Cifrado homomórfico: ejemplos y aplicaciones

por

José Luis Gómez Pardo

Resumen. Con motivo del reciente desarrollo de la «computación en nube», que proporciona servicios de almacenamiento de datos permitiendo al mismo tiempo operar con ellos, ha cobrado gran importancia el *cifrado homomórfico*, cuyo propósito es permitir operar directamente con datos cifrados obteniendo resultados significativos sin necesidad de descifrarlos. En este artículo describimos las conceptos básicos en las que se sustenta el cifrado homomórfico, así como algunas de sus aplicaciones más importantes, incluyendo ejemplos desarrollados con la ayuda de Maple. Finalmente, se hace una breve introducción a las ideas que han llevado al desarrollo de *cifrado completamente homomórfico*, que fue introducido en 2009 por C. Gentry en su tesis doctoral y permite hacer cálculos complejos con los datos cifrados.

Introducción

En los últimos tiempos ha cobrado gran relevancia un modelo de procesamiento y almacenaje de la información denominado *computación en nube* (*cloud computing*, en inglés). «La nube» es una metáfora de Internet y el modelo se basa en la existencia de servicios alojados en servidores externos a los que se puede acceder remotamente usando, por ejemplo, un navegador web. Estos servicios pueden almacenar datos y ejecutar aplicaciones usando infraestructura compartida, lo cual garantiza un alto nivel de eficiencia y funcionalidad y, al mismo tiempo, permite abaratar costes. A pesar de todas estas ventajas, la computación en nube tiene un serio inconveniente derivado del hecho de que sus usuarios carecen de control directo sobre los sistemas que gestionan sus datos, lo cual plantea el problema de cómo pueden proteger la privacidad de dichos datos si no confían en el operador en nube que les proporciona el servicio. La solución obvia sería cifrar los datos que se envían al servidor, pero esto plantea un problema difícil, pues el objetivo de la computación en nube no es solo el de proveer almacenamiento externo para los datos sino también el permitir operar con ellos realizando, por ejemplo, búsquedas o cálculos aritméticos. Si los datos están cifrados, el usuario tendría la posibilidad de bajarlos y luego descifrarlos y operar con ellos a nivel local, pero es obvio también que esto no sería una solución pues así se perderían todas las ventajas de la computación en nube.

CIFRADO HOMOMÓRFICO

Para ilustrar el problema y tratar de vislumbrar una solución al mismo, voy a plantear una situación concreta en la que resulta muy útil poder realizar una computación con datos cifrados que proporcione un resultado significativo sobre los correspondientes datos no cifrados. Supongamos que se va a proceder a una votación electrónica para elegir un candidato entre varios posibles y se pretende garantizar la confidencialidad del voto de tal modo que ninguna otra persona pueda saber a qué candidato concreto ha votado cada elector. Una posibilidad sería hacer que cada elector emita su voto cifrado mediante un esquema de cifrado de clave pública, usando la clave pública de una autoridad que sería la encargada de contabilizar el resultado final de la votación. A tal fin, esta autoridad usaría su clave privada —que nadie más conoce— para descifrar los votos y, suponiendo que el esquema usado es seguro, nadie más podría averiguar a quién ha votado cada elector. Sin embargo, esto no resuelve satisfactoriamente el problema planteado pues lo que se pretendía es que solamente cada elector conociese su voto y, por tanto, la autoridad encargada del recuento de votos no debería tener acceso a los mismos. Así, para resolver el problema, es necesario hacer una computación con los votos cifrados que proporcione información sobre el conjunto de los votos no cifrados, en concreto, se trata de que *se puedan sumar los votos sin necesidad de descifrarlos.*

Una posible forma de alcanzar este objetivo sería usar un esquema de cifrado *homomórfico* en el sentido de que el algoritmo de descifrado define, para una clave privada dada, un homomorfismo de grupos entre el espacio de los criptotextos $\mathcal{C}$ y el de los textos claros (o mensajes) $\mathcal{M}$ (que, para que esto tenga sentido, habrán de tener estructura de grupo). Un esquema con esta propiedad permitiría enviar a la autoridad encargada del recuento el producto (con la operación del grupo de los criptotextos) de todos los votos cifrados correspondientes a un candidato (en lugar de los votos cifrados individuales) y la autoridad obtendría, al descifrar dicho producto, el producto de los correspondientes votos sin cifrar, lo que le permitiría calcular el número de votos favorables alcanzado por el candidato en cuestión, pero no le permitiría averiguar a qué candidato ha votado cada uno de los electores. Pero, ¿existen esquemas de cifrado homomórficos en el sentido que acabo de mencionar? Por supuesto que existen, y el más conocido de ellos es la versión básica de RSA, que es, históricamente, el primer esquema de cifrado de clave pública.

Recordemos que en RSA la clave pública es un par (n, e), donde el *módulo* $n = pq$ es el producto de dos primos distintos grandes y el *exponente de cifrado* e es un entero positivo que satisface $\mathrm{mcd}(e, \phi(n)) = 1$, con $\phi(n) = (p-1)(q-1)$. La correspondiente clave privada es el par (n, d), donde el *exponente de descifrado* d es el inverso de e módulo $\phi(n)$, de modo que $ed \equiv 1 \pmod{(p-1)(q-1)}$. Se puede tomar entonces $\mathcal{M} = \mathcal{C} = \mathbb{Z}_n^*$, el grupo multiplicativo de las unidades de los enteros módulo n, cuyos elementos podemos representar por los enteros m tales que $1 \leq m < n$ y $\mathrm{mcd}(m, n) = 1$, y el algoritmo de cifrado define un homomorfismo de grupos $E : \mathbb{Z}_n^* \to \mathbb{Z}_n^*$ dado por $E(m) = m^e \bmod n$ que es, de hecho, un isomorfismo cuyo inverso D, dado por $D(c) = c^d \bmod n$, corresponde a descifrar c con la correspondiente clave privada (n, d). Por tanto, RSA básico es un esquema de cifrado homomórfico para el que,

como $\mathbb{Z}_n^*$ es un grupo multiplicativo, se tiene que $E(m_1 \cdot m_2) = E(m_1) \cdot E(m_2)$ (con la multiplicación de $\mathbb{Z}_n^*$), lo cual recibe habitualmente el nombre de *propiedad multiplicativa de RSA*.

Veamos, con un ejemplo concreto en Maple, como se puede usar esta propiedad para el recuento de votos. Las funciones de Maple que implementan los algoritmos de cifrado y descifrado serán

```
> RSAEnc := proc(clavepublica::list, mensaje::posint)
  local n, e;
  n := clavepublica[1];
  e := clavepublica[2];
  Power(mensaje, e) mod n
  end proc:


> RSADec := proc(claveprivada::list, criptotexto::posint)
  local n, d;
  n := claveprivada[1];
  d := claveprivada[2];
  Power(criptotexto, d) mod n
  end proc:
```

Usaré una clave RSA de 2048 bits aunque no la imprimiré completa para ahorrar espacio; el ejemplo funcionará igual con cualquier otra clave válida, y el lector interesado podrá repetirlo con su clave favorita. La clave estará determinada por el módulo n (donde $n = pq$, con p y q primos de 1024 bits) y $\phi(n) = (p-1)(q-1)$, dados por los siguientes valores:

```
> n := 2353837249840889802090482051619 48 ... 336700575376126547003051554101801:
  fi_n := 235383724984088980209048205161 ... 655749747482032334978351370258132:
```

La clave pública `cpu` será entonces la lista de Maple $[n, e]$, donde el exponente de cifrado es $e = 2^{16} + 1 = 65537$ (el valor más habitual), y la clave privada `cpr` será la lista $[n, d]$, donde d es el correspondiente exponente de descifrado:

```
> cpu := [n, 2^16+1]:
  cpr := [n, (2^16+1)^(-1) mod fi_n]:
```

Supongamos que existen t votantes y cada uno de ellos emite un voto sobre cada candidato, que puede ser favorable (lo vota) o neutro (no lo vota). Cada elector puede emitir un único voto favorable y el resultado de la elección lo determinará el número de votos favorables de cada candidato. Se puede entonces elegir «2» para representar el voto neutro y «3» para representar el voto favorable, y el número de votos favorables se podrá determinar fácilmente a partir del producto de todos los votos teniendo en cuenta la factorización única de un entero en producto de primos. Siguiendo con nuestro ejemplo, supongamos que hay un total de cien votantes y que 37 de ellos han votado a favor del candidato A mientras que los 63 restantes han emitido voto neutro sobre dicho candidato. Ciframos el voto neutro y el voto favorable, respectivamente, con la clave pública anterior `cpu` y llamamos `x` e `y` a los votos cifrados correspondientes:

```
> x := RSAEnc(cpu, 2):

> y := RSAEnc(cpu, 3):
```

El criptotexto que se transmitiría a la autoridad encargada del recuento es el producto de los votos cifrados que, en este caso, sería

```
> vt := x^63*y^37 mod n:
```

Ahora, usando su clave privada, la autoridad puede calcular

```
> ifactor(RSADec(cpr, vt));
                          (2)^63(3)^37
```

lo cual proporciona el resultado (37 votos favorables) pero no da información sobre el voto emitido por cada elector.

Vemos en el ejemplo que la propiedad homomórfica funciona perfectamente, teniendo cuidado de que el producto de los votos sin cifrar no sea mayor que el módulo n, y podría parecer que se ha conseguido el propósito de garantizar la confidencialidad del voto, pero nada más lejos de la realidad. Aunque la propiedad homomórfica ha cumplido su cometido, existe otro problema más básico que invalida el protocolo esbozado. En efecto, un adversario que observa que el voto cifrado emitido por un votante es `y` puede averiguar fácilmente cuál ha sido el voto haciendo

```
> evalb(RSAEnc(cpu, 2) = y);
                             false
```

que le dice que el voto no es neutro; y, si a continuación calcula

```
> evalb(RSAEnc(cpu, 3) = y);
                              true
```

obtiene la confirmación de que este voto es, efectivamente, favorable. Obsérvese que no tendría sentido confiar en que el adversario podría no conocer qué elementos de $\mathbb{Z}_n^*$ han sido utilizados para representar el voto neutro y el voto favorable, pues esto iría contra el *principio de Kerckhoffs*, que es universalmente aceptado en criptografía y postula que *el adversario conoce el sistema*, es decir, que el sistema tiene que ser seguro frente a un adversario que conoce todo sobre él con excepción de la clave privada.

El ataque descrito es un *ataque con texto claro elegido* que muestra que la versión básica de RSA es un esquema de cifrado extremadamente débil desde el punto de vista de la seguridad. Sería erróneo pensar que, como se sugiere en algunos textos, RSA básico ya es seguro como consecuencia de la hipótesis de que la función RSA

$$\begin{array}{ccc} \mathbb{Z}_n^* & \overset{\mathrm{RSA}_{(n,e)}}{\longrightarrow} & \mathbb{Z}_n^* \\ m & \longmapsto & m^e \text{ mód } n. \end{array}$$

es una función de dirección única. Evidentemente, esta es una condición necesaria para que RSA sea seguro pero, como vemos en el ejemplo, dista mucho de ser suficiente. En este caso ocurre que solo se cifran dos mensajes y ello hace muy fácil averiguar

cuál es el criptotexto que corresponde a cada uno de ellos simplemente por comparación, como se ha hecho antes. Esto es perfectamente compatible con la hipótesis de que la función $\mathrm{RSA}_{(n,e)}$ es de dirección única que, sin entrar en excesivos detalles, solo requiere que sea *difícil* calcular $m \in \mathbb{Z}_n^*$ tal que $\mathrm{RSA}_{(n,e)}(m) = c$ para un c que, a diferencia de los criptotextos de nuestro ejemplo, es elegido aleatoriamente en $\mathbb{Z}_n^*$.

A la vista del ataque descrito, podría parecer que no es viable usar un esquema de cifrado de clave pública para cifrar solo dos votos de manera segura pero, como vamos a ver, esto no es así porque el ataque se aprovecha de manera decisiva de otra característica del esquema empleado, a saber, el hecho de que el algoritmo de cifrado es determinista y tanto el voto neutro como el voto favorable se cifran siempre de la misma manera, dando lugar a dos únicos criptotextos. Más allá de las peculiaridades del ejemplo descrito, el tener un algoritmo de cifrado determinista hace que un esquema de cifrado sea automáticamente inseguro en un sentido que voy a describir brevemente. El concepto de seguridad que se persigue va mucho más allá de prevenir que un adversario pueda recuperar el texto claro a partir del criptotexto, pues lo que se pretende es que el adversario (que se supone que actúa como un algoritmo de tiempo polinómico) no pueda obtener, a partir de un criptotexto, ninguna información adicional sobre el texto claro aparte de la que ya podría tener de antemano, todo ello excepto posiblemente con *probabilidad insignificante*[1]. Esta es la idea intuitiva subyacente al concepto de *seguridad semántica* que es, a su vez, equivalente a otro concepto de seguridad que se basa en la *indistinguibilidad* y es más fácil de formular de manera precisa.

Un esquema de cifrado de clave pública se dice que es IND-CPA seguro[2] (o, simplemente, CPA seguro) cuando un adversario de tiempo polinómico que tiene acceso a la «máquina de cifrar» (como una caja negra a cuyo funcionamiento interno no se accede, es decir, como lo que se llama un *oráculo de cifrado*) y puede, por tanto, obtener criptotextos correspondientes a los textos claros de su elección, es incapaz de distinguir entre dos criptotextos correspondientes a otros dos textos claros que él mismo ha elegido. Esto último significa que si uno de esos dos textos claros, tomado al azar sin conocimiento del adversario, es cifrado y el correspondiente criptotexto es entregado al adversario, este no debe ser capaz de acertar con probabilidad significativamente mayor que $1/2$ cuál de los dos textos claros es el que se ha cifrado. En el ataque anteriormente descrito contra RSA básico, el adversario puede distinguir el voto neutro del voto favorable con probabilidad 1, pero es claro que si pudiera distinguirlos, por ejemplo, con probabilidad $2/3$, el sistema no sería seguro; de hecho, la definición asintótica de probabilidad insignificante implica que si un adversario puede distinguir los mensajes con probabilidad $\frac{1}{2} + \frac{1}{f(k)}$, donde k es la longitud (en bits) del módulo RSA y $f(k)$ una función polinómica, entonces el esquema ya no es CPA seguro.

[1] La probabilidad es insignificante o despreciable cuando decrece asintóticamente más rápido que la inversa de cualquier función polinómica del tamaño de la clave, cf., por ejemplo, [9], para una discusión más detallada de este y otros conceptos relacionados.

[2] En esta terminología, la primera parte (IND) se refiere al objetivo del adversario, en este caso, romper la «indistinguibilidad»; y la segunda (CPA) a la capacidad del adversario: CPA = *Chosen Plaintext Attack*, es decir, ataque con texto claro elegido.

UN ESQUEMA DE CIFRADO HOMOMÓRFICO CPA SEGURO

La propiedad de ser CPA seguro es el requisito de seguridad mínimo que se exige habitualmente a un esquema de cifrado de clave pública, y esto parece razonable teniendo en cuenta que un adversario siempre está en condiciones de montar un ataque con texto claro elegido puesto que conoce la clave pública y, por tanto, tiene acceso a un oráculo de cifrado. Hay que señalar que no se conoce ningún esquema que sea demostrablemente CPA seguro en el sentido estricto del término, pues todas las demostraciones de seguridad en criptografía son *reducciones* que se limitan a probar que un esquema de cifrado es, por ejemplo, CPA seguro bajo la hipótesis de que un cierto problema computacional es *difícil* en el sentido de que un algoritmo de tiempo polinómico solo puede resolverlo con probabilidad insignificante. Por tanto, estas demostraciones se limitan a *reducir* la seguridad del esquema (un problema estrictamente criptográfico o, más bien, criptoanalítico) a un problema matemático como puede ser, en el caso de RSA, el problema de invertir la función RSA sobre un elemento elegido aleatoriamente o un problema relacionado con este como es el problema de la factorización de enteros.

Lo que ocurre es que *es muy difícil demostrar que un problema es difícil*, pues ello implicaría que $P \neq NP$, resolviendo así uno de los problemas del milenio. Por eso el término *demostrablemente seguro* que se aplica a menudo a esquemas que son, por ejemplo, CPA seguros bajo una cierta hipótesis computacional, ha sido criticado por autores como N. Koblitz y A. Menezes [10] porque las demostraciones son solo reducciones y se basan en formalizaciones del concepto de seguridad que no tienen por qué corresponderse exactamente con la seguridad en el mundo real. Además de esto, en ocasiones se han hecho reducciones de seguridad en las que el problema computacional que se presume difícil no es un problema matemático natural, sino una traducción *ad hoc* del problema criptoanalítico correspondiente. No obstante, es generalmente aceptado que una demostración que reduce la seguridad de un esquema a la dificultad de un problema bien conocido (como puede ser el de la factorización y otros similares) proporciona una información valiosa y que, en todo caso, es preferible disponer de una reducción como esta a no tener ninguna.

Teniendo en cuenta las observaciones anteriores, el problema natural que surge es tratar de resolver nuestro objetivo de garantizar la confidencialidad del voto usando un esquema de cifrado que sea, además de homomórfico, CPA seguro bajo alguna hipótesis razonable. Sin embargo, no es ni mucho menos obvio cómo conseguir esto, pues un esquema CPA seguro no puede ser determinista (en ese caso el adversario cifra los dos mensajes que debe distinguir y compara los criptotextos obtenidos exactamente igual que en el ejemplo de la votación con RSA básico), y esto plantea serias dificultades. Por ejemplo, existen variantes de RSA que son CPA seguras bajo hipótesis como la dificultad de invertir la función RSA y para conseguir esto usan un algoritmo de cifrado probabilista. El método básico consiste en usar *relleno aleatorio* que se añade al mensaje antes de cifrarlo mediante la función RSA, pero el problema es que, como resulta obvio, dicho relleno destruye la propiedad multiplicativa de RSA y el esquema ya no es homomórfico. Sin embargo, también existen otros esquemas de cifrado que son a la vez CPA seguros (como siempre, bajo la hipótesis de que

un cierto problema computacional es difícil) y homomórficos y, como vamos a ver, pueden ser usados para conseguir nuestro objetivo sobre la votación.

Un esquema que cumple estas condiciones es el *esquema de cifrado de Paillier* que fue introducido en [13] (véase también [7] para una descripción detallada del mismo que incluye una implementación en Maple) y, de forma similar a RSA, se puede relacionar con el problema de la factorización de enteros, cuya dificultad es una condición necesaria (aunque no se sabe si suficiente) para su seguridad. El esquema de Paillier es homomórfico y, además, se puede demostrar que es CPA seguro bajo la hipótesis de que otro problema computacional, que se mencionará a continuación, es difícil. Los algoritmos que constituyen el esquema de Paillier se pueden describir como sigue:

- Generación de claves: Se generan aleatoriamente dos primos grandes de igual longitud, p y q, y se obtiene la clave pública $n = pq$ y la clave privada $(n, \phi(n))$, donde $\phi(n) = (p-1)(q-1)$.
- Algoritmo de cifrado: El espacio de los textos claros es $\mathbb{Z}_n$ y el de los criptotextos $\mathbb{Z}_{n^2}^*$ y, para cifrar un mensaje $m \in \mathbb{Z}_n$, se elige un $r \in \mathbb{Z}_n^*$ al azar y se calcula el criptotexto mediante

$$c = \mathrm{Enc}(n, m) := ((1+n)^m \cdot r^n) \bmod n^2 \in \mathbb{Z}_{n^2}^*$$

- Algoritmo de descifrado: Para descifrar el criptotexto c con la clave privada $(n, \phi(n))$, se calcula

$$m = \mathrm{Dec}((n, \phi(n)), c) := \frac{(c^{\phi(n)} \bmod n^2) - 1}{n} \cdot \phi(n)^{-1} \bmod n,$$

  donde el cociente $((c^{\phi(n)} \bmod n^2) - 1)/n$ se calcula en $\mathbb{Z}$.

La seguridad del esquema de cifrado de Paillier se basa en el llamado *problema de decisión de la residuosidad compuesta*, que es el problema de distinguir un elemento aleatorio de $\mathbb{Z}_{n^2}^*$ de un elemento elegido aleatoriamente en el conjunto de los *residuos n-simos módulo n^2*, es decir, los elementos de $\mathbb{Z}_{n^2}^*$ que son potencias n-simas (de modo que este problema es difícil si cualquier algoritmo de tiempo polinómico solo puede distinguir estos elementos con probabilidad insignificante en función de la longitud del módulo, n). Se verifica, entonces:

- Si el problema de decisión de la residuosidad compuesta es difícil, entonces el esquema de Paillier es CPA seguro.
- El esquema de Paillier es homomórfico: Dados $m_1, m_2 \in \mathbb{Z}_n$, $c_1 = \mathrm{Enc}(n, m_1)$ y $c_2 = \mathrm{Enc}(n, m_2) \in \mathbb{Z}_{n^2}^*$, se cumple

$$\mathrm{Dec}((n, \phi(n)), (c_1 \cdot c_2) \bmod n^2) = (m_1 + m_2) \bmod n.$$

En lo que se refiere a la seguridad, se tiene la siguiente situación:

Residuosidad compuesta difícil ⇒ Paillier CPA seguro ⇒ Factorización de enteros difícil.

La primera de estas implicaciones[3] está probada, por ejemplo, en [9, Theorem 11.34]; y la segunda implicación es obvia pues la clave privada se puede obtener a partir de la clave pública factorizando el módulo. Se cree que el problema de decisión de la residuosidad compuesta es, efectivamente, difícil (cf., por ejemplo, [9] para una definición precisa de este concepto), y así es razonable suponer que el esquema de Paillier es CPA seguro. Por otra parte, la propiedad homomórfica de este esquema significa que la aplicación de $\mathbb{Z}_{n^2}^*$ a $\mathbb{Z}_n$ inducida por el algoritmo de descifrado con una clave fija es un homomorfismo de grupos (obsérvese que, estrictamente hablando, no podemos decir lo mismo del algoritmo de cifrado pues, al ser este probabilista, no define una aplicación de $\mathbb{Z}_n$ en $\mathbb{Z}_{n^2}^*$, aunque sigue siendo cierto que para cifrar la suma de varios mensajes se puede tomar como criptotexto el producto de los criptotextos correspondientes a cada uno de ellos).

Como $\mathbb{Z}_{n^2}^*$ es un grupo multiplicativo y $\mathbb{Z}_n$ es aditivo, la función de descifrado del esquema de Paillier hace corresponder a un producto de criptotextos en $\mathbb{Z}_{n^2}^*$ la suma de los textos claros correspondientes en $\mathbb{Z}_n$, y así este esquema se adapta especialmente bien al problema de la votación. En efecto, basta considerar que «0» es el voto neutro, «1» es el voto favorable y cada votante emite su voto para un candidato cifrando con el esquema de Paillier uno de estos dos mensajes. Si los votos emitidos por los t votantes para un candidato son $m_1, m_2, \ldots m_t \in \mathbb{Z}_n$, y los correspondientes votos cifrados son $c_1, c_2, \ldots, c_t \in \mathbb{Z}_{n^2}^*$, entonces los votos pueden ser sumados sin necesidad de descifrarlos, calculando $c := \prod_{i=1}^{t} c_i \bmod n^2 \in \mathbb{Z}_{n^2}^*$, que es el resultado que se enviará a la autoridad que realiza el recuento. Esta descifrará el criptotexto c con su clave privada y, como la función de descifrado es un homomorfismo, obtendrá

$$\sum_{i=1}^{t} m_i \bmod n$$

que, suponiendo que el número t de votantes es menor que n, es precisamente el número de votos favorables obtenido por el candidato. Sin embargo, la autoridad no conoce los c_i y por tanto no llega a conocer quién ha votado a cada candidato. Además, los votantes tampoco llegan a conocer los votos emitidos por los restantes votantes.

Vamos a usar Maple para dar un ejemplo de cómo funcionaría este proceso, el cual nos permitirá apreciar las importantes diferencias que existen con respecto al anterior ejemplo que usaba RSA. Comenzaré dando la función de Maple que implementa el algoritmo de cifrado, el cual, al ser probabilista, requiere generar aleatoriamente, en cada operación de cifrado, un elemento de $\mathbb{Z}_n^*$, donde n es la clave pública. En nuestro caso, esto se hará pseudoaleatoriamente mediante el generador pseudoaleatorio de Blum-Blum-Shub[4], del cual Maple ya contiene una implementación. Esto requiere el uso de una semilla aleatoria `sm` que deberá ser obtenida externamente

[3] La implicación recíproca de esta también se verifica y así se tiene una equivalencia, cf. [13, Theorem 5] y también [7].

[4] Este es un generador *impredecible* suponiendo que la factorización de enteros es difícil y, por tanto, adecuado para uso criptográfico, cf. [3] y también [9] para una discusión general de generadores pseudoaleatorios.

pues no puede ser generada usando Maple. Sin embargo, para facilitar el trabajo con ejemplos, he incluido la posibilidad de que la semilla sea obtenida automáticamente del sistema, cosa que ocurrirá por defecto si se omite pasar a la función el tercer argumento. Si se hace así, el esquema ya no es CPA seguro, pero este método sirve igual para ilustrar su funcionamiento. En este caso se usa también otro generador pseudoaleatorio, *Mersenne Twister*, que también está implementado en Maple pero no es criptográficamente seguro, y antes de iniciar las operaciones de cifrado se deberá ejecutar `RandomTools:-MersenneTwister:-SetState()` para inicializar el generador y evitar obtener siempre la misma secuencia de semillas. Las entradas requeridas por la función son una clave pública de Paillier (el módulo n) y el texto claro en la forma de un entero no negativo $< n$ (y, opcionalmente, una semilla aleatoria dada como un entero positivo), y la salida es el criptotexto dado como un entero positivo relativamente primo con n^2:

```
> PaillierEnc := proc(clavepublica::posint, m::nonnegint, sm::posint:=
  RandomTools:-MersenneTwister:-GenerateInteger(':-range'=2^127 .. 2^256))
  local n, l, B, found, r;
  n := clavepublica;
  l := ilog2(n)+1;
  B := RandomTools:-BlumBlumShub:-NewBitGenerator(
       sm, primes=1024, numbits=l, output=integer);
  found := false;
  while not found do
    r := B();
    found := evalb(r<n and igcd(r, n)=1)
  end do;
  (Power(1+n, m)*Power(r, n)) mod n^2
  end proc:
```

La función de descifrado, por su parte, acepta como entradas una clave privada de Paillier y un criptotexto en $\mathbb{Z}_{n^2}^*$, y produce como salida el correspondiente texto claro. Es la siguiente:

```
> PaillierDec := proc(claveprivada::list, c::posint)
  local n, f, t;
  n := claveprivada[1];
  f := claveprivada[2];
  t := ((Power(c, f) mod n^2)-1)/n;
  t/f mod n
  end proc:
```

Vamos ahora a utilizar los valores de n y $\phi(n)$ usados en el ejemplo de RSA para definir una clave pública/privada del esquema de Paillier:

```
> Pcpu := n:
  Pcpr := [n, fi_n]:
```

Supongamos que hay 100 votantes y que la lista de votos (sin cifrar) que han emitido para uno de los candidatos es la siguiente (se da así para poder calcular con ella, sin perjuicio de nuestra hipótesis de que solamente cada elector conoce el voto —sin cifrar— que ha emitido):

```
> votos := [0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 1,
  0, 1, 1, 1, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 1,
  1, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 0, 0,
  0, 1, 1, 0, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1]:
```

La correspondiente lista de votos cifrados, con la clave pública `Pcpu` anterior, es entonces la siguiente (no la imprimimos porque cada voto cifrado es un elemento de $\mathbb{Z}_{n^2}^*$ y la lista ocuparía muchas páginas):

```
> RandomTools:-MersenneTwister:-SetState();
  cvotos := PaillierEnc~(Pcpu, votos):
```

Si estos votos fuesen enviados a la autoridad que posee la clave privada correspondiente a la clave pública que se ha usado para cifrarlos, dicha autoridad podría descifrarlos individualmente y así averiguar el voto de cada uno de los votantes. Pero no se le envían estos votos sino solamente su producto módulo n^2. El producto —que tampoco se muestra en su integridad— es el siguiente:

```
> pvotos := mul(i, i = cvotos) mod Pcpu^2;
401600193789433327338809565415499626786897021970054722821196950545268581426875\
  1173828431901240900156[...1033 digits...]677614103449655684511368992801132 54\
  4347449196142914035505783251898953314696381470903930524297283384 0
```

Aunque este criptotexto no da información sobre los votos individuales, sí permite a la autoridad recuperar el número de votos favorables obtenidos por el candidato, calculando

```
> PaillierDec(Pcpr, pvotos);
                                43
```

Vemos que el número de votos favorables obtenidos por este candidato es 43, y podemos comprobar que coincide con el número de votos favorables en la lista de votos sin cifrar:

```
> ListTools:-Occurrences(1, votos);
                                43
```

Observemos ahora lo que ocurre si tratamos de repetir el ataque con texto claro elegido realizado contra RSA básico. El adversario cifra el voto neutro y el voto favorable con la clave pública y compara los votos cifrados obtenidos con el primer voto cifrado de la lista, `c := cvotos[1]`, tratando de descubrir cuál es el voto (sin cifrar) correspondiente:

```
> c := cvotos[1]:
  evalb(PaillierEnc(Pcpu, 0) = c);
  evalb(PaillierEnc(Pcpu, 1) = c);
                              false
                              false
```

¿Qué ha ocurrido? A diferencia del caso de RSA básico, ninguno de los dos criptotextos así obtenidos coincide con el primer voto cifrado de la lista, y un resultado similar se habría obtenido, con probabilidad muy alta, si hiciésemos la comparación con cualquiera de los 99 restantes votos cifrados. Lo que ha ocurrido es que, como

el primer voto era neutro, la segunda línea del código anterior tenía que dar como resultado `false` con probabilidad 1. La primera línea, en cambio, hubiera dado `true` si el algoritmo de cifrado fuese determinista como en el caso de RSA básico pero, al ser ahora un algoritmo probabilista que elige aleatoriamente el criptotexto entre $\phi(n) = |\mathbb{Z}_n^*|$ posibles valores, la probabilidad de obtener `true` (es decir, de obtener el mismo criptotexto que la primera vez que se cifró este voto) es insignificante (pues es conocido que, para $\epsilon > 0$, $n^{1-\epsilon} = O(\phi(n))$ y, por otra parte, esto también es consecuencia de la hipótesis de que el esquema es CPA seguro).

Vemos que el uso del esquema de cifrado de Paillier permite resolver satisfactoriamente el problema de la votación, aunque lo que hemos descrito son solo las ideas básicas en la suposición de que todas las partes actúan honestamente y no un protocolo completo que requeriría otros ingredientes como, por ejemplo, un método eficiente para probar que los votos emitidos son válidos, lo cual es algo que la criptografía moderna también hace posible (véase, por ejemplo, [4] para una descripción de algunos de estos protocolos).

La propiedad de ser CPA seguro es muy importante, y es lo mínimo que cabe esperar de un esquema de cifrado moderno pero, especialmente en criptografía de clave pública, existe otra propiedad de seguridad más fuerte que suele ser la que se fija como objetivo cuando se diseñan nuevos esquemas. Esta propiedad es la de ser IND-CCA seguro (o, brevemente, CCA seguro), donde las iniciales CCA corresponden a *Chosen Ciphertext Attack* (ataque con criptotexto elegido). Como en el caso CPA, el adversario elige dos mensajes y uno de ellos tomado al azar es cifrado y el criptotexto entregado al adversario, que debe tratar de distinguir a cuál de los dos mensajes corresponde. La diferencia con el ataque CPA es que ahora el adversario, además de tener acceso a la máquina de cifrar, lo tiene también a la máquina de descifrar como una caja negra (es decir, a un *oráculo de descifrado*), y puede obtener los textos claros correspondientes a los criptotextos que él elija, con la única restricción de que no puede preguntar por el criptotexto —correspondiente a uno de los dos mensajes que trata de distinguir— que le ha sido entregado. Además, el adversario puede realizar su ataque de forma *adaptativa*, es decir, puede elegir los textos claros y los criptotextos sobre los que pregunta al oráculo en función de las respuestas que ha ido obteniendo a sus anteriores preguntas y también en función del criptotexto que trata de distinguir.

Puede parecer a primera vista que la propiedad de ser CCA seguro impone una exigencia demasiado fuerte, pues no parece fácil que en la práctica un adversario pueda obtener los textos claros correspondientes a los criptotextos que elija. Sin embargo, esta situación no es inimaginable en el contexto de la criptografía de clave pública, pues a menudo se intercambian mensajes cifrados entre usuarios que no han tenido contacto previo y puede existir la posibilidad de que el adversario suplante a uno de los usuarios honestos y obtenga de esa forma textos claros que podrían ayudarle (por eso, en la obtención de esquemas que sean CCA seguros juega un papel importante la *autenticación*, pero ese es otro tema...). De hecho, un ataque CCA que ha tenido importantes consecuencias prácticas es el ataque de Bleichenbacher [2] contra la implementación de RSA usada en el estándar PKCS #1 v1.5. El ataque se basaba en el envío a un servidor de muchas variantes de un criptotexto construidas

especialmente y en la observación de los mensajes de error devueltos por el servidor, que hacían posible, en principio, la recuperación del texto claro. Esta inseguridad era tan seria que provocó un cambio inmediato en el software de los navegadores web que se usaban en 1998.

Volvamos al esquema de Paillier y al ejemplo de la votación y veamos si el esquema es CCA seguro. Supongamos que un adversario observa el primer voto cifrado de la lista anterior y monta un ataque CCA. El adversario quiere averiguar cuál ha sido el voto del primer votante, es decir, quiere obtener el texto claro correspondiente al criptotexto `c := cvotos[1]` pero no puede pedir que le descifren este criptotexto. Sin embargo, sí puede pedir que le descifren cualquier otro criptotexto y así hace lo siguiente. En primer lugar cifra el texto claro «0», obteniendo

```
> c0 := PaillierEnc(Pcpu, 0);
865373230110139210062426296837972088824728950553791777961986395162796197745144\
  0875233977999568156049[...1032 digits...]144510311238949660828824886507340 43\
  5123897184678981590490722175528463155324411441753406157113790352 1
```

Ahora el adversario sabe que, como consecuencia de la propiedad homomórfica del esquema de Paillier, `c1 := (c0 · c)` mód n^2 es un criptotexto válido y (teniendo además en cuenta que, por construcción, `PaillierDec(Pcpr, c0) = 0`)

```
PaillierDec(Pcpr, c1)
= PaillierDec(Pcpr, c0) + PaillierDec(Pcpr, c) = PaillierDec(Pcpr, c)
```

Nótese que, incluso en caso de ser «0» el texto claro correspondiente a `c`, el criptotexto `c1` es distinto de `c` con probabilidad abrumadora como consecuencia de que a cada texto claro en $\mathbb{Z}_n$ la operación de cifrado le hace corresponder un criptotexto elegido aleatoriamente entre $\phi(n)$ posibles. Por tanto, según las reglas del ataque CCA, el adversario puede pedir el texto claro correspondiente al criptotexto `c1`. La respuesta es la siguiente:

```
> c1 := (c0*c) mod Pcpu^2:
  PaillierDec(Pcpr, c1);
                                0
```

¡El adversario ha descubierto cuál es el voto del primer votante! Esto ha ocurrido porque el esquema de Paillier es *maleable*, en el sentido de que un adversario puede modificar el criptotexto objeto del ataque y obtener otro criptotexto relacionado cuyo texto claro está relacionado con el texto claro objeto del ataque de forma conocida. Esto es precisamente lo que se ha hecho aquí al construir el criptotexto `c1`, que tiene el mismo texto claro que el criptotexto `c` objeto del ataque. En general, la relación entre los textos claros no tiene por qué ser tan sencilla como en este caso, en el cual `c` y `c1` cifran el mismo mensaje como consecuencia de que se ha utilizado el texto claro «0» para modificar `c`. Si, por ejemplo, se hubiese utilizado «1» en su lugar, el proceso sería similar excepto que habría que restar 1 al texto claro correspondiente a `c1` para obtener el correspondiente a `c`. Todo esto no es una peculiaridad especial del esquema de Paillier y, por el contrario, es fácil ver que todo esquema de cifrado homomórfico es maleable y que un esquema maleable no puede ser IND-CCA seguro, usando esencialmente el mismo razonamiento aplicado en este

ejemplo. Por tanto vemos que la propiedad de ser homomórfico no permite alcanzar el nivel de seguridad máximo pero, a cambio, tiene otras ventajas que son muy útiles en casos, como el de la votación, en que es suficiente usar un esquema CPA seguro.

Cifrado completamente homomórfico

Hemos visto la utilidad de un esquema de cifrado homomórfico como el de Paillier en una situación concreta como la de una votación electrónica, pero esto no resuelve el problema planteado al principio en relación con la computación en nube, que requiere poder realizar computaciones con datos cifrados que van mucho más allá que un mero recuento de votos. Sin embargo, la idea de usar un esquema de cifrado homomórfico sigue siendo válida para este propósito, aunque lo que se necesita es un concepto mucho más potente. La formulación de esta posibilidad se remonta ya al año 1978, cuando Rivest, Adleman y Dertouzos [14] plantearon el problema de la existencia de un *esquema de cifrado completamente homomórfico*, que se define de forma similar a los esquemas que hemos visto pero no se limita a permitir computaciones con criptotextos usando la operación de un grupo, sino que permite realizar computaciones arbitrarias expresables mediante circuitos booleanos (véase, p. ej., [1] para una discusión de la computación por circuitos). Así, un *esquema de cifrado completamente homomórfico* está dotado, además de los algoritmos de generación de claves, cifrado y descifrado, de un algoritmo eficiente Eval que, dada una clave (cpu, cpr), un circuito booleano C y criptotextos $c_i = \mathrm{Enc}(cpu, m_i)$, devuelve $c = \mathrm{Eval}(cpu, C, c_1, \ldots, c_t)$, con la propiedad de que $\mathrm{Dec}(cpr, c) = C(m_1, \ldots, m_t)$. Esto permite, sin usar la clave privada ni conocer los textos claros m_i, obtener un criptotexto válido correspondiente al texto claro $C(m_1, \ldots, m_t)$ y, por tanto, un sistema de este tipo permite resolver el problema relativo a la computación en nube planteado al principio. Por ejemplo, esto permitiría a un usuario utilizar una base de datos cifrada y almacenada en un servidor sin perder las ventajas que la computación en nube ofrece al poder operar directamente con los datos cifrados.

El problema de obtener un esquema de cifrado completamente homomórfico era muy difícil pues, como hemos visto, la propiedad homomórfica es como un arma de doble filo, que tiene ventajas computacionales pero no se lleva demasiado bien con las propiedades de seguridad de modo que, como ya he mencionado, un esquema de cifrado homomórfico —incluso en el sentido más débil considerado anteriormente— no puede ser CCA seguro. Por eso el objetivo era obtener un esquema completamente homomórfico que, además, fuese CPA seguro bajo hipótesis razonables.

El problema fue resuelto afirmativamente en 2009 en la tesis doctoral de Craig Gentry [5], en la cual se construye un esquema completamente homomórfico usando la teoría de retículos. Un retículo es un subgrupo aditivo de $\mathbb{R}^n$ generado por un conjunto de vectores linealmente independientes y es, en particular, un grupo abeliano libre de rango finito (cf. [8] para una introducción a la criptografía basada en retículos). Gentry usó los llamados *retículos ideales*, que son a la vez retículos e ideales de un anillo, de modo que, además de la estructura aditiva de retículo, tienen también una estructura multiplicativa. Se trata de los ideales del anillo $\mathbb{Z}[x]/(f)$, donde $f \in \mathbb{Z}[x]$ es un polinomio mónico irreducible, que también definen retículos

puesto que los elementos de este anillo se pueden identificar con los polinomios de grado a lo sumo $n-1$ (siendo n el grado de f), y así el anillo es aditivamente isomorfo a $\mathbb{Z}^n$ (y, de hecho, también lo son sus ideales no nulos como consecuencia de la irreducibilidad de f).

El esquema de cifrado completamente homomórfico de Gentry es CPA seguro suponiendo que el problema SIVP (*Shortest Independent Vectors Problem*) de hallar n vectores linealmente independientes que sean, aproximadamente, lo más cortos posible, es difícil en un retículo ideal de rango n (véase [5] para los detalles). También cumple el requisito habitual de ser eficiente en el sentido de que sus algoritmos son de tiempo polinómico, pero la cuestión importante que queda por resolver es conseguir una versión que sea suficientemente eficiente para su uso en la práctica. La tesis de Gentry ha abierto una línea de trabajo muy prometedora y los resultados iniciales están siendo mejorados regularmente. Por ejemplo, en [6] se describe una implementación concreta del esquema de Gentry, y en [12] se explora otro aspecto interesante y se construye una variante del esquema de Gentry que tiene, bajo una cierta hipótesis de dificultad de un problema de retículos, una propiedad de seguridad más fuerte que IND-CPA. Se trata de IND-CCA1, una versión más débil de IND-CCA[5] en la cual el adversario no puede pedir que se le descifren criptotextos una vez que ha recibido el criptotexto que trata de distinguir.

A pesar de estos avances, el problema de la eficiencia sigue siendo importante y los esquemas completamente homomórficos aún distan de ser usables en la práctica. Sin embargo, existe la posibilidad de usar esquemas *«algo homomórficos»* (*somewhat homomorphic schemes*), que no llegan a ser completamente homomórficos pues no permiten la evaluación homomórfica de funciones computables por circuitos arbitrarios, sino solo de funciones por debajo de un cierto umbral de complejidad, por ejemplo, funciones cuya representación polinómica tiene grado acotado, de modo que se pueden calcular muchas sumas pero solo un pequeño número de multiplicaciones. Un esquema de este tipo ha sido considerado en [11], donde se observa que muchas aplicaciones interesantes (gestión de historiales médicos, análisis financieros, ...) siguen siendo posibles y, al mismo tiempo, que el esquema es suficientemente eficiente para su uso práctico, como se demuestra con una implementación en Magma incluida en dicha referencia.

AGRADECIMIENTOS

El trabajo ha sido financiado parcialmente por los proyectos 04555/GERM/06 (Fundación Séneca de la Región de Murcia) e INCITE09E2R207117ES (Xunta de Galicia).

[5]IND-CCA se llama IND-CCA2 en esta clasificación más fina, y el ataque CCA1 es también llamado *«ataque a la hora del almuerzo»* (*lunchtime attack*), haciendo referencia a la idea de que el adversario tiene acceso a la máquina de cifrar mientras el usuario legítimo está almorzando pero no después de conocer el criptotexto objeto del ataque.

Referencias

[1] S. Arora y B. Barak, *Computational Complexity: A Modern Approach*, Cambridge University Press, 2009.

[2] D. Bleichenbacher, Chosen ciphertext attacks against protocols based on the RSA encryption standard PKCS #1, en *Advances in Cryptology – CRYPTO'98, Lecture Notes in Computer Science* 1462, pp. 1–12, Springer, 1998.

[3] L. Blum, M. Blum y M. Shub, A simple unpredictable random number generator, *SIAM J. Comput.* **15** (1986), 364–383.

[4] I. Damgård, J. Groth y G. Salomonsen, The theory and implementation of an electronic voting system, en *Secure Electronic Voting*, pp. 77–99, Kluwer Academic Publishers, 2002.

[5] C. Gentry, A fully homomorphic encryption scheme, en `http://crypto.stanford.edu/craig/craig-thesis.pdf`.

[6] C. Gentry y S. Halevi, Implementing Gentry's Fully Homomorphic Encryption Scheme, en *Advances in Cryptology – EUROCRYPT 2011, Lecture Notes in Computer Science* 6632, pp. 129–148, Springer, 2011.

[7] J.L. Gómez Pardo, *Introduction to Cryptography with Maple*, Springer, 2012.

[8] J. Hoffstein, J. Pipher y J.H. Silverman, *An Introduction to Mathematical Cryptography*, Springer, 2008.

[9] J. Katz y Y. Lindell, *Introduction to Modern Cryptography*, Chapman & Hall/CRC, 2008.

[10] N. Koblitz y A. Menezes, Another look at "provable security", *J. Cryptology* **20** (2007), 3–37.

[11] K. Lauter, M. Naehrig y V. Vaikuntanathan, Can homomorphic encryption be practical?, en `http://eprint.iacr.org/2011/405`.

[12] J. Loftus, A. May, N.P. Smart y F. Vercauteren, On CCA-Secure Fully Homomorphic Encryption, en `http://eprint.iacr.org/2010/560`.

[13] P. Paillier, Public-key cryptosystems based on composite degree residuosity classes, en *Eurocrypt '99, Lecture Notes in Computer Science* 1592, pp. 223–238, Springer, 1999.

[14] R. Rivest, L. Adleman y M. Dertouzos, On data banks and privacy homomorphisms, en *Foundations of Secure Computation*, pp. 169–180, Academic, 1978.

José Luis Gómez Pardo, Dpto. de Álxebra, Universidade de Santiago de Compostela
Correo electrónico: `gomez.pardo@usc.es`