Protocolo de computación multiparte para funciones simétricas y su aplicación a votación electrónica

Alex Escala
Dept. d'Enginyeria Telemàtica
Universitat Politècnica de Catalunya
Email: alex.escala@ma4.upc.edu

Sandra Guasch
Scytl Secure Electronic Voting
Email: sandra.guasch@scytl.com

Paz Morillo
Dept. de Matemàtica Aplicada IV
Universitat Politècnica de Catalunya
Email: paz@ma4.upc.edu

***Abstract*—La computación multiparte segura es una primitiva criptográfica que permite a un conjunto de participantes evaluar una función de sus entradas de modo que solo obtengan el resultado, es decir, manteniendo sus entradas privadas. En el congreso CRYPTO 2011, Halevi et al. presentaron una familia de protocolos de computación multiparte seguros en el modelo cliente-servidor, con mínima interacción de los participantes. Una de las familias de funciones estudiadas por los autores es la de las funciones simétricas binarias, es decir, funciones binarias cuya salida solo depende del número de entradas con valor 1.**

En este trabajo mejoramos el protocolo de computación multiparte segura de funciones simétricas binarias dado por Halevi et al. Concretamente, nuestro protocolo admite más de un servidor, lo que redunda en un mayor nivel de simultaneidad en las comunicaciones cliente-servidor, sin menoscabar la seguridad del protocolo.

Además, en este artículo usamos el protocolo propuesto como base de un esquema de votación electrónica. La principal ventaja del sistema de voto que proponemos es que, aunque los servidores de voto sean corruptos, se garantiza cierta privacidad del voto.

I. Introducción

La computación multiparte (MPC, del inglés *Multiparty Computation*) es una primitiva criptográfica introducida por primera vez en [2]. El objetivo de los protocolos de MPC es permitir a un conjunto de participantes calcular el resultado de una función de sus entradas privadas, revelando el mínimo de información.

En el artículo de Halevi et al. [1], los autores proponen un sistema de MPC en el modelo cliente-servidor, en el que se requiere una interacción mínima de los participantes con el servidor. Concretamente, se plantea un escenario en el que los participantes se conectan una única vez al servidor de forma no simultánea, reciben información sobre los cálculos parciales hechos por participantes que se han conectado previamente, y ejecutan su parte del protocolo. El sistema propuesto proporciona seguridad en el sentido de que, un atacante que controle ciertos participantes maliciosos, sólo puede llegar a obtener la información que se infiere de evaluar la función cambiando las entradas que maneja (las de los participantes maliciosos).

En dicho artículo se define una descomposición de una función f, que permitirá ejecutar el protocolo de MPC con las propiedades descritas en el párrafo anterior.

Una familia de funciones estudiada en [1] es la de las funciones simétricas binarias, es decir, aquellas en que la salida depende del número de bits a la entrada con valor 1. Ejemplos de estas funciones son las AND, OR, o funciones MAYORÍA, en las que la salida es 1 si la mitad o más de las entradas valen 1.

Una aplicación natural de la computación multiparte para funciones simétricas binarias está en la votación electrónica. Por ejemplo, un referéndum se puede interpretar como el cálculo de la función MAYORÍA.

Usando los protocolos propuestos en [1] se obtiene un esquema de votación electrónica con una mayor privacidad que otros sistemas conocidos en la literatura. En concreto, aunque los servidores de voto fueran corruptos, la privacidad del voto estaría garantizada. No obstante, en un entorno de votación electrónica es importante cumplir con ciertos requisitos que aseguren que el sistema es práctico. Por ejemplo, es de especial importancia soportar un determinado nivel de concurrencia de los participantes para agilizar el proceso electoral. En este artículo proponemos una ampliación del sistema propuesto en [1] para soportar varios participantes concurrentes manteniendo un nivel de seguridad similar. En la sección VI presentamos un sistema de voto basado en este protocolo y algunas soluciones que se le pueden incorporar para que sea apto para ser usado en un entorno de votación real.

II. Descomposición con mínima revelación

Sea $f(x_1, x_2, \ldots, x_n)$ una función, en [1] se define la descomposición de esta función en funciones f_i, tales que:

$f(x_1, x_2, \ldots, x_n) = f_n(\ldots f_2(f_1(x_1), x_2) \ldots, x_n)$.

Así pues, en este esquema el participante P_i recibe el resultado del cálculo parcial efectuado por los participantes anteriores, y_{i-1} y aporta su entrada privada para calcular $y_i = f_i(y_{i-1}, x_i)$.

Para proporcionar privacidad a los participantes, tanto sobre los resultados intermedios como sobre las entradas que aporta cada uno (recordemos que en un esquema de MPC sólo interesa conocer el cálculo final), no todas las descomposiciones de la función f son igual de idóneas, interesan las que revelan la mínima cantidad de información: descomposiciones de mínima revelación (en inglés *Minimum Disclosure decompositions*).

Para definir este tipo de descomposición, primero se define el concepto de función residual. Dado el vector $\vec{x} = (x_1, x_2, \ldots, x_n)$, la i-ésima función residual es

$$g_i^{\vec{x}}(z_{i+1}, z_{i+2}, \ldots, z_n) = f(x_1, \ldots, x_i, z_{i+1}, \ldots, z_n).$$

Observemos que, dada la descomposición de una función y la salida y_i del participante P_i, el adversario que corrompe el resto de participantes $(P_{i+1}, \ldots, P_n)$, puede calcular la función residual. Precisamente, la descomposición es de mínima revelación cuando a partir de la función residual g_i se puede calcular la salida y_i del participante P_i, es decir, $f_i(x_i, f_{i-1}(x_{i-1}, \ldots, f_1(x_1)) \ldots)$.

La intuición de este concepto es la siguiente: la función residual, que recordemos que es algo que el adversario podrá calcular, revela cierta (normalmente poca) información. Lo que queremos es que la descomposición no revele más información que la que revela la función residual. Si no se puede calcular la función parcial a partir de la residual, quiere decir que la función parcial filtra demasiada información.

Un tipo de funciones que se pueden descomponer de forma que se revele la mínima cantidad de información, son las *funciones simétricas binarias*. Como hemos comentado en la sección anterior, las funciones simétricas binarias son aquellas funciones con entrada binaria (0 ó 1) tales que la salida sólo depende del número de entradas que valen 1 (y no de la posición de estas entradas). Las funciones simétricas binarias se pueden representar mediante una tabla de verdad: un vector donde la posición i-ésima toma el valor de la función cuando en la entrada hay i bits de valor 1.

Por ejemplo, para una función PARIDAD de cuatro entradas, el vector de la tabla de verdad sería $(1, 0, 1, 0, 1)$: salida 1 cuando no hay ningún valor a 1 en la entrada, salida 0 cuando únicamente hay un bit a 1, salida 1 cuando hay dos bits a 1,etc.

La descomposición de mínima revelación de una función simétrica binaria es la siguiente: para una función f con tabla de verdad $(h(0), h(1), \ldots, h(n))$ (usamos h para denotar la función que depende únicamente del número de entradas x_i con valor 1), la función parcial f_i es la tabla de verdad de $g_i^{\vec{i}}$, función residual definida anteriormente. Es fácil ver que se trata de una descomposición de mínima revelación.

Veamos un ejemplo: la función MAYORÍA de 3 variables. Esta función tiene por tabla de verdad $(0, 0, 1, 1)$. Supongamos que hay tres participantes con entrada 1 0 y 1 respectivamente. El primer participante calculará la tabla de verdad de la residual g_1^1, que es $(0, 1, 1)$ (efectivamente, si en la función MAYORÍA fijamos la primera entrada en 1, esta es la tabla de verdad que nos queda). El segundo participante tendrá como salida la tabla de la verdad de g_2^{10}, que es $(0, 1)$. Finalmente, el tercer participante tendrá como salida la tabla de verdad de g_3^{101}, que es (1).

Se puede comprobar que, calcular la tabla de verdad de la función residual $g_i^{x_1,\ldots,x_i}$ equivale a tomar la tabla de verdad de la residual $g_{i-1}^{x_1,\ldots,x_{i-1}}$ y quitar el primer o el último componente del vector, según la entrada sea 1 ó 0 respectivamente. Esto es importante, ya que se usará en el protocolo de computación segura explicado en la siguiente sección.

III. Protocolo para funciones simétricas binarias con un servidor

En esta sección, explicamos el protocolo para calcular de forma segura una función binaria simétrica entre varios participantes, según el método de [1].

En el protocolo hay n participantes $P_1, \ldots, P_n$. Asumiremos que en la fase de inicialización los participantes han generado o han recibido de forma segura un par de claves (pk_i, sk_i) correspondientes a un cifrado ElGamal [6]. En el protocolo también participa un servidor S con claves pública y privada (pk_s, sk_s). Es posible utilizar otros algoritmos de cifrado, siempre que tengan ciertas propiedades detalladas en [1].

En una primera fase, el primer participante construye la tabla de la verdad de la función simétrica binaria a calcular $\mathcal{T} = (t_0, \ldots, t_n)$, donde $t_i = h(i)$, por simplicidad supondremos $t_i \in \{0, 1\}$. Después, procede a eliminar un componente del vector según su elección (denotaremos la elección del i-ésimo participante como x_i). En el caso de que el primer participante escoja $x_1 = 1$, procederá a eliminar el primer componente del vector de la tabla de verdad (es decir, t_0). En el caso de que el participante escoja $x_1 = 0$, eliminará el último componente del vector, t_n. Finalmente, el participante P_1 cifrará el resto de la tabla de verdad (cada elemento por separado) con un cifrado ElGamal usando como clave pública el producto de $pk_2, \ldots, pk_n, pk_s$ claves públicas de los participantes restantes y del servidor. El participante P_1 envía el vector cifrado al servidor S.

El comportamiento del resto de participantes es ligeramente distinto, ya que ellos no crean la tabla de verdad desde un principio, sino que reciben del servidor S la tabla de verdad modificada por los participantes anteriores. Concretamente, el participante i-ésimo P_i recibe un vector de elementos cifrados, cuyo contenido en claro está formado por las entradas de la tabla de verdad que no han eliminado los participantes anteriores (imaginemos que de los participantes anteriores uno ha elegido 0, y los otros 1, el vector de cifrados que reciba el participante i-ésimo será el cifrado de cada elemento de $\mathcal{T}' = (t_{i-2}, \ldots, t_{n-1})$). El participante i-ésimo podrá deducir, del tamaño del vector de cifrados, cuántos participantes han actuado, pero no el contenido de sus opciones, ya que las entradas de la tabla de verdad que recibe están cifradas. Siguiendo con el proceso, el participante P_i elimina la última entrada del vector de cifrados si $x_i = 0$, o la primera si $x_i = 1$, obteniendo un vector cifrado de $n - i + 1$ componentes. El participante realiza un descifrado parcial de cada elemento del vector utilizando su clave privada sk_i y hace un recifrado (es decir, una realeatorización) de cada elemento del vector utilizando el producto de las claves públicas del resto de participantes y el servidor $pk_{i+1}, \ldots, pk_n, pk_s$, como en el párrafo anterior. Finalmente, devuelve el resultado al servidor.

El último participante del proceso tiene como salida el cifrado de la única entrada restante de la tabla de verdad, con la clave pública del servidor pk_s. El servidor al recibirla puede descifrarla con su clave secreta sk_s y publicar el resultado.

La seguridad del protocolo contra adversarios pasivos (semi-honestos) se basa en el hecho que la descomposición utilizada es de mínima revelación y que ElGamal es un cifrado IND-CPA. La prueba de seguridad, que se encuentra en [1] (y no vamos a reproducir) se basa en ver que la vista del adversario puede ser simulada sin ayuda de una tercera parte de confianza. Esto es, conociendo el resultado de la función, se puede simular el protocolo con el que el adversario interactúa sin necesidad de conocer las entradas secretas de los adversarios no corruptos. Este procedimiento es muy habitual en pruebas de protocolos MPC.

Concretamente, se diferencian dos casos según el servidor S sea corrupto o no. Si no es corrupto, se llega a la conclusión que lo único que aprende el adversario es el resultado de la función, es decir, no aprende ninguna información adicional. Cuando el servidor es corrupto y los últimos i participantes $P_{n-i+1}, \ldots, P_n$ también, la única información que se revela es la $i-1$-ésima función residual. Es importante remarcar que corromper a un participante cuya posición está entre dos participantes honestos no tienen ningún impacto en la seguridad del esquema.

En el caso de adversarios activos (maliciosos), la seguridad se mantiene usando pruebas de conocimiento cero (en inglés, *zero-knowledge proofs*), que garantizan que todos los adversarios siguen el protocolo.

IV. Protocolo para funciones simétricas binarias con dos servidores

En esta sección presentamos nuestra primera contribución: la generalización del protocolo descrito en [1] para soportar cierto nivel de concurrencia. En este caso, tendremos dos servidores que irán aceptando participantes, de modo que dos participantes pueden ejecutar el proceso en paralelo.

Dividiremos el protocolo en tres partes: una fase de configuración, donde se generan las claves criptográficas necesarias y los vectores formados por los elementos de la tabla de verdad cifrados (uno para cada servidor, S_1 y S_2); una fase de participación donde los participantes acceden a los servidores y realizan su elección ($x_i = 0$ o $x_i = 1$) siguiendo el protocolo explicado en la sección anterior; y una fase de combinación donde los resultados obtenidos por cada servidor se combinan para obtener una salida única del protocolo, tal y como sucede en el original.

Una solución trivial para adaptar el protocolo para dos servidores consiste en construir un vector con los elementos de la tabla de verdad para $n/2$ entradas para cada servidor, dejar a cada mitad de participantes acceder a uno de los servidores para realizar su elección, y al final del protocolo descifrar el resultado obtenido en cada servidor. Sin embargo, de este modo las propiedades del protocolo original no se mantienen, ya que la privacidad de cada participante se mantiene únicamente entre los $n/2$ participantes de su servidor. Para mantener la privacidad entre los n participantes, proponemos un proceso de combinación en el que se inicia el protocolo con el vector completo de elementos de la tabla de verdad en cada servidor y al final del proceso, los dos subvectores residuales (asumiremos que los subvectores tendrán longitud $n/2+1$) se combinan para obtener una única salida, manteniendo las propiedades de seguridad del protocolo original.

Para ello, usaremos una propiedad del esquema que nos permitirá alinear los subvectores con el vector de la tabla de verdad original sin revelar el contenido de estos subvectores de forma individual:

Consideremos el subvector de la tabla de verdad (en claro) que corresponde a S_1 después de que todos los participantes hayan actuado. Este subvector determina el número total de 1's escogidos por los participantes que interactúan con S_1: una vez situado el subvector dentro la tabla de verdad el número de 1's nos lo dará la posición en que se encuentra.

Por ejemplo, la tabla de verdad de una función f de 4 entradas es $(1,0,1,1,0)$. *Lo que queda* de tabla de verdad en el servidor S_1 es $(0,1,1)$. Se puede localizar en la tabla de verdad, como $(1,$ **0, 1, 1**$,0)$. Esto nos dice que el número de 1's que había en la entrada es 1 (se ha suprimido una componente de la izquierda y una de la derecha).

Sin embargo, existe la posibilidad de que este subvector aparezca más de una vez en la tabla de verdad. Veamos un ejemplo: supongamos una función con tabla de verdad $(1,0,1,0,1,0,1)$. En el servidor S_1, se tiene el subvector $(0,1,0,1)$, que puede corresponder a tener 1 o 3 unos a la entrada. Por otra parte, el servidor S_2 tiene como subvector $(1,0,1,0)$, que corresponde a 0 ó 1 unos en la entrada. En cualquier caso, sea cuál sea el número de 1's en total (puede ser $1, 3$ o 5), la salida de la función será 0.

La conclusión es que no importa la posición exacta en la que está el subvector, sino una de las posibles posiciones. La salida de la función será la misma sea cual sea la entrada entre las que dan el mismo subvector.

Para mantener las propiedades de privacidad del protocolo original, trabajaremos en la alineación de los dos subvectores cifrados con la tabla de verdad, para así obtener en claro únicamente el valor final. En el protocolo intervendrán, además de servidores y participantes, las entidades F_1, F_2 y F_3 encargadas de hacer ciertos procesos adicionales necesarios para el desarrollo seguro del protocolo. A continuación se muestra una descripción detallada de las operaciones que se efectúan en cada fase:

A. Fase de configuración

En el protocolo hay n participantes $P_1, \ldots, P_n$. Para una mayor simplicidad, asumiremos que n es un número par y $m = n/2$. Inicialmente, se configuran dos servidores distintos S_1 y S_2 para que cada uno atienda a la mitad de los participantes:

$$\{P_1, P_3, \cdots, P_{2m-1}\} \longleftrightarrow S_1$$

$$\{P_2, P_4, \cdots, P_{2m}\} \longleftrightarrow S_2$$

Se genera un par de claves (pública y privada) (pk_s, sk_s). A diferencia del esquema original, los servidores no tienen sus propias claves, sino éstos reciben la clave pública pk_s y la entidad F_3, que será la encargada de descifrar el resultado al final del protocolo, posee la clave privada sk_s.

Para que el coste del cálculo del resultado final del protocolo no recaiga íntegramente en la última etapa, realizaremos todas las operaciones posibles en esta fase. La entidad F_1:

- Lista los subvectores formados por $m+1$ entradas consecutivas de la tabla de verdad, así como la posición de su primera entrada.
- Para cada uno de estos subvectores, cifra todas sus componentes con la clave pública pk_s y codifica el vector de cifrados resultante, según se explica en la sección IV-D, obteniendo un único cifrado por cada subvector (manteniendo junto a dicho cifrado la información de la posición antes mencionada).
- Para cada posible pareja de estos cifrados, calcula una combinación usando dos exponentes r_1 y r_2 seleccionados de forma aleatoria por F_1 (tal como se explica en la sección IV-E), asociando a la misma un índice k que es la suma de las posiciones de cada cifrado de la pareja.
- Para cada uno de estos índices $k = 0, 1, \ldots, n$, mezcla todas las combinaciones de parejas de cifrados asociados a él para eliminar la relación con los índices individuales de cada subvector (sólo nos interesa k). Este proceso se realiza mediante una mix-net.

B. Fase de participación

El participante P_1 accede al servidor S_1 y suprime la primera o la última componente de la tabla de verdad cifrada, según sea su entrada. A continuación, descifra las componentes de la tabla resultante con su clave privada sk_1 y las recifra con las claves públicas $pk_3, pk_5, \ldots, pk_{2m-1}, pk_s$ (como antes, usa el producto de ellas). Análogamente, el participante P_2 accede al servidor S_2, suprime la primera o la última componente, según sea su entrada, descifra las componentes de la tabla resultante con su clave privada sk_2 y, en este caso, las recifra con las claves públicas $pk_4, pk_6, \ldots, pk_{2m}, pk_s$.

Los siguientes pasos son idénticos a los explicados en la sección III: cada participante recibe de su servidor el cifrado de lo que queda de tabla de verdad, suprime la primera o la última componente, descifra las componentes con su clave privada y recifra, usando como clave pública el producto de claves públicas de los participantes restantes asignados a ese servidor.

Al final de este proceso, tanto el servidor S_1 como el servidor S_2 tendrán un subvector de $m+1$ entradas cifradas.

C. Fase de combinación

En esta fase trataremos de alinear cada subvector compuesto por entradas cifradas con la tabla de verdad original, sin revelar información de cada subvector por separado. Como hemos explicado anteriormente, la alineación de los dos vectores nos dará el valor de la tabla de verdad que hubiera quedado al final de la elección si hubieramos trabajado en un esquema con un único servidor, como en la sección III. Para ello, se realizan los siguientes pasos:

- Cada servidor codifica el subvector de cifrados que ha obtenido, según se explica en la sección IV-D, obteniendo un único cifrado por servidor.
- La entidad F_2 recibe ambos cifrados (uno por servidor), los combina tal como se explica en IV-E usando los mismos exponentes r_1 y r_2 del proceso previo a la interacción con los participantes, y los transmite a la entidad F_3.
- Por último, para cada índice k y para cada combinación de parejas de cifrados asociadas al mismo, la entidad F_3 divide componente a componente (cada componente del cifrado ElGamal) el resultado del item anterior con dicha combinación y descifra el resultado. Este proceso se repite hasta que encuentra un índice k_0 para el cual el descifrado vale 1 (más adelante se justificará este valor). La salida de la función es $h(k_0)$.

D. CODIFICACIÓN DE UN VECTOR DE CIFRADOS

Supongamos, por sencillez, que el rango de la función simétrica binaria es $\{0, 1\}$. Entonces, cada cifrado de un elemento del subvector es un cifrado ElGamal de g^1 o g^0. Interpretando el subvector como un número en binario (por ejemplo, $(0, 1, 0, 1)_{bin} = 5_{dec}$), se calculará el cifrado de g^d, donde d es el número que representa el valor decimal correspondiente al vector. Como ejemplo, a partir del cifrado de (g^0, g^1, g^0, g^1), obtendremos el cifrado de g^5 calculando $E(g^0)^{2^3} \odot E(g^1)^{2^2} \odot E(g^0)^{2^1} \odot E(g^1)^{2^0}$, donde $\odot$ es la multiplicación componente a componente (para tener en cuenta los dos componentes del cifrado), gracias a las propiedades homomórficas del algoritmo de cifrado. Observemos que si la función simétrica binaria tuviera rango de dimensión u, el subvector se interpretaría como un número en base u.

E. COMBINACIÓN DE DOS CIFRADOS

Sean r_1 y r_2 dos números de $\mathbb{Z}_q$, donde q es el orden de g, el elemento que se usa como base en el cifrado ElGamal. Para combinar dos cifrados C_1 y C_2 usando exponentes r_1 y r_2 se calcula $C_{1,2} = C_1^{r_1} \odot C_2^{r_2}$.

V. ANÁLISIS DEL PROTOCOLO

A. Exactitud

Si todos los participantes son honestos, el protocolo da como salida el valor de la función f en las entradas de los participantes. Vamos a verlo.

Recordemos que, antes de la interacción de los participantes, para cada valor de i desde 0 hasta $n/2$, F_1 toma el subvector de la tabla de verdad formado por $(h(i), \ldots, h(m+i))$, cifra cada elemento y codifica el vector de cifrados según lo explicado en la sección IV-D obteniendo el cifrado C_i. Después de la combinación, obtiene $C_{i_1,i_2} = C_{i_1}^{r_1} \odot C_{i_2}^{r_2}$ para todo i_1, i_2 desde 0 hasta $n/2$.

Posteriormente, F_1 mezcla los elementos C_{i_1,i_2} del siguiente modo: para todo k desde 0 hasta n, se definen los conjuntos $A_k = \{C_{i_1,i_2} | i_1 + i_2 = k\}$. Para cada valor de k, se mezclan los elementos de A_k mediante una mix-net.

Por otra parte, el servidor S_j codifica el vector de cifrados obtenido después de la interacción con todos los usuarios, obteniendo el cifrado que llamaremos $R^{(j)}$ y la entidad F_2 combina $R^{(1)}$ y $R^{(2)}$ con los exponentes r_1 y r_2.

Finalmente, F_3 divide (componente a componente) el resultado de la combinación realizada por F_2, por los C_{i_1,i_2} calculados por F_1, obteniendo para todo k desde 0 hasta n los conjuntos $B_k = \{(R^{(1)})^{r_1} \odot (R^{(2)})^{r_2} / C_{i_1,i_2}$ t.q. $i_1 + i_2 = k\}$ (donde abusando de la notación entendemos la división componente a componente). Antes de proseguir, recordemos que los elementos de B_k serán elementos de la forma $\left(E(g^{d^{(1)}})^{r_1} \odot E(g^{d^{(2)}})^{r_2}\right) / \left(E(g^{d_{i_1}})^{r_1} \odot E(g^{d_{i_2}})^{r_2}\right)$ con $i_1 + i_2 = k$, cuyas componentes son $(g^r, h^r g^{r_1(d^{(1)} - d_{i_1}) + r_2(d^{(2)} - d_{i_2})})$ siendo r un aleatorio.

Ahora, un elemento de B_k será el cifrado del 1 cuando $r_1(d^{(1)} - d_{i_1}) + r_2(d^{(2)} - d_{i_2})$ sea 0 y, al ser r_1 y r_2 aleatorios, esto implica que $d^{(1)} = d_{i_1}$ y $d^{(2)} = d_{i_2}$ excepto con probabilidad despreciable. Lo cual implica que el número total de unos es k y, al estar B_k mezclado, no se puede deducir cuantos de los participantes de cada servidor tienen una entrada uno.

Así pues, lo que hará F_3 es descifrar todos los elementos de B_k para todo k hasta encontrar algún elemento, de un cierto B_{k_0}, cuyo descifrado sea igual a 1, con lo que terminará mostrando como salida $h(k_0)$.

B. Seguridad

La seguridad del esquema es la siguiente: si todas las entidades son honestas, la única información que se conoce al final del protocolo es el valor de la función. Por otra parte, aunque todas las autoridades que participan en el protocolo (asumiendo que la generación de claves de los participantes es segura) tengan un comportamiento malicioso, la información que pueden llegar a saber es el número de entradas con valor 1 en cada servidor. Finalmente, para poder llegar a saber la entrada de un participante, todas las entidades menos éste deben tener un comportamiento malicioso (asumiendo, otra vez, que no se compromete la clave secreta del participante en cuestión).

La seguridad depende de las siguientes premisas: el protocolo para un servidor, explicado en la sección III, es seguro. La prueba de seguridad se presenta en [1]. Las entidades F_1, F_2 y F_3 siguen el protocolo de forma segura: F_1 genera los valores aleatorios r_1 y r_2 y los comunica a F_2 de forma secreta. En el caso de ser conocidos por F_3, ésta podría conocer el número de entradas con valor 1 en cada servidor al final del protocolo. La mix-net utilizada por F_1 es segura, entendiendo por ello que mantiene su permutación en secreto y que es capaz de generar pruebas criptográficas de que no ha sustituido ninguna entrada. En caso de que la primera propiedad no se cumpliera, la privacidad de los participantes solamente se mantendría entre los $n/2$ participantes que han usado el mismo servidor. En caso de que no se cumpliera la segunda, la exactitud del protocolo podría verse afectada.

VI. Propuesta de un sistema de voto electrónico basado en el cálculo de la función MAYORÍA

Poder calcular entre varios participantes una función MAYORÍA, con un protocolo que proporciona unas propiedades de seguridad tan buenas, infiere una posible aplicación al voto electónico. Calcular la función MAYORÍA equivale a organizar un referéndum donde se quiere conocer el número de respuestas afirmativas en una votación. La seguridad que ofrece el esquema lo hace atractivo frente a otros sistemas de voto electrónico, como por ejemplo los basados en mixnets [4], [5] y los de recuento homomórfico [3]. En estos sistemas, el voto emitido por cada votante se guarda en una urna digital (cifrado), y antes del descifrado se realiza una operación de *anonimización de los votos* que permite mantener el secreto de voto (no permite relacionar un voto en claro con la identidad del votante que lo ha emitido). Estos sistemas tienen la debilidad de que hay que confiar en que se sigue el procedimiento de primero anonimizar y luego descifrar. Si la entidad que tiene la clave privada necesaria para descifrar, descifra directamente el contenido de la urna digital, la privacidad de los votantes se ve comprometida. En cambio, en un esquema de voto basado en el sistema que se presenta en [1], un servidor malicioso que quiera descubrir qué han votado los votantes no podrá descubrir más que el resultado del agregado de todos los votos emitidos. Además, como hemos visto en la sección III, en el caso de que el servidor se ponga de acuerdo con los últimos participantes, la única información que descubre es el agregado de los votos emitidos por los participantes no corruptos: asumiendo al menos dos participantes honestos, la privacidad de un votante individual no se vería comprometida.

Sin embargo, en un entorno de votación real hay ciertos requisitos que se deben tener en cuenta. Uno de ellos es la posibilidad de gestionar votantes simultáneos, que se solventa utilizando la modificación del protocolo propuesta en la sección IV de este artículo. Por otra parte, en un entorno de votación electrónica real no podemos asumir que los votantes tienen claves criptográficas propias, ni que habrá una participación del 100% de los votantes: en el esquema de [1], el servidor puede obtener el resultado de la función gracias a que, por turno, cada participante *quita* la contribución de su clave privada en los cifrados de las entradas de la tabla de la verdad. En el caso de que no todos los participantes votaran, el servidor no podría descifrar el resultado, ya que no conoce las claves privadas correspondientes a los votantes que no han participado.

A continuación explicaremos cuáles son nuestras propuestas para solventar estos dos problemas.

A. Mejoras y herramientas para usar el protocolo en un entorno de votación real

a) Creación y gestión de claves: Nuestra propuesta consiste en usar un sistema de criptografía de clave pública basada en la identidad (IBE, del inglés *Identity-Based Encryption*) [7], donde las claves públicas pk_i están directamente relacionadas con las identidades de los participantes en el protocolo y una entidad de confianza genera sus claves privadas.

Los sistemas IBE tienen la ventaja de que es posible cifrar un mensaje con la clave pública de una persona antes de que ésta haya interactuado con la entidad de confianza que gestiona las claves para solicitar su clave privada. Esto es de gran

utilidad en el sistema que proponemos para voto electrónico, ya que cada votante, en su turno de ejecución del protocolo, podrá recifrar las entradas de la tabla de verdad con las claves públicas de los otros votantes sin necesidad de que éstos hayan entrado aún en el sistema para solicitar su clave privada.

b) Gestión de la participación incompleta: Al final de la elección, y siempre bajo una autorización específica, la entidad de confianza puede liberar las claves privadas de los votantes que no hayan participado para que el sistema sea capaz de descifrar las entradas restantes de la tabla de verdad y obtener así el resultado del cálculo de la función realizado de forma conjunta por los votantes que sí han participado. Concretamente, en una votación con participación incompleta donde el número de entradas de la tabla de verdad al final de la elección es mayor que 1, se tomará como resultado la primera entrada de las que quedan en la tabla.

La obtención de las claves privadas por parte del sistema para resolver el problema de la participación incompleta no afecta a la privacidad de los votantes, ya que solamente se obtienen las de los votantes que no han participado. El sistema de voto solamente llega a conocer el resultado de las opciones escogidas por el conjunto de votantes que han participado.

B. Breve descripción del protocolo de voto

A continuación vamos a describir cómo sería el sistema de voto utilizando el protocolo de la sección IV para el cálculo de la función MAYORÍA e incorporando el sistema de gestión de claves comentado en esta sección.

Configuración: para establecer el esquema de IBE, una entidad de confianza, F_4, genera su par de claves y, usando su clave privada, genera las claves privadas para los participantes en la elección. Estas claves estarán protegidas hasta el momento de su uso. Las claves públicas correspondientes no dependen de la identidad de los votantes, sino que dependen de un orden. Por ejemplo, *votante_1*, *votante_2*, etc. Se disponen varios servidores para atender a los participantes del protocolo (en esta explicación consideramos dos servidores). Estos servidores generan la tabla de verdad correspondiente a la función MAYORÍA a calcular y cifran sus entradas con las claves públicas de todos los participantes (votantes y el servicio de descifrado). Esta modificación sobre el protocolo explicado en la sección IV permite poder auditar que la tabla de verdad inicial es correcta, y que todos los votantes se comportan del mismo modo (facilita el desarrollo y la auditoría del *software*). Finalmente, se ejecutan los pasos previos a la fase de participación que se han detallado en la sección IV-C.

Votación: durante la fase de votación, cada votante procede a identificarse frente a F_4 que, una vez ha determinado que está autorizado a participar en la elección, le proporciona el par de claves correspondiente, basándose en su orden de llegada al servicio (por ejemplo, el primer votante recibirá la identidad *votante_1* y la clave secreta correspondiente). Este orden también determinará cuál es el servidor que atenderá al votante. Por ejemplo, números de orden del votante impares para el servidor *S1* y pares para el servidor *S2*. El resto del protocolo ejecutado por cada votante en la fase de votación sigue lo explicado en la sección IV-B, con la excepción de que el primer participante para cada servidor se comporta igual que los demás.

Obtención de resultados: en la fase de obtención de resultados F_4 libera las claves privadas de los votantes que no han participado y se descifran las entradas de las tablas de la verdad restantes. A continuación se sigue el protocolo de la sección IV: las entradas restantes de las tablas de la verdad de cada servidor se combinan en un servidor aislado F_2 y se descifran en otro F_3, que tiene la clave privada necesaria (sk_s).

VII. Análisis y conclusiones

En este artículo hemos visto cómo modificar el protocolo para el cálculo de funciones simétricas binarias presentado en [1] para que permita la participación simultánea, sin disminuir la seguridad. Además, hemos presentado una propuesta de sistema de voto electrónico que ofrece unas características de privacidad muy importantes. Utilizando pruebas de conocimiento cero, es posible verificar el correcto seguimiento del protocolo por parte de todas las entidades: generación de vectores de la tabla de verdad cifrados, operaciones de cifrado y descifrado de cada votante, operación de la mix-net, etc.

El principal inconveniente del esquema es el elevado número de operaciones criptográficas que se realizan. Por ejemplo, un votante debe recifrar y descifrar un promedio de $n/2$ elementos cifrados con ElGamal (las entradas de la tabla de verdad), donde n es el número total de votantes. Las pruebas de conocimiento cero aumentarían además el número de cálculos de forma considerable Así pues, el siguiente paso en el estudio de este tipo de protocolos será intentar reducir este número de cálculos, principalmente en la parte cliente (que puede tener un dispositivo con una capacidad de cómputo limitada). En la parte servidor hemos contemplado que determinadas operaciones se hagan por avanzado, en la fase de configuración, para reducir el coste a la hora de obtener los resultados.

Agradecimientos: Este trabajo ha sido parcialmente financiado por el Ministerio de Economía y Competitividad a través del proyecto CONSOLIDER ARES (CSD2007-0004) y el proyecto CICYT (MTM2009-07694).

References

[1] S. Halevi, Y. Lindell and B. Pinkas. Secure Computation on the Web: Computing without Simultaneous Interaction. In CRYPTO'11, pp 132-150.

[2] A. Chi-Chih Yao. Protocols for Secure Computations (Extended Abstract) FOCS 1982, pp 160-164.

[3] R. Cramer, R. Gennaro and B. Schoenmakers. A secure and optimally efficient multi-authority election scheme. 1997. In Proc.of EUROCRYPT'97, pp 103-118.

[4] C. Andrew Neff. A verifiable secret shuffle and its application to e-voting. 2001. In Proc. 8th ACM conf. on Computer and Communications Security (CCS '01), pp 116-125.

[5] K. Sako and J. Kilian. Receipt-free mix-type voting scheme: a practical solution to the implementation of a voting booth. In Proc. of EUROCRYPT'95, pp 393-403.

[6] T. El Gamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In Proc. of CRYPTO 84, pp 10-18.

[7] A. Shamir. Identity-based cryptosystems and signature schemes. In Proc. of CRYPTO 84, pp 47-53.