miembro U_i le envía los

polinomios $h_{i,j}(x) = \delta_i(x, j)$ y $g_{i,j}(y) = \delta_i(j, y)$ al miembro U_j y publica $h_{i,0}(x)$ como compromiso.

Similarmente al ataque anterior, si el trozo correspondiente al miembro U_j en el periodo $(t-1)$. Puede usar los compromisos $h_{i,0}(x)$ para renovar parcialmente el trozo:

$h_{i,0}(j) = g_{i,j}(0)$, luego $g_j^{(t)}(0) = g_j^{(t-1)}(0) + \Sigma_i g_{i,j}(0)$

Esto le permite al adversario renovar $g_j(0) = f(j, 0)$. Luego al repetir este proceso, un miembro a la vez en $k+1$ periodos puede reconstruir el secreto $s = f(0, 0)$ quebrantado nuevamente la proactividad.

Para facilitar la implementación se optó por la primera opción. Esto con el beneficio adicional de dificultar un ataque del adversario. Puesto que, para las opciones 2 y 3, basta con corromper a un administrador para conocer el secreto, mientras que con la opción escogida es necesario corromper k administradores, un número similar a $k+1$ para grandes valores de k.

5.2. Investigación Previa no Usada en la Solución

5.2.1. Firmas de Anillo

Las firmas de anillo son otro esquema que permite generar una firma a nombre de un grupo, ocultando la identidad del firmante. Este difiere de firmas de grupo en dos aspectos:

- No se requiere un administrador, el esquema es inicializado por los mismos usuarios.
- Las firmas de un mismo mensaje son indistinguibles, por lo que es imposible lograr trazabilidad.

Si bien este tipo de esquema soluciona el problema de distribución, la imposibilidad de trazabilidad hace que no sea adecuado para la solución.

5.2.2. Otros Esquemas de Firma de Grupo

Junto con el esquema de BBS se evaluaron otros esquemas de firma de grupo:

- Díaz-Lehmann 2021 [16]
- Garms-Lehmann 2019 [17]
- Pointcheval-Sanders 2016 [18]

Sin embargo, estos no fueron incluidos en la solución debido a que no se encontraban implementados con código abierto o su extensión para incorporar VSS proactivo imponía un desafío teórico superior.

Capítulo 6

Conclusiones

6.1. Revisión de Propiedades

Se puede ver que todas las propiedades que requiere la solución se cumplen a partir de las propiedades de los esquemas en los que se basan:

- Autentificación: Para iniciar sesión en el sistema es necesario ser capaz de generar una firma válida del grupo, como para solicitar una clave de firma hay que autentificarse como miembro de la comunidad, entonces los que pueden producir firmas válidas son miembros auténticos.
- Pseudo-anonimato: La única forma de conocer la identidad del autor de una sesión es trazándola a partir de la firma asociada a dicha sesión. Como esta es una firma de grupo $DistBBS$, sólo un conjunto de $k+1$ administradores del grupo son capaces de conocer la identidad del usuario.
- Trazabilidad: Al igual que en el caso del pseudo-anonimato, trazar una sesión es equivalente a trazar una firma $DistBSS$, por lo que se requiere un conjunto de $k+1$ administradores.
- Distribución: En todo momento la clave de administración msk y los procesos que llevan a cabo los administradores están distribuidos, nunca un administrador tiene acceso a la información privada de otro y la comunicación entre administradores es encriptada.
- Proactividad: La proactividad de los administradores se desprende directamente de la propuesta de [11], por lo tanto es vulnerable al primer ataque de [13].

  Cabe destacar que esto significa que no es seguro para un adversario que corrompe k administradores durante el proceso de renovación. Para el propósito que se está usando proactividad en este sistema, es decir, excluir un ex-administrador, la única forma de ejecutar este ataque es que el ex-administrador se coluda con k administradores que permanecen en el sistema.

6.2. Revisión de Objetivos

Se completaron todos los objetivos propuestos al principio de este trabajo.

Se logró desarrollar una librería que implementa la funcionalidad de $DistBBS$[12], basándose en desarrollo de WhoTooPlus [3]. Se logró incluir proactividad, como es descrita en [11]

y al momento de la elaboración de este informe existe un prototipo funcional en forma de *script*, mas no como aplicación Web.

6.3. Requisitos para Producción

Para lograr un sistema como el descrito en este trabajo que este apto para entrar en producción no basta con atender las extensiones necesarias y evaluar como mitigar las vulnerabilidades de proactividad expuestas en la sección 5.1.

La universidad debe contar con un organismo responsable de la mantención del sistema, la selección de administradores y la asignación de claves de encriptación para todos los miembros de la comunidad.

Además se debe instruir a los usuarios del sistema a iniciar sesión de manera anónima ocultando su dirección IP, por ejemplo mediante TOR o algún servicio de VPN.

Bibliografía

[1] Rivest, R. L., Shamir, A., y Tauman, Y., "How to leak a secret," en International Conference on the Theory and Application of Cryptology and Information Security, pp. 552–565, Springer, 2001.

[2] Bellare, M., Micciancio, D., y Warinschi, B., "Foundations of group signatures: Formal definitions, simplified requirements, and a construction based on general assumptions," en International conference on the theory and applications of cryptographic techniques, pp. 614–629, Springer, 2003.

[3] Mergudich, I., "Implementing secure reporting of sexual misconduct," Master's thesis, Universidad de Chile, 2021.

[4] Kuykendall, B., Krawczyk, H., y Rabin, T., "Cryptography for# metoo.," Proc. Priv. Enhancing Technol., vol. 2019, no. 3, pp. 409–429, 2019.

[5] Gómez, C., "Diseño e implementación de un sistema de votación electrónica modular y dual, con verificación del votante," Master's thesis, Universidad de Chile, 2015.

[6] Jara, M., "Ofuscación de permutaciones en mixnets," Master's thesis, Universidad de Chile, 2012.

[7] ElGamal, T., "A public key cryptosystem and a signature scheme based on discrete logarithms," IEEE transactions on information theory, vol. 31, no. 4, pp. 469–472, 1985.

[8] Boneh, D., Boyen, X., y Shacham, H., "Short group signatures," en Annual international cryptology conference, pp. 41–55, Springer, 2004.

[9] Pedersen, T. P., "Non-interactive and information-theoretic secure verifiable secret sharing," en Annual international cryptology conference, pp. 129–140, Springer, 1991.

[10] Shamir, A., "How to share a secret," Communications of the ACM, vol. 22, no. 11, pp. 612–613, 1979.

[11] Herzberg, A., Jarecki, S., Krawczyk, H., y Yung, M., "Proactive secret sharing or: How to cope with perpetual leakage," en annual international cryptology conference, pp. 339–352, Springer, 1995.

[12] Blömer, J., Juhnke, J., y Löken, N., "Short group signatures with distributed traceability," en International Conference on Mathematical Aspects of Computer and Information Sciences, pp. 166–180, Springer, 2015.

[13] Nikov, V. y Nikova, S., "On proactive secret sharing schemes," en International Workshop on Selected Areas in Cryptography, pp. 308–325, Springer, 2004.

[14] Stinson, D. R. y Wei, R., "Unconditionally secure proactive secret sharing scheme with

combinatorial structures," en International Workshop on Selected Areas in Cryptography, pp. 200–214, Springer, 1999.

[15] D'Arco, P. y Stinson, D., "On unconditionally secure proactive secret sharing scheme and distributed key distribution centers," Manuscript, May, 2002.

[16] Diaz, J. y Lehmann, A., "Group signatures with user-controlled and sequential linkability," en IACR International Conference on Public-Key Cryptography, pp. 360–388, Springer, 2021.

[17] Garms, L. y Lehmann, A., "Group signatures with selective linkability," en IACR International Workshop on Public Key Cryptography, pp. 190–220, Springer, 2019.

[18] Pointcheval, D. y Sanders, O., "Short randomizable signatures," en Cryptographers' Track at the RSA Conference, pp. 111–126, Springer, 2016.