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

OFUSCACIÓN DE PERMUTACIONES EN MIXNETS

MEMORIA PARA OPTAR AL TÍTULO DE INGENIERO CIVIL EN COMPUTACIÓN

MARIO ALEJANDRO JARA RODRÍGUEZ

PROFESOR GUÍA:
ALEJANDRO HEVIA ÁNGULO

MIEMBROS DE LA COMISIÓN:
GONZALO NAVARRO BADINO
MAURICIO PALMA LIZANA

SANTIAGO DE CHILE
NOVIEMBRE 2012

Resumen

Una red de mezcla (o mixnet) es una construcción criptográfica que consiste de una serie de servidores de mezcla que reciben mensajes encriptados, los permutan y luego los reencriptan antes de enviarlos al próximo servidor (o a publicación si es el último servidor). Esto permite enviar mensajes de forma anónima, puesto que para trazar el recorrido de un mensaje en una mixnet hay que conocer la permutación que se realizó en cada uno de los servidores de mezcla.

Esto abre la pregunta: ¿y si se interviniesen todos los servidores de mezcla para extraer las permutaciones? Actualmente ninguna construcción de mixnet ofrece garantías de anonimato ante un adversario pasivo con todos los servidores de mezcla. Esto es lógico pues si el adversario puede ver todo lo que los servidores intervenidos realizan, conoce la permutación que cada uno de ellos aplicó, negando la utilidad de la red de mezcla. Pero, ¿y si los propios servidores no conocieran la permutación que realizan? Entonces aunque los servidores estuvieran intervenidos, el adversario no podría hacer el rastreo. Pero ¿cómo se logra ocultar la permutación al propio servidor que la realiza? Es aquí donde entra el concepto de ofuscación criptográfica.

A grandes rasgos, ofuscar es crear un nuevo programa a partir del programa original tal que la información secreta obtenida a partir del código fuente del programa no es mayor que la producida al simplemente ejecutar el programa como caja negra, teniendo acceso sólo a las entradas y salidas de éste. Así, si se esconde la permutación dentro de un programa ofuscado, ni el propio servidor sabría qué permutacion esta aplicando.

En este trabajo se muestra que una ofuscación de esta naturaleza es posible de realizar, mostrando una construcción genérica de red de mezcla que utiliza ofuscación para esconder el proceso de permutación y re-encriptación dentro de cada servidor de mezcla. Además, a fin de mejorar la eficiencia de la construcción genérica, se propone como prueba de concepto implementable una construcción específica restringida a dos mensajes y un servidor de mezcla. Finalmente, para ambas construcciones se demuestra la seguridad de la ofuscación y la propiedad de anonimato en redes de mezcla.

Agradecimientos

A mi familia, por estar siempre apoyándome, en especial a mi hermano Fabián, que me ayudó en los momentos más críticos de este trabajo.

A mis amigos, por existir, estar ahí y ser mis amigos, en especial a Sebastián Sepúlveda, que me ayudó cuando me sentía perdido.

A mi profesor guía, Alejandro Hevia, por sus sabios consejos y por haberme estado guiando durante este trabajo. Sin él este trabajo no sería posible.

A todo aquél que me haya ayudado, aunque sea incidentalmente, y no lo haya nombrado: gracias.

Índice

1. Introducción

Una red de mezcla (o mixnet) es una construcción criptográfica que consiste de una serie de servidores de mezcla que reciben mensajes encriptados, los permutan y luego los reencriptan antes de enviarlos al próximo servidor (o a publicación si es el último servidor). Esto permite enviar mensajes de forma anónima, puesto que para trazar el recorrido de un mensaje en una mixnet hay que conocer la permutación que se realizó en cada uno de los servidores de mezcla. El concepto fue introducido por Chaum[5] en 1981.

Hay dos tipos de redes de mezcla, las conocidas como redes clásicas y las redes reencriptantes. Las redes de mezcla clásicas corresponden a aquellas que siguen el modelo ideado por Chaum, donde al pasar por el conjunto de servidores de mezcla, los mensajes van desechando capas de encriptación hasta que al finalizar quedan desencriptados. Por otra parte, en las redes de mezcla reencriptantes, los mensajes van con una única capa de encriptación, que a medida que va pasando por los servidores de mezcla va siendo reencriptado, para al final pasar por un proceso de desencriptación.

Uno de los usos más populares de las redes de mezcla es votación electrónica, pues es natural que cada votante envíe su mensaje (voto) de manera encriptada y anónima.

Las redes de mezcla aseguran anonimato mientras las permutaciones realizadas en cada uno de los servidores de mezcla permanezca secreta, puesto que si un adversario desea rastrear uno o más mensajes dentro de la mixnet tiene que conocer todas las permutaciones aplicadas, lo que es imposible a menos que intervenga todos los servidores de mazcla para extraer las permutaciones. Sin embargo, si llegase a ocurrir que un adversario tomara control de todos los servidores de mezcla, ninguna de las construcciones actuales de redes de mezcla podría seguir asegurando anonimato, puesto que el adversario tiene acceso a los servidores y éstos, a su vez, tienen acceso a la permutación que aplicaron. Pero, ¿y si ni los propios servidores conocieran la permutación que realizaron? El adversario por mucho que tenga acceso a todos los servidores de mezcla no podría hacer el rastreo, puesto que éstos no tendrían acceso a la permutación que realizaron. Pero, ¿cómo se logra ocultar la permutación al propio servidor que la realiza? Es aquí donde entra el concepto de ofuscación.

Ofuscar un programa tiene muchas definiciones, pero a rasgos generales es crear un nuevo programa a partir del programa original tal que programa ofuscado revele lo mínimo de información posible, y más específicamente que ofusque todo tipo de claves secretas que se requieran para el funcionamiento del programa. De esta manera, si parte de la información secreta de un programa corresponde a una permutación, al ser ofuscado no revelaría información sobre ésta, ni siquiera al propio servidor que lo está ejecutando.

Por tanto, el objetivo de este trabajo es mostrar la posibilidad de crear un programa ofuscado que se encargue del proceso de permutación y re-encriptación, creando una red de mezcla genérica que utiliza ofuscación en los servidores de mezcla de manera que información que eventualmente sirva para romper el anonimato de la mixnet no pueda ser recuperada por los propios servidores. Además, se muestra como prueba de concepto implementable una red de mezcla de dos mensajes y un servidor de mezcla que es mucho más eficiente que la construcción genérica utilizando otras técnicas de ofuscación. Finalmente, se demuestra tanto la seguridad de las ofuscaciones utilizadas como la propiedad de anonimato de ambas construcciones.

2. Antecedentes criptográficos

2.1. Criptografía

La criptografía es el estudio y práctica de técnicas para una comunicación segura en presencia de actores externos maliciosos (o adversarios). Más generalmente, se ocupa de construir y analizar protocolos que limitan la influencia de dichos agentes externos y que están relacionados con varios aspectos en seguridad informática como confidencialidad de datos, integridad de datos, y autentificación.

A modo de ilustración, supóngase que se tiene un mensaje que se desea enviar a otra persona (por un canal de comunicación de común acuerdo), cuyo contenido es vital que no sea conocido por ninguna persona

que no sean quien creó el mensaje y quien es el destinatario de éste. Para evitar que terceras personas intercepten y lean el mensaje, éste es sometido a un proceso (denominado encriptación) que lo convierte en un texto ilegible para cualquiera que no posea la clave requerida para revertir el proceso (lo que se denomina desencriptación). Así, el texto cifrado es enviado al destinatario, que obviamente posee la clave para revertir el proceso y recuperar el mensaje original.

Con este pequeño ejemplo se ilustran los conceptos básicos de lo que es encriptación, como texto plano y texto cifrado (el mensaje antes y después de ser procesado), la existencia de un valor secreto que permite transformar el mensaje (la clave), y los tres actores fundamentales: el emisor del mensaje, el receptor de éste, y el adversario, que intenta descubrir el contenido del mensaje.[14]

2.2. Esquema de encriptación

A grandes rasgos, un esquema de encriptación permite convertir un texto plano p en un texto cifrado c bajo una llave k_1, de manera que el contenido del mensaje sólo pueda ser desencriptado por quien tenga la llave k_2 pareja de k_1. Los esquemas de encriptación se dividen en dos grandes grupos: esquemas de encriptación de clave simétrica (esquemas simétricos) y de clave asimétrica (esquemas asimétricos).

Los esquemas de clave simétrica son una clase de algoritmos criptográficos, donde las claves de encriptación y desencriptación están relacionadas de manera trivial. Usualmente son la misma clave, pero puede existir una transformación simple que las relacione de manera que no sean idénticas. Estas claves representan un secreto compartido entre 2 o más partes que se pueden utilizar para establecer un canal privado de comunicación.

Los esquemas de clave simétrica son difíciles de usar en la práctica, no por fallos en su seguridad, sino por problemas en el intercambio de claves. Una vez que el remitente y el destinatario hayan intercambiado las claves pueden usarlas para comunicarse con seguridad, pero ¿qué canal de comunicación que sea seguro han usado para transmitirse las claves? Sería mucho más fácil para un atacante intentar interceptar una clave que probar las posibles combinaciones del espacio de claves. Existen protocolos de intercambio de claves como Diffie-Hellman[6] para aliviar el problema, pero no es el único problema que tienen estos esquemas de encriptación.

Otro problema es el número de claves que se necesitan. Si tenemos un número n de personas que necesitan comunicarse entre sí, o bien usan una única clave compartida por todos, o bien se requiere una clave por cada par de actores que deseen comunicarse seguramente. Ambos casos pueden funcionar perfectamente con un grupo reducido de actores, pero para grupos más grandes sería imposible llevarlo a cabo, puesto que o bien mantener una clave única para un grupo grande es impráctico, o bien los costos de crear y mantener una clave para cada posible par de personas que deseen comunicarse seguramente comienzan a crecer exponencialmente.

Los esquemas de clave asimétrica se inventaron con el fin de evitar por completo el problema del intercambio de claves de los sistemas de cifrado simétricos. Estos esquemas, al igual que los anteriores, son una clase de algoritmos criptográficos, donde las claves de encriptación y desencriptación están relacionadas de manera no trivial. Usualmente, la relación entre las llaves es una para la que no se conoce solución eficiente (por ejemplo, logaritmo discreto o factorización de enteros). Bajo este esquema, una de las claves (usada para encriptar textos) se hace pública y la otra (usada para desencriptar textos) se mantiene privada.

Con el sistema de clave pública y privada, no es necesario que dos personas se pongan de acuerdo en la clave a emplear. Todo lo que se requiere es que, antes de iniciar la comunicación secreta, el emisor consiga una copia de la clave pública del destinatario. Es más, esa misma clave pública puede ser usada por cualquiera que desee comunicarse con el propietario de ésta. Por tanto, se elimina tanto el problema del intercambio de claves como el del número de claves requeridas, pero estos sistemas poseen sus propias desventajas: Para una misma longitud de clave y mensaje en promedio se requiere mayor tiempo de proceso; para obtener la misma seguridad que un esquema simétrico, las claves deben ser de mayor tamaño; y por lo general el texto cifrado ocupa más espacio que el original.[14]

Observar que la criptografía de clave pública necesita establecer una confianza en que la clave pública de un usuario (al cual se identifica por una cadena identificativa a la que se llama identidad) es correcta, es

decir el único que posee la clave privada correspondiente es el usuario auténtico al que pertenece. Cuanto más fiable sea el método más seguridad tendrá el sistema. Lo ideal sería que cada persona comunicara y probara de forma directa al resto del grupo cuál es su clave pública. Sin embargo esto es impráctico en aplicaciones reales, por lo que en realidad se utilizan diversos esquemas para establecer confianza. Estos esquemas se pueden agrupar en dos tipos: Esquemas centralizados y descentralizados. En los esquemas descentralizados hay varios nodos y cada uno tiene capacidades y derechos. En los esquemas centralizados, por otra parte, existe una arquitectura cliente-servidor donde los servidores juegan un papel central y proveen servicios a los clientes. Ninguno de los dos tipos de esquema es estrictamente mejor que el otro, cada uno con sus propias ventajas y desventajas, y es usar uno u otro tipo en una aplicación específica requiere cierto análisis para decidir cual es el mejor en ese caso específico. En general, los ataques a sistemas centralizados suelen ser del tipo denegación de servicio debido a que basta con que falle el servidor central para que el sistema de confianza caiga por completo. Los ataques a sistemas descentralizados suelen ser encaminados a publicar claves públicas falsas debido a que al haber varios nodos manejando el sistema de confianza, es más dificil asegurar que las claves falasa van a ser detectadas por toda la red.

Los modelos más usados para establecer confianza son:

- Infraestructura de clave pública (PKI): En este modelo hay una o varias entidades emisoras de certificados (Autoridades de certificación o CA) que aseguran la autenticidad de la clave pública y de ciertos atributos del usuario. Para ello firman con su clave privada ciertos atributos del usuario, incluyendo su clave pública, generando un certificado del usuario.
- Red de confianza: Los propios usuarios recogen claves públicas de otros usuarios y aseguran su autenticidad si están seguros de que la clave privada correspondiente pertenece en exclusiva a ese usuario. Un usuario además puede directamente confiar en el conjunto de claves públicas en las que otro confía ya sea directamente o a través de otras relaciones de confianza. En cada caso es el propio usuario el que decide el conjunto de claves públicas en las que confía y su grado de fiabilidad. Dos usuarios que no se conocen pueden confiar en sus claves públicas si existe una cadena de confianza que enlace ambas partes.
- Red criptográfica basada en identidad: En este modelo existe un generador de claves privadas (PKG) que a partir de una cadena de identificación del usuario genera una clave privada y otra pública para ese usuario. La clave pública es difundida para que el resto de usuarios la sepan y la privada es comunicada en exclusiva al usuario a quién pertenece.
- Red criptográfica basada en certificados: En este modelo el usuario genera de alguna manera sus claves pública y privada. La clave pública es enviada a una autoridad de certificación (CA) la cual, basándose en criptografía basada en identidad, genera un certificado que asegura la validez de la clave.
- Red criptográfica basada en claves parciales: En este modelo existe un centro generador de claves (KGC) que, a petición de un usuario, genera una clave privada parcial. La clave privada real se genera a partir de la clave privada parcial y un valor secreto escogido por el usuario. La clave pública es generada también por el usuario a partir de parámetros públicos del KGC y el valor sercreto escogido.

Ya descritos los esquemas de encriptación simétricos y asimétricos, los definiremos formalmente como una tupla de algoritmos $E = (KeyGen, Enc, Dec)$ definidos de la siguiente manera:

- $KeyGen$: $(pk, sk) \leftarrow KeyGen(1^k)$: Genera un par de claves dado un parám. de seguridad k.
- Enc: $c \leftarrow Enc(pk, m)$: Encripta el texto plano m bajo clave pk.
- Dec: $m \leftarrow Dec(sk, c)$: Desencripta el texto cifrado c si sk es la clave par de pk.

Un esquema es correcto si se cumple que $Dec(sk, Enc(pk, m)) = m$ para todo par de llaves (pk, sk) válidamente generado. Cabe recordar que pk y sk pueden ser el mismo valor, en un esquema de clave simétrica.[14]

En la actualidad, herramientas como PGP, SSH o la capa de seguridad SSL para la jerarquía de protocolos TCP/IP utilizan un híbrido formado por la criptografía asimétrica para intercambiar claves de criptografía simétrica, y la criptografía simétrica para la transmisión de la información.

2.3. Seguridad IND-CPA

Un esquema criptográfico sólo es útil si es seguro de usar, si un adversario no puede romper la encriptación y recuperar el texto plano. En criptografía existen definiciones formales de varios niveles de seguridad de un esquema de encriptación, siendo la más básica la seguridad ante ataques de texto plano escogido (IND-CPA).

La idea básica es que un adversario, dada la encriptación de uno de dos textos planos escogidos por él mismo, no puede determinar a cuál de los textos planos corresponde la encriptación. De manera más formal, para un esquema de encriptación asimétrico $PKEA = (KeyGen, Enc, Dec)$ se define la seguridad ante ataques de texto plano escogido como el siguiente experimento entre un adversario A y un sistema $Sist$:

- $Sist$ genera un par de claves válido $(pk, sk) \leftarrow KeyGen(1^k)$.
- $Sist$ entrega pk a A.
- A puede realizar cualquier número de encriptaciones usando pk u otras operaciones que estime conveniente.
- A eventualmente le entrega dos textos planos m_0 y m_1 a $Sist$.
- $Sist$ escoge un bit $b \xleftarrow{\$} \{0, 1\}$.
- $Sist$ encripta uno de los mensajes: $c \leftarrow Enc(pk, m_b)$.
- $Sist$ entrega c a A.
- A puede nuevamente realizar cualquier número de encriptaciones u otras operaciones.
- A finalmente entrega un bit b' que indica cuál mensaje cree que fue encriptado.

El esquema $PKEA$ es IND-CPA si para todo adversario A, éste adivina correctamente el mensaje que fue encriptado con probabilidad $\frac{1}{2} + \mu(k)$, donde $\mu(k)$ es despreciable, vale decir que es una función tal que para todo polinomial $poly(\cdot)$ distinto de cero existe un k_0 tal que $|\mu(k)| < \left|\frac{1}{poly(k)}\right|$ para todo $k > k_0$.

El experimento es casi el mismo para el caso de un esquema de encriptación simétrico $PKES = (KeyGen, Enc, Dec)$, sólo que en vez de entregar pk a A, $Sist$ le entrega un oráculo de encriptación:

- *Sist* genera una clave válida $key \leftarrow KeyGen(1^k)$.
- *Sist* genera un oráculo de encriptación $OrEnc(x) = Enc(key, x)$
- *Sist* entrega $OrEnc$ a A.
- A puede realizar cualquier número de consultas al oráculo u otras operaciones que estime conveniente.
- A eventualmente le entrega dos textos planos m_0 y m_1 a *Sist*.
- *Sist* escoge un bit $b \xleftarrow{\$} \{0, 1\}$.
- *Sist* encripta uno de los mensajes: $c \leftarrow Enc(key, m_b)$.
- *Sist* entrega c a A.
- A puede nuevamente realizar cualquier número de consultas al oráculo u otras operaciones.
- A finalmente entrega un bit b' que indica cuál mensaje cree que fue encriptado.

El esquema $PKES$ es IND-CPA si para todo adversario A, éste adivina correctamente el mensaje que fue encriptado con probabilidad $\frac{1}{2} + \mu(k)$, donde $\mu(k)$ es despreciable.

2.4. Re-encriptación

Un esquema de encriptación de clave pública aleatorizado corresponde a una tupla de algoritmos $E = (KeyGen, Enc, Dec)$ tales que correr varias veces el algoritmo de encriptación Enc sobre el mismo texto plano m y usando la misma clave pk, genera textos cifrados distintos $c_0, c_1, c_2, c_3, \ldots$ por cada ejecución. Para usar este tipo de esquemas en redes de mezcla re-encriptantes, se requiere un componente adicional conocido como algoritmo de reencriptación, denotado por Re. Este algoritmo realeatoriza la encriptación de un texto cifrado.

De manera más formal, se define un esquema criptográfico de encriptación que permite reencriptación como una tupla de algoritmos $E = (KeyGen, Enc, Dec, Re)$ definidos de la siguiente manera:

- $KeyGen$: $(pk, sk) \leftarrow KeyGen(1^k)$: Genera un par de claves dado un parám. de seguridad k.
- Enc: $c \leftarrow Enc(pk, m)$: Encripta el texto plano m bajo clave pk.
- Dec: $m \leftarrow Dec(sk, c)$: Desencripta el texto cifrado c si sk es la clave par de pk.
- Re: $z \leftarrow Re(pk, c)$: Reencripta el texto cifrado bajo pk.

Un esquema es correcto si se cumple que $Dec(sk, Enc(pk, m)) = m$ y $Dec(sk, Re(pk, Enc(pk, m))) = m$ para todo par de llaves (pk, sk) válidamente generadas.

Cabe notar que, a diferencia de la encriptación, la reencriptación no requiere conocer el texto plano para generar un texto cifrado válido. En términos de seguridad, podemos definir un experimento IND-CRE que considere el proceso de reencriptación de la siguiente manera:

- *Sist* genera un par de claves válido $(pk, sk) \leftarrow KeyGen(1^k)$.
- *Sist* entrega pk a A.
- A puede realizar cualquier número de encriptaciones/reencriptaciones usando pk u otras operaciones que estime conveniente.
- A eventualmente le entrega dos textos cifrados válidos c_0 y c_1 a *Sist*.
- *Sist* escoge un bit $b \xleftarrow{\$} \{0, 1\}$.
- *Sist* reencripta uno de los textos cifrados: $c' \leftarrow Re(pk, c_b)$.
- *Sist* entrega c' a A.
- A puede nuevamente realizar cualquier número de encriptaciones/reencriptaciones u otras operaciones.
- A finalmente entrega un bit b' que indica cuál texto cifrado cree que fue reencriptado.

Si el algoritmo de reencriptación genera textos cifrados con una distribuición indistinguible de la que genera el algoritmo de encriptación, entonces si el esquema de encriptación base es IND-CPA, el esquema con reencriptación es IND-CRE.[11]

La re-encriptación (simple) no es más que un caso específico de lo que se denomina re-encriptación por proxy, que consiste en tomar un texto cifrado c bajo clave pk_A y reencriptarlo usando un agente externo proxy de manera que quede un texto cifrado c' bajo clave pk_B por medio de una clave de reencriptación rk_{AB}.

De manera más formal, se define un esquema criptográfico de encriptación que permite reencriptación por proxy como una tupla de algoritmos $E = (KeyGen, ReGen, Enc, Dec, Re)$ definidos de la siguiente manera:

- $KeyGen$: $(pk, sk) \leftarrow KeyGen(1^k)$: Genera un par de claves dado un parám. de seguridad k.
- $ReGen$: $rk_{AB} \leftarrow ReGen(pk_A, sk_A, pk_B, sk_B)$: Genera una clave de reencriptación de A a B.
- Enc: $c \leftarrow Enc(pk, m)$: Encripta el texto plano m bajo clave pk.
- Dec: $m \leftarrow Dec(sk, c)$: Desencripta el texto cifrado c si sk es la clave par de pk.
- Re: $z \leftarrow Re(rk, c)$: Reencripta el texto cifrado bajo una nueva clave definida por rk.

Un esquema es correcto si se cumple que $Dec(sk_A, Enc(pk_A, m)) = m$ y $Dec(sk_B, Re(rk_{AB}, Enc(pk_A, m))) = m$ para todas las claves $[(pk_A, sk_A), (pk_B, sk_B), rk_{AB}]$ válidamente generadas.

Los esquemas de reencriptación por proxy pueden ser de dos tipos: esquemas de reencriptación unidireccional o bidireccional.

En los esquemas unidireccionales, la clave de reencriptación rk_{AB} sirve sólo para reencriptar textos encriptados bajo la clave de A a textos encriptados bajo la clave de B. En los esquemas bidireccionales, rk_{AB} también sirve para reencriptar textos encriptados bajo la clave de B a textos encriptados bajo la clave de A.

Además de direccionalidad, los esquemas de reencriptación se definen como activos o pasivos, donde en los esquemas activos las cuatro claves de A y B ($[(pk_A, sk_A), (pk_B, sk_B)]$) se requieren para generar la clave

de reencriptación rk_{AB}. En los esquemas pasivos, por otra parte, una de las claves secretas (usualmente sk_B) no es requerida para generar la clave de reencriptación rk_{AB}.[3]

2.5. Encriptación completamente homomórfica

Un esquema de encriptación homomórfico es un esquema donde se puede realizar una operación algebraica ($\cdot$ ó $+$) sobre los textos planos mediante la aplicación de una operación algebraica ($\cdot$ ó $+$) sobre los textos cifrados. Como ilustración, supóngase que se tienen dos textos cifrados $c_1 = Enc_H(m_1, r_1)$ y $c_2 = Enc_H(m_2, r_2)$ encriptados bajo un sistema homomórfico con respecto a $(\cdot,\cdot)$, entonces se cumple que $c_1 \cdot c_2 = Enc_H(m_1, r_1) \cdot Enc_H(m_2, r_2) = Enc_H(m_1 \cdot m_2, r_1 \cdot r_2)$.

Un esquema de encriptación completamente homomórfico es un esquema que es homomórfico tanto para $+$ como para $\cdot$, lo que implica que cualquier función que pueda ser expresada como polinomio puede ser evaluada homomórficamente. El primer esquema de encriptación completemente homomórfico fue propuesto por Gentry[9].

De manera más formal, se define un esquema criptográfico de encriptación completamente homomórfico como una tupla de algoritmos $E = (KeyGen, Enc, Dec, Eval)$ definidos de la siguiente manera:

- $KeyGen$: $(pk, sk) \leftarrow KeyGen(1^k)$: Genera un par de claves dado un parám. de seguridad k.
- Enc: $c \leftarrow Enc(pk, m)$: Encripta el texto plano m bajo clave pk.
- Dec: $m \leftarrow Dec(sk, c)$: Desencripta el texto cifrado c si sk es la clave par de pk.
- $Eval$: $z \leftarrow Eval(pk, f, c)$: Evalúa la función f en el texto plano correspondiente a c y encripta el resultado bajo la clave pk.

Un esquema es correcto si se cumple que $Dec(sk, Eval(pk, f, c)) = f(m)$ para $c = Enc(pk, m)$ y todas las claves (pk, sk) válidamente generadas. Además, debe darse que el tiempo de ejecución de Dec sobre un texto cifrado $c = Eval(pk, f, Enc(pk, m))$ no debe depender de la complejidad de f.[9]

3. Descripción del problema

En una red de mezcla (re-encriptante), cada servidor de mezcla (o 'mixer') realiza una permutación y re-encriptación de los mensajes que le llegan, permitiendo así una comunicación anónima. Sin embargo, esto es posible sólo si los mixers no revelan la permutación de mensajes que hicieron. En el caso de que existan suficientes servidores deshonestos o intervenidos (por ejemplo todos), sería posible relacionar un mensaje dado con su emisor.

Pero, ¿qué pasaría si los propios servidores no supieran la permutación que aplicaron? En ese caso, aunque los servidores fueran deshonestos o intervenidos, no se podría obtener más información sobre los mensajes que si simplemente se estuviera monitoreando el tráfico de entrada y salida, como si el proceso de permutación fuese una caja negra. Este proceso de ocultamiento se realiza utilizando técnicas de ofuscación. En particular, el programa ejecutado por el mixer estaría ofuscado, lo que le impediría revelar la permutación utilizada.

A continuación se definen de manera más formal los conceptos de red de mezcla y ofuscación, además de mostrar el trabajo previo que se ha realizado en esta área.

3.1. Mix Networks

Una red de mezcla o mix network (mixnet) es una red de comunicación anónima por medio de servidores proxy, donde se desea ocultar no sólo el contenido del mensaje, sino que también la identidad del emisor del mensaje. Esto se realiza por medio de los servidores que van recibiendo, reencriptando, reordenando y reenviando los mensajes cifrados que les llegan de manera de que sea computacionalmente difícil determinar el emisor de un mensaje que ha pasado por varios proxy sin tener que intervenir directamente los servidores. El concepto de mixnet fue propuesto por Chaum en el año 1981[5].

Existen dos tipos de mixnets: las clásicas, donde al pasar por los distintos servidores de mezcla, el conjunto de mensajes es re-encriptado y desencriptado parcialmente, lo cual permite obtener el conjunto de mensajes desencriptados y permutados al final; y las re-encriptantes, donde el conjunto de mensajes es re-encriptado en cada servidor de mezcla con algún esquema de encriptación de clave pública como ElGamal[8].

Para efectos de este trabajo, se describirá el funcionamiento de una red de mezcla re-encriptante basada en los trabajos de Park et al.[17], y Sako y Kilian[18] principalmente. Se asumirá que existe una manera de generar un par de claves (pk, sk) de un esquema de clave pública que permite re-encriptación simple tal que ningún servidor de mezcla posea la capacidad de desencriptar textos por su cuenta y asimismo que existe una manera de desencriptar textos encriptados con la clave pública del par (en la práctica esto se logra con generación distribuida de claves o un servidor completamente honesto aparte). Sean n usuarios y m servidores de mezcla, donde cada usuario encripta su mensaje bajo la clave pk y luego lo envía a la red de mezcla. Una vez recibidos todos los mensajes encriptados, denotado por el vector $\vec{c}$, éstos son recibidos por el primer servidor de mezcla, que procede a generar y aplicar una permutación a $\vec{c}$, generando $\vec{c'}$, y luego aplica una re-encriptación sobre cada uno de los elementos de $\vec{c'}$, generando $\vec{c''}$. Finalmente, el servidor genera una prueba de que hizo correctamente la permutación y re-encriptación tal que cualquier verificador pueda corroborar la correctitud del proceso usando $\vec{c''}$ y la prueba, pero sin obtener información que le permita encontrar la permutación y/o re-encriptación realizada. Una vez verificado correctamente el proceso (ya sea por el resto de los servidores o por un agente de verificación externo), el servidor le envía $\vec{c''}$ al próximo servidor de mezcla, que realizará el mismo proceso, y así sucesivamente hasta que el servidor final entrega el vector final $\vec{r}$, que es desencriptado y sus contenidos publicados.

Cabe notar, pues importa en este trabajo, que si bien para cualquier agente externo la permutación y re-encriptación realizada en cada servidor de mezcla es desconocida (necesario para mantener el anonimato), un adversario que sea capaz de tomar control de uno o más servidores podría tomar nota de los pasos que realizan los servidores y así conocer la permutación y/o la re-encriptación usada en esos servidores. El anonimato puede no verse comprometido incluso si existen servidores comprometidos, pues las redes de mezcla están construidas para soportar hasta cierto número de servidores deshonestos, pudiendo ser desde requerir una mayoría de servidores honestos hasta requerir al menos un servidor honesto. Obviamente, una vez que todos los servidores son comprometidos, ya no existe anonimato puesto que el adversario es capaz de conocer todas las permutaciones realizadas por todos los servidores, haciendo el proceso de trazar un mensaje a su emisor relativamente trivial.

3.2. Ofuscación

Ofuscar es alterar el código de un programa para que sea ilegible; una definición de ofuscación más específica depende del contexto en que se aplique y las definiciones de seguridad que se usen, pero usualmente requieren que el programa ofuscado revele lo mínimo de información posible, y más específicamente que ofusque todo tipo de claves secretas que se requieran para el funcionamiento del programa.

Hasta antes del 2001, no existía una base teórica sobre la cual definir formalmente ofuscación, por lo que el proceso de ofuscar un programa era largamente artesanal y sin capacidad de demostrar su posible efectividad. En general, ofuscar un programa bajo este paradigma (o ausencia de uno) consiste en escribir la funcionalidad del programa de la manera más compleja que el lenguaje de programación permitiese, abusar de la notación permitida, usar código generado al vuelo, y/o abusar de técnicas de compresión. Estrictamente hablando, este tipo de ofuscación no hacen imposible descubrir lo que está haciendo internamente el programa, pues

cualquiera que posea el conocimiento suficiente del lenguaje de programación utilizado podría ver el código y descifrar las instrucciones del programa en un proceso probablemente tedioso, pero realizable.

El 2001 se publicó el primer artículo, de Barak et al.[2], donde se aborda de manera formal el problema de ofuscar un programa. Más específicamente, se define ofuscación como un experimento donde un adversario A con oráculo C desea obtener toda la información posible de su oráculo. Sea Inf el conjunto de información que A puede obtener de C como oráculo, y sea ObC un programa que ofusca a C; si ahora A además de tener como oráculo a C, recibe el código fuente de ObC, puede obtener un nuevo conjunto de información sobre C que se denominará Inf'. Si ObC es efectivamente una ofuscación de C, entonces Inf' debe ser un subconjunto de Inf; o dicho de otra manera, la ofuscación no le entrega ninguna información a A que éste no pudiera haber obtenido de su oráculo C. En este mismo artículo se demuestra que, aún bajo supuestos débiles, la ofuscación es imposible.

A pesar de este resultado, la investigación sobre ofuscación de programas no cesó, con la investigación centrándose en buscar definiciones de ofuscación que fueran lo suficientemente débiles y/o especializadas para que no cayeran en el resultado de imposibilidad y aún así fueran de utilidad. Así, el 2005 se publica uno de los primeros artículos con resultados positivos de ofuscación, específicamente de funciones punto, posteriormente extendido a funciones punto con output multibit[4, 19]. Posteriormente, el 2007 Hohenberger et al.[13] muestran el primer resultado positivo de ofuscación de una función criptográfica compleja, y el 2011 Ding y Gu[7] muestran cómo usar encriptación completamente homomórfica para generar ofuscaciones.

3.3. Trabajo previo

Existe sólo un trabajo previo que trata precisamente sobre ofuscación de permutaciones en redes de mezcla, de Adida y Wikström[1] basado en una definición de ofuscación más débil usada por Ostrovsky y Skeith[16]. A grandes rasgos, en este trabajo, cada servidor de mezcla de la red puede escoger una permutación para luego crear un circuito ofuscado que la aplica. Este circuito es publicado de manera que cualquier otro agente pueda aplicar la permutación sin llegar a conocerla.

La gran diferencia entre el trabajo de Adida y Wikström y el presentado aquí, es en la capacidad de los circuitos ofuscados de poder ser reutilizados. En el trabajo previo, cada circuito ofuscado posee una matriz de permutación fija, por lo que cada vez que se utiliza el circuito, la permutación es la misma. En cambio, en el trabajo presentado aquí, la permutación es definida cuando ingresa el vector de textos cifrados al circuito, por lo que, a menos que se reingrese el mismo vector, cada uso del circuito debería generar una permutación distinta, lo que implica que no hay que estar recalculando circuitos cada vez que se quiera realizar una nueva mezcla.

Por lo tanto, a pesar de que ambos trabajos abordan el tema de la ofuscación de permutaciones en redes de mezcla, los métodos son lo suficientemente distintos como para no calificar este trabajo como redundante.

4. Solución propuesta

La construcción descrita en este trabajo se basa en los resultados de Hohenberger et al.[13] y Ding y Gu[7], por lo que la definición de ofuscación utilizada es la descrita en estos trabajos.

4.1. Definición formal de ofuscación

Un algoritmo eficiente Obf es un ofuscador para la familia de circuitos $\{C_n\}_{n\in\mathbb{N}}$ si cumple con las siguientes tres propiedades:

- Ralentización polinomial: Existe un polinomial $p(k)$ tal que para input n suficientemente grande, para cualquier $C \in \{C_n\}_{n\in\mathbb{N}}$, se tiene que:
$$|Obf(C)| \leq p(|C|)$$

- Preservación de funcionalidad: Existe una función despreciable $\mu(k)$ tal que para toda longitud de input k, para cualquier $C \in \{C_n\}_{n\in\mathbb{N}}$, se tiene que:

$$Pr\left[\exists x \in \{0,1\}^k : (Obf(C))(x) \neq C(x)\right] \leq \mu(k)$$

- Propiedad de caja negra (caso promedio): Existe una máquina PPT S (simulador) tal que, para toda máquina PPT D (distinguidor), todo polinomial $p(\cdot)$, todos los n suficientemente grandes, para un $C \in \{C_n\}_{n\in\mathbb{N}}$ al azar, y todo $z \in \{0,1\}^{poly(n)}$ se tiene que:

$$\begin{aligned}\Big|&Pr\left[C \leftarrow \{C_n\}_{n\in\mathbb{N}}; C' \leftarrow Obf(C); b \leftarrow D^C(C', z) : b = 1\right] \\ &- Pr\left[C \leftarrow \{C_n\}_{n\in\mathbb{N}}; C' \leftarrow S^C(1^n, z); b \leftarrow D^C(C', z) : b = 1\right]\Big| < \frac{1}{p(n)}\end{aligned}$$

4.2. Solución genérica

Sea una red de mezcla para n inputs y m servidores de mezcla con esquemas de encriptación homomórfico $PKE = (KeyGen, Enc, Dec)$ y completamente homomórfico $FHES = (KeyGen_F, Enc_F, Dec_F, Eval_F)$, ambos IND-CPA, con claves (sk, pk) y (sk_F, pk_F) respectivamente. Cada servidor de mezcla posee un secreto K_i y un programa ofuscado $ObC_{K_i,pk_F}(\cdot)$ que se detallará más adelante.

Para simplificar la construcción y mostrar de manera más notoria los resultados del trabajo, se asumirá que existen servidores de confianza que se encargan de generar y entregar a los servidores de mezcla su secreto y su programa ofuscado, y además se encargan de desencriptar los textos encriptados que entrega en último servidor de mezcla. Además, se asumirá que los textos cifrados que llegan a la red de mezcla son correctos, es decir que cada emisor conoce el texto plano de su texto cifrado; esto se puede comprobar con pruebas de conocimiento, por ejemplo.

4.2.1. Funcionalidad de mezcla

La funcionalidad básica de un servidor de mezcla, dado un vector de textos cifrados $\vec{x}$ es la siguiente:

$C_{n,K_i}(\vec{x})$:

- $(r, \pi) \leftarrow F_{K_i}(\vec{x})$
- $\vec{x}' \leftarrow \vec{x} : x'_j \leftarrow x_{\pi(j)}$ for all j
- $\vec{x}'' \leftarrow \vec{x}' \cdot Enc(pk, 1, r)$
- $\vec{z} \leftarrow \vec{x}''$
- output $\vec{z}$

Donde $F_{K_i}(\cdot)$ es una función pseudoaleatoria evaluable bajo homomorfismo (por ejemplo, una evaluación $AES[10]$) y π es una permutación aleatoria para el vector $\vec{x}$:

$\pi(\vec{x})$:

- $i \leftarrow len(\vec{x})$
- $while(i > 1)\{$
- $swap(x_{r_i}, x_i)$
- $i \leftarrow i - 1$
- $\}$
- Output $\vec{x}$

Donde r_i es un número entre 1 e i, definido por la función pseudoaleatoria $F_{K_i}(\cdot)$.

Esta funcionalidad, tal como está, no hay certeza de que sea ofuscable, pero se puede usar para construir una funcionalidad que sí lo es, al menos de acuerdo al resultado de Ding y Gu[7]:

$C_{K_i,pk_F}(\vec{x})$:

- Si $\vec{x} = RetrieveKey$: Output pk_F
- $\vec{x'} \leftarrow C_{n,K_i}(\vec{x})$
- $\vec{c_x} \leftarrow Enc_F(pk_F, \vec{x'})$
- Output $\vec{c_x}$

Esto sí se sabe que es ofuscable, pero el problema es que si se quiere aplicar en varios servidores de mezcla, cada uno esperaría como input un vector de textos encriptados sólo con PKE, por lo que habría que desencriptar $FHES$ después de cada servidor, lo que es altamente costoso y molesto. Luego, se tiene que buscar una forma de modificar la funcionalidad de manera que sirva para varios servidores de mezcla sin tener que desencriptar de por medio. Pero por ahora, la construcción sirve al menos para el primer servidor:

$C_{K_1,pk_F}(\vec{x})$:

- Si $\vec{x} = RetrieveKey$: Output pk_F
- $\vec{x'} \leftarrow C_{n,K_1}(\vec{x})$
- $\vec{c_x} \leftarrow Enc_F(pk_F, \vec{x'})$
- Output $\vec{c_x}$

4.2.2. Ofuscación de la funcionalidad (1er mixer)

Para ofuscar la funcionalidad descrita por $C_{K_1,pk_F}(\cdot)$, se debe tener que la profundidad un circuito U_n perteneciente a una familia de circuitos $U_{n n\in\mathbb{N}}$ tal que $U_n(C_{n,K_1}, \vec{x})$ emule el comportamiento de $C_{n,K_1}(\vec{x})$, debe poder ser estimada con antelación para poder generar un esquema $FHES$ que soporte evaluar este circuito. Pero esto se tiene puesto que C_{n,K_1} pertenece a una familia de circuitos públicamente conocida, por lo que se puede computar su tamaño con antelación y, por ende, estimar la profundidad de U_n.

Con todo esto, se puede ofuscar el circuito C_{K_1,pk_F} del primer mezclador de la siguiente manera:

$GenObf(C_{K_1,pk_F})$:

- Obtiene pk_F y C_{n,K_1} de la descripción de C_{K_1,pk_F}
- $c_{C_{n,K_1}} \leftarrow Enc_F(pk_F, C_{n,K_1})$
- $\alpha \leftarrow c_{C_{n,K_1}}$
- Output $ObC_\alpha(\cdot)$

$ObC_\alpha(\vec{x})$:

- Si $\vec{x} = RetrieveKey$: Output pk_F
- r_0, r_1 al azar
- $c_x \leftarrow Enc_F(pk_F, \vec{x}, r_0)$
- $z \leftarrow Eval_F(pk_F, U_n, \alpha, c_x, r_1)$
- Output z, r_0, r_1

4.2.3. Demostración de seguridad (1er mixer)

Una vez construído $ObC_\alpha(\cdot)$, hay que demostrar que efectivamente es una ofuscación, lo que significa probar que cumple con las tres propiedades descritas en su definición. Las demostraciones de las propiedades de preservación de funcionalidad y ralentización polinomial se pueden obviar puesto que si U_n fue bien elegido y $Eval_F$ soporta circuitos de la envergadura de U_n, entonces las propiedades se obtienen simplemente por la definición de $Eval_F$. La propiedad de caja negra se demostrará primero creando el algoritmo S y luego mostrando que si existe un algoritmo D que puede distinguir entre la ofuscación real y la generada por S, entonces existe un adversario A que, usando como subrutina a D, puede romper la seguridad IND-CPA de $FHES$.

El algoritmo S que genera un circuito de ofuscación falso es el siguiente:

$S^{C_{K_1,pk_F}}(n)$:

- $pk_F \leftarrow C_{K_1,pk_F}(RetrieveKey)$
- Genera K_1' falso.
- Genera $C_{n,K_1'}$ usando K_1'
- $c_{C_{n,K_1'}} \leftarrow Enc_F(pk_F, C_{n,K_1'})$
- $\beta \leftarrow c_{C_{n,K_1'}}$
- Output $ObC_\beta(\cdot)$

Cabe notar que el simulador S genera una ofuscación falsa que tiene la misma forma que la ofuscación verdadera (que es conocida), sólo que con un secreto falso. Ahora, si $D^{C_{K_1,pk_F}}(ObC_a(\cdot), z')$ distingue entre la ofuscación verdadera y la creada por el simulador con probabilidad no despreciable, entonces existe adversario A que puede ganar en el experimento IND-CPA de $FHES$ usando D como subrutina y con su misma probabilidad de éxito.

El adversario A que rompe la seguridad IND-CPA de $FHES$ es el siguiente:

$A^{LoR_F}(n, pk_F, z')$:

- Genera K y K' secretos.
- Genera $C_{n,K}$ y $C_{n,K'}$
- Genera C_{K,pk_F} usando $C_{n,K}$
- $c_{C_{n,K_?,1}} \leftarrow LoR_F(C_{n,K}, C_{n,K'})$
- $\gamma \leftarrow c_{C_{n,K_?,1}}$
- Genera $ObC_\gamma(\cdot)$
- $r \leftarrow D^{C_{K,pk_F}}(ObC_\gamma(\cdot), z')$
- Output r

De la descripción de A se puede ver fácilmente que si D distingue entre las ofuscaciones con probabilidad p, entonces A distingue entre las encriptaciones de $C_{n,K}$ y $C_{n,K'}$ con la misma probabilidad p. Luego, si p no es despreciable, A rompe la seguridad IND-CPA del esquema $FHES$, lo que es una contradicción, lo que implica que p es despreciable. Por lo tanto, D distingue entre la ofuscación real y la generada por un simulador sólo con probabilidad despreciable, por lo que la propiedad de caja negra se cumple.

4.2.4. Ofuscación de la funcionalidad (resto de los mixers)

Se ha demostrado que $GenObf$ efectivamente genera una ofuscación, pero sólo para en primer servidor de mezcla. Para el resto de los servidores se mostrará primero la ofuscación y luego se explicará porqué es así.

Dados m servidores de mezcla, y definiendo como OP_i el resultado entregado por el servidor i, la ofuscación para los servidores $2 \leq i \leq m$ se genera así:

$GenObf(C_{K_1,pk_F}, C_{K_2,pk_F}, \ldots, C_{K_m,pk_F})$:

- $pk_F \leftarrow C_{K_1,pk_F}(RetrieveKey)$
- Obtiene C_{n,K_i} de la descripción de C_{K_i,pk_F} para $1 \leq i \leq m$
- $c_{C_{n,K_i}} \leftarrow Enc_F(pk_F, C_{n,K_i})$ para $1 \leq i \leq m$
- $\vec{\theta} \leftarrow \{c_{C_{n,K_1}}, c_{C_{n,K_2}}, \ldots, c_{C_{n,K_m}}\}$
- Genera $ObC_{\vec{\theta},i}(\cdot)$ para $2 \leq i \leq m$
- Genera $ObC_{\theta_1}(\cdot)$
- $\vec{ObC} \leftarrow \{ObC_{\theta_1}(\cdot), ObC_{\vec{\theta},2}(\cdot), \ldots, ObC_{\vec{\theta},m}(\cdot)\}$
- Output $\vec{ObC}$

$ObC_{\vec{\theta},i}(\vec{x}, OP_1, OP_2, \ldots, OP_{i-1})$:

- Si $\vec{x} = RetrieveKey$: Output pk_F
- Si $Verify_{\vec{\theta},i-1}(\vec{x}, OP_1, OP_2, \ldots, OP_{i-1}) = false$: Output $\perp$
- r_i al azar
- $z_{i-1} \leftarrow OP_{i-1}$
- $z_i \leftarrow Eval_F(pk_F, U_n, \theta_i, z_{i-1}, r_i)$
- Output z_i, r_i

Todavía no se han definido $Verify_{\vec{\theta},i}(\cdot)$ ni C_{K_i,pk_F} para $2 \leq i \leq m$, pero se puede ver que las ofuscaciones (si efectivamente lo son) para los servidores de mezcla i, son muy parecidas a la ofuscación para el primer sevidor de mezcla, sólo que en vez de encriptar el vector $\vec{x}$, se recibe el encriptado del servidor anterior.

Las funciones $Verify_{\vec{\theta},i}(\cdot)$ están sólo para asegurarse de que los servidores de mezcla anteriores realizaron correctamente el mezclado y reencriptado, y si existe un ente externo verificador (un servidor de confianza o el resto de los servidores de mezcla) no es necesario ejecutarlas, pero están descritas para asegurar la correctitud

del proceso. Las funciones $Verify_{\vec{\theta},i}(\cdot)$ hacen esencialmente lo mismo que las ofuscaciones, lo que permite asegurar la correctitud del trabajo hecho sin necesidad de alguna demostración externa como pruebas de cero conocimiento.

$Verify_{\vec{\theta},i}(\vec{x}, OP_1, OP_2, \ldots, OP_i)$:

- If $Verify_{\vec{\theta},i-1}(\vec{x}, OP_1, OP_2, \ldots, OP_{i-1}) = false$: Output $false$
- $z_i, r_i \leftarrow OP_i$
- $z_{i-1} \leftarrow OP_{i-1}$
- $z_i \stackrel{?}{=} Eval_F(pk_F, U_n, \theta_i, z_{i-1}, r_i)$

$Verify_{\vec{\theta},1}(\vec{x}, OP_1)$:

- $z, r_0, r_1 \leftarrow OP_1$
- $c_x \leftarrow Enc_F(pk_F, \vec{x}, r_0)$
- $z \stackrel{?}{=} Eval_F(pk_F, U_n, \theta_1, c_x, r_1)$

Ahora sólo falta definir los circuitos C_{K_i,pk_F} para $2 \leq i \leq m$, los que son estrictamente artificiales, existen sólo por dos motivos: entregar el circuito C_{n,K_i} al generador de ofuscaciones, y justificar la existencia de las mismas. Estos circuitos nunca van a ser ejecutados bajo ningún motivo, y hacen básicamente lo mismo que los circuitos ofuscados (para que las ofuscaciones cumplan las propiedades).

$C_{K_i,pk_F}(\vec{x}, OP_1, OP_2, \ldots, OP_{i-1})$:

- Si $\vec{x} = RetrieveKey$: Output pk_F
- $c_x \leftarrow OP_{i-1}$
- $c_{C_{n,K_j}} \leftarrow Enc_F(pk_F, C_{n,K_j})$ para $1 \leq j \leq i$
- $\vec{\theta} = \{c_{C_{n,K_1}}, c_{C_{n,K_2}}, \ldots, c_{C_{n,K_i}}\}$
- if $Verify_{\vec{\theta},i-1}(\vec{x}, OP_1, OP_2, \ldots, OP_{i-1}) = false$: Output $\perp$
- $\vec{x}' \leftarrow Dec_F(sk_F, c_x)$
- $\vec{x}'' \leftarrow C_{n,K_i}(\vec{x}')$
- $c_x \leftarrow Enc_F(pk_F, \vec{x}'')$
- Output c_x

Cabe notar que el circuito tiene acceso a todos los C_{n,K_j} con $1 \leq j \leq i$ y además tiene acceso a sk_F para desencriptar, ambas suposiciones que el ofuscador no posee, puesto que el ofuscador obtiene C_{n,K_i} del circuito correspondiente C_{K_i,pk_F}, y usa $Eval_F(\cdot)$ para saltarse el paso de desencriptación.

4.2.5. Demostración de seguridad (resto de los mixers)

Con todos los circuitos ya definidos, falta demostrar que las construcciones definidas anteriormente son efectivamente ofuscaciones. Para las propiedades de preservación de funcionalidad y ralentización polinomial, los circuitos C_{K_i,pk_F} están construidos de manera que se cumplan, por lo que sólo se mostrará la demostración para la propiedad de caja negra. Para eso primero mostraremos el circuito S dados m servidores de mezcla:

$S^{C_{K_1,pk_F},\ldots,C_{K_m,pk_F}}(n)$:

- $pk_F \leftarrow C_{K_1,pk_F}(RetrieveKey)$
- Genera K'_i falsos para $1 \leq i \leq m$
- Genera C_{n,K'_i} usando K'_i para $1 \leq i \leq m$
- $c_{C_{n,K'_i}} \leftarrow Enc_F(pk_F, C_{n,K'_j})$ para $1 \leq i \leq m$
- $\vec{\theta'} \leftarrow \{c_{C_{n,K'_1}}, c_{C_{n,K'_2}}, \ldots, c_{C_{n,K'_m}}\}$
- Genera $ObC_{\vec{\theta'},i}(\cdot)$ para $2 \leq i \leq m$
- Genera $ObC_{\theta'_1}(\cdot)$
- $\vec{ObC}' \leftarrow \{ObC_{\theta'_1}(\cdot), ObC_{\vec{\theta'},2}(\cdot), \ldots, ObC_{\vec{\theta'},m}(\cdot)\}$
- Output $\vec{ObC}'$

Al igual que en el caso del primer servidor de mezcla, el simulador S entrega ofuscaciones falsas que tienen la misma forma que las reales.

Sea un distinguidor $D^{\{C_{K_i,pk_F}\}_{i=1}^m}(\vec{ObC}, z')$. Si se define $\vec{ObC}_R$ como el vector compuesto por las ofuscaciones reales, y $\vec{ObC}_S$ como el vector compuesto por las ofuscaciones generadas por el simulador, se tiene que la probabilidad de que D funcione está dada por:

$$\left| Pr\left[b \leftarrow D^{\{C_{K_i,pk_F}\}_{i=1}^m}(\vec{ObC}_R, z') : b = 1\right] - Pr\left[b \leftarrow D^{\{C_{K_i,pk_F}\}_{i=1}^m}(\vec{ObC}_S, z') : b = 1\right] \right|$$

Si se definen ahora vectores $\vec{\alpha}(i)$ tales que:

$$\vec{\alpha}(0) = \vec{ObC}_S$$

$$\vec{\alpha}(m) = \vec{ObC}_R$$

$$\vec{\alpha}(k) = \{ObC_{\vec{\theta},K_1,pk_F}(\cdot), \ldots, ObC_{\vec{\theta},K_k,pk_F}(\cdot), ObC_{\vec{\theta},K'_{k+1},pk_F}(\cdot), \ldots, ObC_{\vec{\theta},K'_m,pk_F}(\cdot)\}$$

Y se define $Pr\left[b \leftarrow D^{\{C_{K_i,pk_F}\}_{i=1}^m}(\vec{\alpha}(j), z') : b = 1\right]$ como $Pr(j)$, se tiene que la probabilidad de que D funcione está dada por:

$$|Pr(m) - Pr(0)|$$

A este resultado se le pueden sumar y restar el resto de las probabilidades de la siguiente manera:

$$\left| Pr(m) + \sum_{i=1}^{m-1} [Pr(i) - Pr(i)] - Pr(0) \right|$$

Lo que finalmente se puede expresar como:

$$\left| \sum_{i=1}^{m} Pr(i) - \sum_{i=0}^{m-1} Pr(i) \right|$$

Con esto, se puede expresar el adversario A que rompe la seguridad IND-CPA del esquema $FHES$:

$A^{LoR_F}(n, pk_F, z')$:

- Genera K_i y K'_i para $1 \leq i \leq m$
- Genera C_{n,K_i} y C_{n,K'_i} para $1 \leq i \leq m$
- Genera C_{K_i,pk_F} usando C_{n,K_i} para $1 \leq i \leq m$
- $Y \xleftarrow{\$} \{1, 2, \ldots, m\}$
- $c_{C_{n,K_i^?}} \leftarrow OR^{LoR_F}(j, Y, C_{n,K_i}, C_{n,K'_i})$ para $1 \leq i \leq m$
- $\vec{\theta} \leftarrow \{c_{C_{n,K_1^?}}, c_{C_{n,K_2^?}}, \ldots, c_{C_{n,K_m^?}}\}$
- Genera $ObC_{\vec{\theta},i}(\cdot)$ para $2 \leq i \leq m$
- Genera $ObC_{\theta_1}(\cdot)$
- $\vec{\beta} \leftarrow \{ObC_{\theta_1}(\cdot), ObC_{\vec{\theta},2}(\cdot), \ldots, ObC_{\vec{\theta},m}(\cdot)\}$
- $r \leftarrow D^{\{C_{K_i,pk_F}\}_{i=1}^m}(\vec{\beta}, z')$
- Output r

$OR^{LoR_F}(j, Y, C_1, C_2)$:

- Si $j < Y$: Output $Enc_F(pk_F, C_1)$
- Si $j = Y$: Output $LoR_F(C_1, C_2)$
- Si $j > Y$: Output $Enc_F(pk_F, C_2)$

Cabe notar que cuando $Y = i$ y $LoR_F(\cdot)$ encripta el mensaje de la izquierda ($b = 0$), $\vec{\beta} = \vec{\alpha}(i)$; y similarmente cuando $Y = 1$ y $LoR_F(\cdot)$ encripta el mensaje de la derecha ($b = 1$), $\vec{\beta} = \vec{\alpha}(0)$.

Sea A corriendo en el caso que LoR encripte el mensaje de la izquierda, entonces OR encripta los C_{n,K_i} las primeras Y veces que es llamado, y encripta los C_{n,K'_i} el resto de las veces. En el otro caso, cuando LoR encripta el mensaje de la derecha, OR encripta los C_{n,K_i} las primeras $Y - 1$ veces que es llamado, y encripta los C_{n,K'_i} el resto de las veces. Si Y es una variable aleatoria en $\{1, \ldots, i\}$, entonces para cada $i \in \{1, \ldots, m\}$:

$$Pr\left[Exp^{IND-CPA-LEFT}(A) = 1 | Y = i\right] = Pr(i)$$

$$Pr\left[Exp^{IND-CPA-RIGHT}(A) = 1 | Y = i\right] = Pr(i-1)$$

Dado que la variable Y está distrbuida uniformemente en su rango, tenemos que:

$$\begin{aligned} Pr\left[Exp^{IND-CPA-LEFT}(A) = 1\right] &= \sum_{i=1}^{m} Pr\left[Exp^{IND-CPA-LEFT}(A) = 1 | Y = i\right] \cdot Pr\left[Y = i\right] \\ &= \sum_{i=1}^{m} Pr(i) \cdot \frac{1}{m} \\ Pr\left[Exp^{IND-CPA-RIGHT}(A) = 1\right] &= \sum_{i=1}^{m} Pr\left[Exp^{IND-CPA-RIGHT}(A) = 1 | Y = i\right] \cdot Pr\left[Y = i\right] \\ &= \sum_{i=1}^{m} Pr(i-1) \cdot \frac{1}{m} \\ &= \sum_{i=0}^{m-1} Pr(i) \cdot \frac{1}{m} \end{aligned}$$

Por lo tanto, la probabilidad de que A gane en el experimento IND-CPA de $FHES$ está dada por:

$$\frac{1}{m} \cdot \left| \sum_{i=1}^{m} Pr(i) - \sum_{i=0}^{m-1} Pr(i) \right|$$

Como la resta de sumatorias es la probabilidad de que D pueda distinguir, si ésta no es despreciable, la probabilidad de que A gane tampoco lo es. Luego, D debe distinguir sólo con probabilidad despreciable, puesto que $FHES$ es efectivamente IND-CPA, lo que significa que las ofuscaciones efectivamente satisfacen la propiedad de caja negra.

Con esto, se tiene una construcción de red de mezcla donde la permutación usada por cada servidor de mezcla es desconocida para el mismo, garantizado por la ofuscación, por lo que aún si todos los servidores son intervenidos por un adversario que desee recuperar las permutaciones, no le es posible hacerlo.

4.3. Solución particular

La construcción descrita anteriormente es un ejemplo general de ofuscación de permutaciones en mixnets, la que de por sí no es precisamente eficiente, debido principalmente al uso de un esquema completamente homomórfico $FHES$. Luego, se verá a continuación una red de mezcla reducida ($n = 2$, $m = 1$) donde se puede ocupar una solución alternativa a ocupar $FHES$.

4.3.1. Encriptación de dos formatos

En el artículo de Hohenberger et al.[13] se describe un esquema de encriptación que posee dos formatos. A grandes rasgos, la idea es que al momento de encriptar un texto plano, se escoge uno de los dos formatos de encriptación para el texto de manera que cualquiera que sea el formato escogido, al desencriptar se obtiene el texto plano original.

Más formalmente, se define un esquema de encriptación de dos formatos como una tupla de algoritmos $TFE = (KeyGen, Enc, Dec)$ definidos de la siguiente manera:

- $KeyGen$: $(pk, sk) \leftarrow KeyGen(1^k)$: Genera un par de claves dado un parám. de seguridad k.
- Enc: $c \leftarrow Enc(pk, \beta, m)$: Encripta el texto plano m bajo clave pk y formato de encriptación β.
- Dec: $m \leftarrow Dec(sk, c)$: Desencripta el texto cifrado c si sk es la clave par de pk.

Un esquema es correcto si se cumple que $Dec(sk, Enc(pk, \beta, m)) = m$ para todo par de llaves (pk, sk) válidamente generado y para cualquier $\beta \in \{0, 1\}$.

La idea expuesta en el artículo[13] es reencriptar un texto cifrado convirtiéndolo de un formato ($\beta = 0$) al otro, y ofuscar esa reencriptación. El esquema usado se describe a continuación:

$(q, g, \mathbb{G}, \mathbb{G}_T, e) \leftarrow BMsetup(1^k)$ parámetro común.
$KeyGen(1^k, (q, g, \mathbb{G}, \mathbb{G}_T, e))$:

- $h \xleftarrow{\$} \mathbb{G}$
- $a, b \xleftarrow{\$} \mathbb{Z}_q$
- $pk \leftarrow (h^a, h^b, h)$
- $sk \leftarrow (a, b, h)$
- Output (pk, sk)

$Enc(pk, \beta, m)$:

- $r, s \xleftarrow{\$} \mathbb{Z}_q$
- $t \xleftarrow{\$} \mathbb{G}$
- Si $\beta = 0$, $c \leftarrow \left[0, (h^a)^r, (h^b)^s, h^{r+s} \cdot m, 0\right]$
- Si $\beta = 1$, $c \leftarrow \left[1, e((h^a)^r, t), e((h^b)^s, t), e(h^{r+s} \cdot m, t), t\right]$
- Output c

$Dec(sk, [s, W, X, Y, Z])$:

- $Q \leftarrow Y/(W^{\frac{1}{a}} \cdot X^{\frac{1}{b}})$
- Si $s = 0$, output Q
- Si $s = 1$, por cada m posible, si $e(m, Z) = Q$, output m.

Los dos formatos de encriptación sólo difieren en la existencia de un factor bilinear t en el segundo formato. A grandes rasgos, la idea de re-encriptación proxy ofuscada en el artículo se basa en combinar el cambio de claves y el cambio de formato en una única operacion bilinear sobre cada elemento del texto cifrado. La idea de la solución particular es usar una idea similar, donde el circuito de reencriptación reciba dos textos cifrados bajo el primer formato, realice una operación sobre los elementos del texto cifrado que combinen el cambio de formato y la permutación, y entregue los textos cifrados bajo el segundo formato, pero permutados.

4.3.2. Descripción de la solución

Para crear la solución, primero el generador de claves debe además generar un valor secreto x y calcular $h = d^x$, $g = e^x$ y $\delta = \gamma^{x^{-1}}$ donde d, e y γ son generadores del grupo en el que trabaja el esquema de encriptación. Además, redefinimos la encriptación de formato 0 de la siguiente manera:

$$Enc((h^a, h^b, h, d^a, d^b, d), m) = (0, (h^a)^r, (d^a)^r, (h^b)^s, (d^b)^s, h^{r+s} \cdot g^m, d^{r+s} \cdot e^m, 0)$$

Con esto, se puede comenzar a describir la solución particular.

Primero, el generador de las claves publica una secuencia de bits aleatorios $\sigma_1, \sigma_2, \ldots$ de la siguiente manera:

$$\alpha_i = (\delta^{z_i}, \gamma^{z_i} \cdot {t_i}^{\sigma_i})$$

$$\alpha'_i = (\delta^{z'_i}, \gamma^{z'_i} \cdot {t_i}^{1-\sigma_i})$$

Donde (z_i, t_i) son escogidos al azar y $z_i + z_i' = 1$.

Esta manera de publicar los bits aleatorios es IND-CPA segura puesto que α_i es esencialmente el esquema de ElGamal, donde δ y γ son generadores y ${t_i}^{\sigma_i}$ el mensaje encriptado. Los componentes de α_i' se pueden calcular a partir de los valores de α_i e información pública, por lo que no agrega más información.

Sean los textos cifrados C_1 y C_2 siguientes:

$$C_1 = (0, (h^a)^r, (d^a)^r, (h^b)^s, (d^b)^s, h^{r+s} \cdot g^{m_1}, d^{r+s} \cdot e^{m_1}, 0)$$

$$C_2 = (0, (h^a)^{r'}, (d^a)^{r'}, (h^b)^{s'}, (d^b)^{s'}, h^{r'+s'} \cdot g^{m_2}, d^{r'+s'} \cdot e^{m_2}, 0)$$

Con esto, para calcular la cuarta componente del texto encriptado correspondiente a la mezcla de C_1 y C_2 usando el bit σ_i se calcula lo siguiente:

$$\frac{e(h^{r+s} \cdot g^{m_1}, \gamma^{z_i} \cdot {t_i}^{\sigma_i}) \cdot e(h^{r'+s'} \cdot g^{m_2}, \gamma^{z_i'} \cdot {t_i}^{1-\sigma_i}) \cdot e(d^{r+s} \cdot e^{m_1}, \delta^{z_i'}) \cdot e(d^{r'+s'} \cdot e^{m_2}, \delta^{z_i})}{e(d^{r+s} \cdot e^{m_1} \cdot d^{r'+s'} \cdot e^{m_2}, \delta)}$$

$$= \frac{e(h^{r+s} \cdot g^{m_1}, \gamma^{z_i} \cdot {t_i}^{\sigma_i}) \cdot e(h^{r'+s'} \cdot g^{m_2}, \gamma^{z_i'} \cdot {t_i}^{1-\sigma_i}) \cdot e(d^{r+s} \cdot e^{m_1}, \delta^{z_i'}) \cdot e(d^{r'+s'} \cdot e^{m_2}, \delta^{z_i})}{e(d^{r+s} \cdot e^{m_1}, \delta) \cdot e(d^{r'+s'} \cdot e^{m_2}, \delta)}$$

$$= e(h^{r+s} \cdot g^{m_1}, \gamma^{z_i} \cdot {t_i}^{\sigma_i}) \cdot e(h^{r'+s'} \cdot g^{m_2}, \gamma^{z_i'} \cdot {t_i}^{1-\sigma_i}) \cdot e(d^{r+s} \cdot e^{m_1}, \delta^{z_i'-1}) \cdot e(d^{r'+s'} \cdot e^{m_2}, \delta^{z_i-1})$$

$$= e(h^{r+s} \cdot g^{m_1}, \gamma^{z_i} \cdot {t_i}^{\sigma_i}) \cdot e(h^{r'+s'} \cdot g^{m_2}, \gamma^{z_i'} \cdot {t_i}^{1-\sigma_i}) \cdot e(h^{r+s} \cdot g^{m_1}, \gamma^{z_i'-1}) \cdot e(h^{r'+s'} \cdot g^{m_2}, \gamma^{z_i-1})$$

$$= e(h^{r+s} \cdot g^{m_1}, \gamma^{z_i+z_i'-1} \cdot {t_i}^{\sigma_i}) \cdot e(h^{r'+s'} \cdot g^{m_2}, \gamma^{z_i'+z_1-1} \cdot {t_i}^{1-\sigma_i})$$

$$= e(h^{r+s} \cdot g^{m_1}, {t_i}^{\sigma_i}) \cdot e(h^{r'+s'} \cdot g^{m_2}, {t_i}^{1-\sigma_i})$$

$$= e((h^{r+s} \cdot g^{m_1})^{\sigma_i}, t_i) \cdot e((h^{r'+s'} \cdot g^{m_2})^{1-\sigma_i}, t_i)$$

$$= e(h^{(r+s)\cdot\sigma_i} \cdot g^{m_1 \cdot \sigma_i} \cdot h^{(r'+s')\cdot(1-\sigma_i)} \cdot g^{m_2 \cdot (1-\sigma_i)}, t_i)$$

$$= e(h^{((r+s)\cdot\sigma_i)+((r'+s')\cdot(1-\sigma_i))} \cdot g^{(m_1\cdot\sigma_i)+(m_2\cdot(1-\sigma_i))}, t_i)$$

Calcular las dos componentes restantes es muy similar:

$$\frac{e((h^a)^r, \gamma^{z_i} \cdot {t_i}^{\sigma_i}) \cdot e((h^a)^{r'}, \gamma^{z_i'} \cdot {t_i}^{1-\sigma_i}) \cdot e((d^a)^r, \delta^{z_i'}) \cdot e((d^a)^{r'}, \delta^{z_i})}{e((d^a)^r \cdot (d^a)^{r'}, \delta)}$$

$$\frac{e((h^b)^s, \gamma^{z_i} \cdot {t_i}^{\sigma_i}) \cdot e((h^b)^{s'}, \gamma^{z_i'} \cdot {t_i}^{1-\sigma_i}) \cdot e((d^b)^s, \delta^{z_i'}) \cdot e((d^b)^{s'}, \delta^{z_i})}{e((d^b)^s \cdot (d^b)^{s'}, \delta)}$$

Con esto queda uno de los dos mensajes encriptados en el formato 1 de la siguiente manera:

$$C_1' = (1, e((h^a)^{(r\cdot\sigma_i)+(r'\cdot(1-\sigma_i))}, t_i), e((h^b)^{(s\cdot\sigma_i)+(s'\cdot(1-\sigma_i))}, t_i), e(h^{((r+s)\cdot\sigma_i)+((r'+s')\cdot(1-\sigma_i))} \cdot g^{(m_1\cdot\sigma_i)+(m_2\cdot(1-\sigma_i))}, t_i), t_i)$$

Para calcular C_2' se procede de la misma manera que para C_1', pero intercambiando los roles de las componentes de α_i y α_i'.

Para asegurar la seguridad de esta construcción, los textos cifrados son reencriptados (sin cambio de formato) antes y después del proceso, tal y como se describe en el artículo de Hohenberger[13].

La demostración de que esto es efectivamente una ofuscación es similar a la usada en el artículo de Hohenberger.

4.4. Ofuscación implica privacidad de mezcla

Se ha mostrado una construcción de red de mezcla que utiliza ofuscación para esconder a los propios servidores de mezcla la permutación que usan, pero no se ha dicho nada sobre cómo es la permutación misma. Para que la red de mezcla garantice anonimato, las permutaciones que se utilizan en cada servidor de mezcla deben ser escogidas de manera uniforme, y además la permutación final resultante debe ser uniforme, o de lo contrario un adversario podría obtener información usando la distribución de las permutaciones.

Para demostrar que una red de mezcla garantiza anonimato, se puede usar un experimento de indistinguibilidad con adversario A y sistema $Sist$:

- $Sist$ construye una red de mezcla Mix que recibe un vector de textos cifrados $\vec{C}_{pre}$, y entrega un vector de textos cifrados permutados $\vec{C}_{post}$ e información adicional sobre la mezcla ρ (Red de mezcla real).
- $Sist$ construye un permutador *Shuffle* que recibe un vector de mensajes, los mezcla usando una permutación escogida uniformemente, y entrega el vector de mensajes permutados (Red de mezcla falsa).
- A le entrega un vector de textos planos $\vec{m}$ a $Sist$.
- $Sist$ encripta $\vec{m}$, generando $\vec{C}_{pre}$ y se lo envía a Mix.
- $Sist$ envía $\vec{m}$ a *Shuffle*.
- Mix envía $\vec{C}_{post}$ y ρ a $Sist$.
- $Sist$ desencripta $\vec{C}_{post}$ obteniendo $\vec{m}_0$.
- $Shuffle$ envía $\vec{m}_1$ a $Sist$.
- $Sist$ escoge un bit $b \xleftarrow{\$} \{0,1\}$.
- $Sist$ entrega $\vec{m}_b$, $\vec{C}_{pre}$, $\vec{C}_{post}$ y ρ a A.
- A entrega un bit b' que indica por cuál red de mezcla cree que su vector $\vec{m}$ fue permutado.

La idea es que si un adversario puede distinguir la red de mezcla real de la falsa, entonces o bien la información adicional ρ entrega datos sobre la permutación usada, o bien la permutación no es uniforme. La idea es que si la red de mezcla está bien construida y la ofuscación es correcta, la información adicional ρ no debería divulgar información alguna sobre la permutación escogida, pero el método para escoger esa permutación es el que podría estar entregando permutaciones no uniformes.

De manera más formal, el experimento para el presente trabajo sería el siguiente:

$EXP^{A,Sist}_{IND-MIX}(sk, pk, pk_F, O\vec{b}C)$:

- $\{\vec{m}, st\} \leftarrow A(choose, pk, pk_F, O\vec{b}C)$
- $\vec{C_{pre}} \leftarrow Enc(pk, \vec{m})$
- $\{\vec{C_{post}}, \rho\} \leftarrow Mix(\vec{C_{pre}})$
- $\vec{m_0} \leftarrow Dec(sk, \vec{C_{post}})$
- $\vec{m_1} \leftarrow Shuffle(\vec{m})$
- $b \xleftarrow{\$} \{0, 1\}$
- $g \leftarrow A(find, \vec{C_{pre}}, \vec{C_{post}}, \rho, \vec{m_b}, st)$
- Si $g = b$: Output 1
- Output 0

Para demostrar que la red de mezcla cumple con el anonimato, supóngase primero que existe un adversario A que gana en el experimento IND-MIX con probabilidad no despreciable $\frac{1}{2} + P_A$. Por el teorema 2.3 del artículo de Hohenberger[13], existe un adversario A' que gana en el experimento IND-MIX modificado donde el adversario sólo posee acceso de caja negra al circuito real. Luego, podemos crear un adversario B que gana en el experimento IND-CPA del esquema de encriptación inicial de la siguiente manera:

$EXP^{B}(pk)$:

- B ejecuta $A'(choose)$: $(\vec{m}, st) \leftarrow A'^{C}(choose, pk, pk_F)$.
- B crea dos mensajes a partir de $\vec{m}$: $(\vec{m_0}, \vec{m_1}) \leftarrow (\pi_0(\vec{m}), \pi_1(\vec{m}))$.
- B encripta $\vec{m_0}$: $\vec{C_{pre}} \leftarrow Enc(pk, \vec{m_0})$.
- B envía $(\vec{m_0}, \vec{m_1})$ a su oráculo LoR: $\vec{C_{post}} \leftarrow LoR(\vec{m_0}, \vec{m_1})$.
- B ejecuta $A'(find)$: $g \leftarrow A'^{C}(find, \vec{C_{pre}}, \vec{C_{post}}, \vec{m_0}, st)$.
- B entrega g.

Si A efectivamente rompe IND-MIX con ventaja P_A, entonces A' también rompe IND-MIX con ventaja P_A, y por ende B rompe IND-CPA con la misma ventaja P_A. Por lo tanto, si el esquema de encriptación es efectivamente IND-CPA, entonces P_A debe ser despreciable, y por ende, A' (y luego A) no puede ganar el experimento IND-MIX con probabilidad muy superior a $\frac{1}{2}$. Por lo tanto, si la permutación usada es efectivamente uniforme, entonces ofuscarla en una red de mezcla implica anonimato.

5. Conclusiones

En el presente trabajo, se mostró la manera de ofuscar una permutación y su aplicación para la construcción de una red de mezcla totalmente resistente a adversarios que deseen rastrear los mensajes. Para ello primero se creó una ofuscación para realizar una permutación y luego se extendió para realizar varias permutaciones seguidas.

Con esto, se muestra que, si bien no existe un ofuscador genérico para primitivas criptográficas, sí se pueden utilizar los resultados positivos más abiertos como el de Ding y Gu[7], para crear ofuscaciones de construcciones tan complejas como una red de mezcla.

La construcción general no debe tomarse más que como una prueba de concepto, pues el uso de un esquema de encriptación completamente homomórfico (y usarlo además para algo como evaluaciones de AES) lo hacen sumamente impráctico para el uso en el mundo real. La solución particular, por otra parte, muestra que si se reducen las entradas y los servidores de mezcla a números pequeños, se puede obtener algo que funciona, aunque queda para trabajo futuro qué tan escalable puede ser.

A futuro, sería deseable encontrar un método para distribuir la generación de las claves secretas, los circuitos ofuscados y la desencriptación final. Para lo último existen esquemas de encriptación umbrales completamente homomórficos, como el de Myers et al.[15], pero falta por ver si son compatibles con las evaluaciones que se realizan en la construcción. Para los primeros, distribuir la generación quita la dependencia de la construcción de un agente externo confiable, que es una suposición fuerte, y deposita la confianza en un grupo de servidores donde no es necesario confiar en todos.

6. Referencias

[1] B. Adida, D. Wikström, *How to shuffle in public*. Proc. TCC 2007. Págs.555-574, Springer-Verlag, 2007.

[2] B. Barak, O. Goldreich, R. Impagliazzo, S. Rudich, A. Sahai, S. Vadhan, K. Yang, *On the (im)possibility of obfuscating programs*. Proc. CRYPTO '01. Págs.1-18, Springer-Verlag, 2001.

[3] M. Blaze, G. Bleumer, M. Strauss, *Divertible protocols and atomic proxy cryptography*. EUROCRYPT '98. LNCS, Vol.1403, págs.127-144, Springer-Verlag, 1998.

[4] R. Canetti, R.R. Dakdouk, *Obfuscating point functions with multibit output*. N.P. Smart (ed.) EUROCRYPT 2008. LNCS, Vol.4965, págs.489-508, 2008.

[5] D. Chaum, *Untraceable electronic mail, return addresses, and digital pseudonyms*. Communications of the ACM, Vol.24, No. 2, págs.84-88, 1981.

[6] W. Diffie, M.E. Hellman, *New directions in cryptography*. IEEE Transactions on Information Theory, Vol.22, No. 6, págs.644-654, 1976.

[7] N. Ding, D. Gu, *A note on obfuscation for cryptographic functionalities of secret-operation then public-encryption* M. Ogihara, J. Tarui (eds.) TAMC 2011, LNCS 6648, págs.377-389, 2011.

[8] T. El Gamal, *A public key cryptosystem and a signature scheme based on discrete logarithms*. IEEE Transactions on Information Theory, Vol.31, No. 4, págs.469-472, 1985.

[9] C. Gentry, *Fully homomorphic encryption using ideal lattices*. Proc. STOC 2009, págs.169-178, 2009.

[10] C. Gentry, S. Halevi, N.P. Smart, *Homomorphic evaluation of the AES circuit*. IACR Cryptology ePrint Archive, Report 2012/099, 2012. http://eprint.iacr.org/

[11] P. Golle, M. Jakobsson, A. Juels, P. Syverson, *Universal re-encryption for mixnets*. Topics in Cryptology (CT-RSA), Vol.2964, págs.163-178, 2004.

[12] S. Hada, *Secure obfuscation for encrypted signatures*. H. Gilbert (ed.) EUROCRYPT 2010. LNCS, Vol.6110, págs.92-112, 2010.

[13] S. Hohenberger, G. Rothblum, A. Shelat, V. Vaikuntanathan, *Securely obfuscating re-encryption*. S. Vadhan (ed.) TCC 2007. LNCS, Vol.4392, págs.233-252, 2007.

[14] J. Katz, Y. Lindell, *Introduction to modern cryptography*. Chapman & Hall/CRC Press, 2007.

[15] S. Myers, M. Sergi, A. Shelat, *Threshold fully homomorphic encryption and secure computation*. IACR Cryptology ePrint Archive, Report 2011/454, 2011. http://eprint.iacr.org/

[16] R. Ostrovsky, W.E. Skeith III, *Private searching on streaming data*. IACR Cryptology ePrint Archive, Report 2005/242, 2005. http://eprint.iacr.org/

[17] C. Park, K. Itoh, K. Kurosawa, *Efficient anonymous channel and all/nothing election scheme*. EUROCRYPT '93. LNCS, Vol.765, págs.248-259, Springer-Verlag, 1994.

[18] K. Sako, J. Kilian, *Receipt-free mix-type voting scheme*. EUROCRYPT '95. LNCS, Vol.921, págs.393-403, Springer-Verlag, 1995.

[19] H. Wee, *On obfuscating point functions* H.N. Gabow, R. Fagin (eds.) STOC 2005, págs.523-532, 2005.