Universidad de Buenos Aires
Facultades de Ciencias Económicas,
Cs. Exactas y Naturales e Ingeniería

Carrera de Especialización en Seguridad Informática

Investigación de Cifrados Homomórficos y Lecciones para la Seguridad Informática

Autor:
Lukas Udstuen
Tutor:
Dr. Pedro Hecht

Marzo, 2021
Cohorte 2020

Por medio de la presente, el autor manifiesta conocer y aceptar el Reglamento de Trabajos Finales vigente y se hace responsable que la totalidad de los contenidos del presente documento son originales y de su creación exclusiva, o bien pertenecen a terceros u otras fuentes, que han sido adecuadamente referenciados y cuya inclusión no infringe la legislación Nacional e Internacional de Propiedad Intelectual.

Firmado,
Lukas Harry Udstuen
DNI: 95961401

RESUMEN

Mientras el mundo está cada vez más conectado, con cada vez más datos migrándose a la nube, cuestiones de seguridad todavía existen. Usuarios que eligen procesar datos en la nube muchas veces tienen que aceptar un cierto nivel de riesgo – que, para poder aprovechar de la nube, ceden la confidencialidad de sus datos, los datos descifrados al proveedor de los servicios. Los cifrados homomórficos pretenden ofrecer una solución: nos proveen la posibilidad de procesar datos cifrados sin tener que descifrarlos. Acá realizamos una investigación de los cifrados homomórficos conocidos hoy en día.

Palabras Claves: Cifrados homomórficos, privacidad, computación en la nube, confidencialidad

Índice

Introducción

En la gran mayoría de los casos, los usuarios no navegan el internet sin dejar huellas de sus actividades. En esta época de la información, muchas empresas habían identificado maneras para convertir la información de los usuarios a ingresos para sus compañías. Si bien es difícil calcular cuanto vale exactamente las huellas generadas por los usuarios, hay cifras que demuestran que el valor es importante. En el segundo cuarto de 2019, Google generó U$S 32,6 mil millones, mientras en el mismo periodo, Facebook generó U$S 16 mil millones [1]. Además, Facebook indicó que con la actividad de las publicidades y otros pagos, cada uno de sus 2,41 mil millones usuarios generó, en promedio, U$S 7,05 durante el mismo cuarto. Hoy en día, usuarios en general reconoce que dejamos rastros en el internet cuando aprovechamos de los servicios. Pero un campo de criptografía en particular podría dar más anonimidad a los usuarios y desarrolladores mientras naveguen el internet.

Los cifrados homomórficos son los que, más allá de cifrar los datos, permiten que las aplicaciones puedan realizar calculaciones con los datos cifrados. Históricamente, los cifrados podían cifrar y descifrar datos – pero cuando los datos estaban cifrados, no era posible sacar información, o cambiarlos [2]. Los cifrados funcionaban como una especie de caja negra, guardando los contenidos dentro de ella, pero prohibiendo cualquier tipo de cálculo contra los contenidos de la caja. Los cifrados homomórficos, por otro lado, nos permiten separar el deseo de descifrar los datos, y de realizar cálculos contra los mismos. Es decir, podemos realizar cálculos contra los datos cifrados sin la necesidad de descifrarlos. Este pequeño cambio de procedimiento, la habilidad de realizar cálculos contra los datos cifrados, había sido reconocido como el “Santo Grial” de la criptografía [3]–[6].

Por ejemplo, supongamos que, en una elección, tenemos los votos cifrados de todos los residentes de un país. Queremos contar los votos para averiguar quien ganó la elección. En este caso, la cantidad de votos es grande, y en este ejemplo, asumimos que

tenemos que utilizar la nube para realizar el conteo. Sin los cifrados homomórficos, podemos cifrar los datos mientras que los transmitamos al proveedor de la nube. Pero para realizar el conteo de votos, los servidores en la nube tendrían que descifrar los datos. Dado que es una elección, y los votos son confidenciales, no podemos permitir que nadie vea ni tenga la posibilidad de alterar ningún voto individual. Acá, los cifrados homomórficos nos ofrecen una solución; podemos mandar los valores de todos los votos de la elección a un servidor, y solicitar que el servidor cuente los votos. En este caso, si bien el servidor en la nube había contado todos los votos, nunca sabrá los valores originales de los votos en sí, porque no tendría la clave de desencriptación. Por ende, sería más difícil que alguien haya alterado los votos que un servidor que tuviera los votos guardados en texto plano.

Más allá de la privacidad, los cifrados homomórficos también tienen la posibilidad de crear nuevas oportunidades para negocios. Imaginamos dos empresas: una que tiene acciones, y otra tiene un algoritmo que puede indicar cual es el momento oportuno para comprar o vender las acciones. Para que las dos empresas puedan colaborar, hasta ahora era necesario cierto intercambio de información: o la primera empresa tiene que revelar los datos de sus acciones a la segunda, o la segunda tiene que revelar el algoritmo a la primera [3]. Si ninguna de estas situaciones resulta posible, las dos empresas no podrían trabajar juntos. La primera no podrá aprender cual sería el momento oportuno para comprar o vender acciones, y la segunda no puede monetizar su algoritmo.

Podemos imaginar situaciones en las que la cooperación entre dos entidades que, si bien no tienen suficiente confianza en el otro para divulgar su propiedad intelectual, podría producir nuevas oportunidades para descubrimiento [3].

Por último, también vemos la posibilidad de que los cifrados homomórficos podrían tener mucho impacto en medicina. Una aplicación usando cifrados homomórficos podría recibir toda la

información del ADN de una persona, y analizarlo por posibles enfermedades – sin saber el contenido del ADN en sí.

En este trabajo, realizaremos una investigación de los cifrados homomórficos conocidos hoy en día. Por el transcurso de los años, vemos que los cifrados homomórficos han empezado como una teoría nomás, y que paulatinamente se han convertido en una realidad. En este trabajo, exploraremos la evolución de los cifrados homomórficos desde la teoría hasta producción.

Se cree que esta clase de criptografía empezó en el año 1978 con la investigación de Rivest, Adleman y Dertouzos, en el Massachusetts Institute of Technology [2]. Si bien era posible cifrar datos para transmisión, y almacenamiento, los autores pensaban que existía otra posibilidad. Durante más de 30 años, había avances pequeños, pero mayormente el concepto quedó como teoría. Aparecieron varios cifrados conocidos como parcialmente homomórficos. Pero en el año 2009, hubo un salto importante: Craig Gentry publicó su tesis, dónde efectivamente logró diseñar el primer cifrado totalmente homomórfico [7]. Si bien era demasiado lento para estar usado en realidad, mostró que era posible juntar fragmentos de lo previamente desarrollado para diseñar un cifrado totalmente homomórfico.

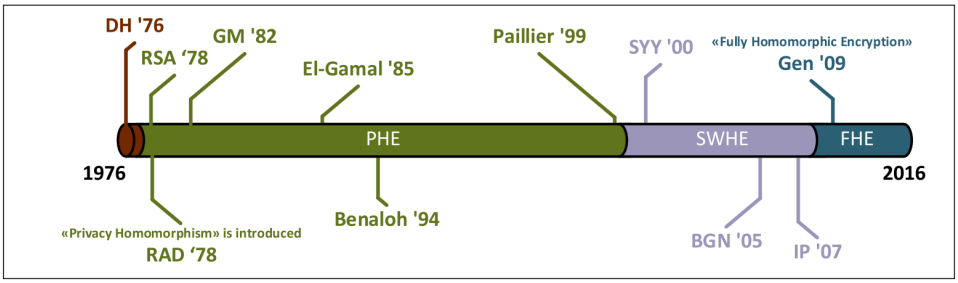

Figura 1.0: El desarrollo de los cifrados homomórficos a través de los años

Fuente: Adaptado de [8]

En este trabajo, trazaremos el desarrollo de los cifrados homomórficos por los años, y realizamos una investigación del funcionamiento de cada cifrado. Después, dejaremos el lector con una

vista hacía el futuro, mostrando los avances más recientes en el campo de los cifrados homomórficos.

Un primer acercamiento

Muchas veces, los problemas más complicados pueden ser entendidos más fácilmente a través de la simplificación. En la computación, podemos hacernos un gran favor separando los resultados – lo que estamos cifrando (sean documentos, imágenes o películas) – de los datos. Gentry, explicando los cifrados homomórficos, hace una simplificación, explicando que al final, los datos que queremos cifrar son ni más ni menos bits – 0s y 1s [9]. Entendiendo eso, los cifrados homomórficos, entonces, son los que nos permiten realizar las operaciones de adición y multiplicación en los textos cifrados – los bits [9].

Expresando la misma idea en términos matemáticos, podemos entender que un cifrado es homomórfico para la operación de multiplicación si cumple con la siguiente ecuación [8]:

$$E(m_1) \cdot E(m_2) = E(m_1 \cdot m_2), \forall\, m_1, m_2 \in M$$

Acá, indicamos que si tenemos dos mensajes (m_1 y m_2), si multiplicamos los dos mensajes, y después los ciframos, tendremos el mismo resultado (en el texto plano, obviamente) que si primero habíamos cifrado los mensajes, multiplicado los mensajes, y después evaluado resultado descifrado.

Como vemos en *figura 1.0*, hay varias clases de cifrados homomórficos que había aparecido a través de los años. Existen tres categorías de cifrados homomórficos [8]. Las variables que cambian entre las tres clases son dos: el número de operaciones soportadas, y la cantidad de veces que podemos realizar operaciones. Primero, los cifrados *parcialmente homomórficos* (*partially homomorphic*) permiten una operación una cantidad ilimitada de veces, pero solo con un solo tipo de operación (adición o multiplicación). Después, existen los cifrados bastantes homomórficos (*somewhat homomorphic*). Esta clase de cifrados permite realizar algunas operaciones homomórficas

(no todas) en una cantidad limitada de veces [8]. Exploraremos las limitaciones de los cifrados parcialmente homomórficos (CPH) más a continuación. Por último, existen los cifrados totalmente homomórphicos (*fully homomorphic*), que permiten todas las operaciones (adición y multiplicación) una cantidad ilimitada de veces.

Aclaraciones – lo que no es un cifrado homomórfico

Cabe mencionar que en un análisis de los cifrados homomórficos, aparece otro fenómeno similar, que sería la ofuscación [3]. Fundamentalmente, el objetivo de los cifrados homomórficos es permitir que los usuarios puedan realizar operaciones contra datos cifrados. Los cifrados homomórficos suelen ofrecer varias claves, cada una con un propósito distinto. Tanto como los cifrados comunes, los cifrados homomórficos requieren una clave para descifrar el texto cifrado [10]. Pero los cifrados homomórficos también incluyen otra clave para realizar operaciones contra los datos cifrados, una que no ofrece la habilidad de ver los datos cifrados.

Por otro lado, ofuscación sería conceptualmente similar a la idea de la criptografía homomórfica, pero con unos aspectos distintos. Primero, la ofuscación representa una especie de caja negra, dónde un usuario puede mandar sus datos a un programa, y el programa realice ciertas operaciones contra los datos cifrados. Autores habían mencionado, por ejemplo, la posibilidad de esconder una clave en un programa, y dejar que el programa entregara los resultados al final [3]. Teóricamente esta especie de programa podría sustituir los cifrados homomórficos, pero hay que analizar ciertas restricciones de seguridad que aparecen cuando uno esté escondiendo llaves adentro de un programa ya publicada [3], [7].

Revisión de los cifrados homomórficos conocidos hoy en día

A continuación, repasamos muchos de los cifrados homomórficos reconocidos hoy en día. En general, como vemos en la Figura 1.0, vemos que por los años, los cifrados homomórficos

soportan cada vez más operaciones, con la habilidad de realizar cálculos cada vez más rápido.

RSA: El primer cifrado usando llaves públicas que tiene propiedades homomórficas

La noción de un cifrado homomórfico apareció un poco después del invento de RSA, por Rivest, Shamir y Adleman en los finales de los años 70 [7]. Cifrando los mensajes con el uso de los números primos, RSA es el primer cifrado de tipo clave pública que era reconocido por tener propiedades homomórficas [11]. RSA es un cifrado parcialmente homomórfico – es decir, que permite operaciones homomórficos una cantidad ilimitada de veces, pero solo con algunas operaciones [8]. Específicamente, RSA permite operaciones homomórficas con multiplicación. Para entender como semejantes operaciones podrían funcionar, revisamos los pasos del cifrado RSA:

Generación de claves: Primero, es necesario seleccionar números primos grandes, p y q. Después calculamos otro número, $n = pq$. También, usando el phi de Euler, calculamos la cantidad de números coprimos con p y q, y los multiplicamos. $\varphi = (p-1)(q-1)$. Finalmente, buscamos un número, e, que tenga $mcd(e, \varphi)$ y d usando el inverso multiplicativo de $ed \equiv 1 \; mod \; \varphi$. Con esto, tenemos los dos pares que necesitamos: (e, n) sería la clave pública, y (d, n) la clave privada. [8]

Cifrando un mensaje: En RSA, cuando estamos cifrando mensajes, realmente lo que estamos haciendo es elevar el número del mensaje al número que habíamos declarado como e, y tomando el resultado modulo n. Es decir:

$$c = E(m) = m^e \; (mod \; n), \forall m \in M$$

[8]

RSA nos sirve como cifrado porque el proceso indicado anteriormente para cifrar los mensajes es relativamente fácil de computar. Por otro lado, el proceso de intentar descifrar el mensaje sin saber el inverso multiplicativo que habíamos declarado en e sería más complicado. Rivest, Shamir y Adleman marcan eso como requerimiento de un sistema de criptografía público, un problema que sea fácil solucionar con todos los números del problema, pero muy difícil solucionar con solo un subconjunto de los números, por ejemplo los datos de la clave pública [12].

Descifrar: Los mensajes cifrados con RSA, entonces, pueden ser recuperados elevando el mensaje cifrado al inverso multiplicativo, modulo n. Recordemos que habíamos identificado previamente el multiplicativo inverso, y lo teníamos guardado como d.

$$m = D = c^d \ (mod\ n)$$

[8]

Las varias décadas usando RSA demuestran que el cifrado nos provee una herramienta segura para cifrar los mensajes, no es un cifrado seguro en sí. Es necesario agregar algún tipo de padding para que el cifrado sea seguro contra ataques. Este tipo de RSA no es seguro porque no contiene una protección contra un ataque de texto plano elegido; por ello, es muy común que en la práctica, RSA incluye algún tipo de relleno [13], [14].

Mayormente, cuando se habla de las propiedades homomórficas de RSA, hablan de operaciones de multiplicación; operaciones de adición romperán la posibilidad de descifrar los mensajes [8]. En ciertos casos, se han demostrado que RSA también tiene propiedades homomórficos con división [15]. Aunque es muy común que los investigadores mencionen que RSA es un cifrado homomórfico con operaciones de multiplicación, las

operaciones de división no han sido ampliamente investigadas [8], [16].

Cifrado Goldwasser-Micali

El cifrado Goldwasser-Micali (GM) fue creado en en 1982, unos cuatro años después de la introducción de RSA. El cifrado fue introducido por Shafi Goldwasser y Silvio Micali [11]. A diferencia que el esquema de RSA mencionada anteriormente, que no incluye un tipo de procesamiento previo para agregar un relleno, el cifrado GM es un cifrado probabilístico. Esto quiere decir que el cifrado incluye un elemento de aleatoriedad, implicando que el mismo mensaje, cifrado dos veces, no tendrá el mismo texto cifrado [11]. Es más, en todos los cifrados que usan una clave pública, el cifrado GM fue el primer en lograr crear un cifrado probabilístico [17], [18].

El funcionamiento de este cifrado depende de que estemos algo que tenga dos posibilidades del resultado – o sea, bits. Al final, evaluando el mensaje cifrado, determinamos si el número es un residuo cuadrático; en el caso de sí, $m_i = 0$, y si no, $m_i = 1$ [11].
Cabe mencionar también que si bien el cifrado GM presenta un valor académico importante, es un cifrado impráctico para usar en aplicaciones en internet; debido al funcionamiento del cifrado, para mantener un nivel de seguridad aceptable, los textos cifrados son 1000 más largo que los textos planos[18], [19].

En cambio del cifrado RSA, el cifrado GM permite las operaciones de adición. Entre todos los cifrados parcialmente homomórficos, los cifrados que permiten operaciones homomórficas de adición son mucho más comunes de los que permiten operaciones de multiplicación [8].

Generación de claves: El modulo que usamos en GM es el mismo que usamos en RSA [11]. Primero, tenemos que buscar dos números primos que estén iguales, cada uno, a $3\ mod\ 4$. Similar también a RSA, calculamos un valor N, multiplicando los dos números

primos mencionados anteriormente $(p,\ q)$. Por último, buscamos un valor "a" que cumple con dos criterios[11]:

$$a_p^{\frac{p-1}{2}} = -1(mod\ p), a_q^{\frac{q-1}{2}} = -1(mod\ q)$$

[11]

Con este proceso tenemos toda la información necesaria para cifrar y descifrar los mensajes: la clave pública sería (a, N) y para descifrar, sería la factorización de (p, q) [11].

Cifrar los mensajes: Primero, convertimos nuestro mensaje a un *string* de bits [11]. Este paso es necesario, porque como habíamos mencionado antes, cuando estamos descifrando el mensaje, el cifrado nos da dos posibilidades (0 o 1) dependiendo de si el resultado es un residuo cuadrático. Entonces, para cada bit del mensaje, tenemos que buscar un valor, b_i, para que $\gcd(b_i, N) = 1$ [8]. Entonces, procedemos a cifrar los mensajes:

$$c_i = E(m_i) = b_1^2 x^{m1}(mod\ N) \forall\ mi = \{0,1\}$$

[8]

Descifrando los mensajes: La clave para descifrar los mensajes reside en la utilización de nuestra clave privada, *(p, q)*. Iteramos, bit por bit – y si encontramos que un bit (c_i) es un residuo cuadrático, podemos deducir que $m_i = 0$. Caso contrario, $m_i = 1$.

Este cifrado es homomórfico con operaciones de adición. En este caso, estamos operando en un mundo de dos posibilidades, 0 y 1. Por lo tanto, vemos que cuando , $m_0 = m_1 = 1, m_0 + m_1 = 2$ y $c_0 c_1 (mod\ n)$ es un residuo cuadrático, y entonces, es el equivalente de 0 cifrado [11].

ElGamal

El algoritmo ElGamal apareció en el año 1985, y desde su primera concepción, distintas variaciones habían aparecido [8], [20], [21]. ElGamal fue el primer cifrado de tipo clave pública que ofrecía seguridad basado en los problemas de los logaritmos discretos [20], [22].

Un año después de su publicación, aparecieron dos artículos importantes, escritos por Neal Koblitz y V.S. Miller, que propusieron el uso de curvas elípticas en criptografía – y concretamente, incluyeron varias posibilidades de curvas elípticas usando el cifrado ElGamal [23]–[25].

Las curvas elípticas presentan posibilidades muy interesantes para el futuro de criptografía porque se cree que son problemas computacionalmente imposible de romper; junto con ElGamal, Diffie-Hellman es otro algoritmo prometedor que se puede usar con curvas elípticas [26].

Generación de Claves: Primero, es necesario crear un grupo cíclico, G. Con eso, calculamos $H = g^y$ para un número y, que forme parte del grupo $\mathbb{Z}_n^*$ [8]. Con esto, podemos crear la clave pública, que es (G, n, g, h) y x, que es la clave secreta de la esquema [8].

Cifrar los mensajes: Los cifrados, en el cifrado ElGamal, toman la forma de un par: $c = (c_1, c_2)$ [8]. Para cifrar los mensajes, necesitamos dos números, g y x; x sería un número escogido al azar entre la colección $\{1, 2..n-1\}$ [8]. Entonces, el algoritmo para cifrar los mensajes sería:

$$c = E(m) = (g^x, mh^x) = (g^x, mg^{xy}) = (c_1, c_2)$$

[8]

Descifrar los mensajes: Para descifrar los mensajes, primero tenemos que calcular un valor s usando los valores de la llave privada; calculamos $s = c_1^y$. Poseyendo dicho valor, podemos descifrar el texto cifrado así:

$$c_2 \cdot s^{-1} = mg^{xy} \cdot g^{-xy} = m$$

[8]

Propiedad homomórfico: Vemos, entonces, que el cifrado ElGamal posee una propiedad homomórfica, respecto a operaciones multiplicativas.

$$E(m_1) \cdot E(m_2) = (g^{x_1}, m_1 h^{x_1}) \cdot (g^{x_2}, m_2 h^{x_2}) = (g^{x_1+x_2}, m_1 \cdot m_2 h^{x_1+x_2})$$
$$= E(m_1 \cdot m_2)$$

[8]

Cabe mencionar que el cifrado ElGamal explicado acá (el que se suele enseñar en los libros de texto) no es un cifrado seguro a menos que haya cierto tipo de pre-procesamiento en el texto descifrado [14].

Benaloh

El cifrado Benaloh es similar al cifrado GM (Goldwasser-Micali), pero con algunas mejoras; el cifrado Benaloh permite cifrar los mensajes en bloques, en vez de cada bit individualmente [8]. Debido a las mejoras en rendimiento a comparación a GM, y a sus propiedades homomórficas, varios investigadores sí habían identificado usos factibles para el cifrado Benaloh, incluyendo por ejemplo esquemas de votación, calculo de confianza entre varios usuarios, y juegos de póker online [27]. El cifrado está basado en el problema de residuosidad superior (*higher residuosity problem)*, que está relacionado al problema de la residuosidad cuadrática que se usa en los cifrados GM [8], [28].

Generación de claves: Este cifrado, como muchos mencionados anteriormente, empieza con la generación de dos números primos que sean grandes. Acá, indicamos esos valores con el nombre p y q [29]. Después, similar al cifrado RSA, calculamos $n = pq$. Después, calculamos el $\varphi(n)$, que sabemos que, por tratar de dos números primos, es el equivalente de $(p-1)(q-1)$ [29].

Después, tenemos que indicar el tamaño de los bloques (r) que vamos a usar. El tamaño que indicamos para r tiene que cumplir con tres condiciones [29]:

- r es divisible por $p-1$
- $\gcd\left(r, \frac{p-1}{r}\right) = 1$
- $\gcd(r, q-1) = 1$

También, buscamos un valor *y*, para que $x = y^{\frac{\varphi(n)}{r}} mod\ 1\ \neq 1$ [29] Finalmente, tenemos nuestros valores [29]:

- Llave privada: (p, q)
- Llave pública: (y, r, n)

Cifrar mensajes: Para cifrar los mensajes, primero, seleccionamos un número *u*, que cumpla con la condición $0 < u < n$ [29].

El texto, entonces, lo ciframos con la siguiente ecuación:

$$c = y^m u^r \ mod\ n$$

[29]

Descifrar Mensajes: Podemos descifrar los mensajes con la siguiente ecuación:

$$a = c^{\frac{\varphi(n)}{r}} mod\ n$$

[29]

Propiedad homomórfica: Analizando el cifrado, vemos que cualquier operación multiplicativa en el texto cifrado corresponde a una operación aditiva en el texto descifrado[29].

Expresado matemáticamente, para cada bit $m_i \in \{0, 1\}$:

$$E(m_1) \cdot E(m_2) = \left(y_1^2 x^{m_1}\ (mod\ n)\right) \cdot \left(y_2^2 x^{m_2}\ (mod\ n)\right)$$
$$= (y_1 \cdot y_2)^2 x^{m_1 + m_2}\ \ (mod\ n) = E(m_1 +\ m_2)$$

[8]

Naccache & Stern

En 1998, David Naccache y Jaques Stern publicaron un cifrado basado en el cifrado *Benaloh*, pero que ofrecía varias ventajas en términos de eficiencia [8]. Este cifrado, aunque más lento que RSA, todavía representa un logro importante, y podría servir para varias aplicaciones [30]. Hay dos versiones del cifrado, uno es determinístico, y el otro probabilístico. Cómo habíamos visto antes, usando la versión determinística, si el mismo mensaje es cifrado dos veces, tendrá el mismo texto cifrado o no.

Cabe mencionar que un par de años antes, en 1995, Naccache & Stern publicaron un cifrado *knapsack* (mochila), que es distinto que el cifrado analizado acá [31].

Cifrar los mensajes: Primero, creamos una familia p_i de k primos impares distintos, con el número k siendo par [30]. Después calculamos $u = \prod_{i=1}^{\frac{k}{2}} = p_i$, $v = \prod_{\frac{k}{2}+1}^{k} p_i$ y finalmente $\sigma = uv = \prod_{i=1}^{i} p_i$.

Después, tenemos que escoger dos números, a y b, que el resultado de las siguientes dos ecuaciones también son números primos [30]:

- $p = 2au + 1$
- $q = 2bv + 1$

El proceso de seleccionar números que cumplan con dichas condiciones podría resultar complicado; Naccache y Stern ofrecen algunas sugerencias en su paper, *A New Public Key Cryptosystem Based on Higher Residues* [30].

Por último, como en otros cifrados, calculamos $n = pq$, y buscamos un número g para que $g \, mod \, n$ sea del orden $\frac{\varphi(n)}{4}$. Lo importante es que para cada número $i \leq k$, probando para cada número que $g^{\frac{\varphi(n)}{p_i}} \neq 1 \, mod \, n$ [30].

Cifrar los mensajes: El proceso de cifrar los mensajes es sencillo; consiste en una exponenciación modular [30]. En el caso de que un mensaje, *m*, sea menos que σ, podemos cifrar el mensaje haciendo $g^m \, mod \, n$.

Descifrar los mensajes: Para descifrar los mensajes, seguimos un proceso que tiene varias fases, y termina con el teorema chino del resto. Naccache y Stern explica la primera fase a través de un algoritmo:

$$\begin{aligned}
&\texttt{for } i = 1 \texttt{ to } k \\
&\{ \\
&\quad \texttt{let } c_i = c^{\phi(n)/p_i} \bmod n \\
&\quad \texttt{for } j = 0 \texttt{ to } p_i - 1 \\
&\{\texttt{if } c_i == g^{j\phi(n)/p_i} \bmod n \texttt{ let } m_i = j\} \\
&\} \\
&x = \texttt{ChineseRemainder}(\{m_i\}, \{p_i\})
\end{aligned}$$

Figura 1.2: Algoritmo para descifrar mensajes cifrado con el cifrado Naccache/Stern.

Source: Adaptado de [30]

Una vez que el algoritmo encuentre el resultado que cumpla con $c_i = g^{\frac{j\varphi(n)}{p_i}}$, aplicamos el teorema chino del resto a todos los resultados m_i para recuperar el valor del mensaje m.

Propiedad homomórfica: El cifrado es homomórfico con operaciones de adición [8].

Okamoto & Uchiyama

El cifrado Okamoto & Uchiyama (OU) no solo tiene la distinción de ser un cifrado homomórfico, sino que también un cifrado probabilístico [32]. El cifrado, introducido en 1998, tenía el objetivo de mejorar el rendimiento de los cifrados, cambiando la colección dónde opera el cifrado; ese cifrado todavía trabaja con $\mathbb{Z}_n^*$, pero establece $n = p^2q$ [8].

El cifrado OU tenía el objetivo de agregar un "trapdoor" al problema de la curva elíptica de logaritmo discreto, en una curva elíptica en un anillo $\frac{\mathbb{Z}}{N\mathbb{Z}}$ donde N es el producto de los primos pq [33]. Al final, los autores aceptaron que su cifrado carecía seguridad, porque era posible derivar la factorización de N de la información pública en el cifrado [33].

Generar las claves: Las claves secretas están representadas por las letras p y q, y la clave pública está formada del cálculo $n = p^2q$ [32]. Para empezar, tenemos que seleccionar dos números primos, grandes, que tienen el mismo longitud (*k*-bits) [32]. Cómo habíamos indicado anteriormente, la clave pública es $n = p^2q$ [32].
Mientras tanto, también tenemos que elegir un generador

$$g \in \left(\frac{\mathbb{Z}}{p^2\mathbb{Z}}\right)*$$

tal que $g_p = g^{p-1} mod\ p^2 \neq 1$ y $g^{p(p-1)}\ mod\ p^2 = 1$. Después, asignamos otra clave pública $h = g^n \in \mathbb{Z}/n\mathbb{Z}$ [32]. Al final, tendremos las claves públicas g, h, y n

Cifrar los mensajes: Para m que ciframos, el cifrado OU también exige que agregamos un elemento de aleatoriedad, r. Entonces, para cifrar los mensajes, usamos el algoritmo:

$$C = E(m, r) = g^m \cdot h^r \; mod \; n$$

[34]

Si simplificamos el proceso de cifrar el mensaje, esencialmente estamos siguiendo el fórmula $c = E(m, r)$ [32].

Descifrar el mensaje: Para descifrar nuestro mensaje, tenemos el siguiente algoritmo:

$$m = \frac{L(C_p)}{L(g_p)} mod \; p$$

[34]

Propiedad Homomórfica: El es uno de los varios cifrados que tiene una propiedad homomórfica.

$$E(m_0, r_0) \cdot E(m_1, r_1) = E(m_0 + m_1, r_2)$$

[34]

La propiedad homomórfica depende de que $m_0 + m_1 < p$ [34].

Paillier

El cifrado Paillier es similar a dos otros cifrados analizado anteriormente, Benaloh y Goldwasser-Micali [8]. El cifrado Paillier está basado en el problema de decisión de la residuosidad compuesta [8], [35]. Este problema tiene que ver con la búsqueda de un entero, x, que satisfaga la ecuación $x^n \equiv a(mod \; n^2)$ para cada entero, a [8].

El cifrado Paillier fue introducido en el año 1999 por Pascal Paillier, y es un cifrado seguro contra los ataques CPA (*chosen plaintext attack*) [36].

Generación de claves: Primero, como en muchos otros cifrados, empezamos eligiendo dos números primos grandes, p y q, y los combinamos para crear la clave pública, $n = pq$ [35]. La clave privada, entonces, requiere el phi de n, que podemos hallar usando $\varphi(n) = (p-1)(q-1)$; la clave privada es $(n, \varphi(n))$ [35].

Para cifrar los mensajes: Tanto como en el cifrado OU, el cifrado Paillier incluye un elemento de aleatoriedad. Para cifrar un mensaje $m \in \mathbb{Z}_n$, elegimos primero un número $r \in \mathbb{Z}_n^*$ al azar [35].

Todos nuestros textos claros se encuentran en $\mathbb{Z}_n$, y los textos cifrados en $\mathbb{Z}_{n^2}^*$ [35]. El algoritmo, entonces, para cifrar los mensajes es el siguiente:

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

[35]

Descifrar los mensajes: Para descifrar los mensajes, utilizamos la clave privada, y calculamos:

$$m = Dec((n, \varphi(n)), c) = \frac{(c^{\varphi(n)} mod\ n^2) - 1}{n} \cdot \varphi(n)^{-1}\ mod\ n$$

Propiedad Homomórfica: Paillier es uno de los varios cifrados contra operaciones de adición. Para lograr la adición contra los mensajes descifrados, tenemos que multiplicar los mensajes cifrados.

$$E(m_1) \cdot E(m_2) = \left((1+n)^{m1} r_1^n\ (mod\ n^2)\right) \cdot \left((1+n)^{m2} r_2^n\ (mod\ n^2)\right)$$
$$= (1+n)^{m_1+m_2} (r_1 \cdot r_2)^n\ (mod\ n^2) = E(m_1 + m_2)$$

[8], [35]

Damgård & Jurik

El cifrado *Damgård and Jurik* (DJ) es una generalización del cifrado Paillier [8]. Tanto Paillier como DJ permiten operaciones homomórficas con adición. El cifrado DJ entonces está calificado también como un cifrado parcialmente homomórfico. En el trabajo que presentó su cifrado epónimo, los autores, Dåmgard y Jurik, mencionan que el cifrado Paillier tiene varias ventajas – principalmente, que permite el usuario cifrar varios bits en una sola operación [37]. Sin embargo, los autores indicaron que encontraron una manera de simplificar el cifrado de Paillier en varios aspectos, mientras todavía manteniendo las partes más atractivas del cifrado, y sin introducir problemas de seguridad [37]. La simplificación del cifrado significa que, por ejemplo, si el cifrado estuviera usado para contar los votos en una elección, el poder computacional requerido para contar los votos es reducido considerablemente [37].

El cifrado DJ opera en base a un modulo n^{s+1}, para cada $s \geq 1$. El cifrado DJ también permite que el factor de expansión puede

ser reducido considerablemente también, de un factor de 2 a casi 1 [37].

Generación de Clave: Para empezar, elegimos un módulo RSA, $n = pq$, que sea de longitud $k\ bits^2$ [37]. Acá, p y q tienen que ser primos impares; sigue, entonces, que $\mathbb{Z}^*_{n^{s+1}}$ como grupo multiplicativo es un producto de $G \cdot H$, dónde G es cíclico del orden n^sy H es isomórfico a $\mathbb{Z}^*_n$ [37]. Buscamos también, un elemento $g \in \mathbb{Z}^*_{n^{s+1}}$, tal que $g = (1+n)^j x\ mod\ n^{s+1}$, donde j es relativamente primo a n, y $x \in H$ [37]. Declaramos γ el mínimo común múltiplo entre p y q; dado que los dos son primos, podemos deducir que $\gamma = (p-1)(q-1)$. Por último, con el teorema chino del resto, buscamos un valor d tal que $d\ mod\ n\ \in \mathbb{Z}^*_n$ y $d = 0\ mod\ \gamma$ [37].

Cifrar los mensajes: Entendemos que el texto plano es Z_{n^s}, y el texto plano es i. Entonces, tanto como en Paillier, elegimos un número al azar, $r \in \mathbb{Z}^*_{n^{s+1}}$ [37]. Calculamos el texto cifrado entonces, con el algoritmo: $E(i,r) = g^i r^{n^s} mod\ n^{s+1}$ [37].

Descifrar los mensajes: Para descifrar los mensajes, solo tenemos que computar $c^d mod\ n^{s+1}$ [37]. Los autores demuestran el siguiente:

$$c^d = (g^i r^{n^s})^d = ((1+n)^{ji} x^i r^{n^s})^d = (1+n)^{jid\ mod n^s} (x^i r^{n^s})^{d\ mod\ \gamma}$$
$$= (1+n)^{jid\ mod\ n^s}$$

[37]

Propiedad homomórfica: Vemos que el cifrado es cifrado; el producto de los mensajes cifrados i, i' es equivalente al texto cifrado de $i + i' mod\ n^s$ [37].

Kawachi, et. al.

El esquema conocido como Kawachi, et. al. es reconocido por tener propiedades homomórficas, aunque en el proceso de descifrar el texto cifrado, aparecen pequeños errores [8]. Publicado en 2007 por los autores Kawachi, Tanaka y Xagawa, el esquema usa grupos cíclicos grandes, y está basado en los problemas de látices [5]. Los autores en efecto, proponen cifrados multi-bit de varios cifrados

existentes, que usan un solo bit [8]. La técnica no aumenta el tamaño de los textos cifrados [38]. Dado que su técnica produce pequeños errores cuando el texto está descifrado, los autores eligieron el nombre *pseudohomomórfico* [8].

Galbraith

Galbraith presentó una continuación del trabajo de Paillier. Paillier había creado un cifrado probabilístico, que tenía propiedades homomórficas – el cifrado encontró varias aplicaciones [33]. Después, Paillier había intentando crear otro cifrado, una generalización del cifrado usando curvas elípticas anómalas en anillos; este cifrado no era seguro, pero sirvió como la fundación para el trabajo de Galbraith [33].

Generación de claves: Para empezar, cada usuario selecciona un módulo $N = pq$, que es el producto de dos primos impares. Después, los usuarios seleccionan una curva elíptica al azar: $E: y^2z = x^3 + axz^2 + bz^3$en $\frac{\mathbb{Z}}{N\mathbb{Z}}$ (esto es, $mcd\big(N, 6(4a^3 + 27b^2)\big) = 1$), [33]

Después, calculamos $M = mcm(\#E(\mathrm{F}_p), \#E(\mathrm{F}_q))$ [33]. Galbraith comenta que este número podemos encontrar usando el algoritmo Schoof-Atkin-Elikes en tiempo polinomial si sabemos los valores p y q.

Por último, es necesario obtener un valor $Q = (x: y: z)$, que divide M en $E(\frac{\mathbb{Z}}{N^2\mathbb{Z}})$ [33]. Galbraith indica que podemos tomar un punto al azar $Q' = (x': y': z')$, y dejar $Q = NQ'$ [33].

Cifrar un mensaje: Primero, tenemos que seleccionar un número al azar, $1 \leq r < N$, para poder calcular $S = rQ + P_m$ [33].

Descifrar un mensaje: Para descifrar el mensaje, es fundamental que tengamos el valor M, que sirve como la clave secreta. Entonces, calculamos $MS = r(MQ) + MP_m = P_{mM} = (mMN: 1: 0)$ [33] Si tenemos el coordinado x, podemos dividirlo por N, y multiplicarlo por el inverso de $M\ mod\ N$ para recuperar $m \in \frac{\mathbb{Z}}{N\mathbb{Z}}$ [33].

Propiedad homomórfica: Como otros cifrados analizados, este cifrado tiene la ventaja de ser cifrado con operaciones de adición. Si entendemos que S_1es el mensaje cifrado de m_1, y S_2 es el mensaje cifrado de m_2, entonces $(S_1 + S_2)$ es equivalente al texto cifrado de $(m_1 + m_2)$ [33]

Galbraith presentó una generalización de un cifrado de Paillier, que está basado en la curvas elípticas [5].

Polly Cracker

Había varios intentos de usar las bases Gröbner para construir cifrados a través de los años; muchos de los intentos hasta la fecha habían fallados [39]. Investigadores habían incluido todos los intentos bajo el nombre “Polly Cracker” [40]. El interés en el uso de las Bases de Gröbner empezó en los años 90, cuando las vieron como una alternativa a la algebra multivariante. En 1994, Barkee, Can, et. al. publicaron una carta abierta explicando por que el uso de las Bases de Gröbner no era factible.

Cuando estamos analizando cifrados, es fundamental que la información confidencial adentro de nuestro algoritmo no sea fácilmente hallado. En las primeras páginas, explican que la suposición que las Bases de Gröbner son difíciles de computar es falso [41]. Los autores muestran un procedimiento, que es similar a la reducción de matrices usando el método de Gauss. Sin embargo, el cifrado tiene valor académico, y presenta un caso interesante con los cifrados homomórficos. Al final, los autores recomendaron al usuario que si un investigador quería seguir investigando los cifrados que usen las ecuaciones multivariantes, que investiguen algoritmos más dispersos (*sparse*) [41].

Cuando evaluamos lo interesante de la seguridad de los cifrados Polly Cracker reside en el uso de las Bases de Gröbner como una función *trap-door*. Esto significa que si tenemos todos los valores privados, es muy fácil calcular el resultado de la función – pero si solo contamos con los valores públicos, resulta muy difícil hacer lo mismo.

La clave para los cifrados Polly Cracker es un par de ideales en un anillo polinomial multivariante [39]. Mantenemos una de las bases como secreta, y la otra es muy difícil hallar. Ciframos los mensajes en un vector de polinomiales [39]. Pero este tipo de cifrado sufre de dos falacias. Primero, la seguridad solo existe asintóticamente. Segundo, se puede realizar un ataque contra solo solucionando un problema de los mensajes en su forma normal, no requiere la solución de un problema de forma normal genérica [39].

El cifrado Polly-Cracker genérico es muy sencillo. Alice genera un punto en un vector, y, y un conjunto de polinomiales $\{q_i\}$ que llegan a 0 en el punto y [42]. Para mandar un bit, entonces, Bob tiene que generar una suma $p = \sum g_i q_i$, y le manda a Alice el polinomial $p + m$ [42]. Para descifrar, Alice evalúa el texto cifrado en el secreto, $c \; mod \; q = m$ para reccibir el mensaje, m [9]. Analizando lo descrito acá, vemos que el cifrado Polly Cracker tiene la distinción de tener una propiedad homomórfica tanto de adición como de multiplicación. El cifrado, como menciona Gentry, no es seguro porque existen varios tipos de ataques. Uno de los ataques más importantes es la Eliminación de Gauss; usando este método, que en general se usa para solucionar ecuaciones lineales, podemos hallar el texto cifrado [9]. Adicionalmente, si podemos juntar muchos textos cifrados del número 0, podemos hallar información importante del texto cifrado. Los mensajes generan una cuadrícula, que podemos llamar L; tenemos que calcular la Forma Normal de Hermite en base a nuestra cuadrícula (L). Con el resultado, reducimos $c_1 - c_2 \; mod \; HNF(L) -$ si el resultado sería 0 si $m_1 = m_2$ [9].

Noisy Polly Cracker

Si bien las vulnerabilidades del cifrado representan un riesgo considerable, tenemos una forma de obviarlas: si agregamos ruido al cifrado, el mensaje original se mantiene escondido incluso frente a los ataques anteriormente mencionado [9]. Similar a lo que pasa con el ruido en un entorno normal con dos locutores, podemos captar un mensaje si bien tenemos que competir con ruido al fondo. En este

caso del código específicamente, el ruido consiste en códigos "escondidos" que en la mayoría de los casos, los podemos sacar al momento de descifrar el mensaje [9]. Pero tanto como en la vida real, existe la posibilidad de agregar demasiado ruido; si pasamos el umbral, no podremos recuperar ni el texto plano del mensaje ni la anécdota que un amigo nos estaba intentando contar [9].

En este contexto, nos interesa analizar las propiedades homomórficas de los cifrados. Aunque los cifrados Polly Cracker soportan operaciones homomórficas de adición y multiplicación, estas dos operaciones no tienen el mismo impacto en la presencia del ruido. Si sumamos los textos cifrados, en efecto estamos agregando el ruido. Similarmente, si multiplicamos los textos cifrados, estamos multiplicando también el ruido. Estas operaciones las podemos realizar, pero después de cierto punto, pasamos el umbral y el mensaje se vuelve corrupto [9].

Si analizamos el funcionamiento del cifrado Noisy Polly Cracker, es muy similar al cifrado explicado anteriormente, Polly Cracker; Gentry indica que los codificaciones de 0 son esencialmente polinomiales que evalúan a 0 con la clave secreta, pero acá, los polinomiales también tienen que ser números pares y pequeños (*smevens*) [9].

Para generar una clave, primero elegimos un punto en un vector, lo cual podemos representar matemáticamente $(s_1, \dots, s_n)2\ Z_q^n$. Cuando mencionamos los números pares y pequeños, tienen que ser "smeven" en relación a q en la clave secreta.

La clave pública, entonces, consistiría en varias codificaciones del número 0 en el punto elegido en el vector. Para cifrar los mensajes, primero seleccionamos un conjunto de números de la clave pública, para obtener una codificación de 0 al azar [9]. Para cifrar el mensaje, entonces, agregamos el mensaje a la codificación de 0, usando la clave secreta. En este caso, sería $c(x_1, \dots, x_n) = m + g(x_1, \dots, x_n)\ mod\ q\ \forall\ m \in \{0, 1\}$ [9]. Para descifrar entonces nuestro mensaje, solo tenemos que volver a calcular el mensaje usando la clave secreta; el resultado que tenemos es el equivalente al mensaje

más el número par y pequeño; entonces, podemos obtener el mensaje original reduciendo el resultado modulo 2 [9]. El “ruido” que habíamos mencionado, entonces, es ese mismo número, pequeño y par. Cabe mencionar que este cifrado es homomórfico para operaciones de multiplicación y adición; pero mientras más veces que hagamos operaciones homomórficas, el ruido sigue creciendo; si llega a ser más grande que el número q, el cifrado deja de funcionar [9].

Dificultades con los cifrados homomórficos

Si bien los cifrados homomórficos habían logrado captar el interés de muchos criptógrafos, existen varias dificultades con una gran porción de los cifrados homomórficos en los últimos años. Las dificultades cubren toda la gama desde problemas del rendimiento hasta problemas de seguridad que amenazan socavar los cifrados en sí [43]. Sin embargo, hay algunos cifrados totalmente homomórficos que sí lograron a ser prácticos para el uso en producción.

Los cifrados con que no permiten todas las operaciones homomórficas (cifrados parcialmente homomórficos), o que solo permiten las operaciones homomórficas una cierta cantidad de veces (cifrados bastantes homomórficos), en general durante los últimos años, habían sido más rápidos que los cifrados totalmente homomórficos, con algunas excepciones [8], [44]. Dependiendo del caso de uso, los cifrados que no logran ser homomórficos totalmente podría ser útiles. Gentry, por ejemplo, indicó que mientras un cifrado parcialmente homomórfico podría realizar una operación de multiplicación en la dimensión 32768 (números enteros de 13.000.000 bits) en 0,6 segundos, la misma operación con un cifrado totalmente homomórfico conocido en ese momento en 2012 tardaría 30 minutos – más de 1.000 veces más lento (*Estos resultados eran obtenidos en una máquina con 1 CPU, Xeon E5440/2,83 GHz — 64-bit, quad-core — memoria 24GB) [9]. Gentry explica que la operación de multiplicación tarda porque después de cada operación de multiplicación, era necesario realizar una operación para cifrar de nuevo el texto cifrado, en su operación de *bootstrapping*.

En 2009, Gentry introdujo el proceso de *bootstrapping*, por medio del cual es posible reducir los errores introducidos a través de operaciones homomórficas; Gentry efectivamente descifra el text cifrado homomórficamente usando la clave cifrada, ergo produciendo un texto cifrado igual al texto anterior, pero reduciendo los errores generados a través de los operaciones homomórficas iniciales [7]. El proceso de bootstrapping de Gentry, si bien permitía las operaciones totalmente homomórficas, tardaba bastante en realizar. Aunque no es algo específico al procedimiento de bootstrapping, es importante mencionar que los cifrados homomórficos, cada vez que el cifrado realice una operación homomórfica, tiene que manipular todo el texto cifrado; si, dado una operación, el cifrado pudiera saber cual porción del texto cifrado tendría que utilizar para efectuar la operación, tendríamos un error de seguridad: implicaría que el cifrado supiera algo de nuestro texto cifrado, y por ende, que el cifrado no fuera totalmente seguro [9]. Entonces, el acceso aleatorio a los datos no está soportado con nuestros textos cifrados.

Cuando hablamos del tiempo necesario para realizar operaciones homomórficas, estamos haciendo una referencia indirecta a uno de los otro problemas que existe con muchos de los cifrados conocidos: el tamaño del texto cifrado. En el cifrado creado por Gentry, vemos que en la dimensión 32768, vemos que el tamaño de la clave pública es 2,25 GB [45]. Gentry indica que la clave pública es más grande por dos razones: primero, la necesidad de especificar una instancia del problema de la suma de subconjuntos dispersos (*Sparse-subset-sum problem)*, y la necesidad de incluir en la clave pública la codificación de todos los bits de la clave secreta [45]. Claramente, el tamaño de este cifrado representa una barrera importante para el uso de los cifrados homomórficos. Aunque Gentry describió algunos pasos que habían tomado para reducir el tamaño de la clave pública, sigue representando una barrera importante. Es más, Gentry evidenció que en su investigación, un servidor similar al mencionado anteriormente tardaría más de dos horas en generar la clave pública[45].

El tamaño de la clave pública representa solo una parte de las dificultades. Para poder operar contra 4MB de datos, tendríamos en ese momento que mandar alrededor de 73TB a la nube, usando uno de los cifrados modernos; aunque podemos reducir el tamaño del texto que necesitamos mandar a 280GB usando *batching*, veremos que esto todavía no representa una solución práctica en muchos casos [46]. Peor, es que en el cifrado mencionado, [47], la cantidad de ruido también crece exponencialmente con la profanidad de los circuitos siendo evaluados. Hay técnicas que existen para reducir el crecimiento del ruido, por ejemplo el modelo de los cifrados homomórficos nivelados (*leveled homomorphic encryption)*, pero muestra igual las dificultades que tienen muchos cifrados homomórficos [48].

En 2019, investigadores marcaron que todos los cifrados homomórficos conocidos carecían de seguridad IND-CCA [49].
La seguridad IND-CCA refiere a una situación en la cual el atacante tiene la habilidad de cifrar o descifrar mensajes arbitrarios con el cifrado elegido; después, usando lo que sabe de su acceso a los algoritmos de cifrar y descifrar, y el atacante tiene que adivinar si un texto cifrado coincide con cierto mensaje en texto plano[50].

Criptografía, como mucho trabajo en sistemas, requiere vigilancia constante en contra de las vulnerabilidades; por el contrario, los sistemas más nuevos no son excepciones. Microsoft SEAL (*Simple Encrypted Arithmetic Library*) es una tecnología *open-source* que provee un conjunto de bibliotecas que contienen cifrados homomórficos, ofreciendo a los desarrolladores la posibilidad de realizar cálculos en los datos cifrados [51]. La biblioteca de cifrados ha sido usado en varios proyectos, incluyendo el *Nueral Network Compiler nGraph* de Intel [49]. Específicamente, SEAL usa dos esquemas de cifrados, BVF y CKKS [49]. Peng demostró que con BVF, un esquema basado en el problema Ring-LWE de celosías ideales (*Ring-LWE problem of ideal lattice)*, se puede recuperar la clave secreta con un solo query. Fundamentalmente, el autor plantea que si un atacante logra ver los datos descifrados, puede comparar

esa información con los datos cifrados para romper el cifrado [49]. El investigador elabora que, incluso si pudiéramos prevenir que los atacantes tuvieran acceso a hacer queries para descifrar los mensajes, el investigador revela que incluso que filtraciones parciales todavía revelan información de la clave secreta; una filtración de 1 bit revela 1 bit de la clave secreta [49].

Vista hacia el futuro

Si bien Microsoft SEAL había recibido elogios por ser un avance en los cifrados homomórficos, no es la única opción. TFHE, *Fully Fast Homomorphic Encryption over the Torus*, fue publicado en abril, 2019. El cifrado muestra que, con algunas mejoras en el cifrado FHEW publicado anteriormente de Ducas and Micciancio, es posible reducir el tiempo de realizar *bootstrapping* de 690ms a 13ms, y el tamaño de la clave de boostrapping de 1GB a solo 16MB [44]. Esto representa un avance tremendo en el tamaño de la clave.

Similarmente, CONCRETE, un library *open-source* basado en TFHE, también muestra resultados impresionantes en la multiplicación de textos cifrados. Por ejemplo, multiplicación usando enteros de 64 bits, CONCRETE logra multiplicar datos cifrados en 127,8 milisegundos, en la dimensión 2048, con n=1024. Mientras tanto, Gentry había indicado con una dimensión similar, la operación de multiplicación tomó 31 segundos. Es importante mencionar también que los procesadores usados para probar los dos cifrados homomórficos eran distintos; en el caso de CONCRETE, usaron un procesador 2,6 GHz 6-Core Intel Core i7, y en el caso del cifrado homomórfico de Gentry, usaron el Xeon E5440 / 2,83 GHz (g2-bit, quad-core) mencionado anteriormente [9], [52]. A pesar de las diferencias en las mediciones, CONCRETE representa un avance importante en la posibilidad de usar cifrados homomórficos en cada vez más situaciones.

Hasta ahora, habíamos analizado mayormente código, pero los avances con los cifrados homomórficos también extienden más allá de los algoritmos. Intel anunció hace unos meses que empezaron a

colaborar con Microsoft en el programa DARPA (*Defense Advanced Research Projects Agency)* para generar un acelerador para usar con los cifrados totalmente homomórficos [6]. El programa tiene como objetivo crear un chip que usa un circuito integrado específico para esta aplicación ("ASIC," por sus siglas en inglés; creando infraestructura específica para este propósito tiene la potencial de agilizar dramáticamente las operaciones costosas que los cifrados homomórficos requieren. Cabe mencionar también que CONCRETE también logró mejorar sus resultados soportando la aceleración por GPU [53]. Esto demuestra que más allá de las optimizaciones que podemos hacer en los algoritmos, el hardware también podría ser una parte importante en el avance de los cifrados homomórficos.

CONCRETE no es el único library en su género; similar a CONCRETE, lattigo también usa un cifrado homomórfico basado en Ring-LWE[54]. Lo más notable del cifrado es que permite la comunicación segura entre múltiples partes ("SMC," por sus siglas en inglés). El objetivo de SMC es permitir que varias computadoras pueden colaborar en la computación de alguna función de manera segura [55]. Este tipo de comunicación nos permite revelar solo los datos relevantes a nuestro objetivo; por ejemplo, si quisiéramos permitir que un usuario pudiera comprar su ADN contra una base de datos de ADN, sin relevar el contenido de su ADN, SMC nos permitiera hacer el cálculo, y solo relevar al final el tipo de cáncer u otra enfermedad que fuese reconocido en el ADN del paciente [55]. Aunque lattigo tiene un propósito un poco distinto que CONCRETE, es otro ejemplo claro de que las mejoras en el rendimiento de los cifrados homomórficos significa que ahora los cifrados homomórficos representan soluciones prácticas [54].

Durante la última década, IBM también hecho contribuciones al desarrollo de los cifrados homomórficos también. Publicaron un juego de herramientas que también tiene el objetivo de fomentar el uso de los cifrados totalmente homomórficos en distintos proyectos; actualmente, tienen herramientas disponibles para iOS, MacOS, Linux, e indican que brevemente también tendrán otra opción para

Android [56]. Las herramientas están basadas en su producto, HElib, que uno de los libraries de cifrados homomórficos más antiguos, que fue originalmente publicado en 2013 [56], [57]. Originalmente, HElib implementó el cifrado Brakerski-Gentry-Vaikuntanathan (BGV), que era uno de los cifrados que no requería el procedimiento de Bootstrapping que Gentry había introducido con el cifrado homomórfico descrito en su tesis en 2009 [48], [57]. Siguiendo los avances de todos los cifrados homomórficos conocidos en los últimos años, investigadores de IBM indican que habían logrado aumentar la velocidad de los cálculos más de 100x [58]. Con un enfoque a la capacitación, el juego de herramientas también incluye una aplicación que sirve como una demostración, y permite que los usuarios puedan realizar queries en una base de datos con cifrados homomórficos; indican que en un MacBook Pro, el query tarda 80 segundos en ejecutarse [58].

Más allá de los cifrados homomórficos desarrollado por investigadores, vemos que también varias empresas habían logrado usar cifrados homomórficos en producción. Uno de los primeros ejemplos sucedió en el año 2013. Fujitsu Laboratories Ltd. utilizó cifrados homomórficos para realizar cálculos estadísticos y autenticación biométrica en los datos cifrados, y con tiempos de ejecución en un tiempo razonable [59]. Esto fue aproximadamente cinco años antes de que Microsoft anunciara que iba a publicar su library SEAL en GitHub, bajo una licencia open source [60].

Además, en los últimos años estamos viendo empresas dedicadas a ofrecer servicios que usan los cifrados homomórficos. Duality Technolgoies también empezó en 2016, y fue fundado por criptógrafos reconocido mundialmente [61]. Actualmente, tiene en su equipo Prof. Shafi Goldwasser, inventora del cifrado Goldewasser-Micali analizado previamente en este trabajo, y Prof. Vinod Vaikuntanathan, que colaboró con Craig Gentry para crear el cifrado homomórfico nivelado (leveled), también mencionado previamente en este trabajo [62]. Duality Technologies ofrece servicios que permiten

a organizaciones en industrias reguladas aprovechar de los datos en todo su ecosistema [63].

Por otro lado, fundado en 2016, Enveil usa los cifrados homomórficos para generar nuevas oportunidades en negocios [64]. En un caso, por ejemplo, la empresa quiere utilizar los cifrados homomórficos para permitir la monetización seguro y ético de los datos en una empresa [65]. La empresa también se dedica a la colaboración segura entre empresas; esto permitiría que empresas grandes, por ejemplo dos o más bancos, pudieran consultar los datos de otros bancos sin comprometer la privacidad de los usuarios [65]. Por ejemplo, si un banco recibía una aplicación para un préstamo, el banco podría contactar a otros bancos para preguntarles si ese cliente había participado alguna vez en una estafa financiera. Los otros bancos, entonces, serían capaz de buscar en sus bases de datos para ver si los datos del cliente (cifrados cuando los recibió) estaban guardados en la base de datos del banco; después, el banco devolvería el resultado (otro mensaje cifrado) al banco principal. Al final, los bancos estarían colaborando, pero los datos confidenciales de los clientes nunca estarían expuestos [66]. Un área clave también de la empresa tiene que ver con el cumplimiento de las regulaciones; si una empresa puede interactuar con sus propios datos personales de sus clientes sin descifrarlos, sería más fácil cumplir con las regulaciones vigentes, por ejemplo el GDPR que empezó a regir en 2018 [65].

Conclusión

La fundadora de Enveil, Ellison Anne Williams, publicó el año pasado en Twitter: “Los cifrados homomórficos no se trata de mejorar algo; se trata de crear algo totalmente nuevo. #homomorphicencryption” [67]. Cuando imaginamos el futuro de criptografía, con los avances recién demostrados con los cifrados homomórficos, es difícil imaginar los límites de lo que podría ser posible. Dos bancos multinacionales, que ahora son contrincantes, con la garantía de que sus datos no dejarían de ser cifrados, podrían

llegar a trabajar en conjunto para mejorar la seguridad de todos. Un hospital, que tiene almacenado todos los datos de sus pacientes, podría colaborar con investigadores científicos para apoyar a la lucha contra las enfermedades más graves, como COVID-19 o cáncer, dejando que los investigadores analicen los datos del hospital en tiempo real, pero de forma anónima y segura. Usuarios podrían realizar búsquedas en internet sin temer que los proveedores se estarían enterado de lo que están buscando.

Las oportunidades creadas con esta nueva promesa de privacidad son muchas. Pero este nuevo avance nos trae también dilemas éticos y morales. El Tor Browser, por ejemplo, con su red entrelazada de conexiones VPN permite que los usuarios puedan navegar el internet de forma anónima [68]. Pero con el tiempo, vimos que el Tor Browser fue usado tanto por periodistas luchando por la verdad como por usuarios buscando pornografía infantil. Si bien los cifrados homomórficos podrían posibilitar el acceso a datos confidenciales de hospitales o bancos importantes sin revelar los datos confidenciales, aparece otra dilema: la posibilidad de sacar información sobre la “meta información” en los queries. Por ejemplo, existen vulnerabilidades importantes en criptografía, como el ataque POODLE, que demuestran que sin disponer del texto plano, uno puede deducir mucha información – en el caso del ataque POODLE, uno puede hacer pruebas, alterando apenas los datos cifrados, y observar los resultados para poder al final deducir hasta contraseñas, cookies, y todo tipo de información confidencial [69]. Podemos imaginar, entonces, que sin disponer de los datos en texto plano, que todavía hay la posibilidad de deducir información importante.

Al final, aunque lo que estemos analizando acá queda a la vanguardia de lo más nuevo, no dejemos de pensar que con la invención de lo nuevo, casi siempre después vienen nuevas variaciones de los mismos riesgos de siempre.

Bibliografía

[1] M. C. Baca, “What you do on the Internet is worth a lot . Exactly how much , nobody knows .,” Washington, DC, Oct. 14, 2019.

[2] R. L. Rivest, L. Adleman, and M. L. Dertouzos, “On Data Banks and Privacy Homomorphisms,” Cambridge, 1978.

[3] F. Armknecht *et al.*, “A Guide to Fully Homomorphic Encryption,” *Cryptol. ePrint Arch.*, pp. 1–35, 2015, [Online]. Available: https://eprint.iacr.org/2015/1192.

[4] N. P. Smart and F. Vercauteren, “Fully homomorphic encryption with relatively small key and ciphertext sizes,” *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 6056 LNCS, pp. 420–443, 2010, doi: 10.1007/978-3-642-13013-7_25.

[5] A. Wainakh, “Homomorphic Encryption for Data Security in Cloud Computing,” *Middle East Tech. Univ.*, no. June, p. 121, 2018.

[6] “Intel to Collaborate with Microsoft on DARPA Program,” *Intel*, Mar. 08, 2021. https://newsroom.intel.com/news/intel-collaborate-microsoft-darpa-program/#gs.xz344r.

[7] C. Gentry, “A Fully Homomorphic Encryption Scheme,” *Dissertation*, no. September, p. 189, 2009, [Online]. Available: http://cs.au.dk/~stm/local-cache/gentry-thesis.pdf.

[8] A. Acar, H. Aksu, A. S. Uluagac, and M. Conti, “A survey on homomorphic encryption schemes: Theory and implementation,” *arXiv*, pp. 1–35, 2017.

[9] C. Gentry, “Winter School on Cryptography: Fully Homomorphic Encryption - Craig Gentry,” *Bar-Ilan Univ.*, 2012, [Online]. Available: https://www.youtube.com/watch?v=Y1TxCiOuoYY.

[10] A. Arampatzsis, “Homomorphic Encryption : What Is It and How Is It Used,” *Venafi*, 2020. https://www.venafi.com/blog/homomorphic-encryption-what-it-and-how-it-used#:~:text=A homomorphic cryptosystem is like,to access its unencrypted data.

[11] X. Yi, R. Paulet, and E. Bertino, “Homomorphic encryption,” *SpringerBriefs Comput. Sci.*, vol. 0, no. 9783319122281, pp. 27–46, 2014, doi: 10.1007/978-3-319-12229-8_2.

[12] R. L. Rivest, A. Shamir, and L. Adleman, “A method for obtaining digital signatures and public-key cryptosystems,” *Commun. ACM*, vol. 21, no. 2, pp. 120–126, Feb. 1978, doi: 10.1145/359340.359342.

[13] D. Boneh, “Twenty years of attacks on the RSA cryptosystem,” *Not. AMS*, vol. 46, no. 2, pp. 203–213, 1999.

[14] D. Boneh, A. Joux, and P. Q. Nguyen, “Why textbook elgamal and RSA encryption are insecure: (Extended abstract),” *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 1976, pp. 30–43, 2000, doi: 10.1007/3-540-44448-3_3.

[15] B. Buchanan, “Homomorphic Encryption For Division With RSA,” *Medium*, 2018. https://medium.com/asecuritysite-when-bob-met-alice/homomorphic-encryption-for-division-with-rsa-2f1e97058f91 (accessed Mar. 13, 2021).

[16] D. Chandravathi and P. V. Lakshmi, “A New Hybrid Homomorphic Encryption Scheme for Cloud Data Security,” *Adv. Comput. Sci. Technol.*, vol. 10, no. 5, pp. 825–837, 2017, [Online]. Available: https://www.ripublication.com/acst17/acstv10n5_16.pdf.

[17] R. Shruthi, P. Sumana, and A. K. Koundinya, “Performance Analysis of Goldwasser-Micali Cryptosystem,” *Int. J. Adv. Res. Comput. Commun. Eng.*, vol. 2, no. 7, pp. 2818–2822, 2013,

[Online]. Available: http://pages.cs.wisc.edu/~shruthir/Documents/PerformanceAnalysisOfGoldwasserMicaliCryptosystem.pdf.

[18] C. Pleşca, M. Togan, and C. Lupaşcu, “Homomorphic Encryption Based on Group Algebras and Goldwasser-Micali Scheme,” 2016. doi: 10.1007/978-3-319-47238-6_11.

[19] A. Martínez and M. Marfia, “Una breve introducción a la criptografía matemática,” 2015. [Online]. Available: http://www.mate.unlp.edu.ar/~demetrio/Monografias/Materias/EA/33. Criptografia - Marfia & Martinez Lopez.pdf.

[20] R. Darrel and A. Scott, *Guide to Elliptic Curve Cryptography*. 2004.

[21] T. ElGamal, “A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms,” in *Advances in Cryptology*, Berlin, Heidelberg: Springer Berlin Heidelberg, 1976, pp. 10–18.

[22] School of Mathematical and Statistical Sciences, “ElGamal Cryptosystem,” *Arizona State Univ.*, [Online]. Available: https://math.asu.edu/sites/default/files/elgamal.pdf.

[23] V. S. Miller, “Use of Elliptic Curves in Cryptography,” *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 218 LNCS, pp. 417–426, 1986, doi: 10.1007/3-540-39799-X_31.

[24] N. Koblitz, “Elliptic curve cryptosystems,” *Math. Comput.*, vol. 48, no. 177, pp. 203–203, Jan. 1987, doi: 10.1090/S0025-5718-1987-0866109-5.

[25] “Criptografía con curvas elípticas,” *Universidad Politécnica de Madrid*. http://www.criptored.upm.es/crypt4you/temas/ECC/leccion1/leccion1.html#04 (accessed Mar. 20, 2021).

[26] R. Ranasinghe, “A generalized elliptic curve elgamal cryptosystem,” *Univ. Perad.*, no. November, 2020.

[27] L. Fousse, P. Lafourcade, and M. Alnuaimi, “Benaloh’s Dense Probabilistic Encryption Revisited,” in *Proc. Work. Sel. areas {…}*, 2011, pp. 348–362.

[28] Y. Zheng, T. Matsumoto, and H. Imai, “Residuosity problem and its application in cryptography,” *Ieice Trans.*, vol. E71, no. 8, pp. 759–767, 1988, [Online]. Available: http://search.ieice.org/bin/summary.php?id=e71-e_8_759&category=E&year=1988&lang=E&abst=.

[29] M. A. Budiman and D. Rachmawati, “A tutorial on using Benaloh public key cryptosystem to encrypt text,” *J. Phys. Conf. Ser.*, vol. 1542, no. 1, 2020, doi: 10.1088/1742-6596/1542/1/012039.

[30] D. Naccache and J. Stern, “A new public key cryptosystem based on higher residues,” in *Proceedings of the 5th ACM conference on Computer and communications security - CCS ’98*, 1998, vol. 53, no. 9, pp. 59–66, doi: 10.1145/288090.288106.

[31] É. Brier, R. Géraud, and D. Naccache, “Exploring Naccache-Stern Knapsack Encryption,” *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 10543 LNCS, pp. 67–82, 2017, doi: 10.1007/978-3-319-69284-5_6.

[32] L. Haryanto, A. Lawi, S. Suhastina, and P. Qarynah, “On the Okamoto-Uchiyama cryptosystem: (A brief essay on basic mathematics applied in cryptography),” *J. Phys. Conf. Ser.*, vol. 1341, no. 4, 2019, doi: 10.1088/1742-6596/1341/4/042013.

[33] S. D. Galbraith, “Elliptic Curve Paillier Schemes,” *J. Cryptol.*, vol. 15, no. 2, pp. 129–138, 2002, doi: 10.1007/s00145-001-0015-6.

[34] T. Okamoto and S. Uchiyama, “A new public-key cryptosystem as secure as factoring,” 1998, pp. 308–318.

[35] J. L. Gómez Pardo, “Cifrado homomórfico: ejemplos y aplicaciones,” *La Gac. la RSME*, vol. 15, pp. 697–711, 2012, [Online]. Available: https://gaceta.rsme.es/abrir.php?id=1111.

[36] P. Paillier, “Public-Key Cryptosystems Based on Composite Degree Residuosity Classes,” in *Advances in Cryptology — EUROCRYPT ’99*, vol. 1592, Berlin, Heidelberg: Springer Berlin Heidelberg, 1999, pp. 223–238.

[37] I. Damgård and M. Jurik, “A Generalisation, a Simplification and Some Applications of Paillier’s Probabilistic Public-Key System,” in *Proceedings of the 4th International Workshop on Practice and Theory in Public Key Cryptography: Public Key Cryptography*, no. December, 2001, pp. 119–136.

[38] A. Kawachi, K. Tanaka, and K. Xagawa, “Multi-bit cryptosystems based on lattice problems,” *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 4450 LNCS, pp. 315–329, 2007, doi: 10.1007/978-3-540-71677-8_21.

[39] M. Caboara, F. Caruso, and C. Traverso, “Lattice Polly Cracker cryptosystems,” *J. Symb. Comput.*, vol. 46, no. 5, pp. 534–549, May 2011, doi: 10.1016/j.jsc.2010.10.004.

[40] F. Levy-dit-Vehel, M. G. Marinari, L. Perret, and C. Traverso, “A Survey on Polly Cracker Systems,” in *Gröbner Bases, Coding, and Cryptography*, Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 285–305.

[41] B. Barkee, D. C. Can, J. Ecks, T. Moriarty, and R. F. Ree, “Why You Cannot Even Hope to Use Gröbner Bases in Public-Key Cryptography ?,” *J. Symb. Comput.*, vol. 18, no. 6, pp. 497--501, 1994.

[42] M. Fellows and N. Koblitz, “Combinatorial cryptosystems

galore!," no. February, pp. 51–61, 1994, doi: 10.1090/conm/168/01688.

[43] M. Chenal and Q. Tang, "On key recovery attacks against existing somewhat homomorphic encryption schemes," *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 8895, pp. 239–258, 2015, doi: 10.1007/978-3-319-16295-9_13.

[44] I. Chillotti, N. Gama, M. Georgieva, and M. Izabachène, "TFHE: Fast Fully Homomorphic Encryption Over the Torus," *J. Cryptol.*, vol. 33, no. 1, pp. 34–91, 2020, doi: 10.1007/s00145-019-09319-x.

[45] C. Gentry, "Implementing Gentry ' s Fully-Homomorphic Encryption Scheme Preliminary Report," *Organization*, pp. 1–30, 2010, [Online]. Available: http://eprint.iacr.org/2010/520.

[46] T. Lepoint and M. Naehrig, "A comparison of the homomorphic encryption schemes FV and YASHE," *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 8469 LNCS, pp. 318–335, 2014, doi: 10.1007/978-3-319-06734-6_20.

[47] J. S. Coron, D. Naccache, and M. Tibouchi, "Public key compression and modulus switching for fully homomorphic encryption over the integers," *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 7237 LNCS, pp. 446–464, 2012, doi: 10.1007/978-3-642-29011-4_27.

[48] Z. Brakerski, C. Gentry, and V. Vaikuntanathan, "(Leveled) fully homomorphic encryption without bootstrapping," *ITCS 2012 - Innov. Theor. Comput. Sci. Conf.*, vol. 2021, pp. 309–325, 2012, doi: 10.1145/2090236.2090262.

[49] Z. Peng, "Danger of using fully homomorphic encryption: A look at Microsoft SEAL," *arXiv*, pp. 1–14, 2019.

[50] S. A. Figueroa, “Easy explanation of ‘ IND- ’ security notions ?,” *Stack Overflow*, 2015. https://crypto.stackexchange.com/questions/26689/easy-explanation-of-ind-security-notions.

[51] Microsoft, “Microsoft SEAL,” *Microsoft*. https://www.microsoft.com/en-us/research/project/microsoft-seal/.

[52] I. Chillotti, M. Joye, D. Ligier, J.-B. Orfila, and S. Tap, “CONCRETE : Concrete Operates oN Ciphertexts Rapidly by Extending TfhE,” Zama. [Online]. Available: https://whitepaper.zama.ai/concrete/WAHC2020Demo.pdf.

[53] “Build fast homomorphic programs easily,” *ZAMA, Inc.*, 2021. https://zama.ai/concrete/.

[54] J.-P. Bossuat, C. Mouchet, J. Troncoso-Pastoriza, and J.-P. Hubaux, “The Lattigo lattice-based cryptographic library,” *Homomorphic Encryption*, 2019. https://homomorphicencryption.org/wp-content/uploads/2019/08/poster_2.pdf.

[55] Y. Lindell, “Secure multiparty computation,” *Commun. ACM*, vol. 64, no. 1, pp. 86–98, 2021, doi: 10.1145/3387108.

[56] I. Research, “Never decrypt your data , even during computation.” https://fhe-website.mybluemix.net/#toolskits.

[57] P. Ducklin, “IBM takes a big new step in cryptography: practical homomorphic encryption,” *Naked Security by Sophos*, 2013. https://nakedsecurity.sophos.com/2013/05/05/ibm-takes-big-new-step-in-cryptography/.

[58] S. De Simone, “IBM Fully Homomorphic Encryption Toolkit Now Available for MacOS and iOS,” 2020. https://www.infoq.com/news/2020/06/ibm-fully-homomorphic-encryption/.

[59] Fujitsu Limited, “Fujitsu Develops World’s First Homomorphic Encryption Technology that Enables Statistical Calculations and Biometric Authentication,” *Fujitsu Laboratories Ltd.*, 2013. https://www.fujitsu.com/global/about/resources/news/press-releases/2013/0828-01.html.

[60] K. Laine and S. Kannepalli, “The Microsoft Simple Encrypted Arithmetic Library goes open source,” *Microsoft Research Blog*, 2018. https://www.microsoft.com/en-us/research/blog/the-microsoft-simple-encrypted-arithmetic-library-goes-open-source/.

[61] Crunchbase, “Duality Technologies.” https://www.crunchbase.com/organization/duality.

[62] “About Us,” *Duality Tech*. https://dualitytech.com/about-us/.

[63] “Our Solutions,” *Duality Technologies*. https://dualitytech.com/duality-solutions-overview/.

[64] Crunchbase, “Enveil.” https://www.crunchbase.com/organization/enveil-inc.

[65] Enveil, “Use cases.” https://www.enveil.com/use-cases.

[66] E. A. Williams, “Unlocking The Power Of Secure Data Collaboration,” *Forbes*, 2019. https://www.forbes.com/sites/forbestechcouncil/2019/12/19/unlocking-the-power-of-secure-data-collaboration/?sh=6905ae403030.

[67] E. A. Williams, “HE is not making something better; it’s making something entirely new possible. #homomorphicencryption,” *Twitter*, 2020. https://twitter.com/ellisonanne5/status/1316510071052959751.

[68] I. The Tor Project, “About History.” https://www.torproject.org/about/history/.

[69] T. A. Nidecki, “What Is the POODLE Attack?,” *Acunetix*, 2020.

https://www.acunetix.com/blog/web-security-zone/what-is-poodle-attack/.