Universidad Autónoma de Querétaro
Facultad de Ingeniería
Maestría en Ciencias (Ingeniería Matemática)

Criptografía con curvas elípticas aplicada a la firma digital

Tesis

Que como parte de los requisitos para obtener el Grado de
Maestro en Ciencias en Ingeniería Matemática

Presenta
Diego Abraham Olvera Mendoza

Dirigido por:
M. en C. Fidel González Gutiérrez

Co-dirigido por:
Dr. Arturo González Gutiérrez

M. en C. Fidel González Gutiérrez
Presidente

Dr. Arturo González Gutiérrez
Secretario

M. en D. M. Carmen Sosa Garza
Vocal

Dr. Samuel Estala Arias
Suplente

M. en C. Guillermo Díaz Delgado
Suplente

Centro Universitario, Querétaro Qro.
Fecha de aprobación por el Consejo Universitario (mes y año)
México

A mi Señor y Salvador Jesús
¡Toda la gloria a Él!

Agradecimientos

Mi agradecimiento al M. en C. Fidel González Gutiérrez por su exigencia, recomendación, apoyo y aliento para terminar esta investigación. Quién habría imaginado nos volveríamos a encontrar cuatro años después y me dirigiría para titularme como maestro.

Agradezco a mi codirector de tesis el Dr. Arturo González Gutiérrez por su acertada orientación para realizar este trabajo con el maestro Fidel. Son pocos los profesores en la universidad especializados en temas de criptografía y dar con él fue gracias a usted.

También quisiera agradecr a los miembros de mi sínodo por el tiempo destinado a la revisión de mi tesis, así como sus invaluables comentarios para mejorar el presente trabajo

La financiación para la realización de este estudio ha sido gracias al Consejo Nacional de Ciencia y Tecnología. Sin su apoyo económico esto no habría sido posible. También a la Universidad Autónoma de Querétaro por absorber los gastos de la matrícula durante estos años.

Nunca podré agradecer lo suficiente a mis padres, José Luis Olvera Moya y Virginia Alfonsina Mendoza Saldaña. Su cuidado, enseñanzas y amor siempre me alentó para finalizar mis estudios de posgrado ¡Este logro también es de ustedes!

Dudo haber tenido mejor compañía que la de mis compañeros Iliana Paternina y Francisco Ortiz. Gracias por su amistad y consejos a lo largo de este camino para convertirnos en maestros en ciencias.

A todos, de corazón ¡Gracias!

Índice general

Índice de figuras

Índice de tablas

Glosario

ancho de banda Tasa de datos máxima de un canal de telecomunicaciones. Esta cantidad se mide en bits/segundo

criptografía postcuántica Técnicas para proteger computadores clásicos de criptoanálisis realizados con una computadora cuántica

Mathematica Sistema de computación técnica basado en Wolfram Language. Desarrollado por Stephen Wolfram en 1998. Mathematica® es una marca registrada de Wolfram Research Inc

Raspberry Pi Computadora de una sola placa del tamaño de una tarjeta de crédito. Se caracteriza por tener un precio accesible. Raspberry Pi® es una marca registrada de la Raspberry Pi Foundation

tasa de transmisión Cantidad de bits que se transmiten en un segundo por un canal. Es la velocidad de transmisión real de la red. También conocido como rendimiento de red. Está limitado por el ancho de banda

Siglas

AMEXIPAC	Asociación Mexicana de Proveedores Autorizados de Certificación
ANSI	American National Standards Institute (Español: Instituto Nacional de Normas de EE. UU.)
CFD	Comprobante Fiscal Digital
CFDI	Comprobante Fiscal Digital por Internet
CRT	Chinese Remainder Theorem (Español: Teorema chino del residuo)
CSD	Certificado de Sello Digital
DSA	Digital Signature Algorithm (Español: Algoritmo de firma digital)
DSS	Digital Signature Standard (Español: Estándar de firma digital)
ECC	Elliptic Curve Cryptography (Español: Criptografía de curva elíptica)
ECDLP	Elliptic Curve Discrete Logarithm Problem (Español: Problema del logaritmo discreto en curva elíptica)
ECDSA	Elliptic Curve Digital Signature Algorithm (Español: Algoritmo de firma digital con curva elíptica)
ERS	Evidence Record Syntax (Español: Sintaxis del Registro de Evidencias)
GNFS	General Number Field Sieve (Español: Algoritmo de la criba general de campo numérico)
IEEE	Institute of Electrical and Electronics Engineers (Español: Instituto de Ingenieros Eléctricos y Electrónicos)
IoT	Internet of things (Español: Internet de las cosas)
IP	Internet Protocol (Español: Protocolo de Internet)

ISO	International Organization for Standardization (Español: Organización Internacional de Normalización)
LTAP	Long-term Archive Protocol (Español: Protocolo para archivar a largo plazo)
MD5	Message-Digest Algorithm 5 (Español: Algoritmo de Resumen del Mensaje 5)
NIST	National Institute of Standards and Technology (Español: Instituto Nacional de Estándares y Tecnología de EE. UU.)
PAC	Proveedor Autorizado de Certificación
PKCS	Public-Key Cryptography Standards (Español: Normas para criptografía de clave pública)
PKI	Public Key Infrastructure (Español: Infaestructura de clave pública)
RFC	Registro Federal de Contribuyente
RSA	Criptosistema de Rivest, Shamir y Adleman
SAT	Servicio de Administración Tributaria
SE	Secretaría de Economía
SEA	Algoritmo de Schoof-Elkies-Atkin
SEC	Standards for Efficient Cryptogrpahy (Español: Normas para criptografía eficiente)
SFP	Secretaría de la Función Pública
SHA	Secure Hash Algorithm (Español: Algoritmo de digestión seguro)
SHA-1	Secure Hash Algorithm 1 (Español: Algoritmo de digestión seguro 1)
SI	Sistema Internacional de Unidades
TI	Tecnologías de la Información
XML	eXtensible Markup Language (Español: Lenguaje de marcado extensible)
XSD	XML Schema Definition (Español: Definición de esquema XML)

Símbolos matemáticos

$\mathcal{E}$ Proceso de cifrado en un criptosistema de clave pública

h Cofactor de una curva elíptica

$\mathbb{C}$ Números complejos

E Curva elíptica

$\mathcal{D}$ Proceso de descifrado en un criptosistema de clave pública

$\mathbb{Z}^+$ Números enteros igual o mayores que uno

ϕ La función fi de Euler

F_p Campo finito con p elementos

s Firma digital

$\mathcal{H}$ Función de digestión o *hash*

M Un mensaje, representado en base 2

$\#E$ Número de puntos de una curva elíptica

π Función contador de números primos

p Número primo

$\mathcal{O}$ Punto al infinito

$\mathbb{Q}$ Números racionales

$\mathbb{R}$ Números reales

t Traza de una curva elíptica

Resumen

El comprobante fiscal digital por internet (CFDI) da confianza a los involucrados en transacciones comerciales y permite a la Servicio de Administración Tributaria (SAT) automatizar los procesos de contabilidad y recaudación de impuestos. Para garantizar la autenticidad, integridad y evitar el repudio de los CFDI, se utiliza un sello digital basado en el algoritmo de firma digital RSA. Ante el aumento de emisiones de CFDI, las firmas RSA ocupan grandes espacios de almacenamiento, tiempos de procesamiento y ancho de banda. En este trabajo, se presenta la propuesta de un sello digital basado en el algoritmo de firma digital con curva elíptica (ECDSA). El prototipo se desarrolló en el sistema de computación técnica Mathematica® con la curva elíptica *secp256k1*. Los nuevos sellos resultan más seguros, utilizan menor espacio de almacenamiento, procesamiento y ancho de banda. Se generaron en una laptop Dell Inspiron 7537 con procesador Intel i7-4500U a 1.8 GHz, 8 GB de memoria RAM y sistema operativo Windows 10 Home de 64 bits. La medición del uso de ancho de banda se realizó entre dos Raspberrys Pi conectados por un cable Ethernet categoría seis. Contrario a lo expuesto en la literatura, la implementación en Mathematica® del algoritmo de verificación de firma digital con curva elíptica resultó más rápido que la verificación del algoritmo RSA. A pesar de sus beneficios, nuestra propuesta es vulnerable a computadores cuánticos capaces de ejecutar el algoritmo de factorización polinomial de Shor. Como alternativa, sugerimos el problema de isogenia entre curvas elípticas para generar sellos seguros a largo plazo.

Palabras clave: Comprobante fiscal digital por Internet (CFDI), sello digital, criptografía de curva elíptica (ECC), algoritmo de firma digital con curva elíptica (ECDSA).

Abstract

In Mexico, the *comprobante fiscal digital por Internet* (CFDI) is a digital invoice that gives confidence to the parties involved in a comercial transactions. It allows the *Secretaría de Administración Tributaria* (SAT) to automate the accounting process and tax collection. To guarantee the authenticity, integrity and avoid disclaim when a CFDI is generated a stamp based on the RSA signature algorithm is used. In this work, we propose a stamp based on the Elliptic Curve Digital Signature Algorithm (ECDSA). The prototype was developed in Mathematica® using the elliptic curve *secp256k1*. The new stamp is more secure, requieres less computing time, storage space and bandwidth. The stamp was generated on a Dell Inspiron 7537 laptop with a 1.8 GHz Intel i7-4500U processor, 8 GB of RAM and Windows 10 Home edition (64-bit). Bandwidth usage was measured between two Raspberrys Pi 4 model B connected by a category 6 Ethernet cable. Despite their benefits, we discuss the imminent arrival of quantum computers that could execute Shor's polynomial time factoring algorithm. As an alternative, we suggest the problem of finding isogeny between elliptic curves to generate long-term secure stamps for the CFDI.

Keywords: Comprobante fiscal digital por Internet, digital stamp, elliptic curve cryptography (ECC), Elliptic Curve Digital Signature Algorithm (ECDSA).

Prefacio

Mi principal interés personal para elegir este tema fue el misticismo que envuelve la criptografía y los años de mi adolescencia ayudando a mi padre en su despacho contable. Sorprendentemente estás dos áreas se unen en el sello del comprobante fiscal. Para ser precisos, en la firma digital que ostentan como *sello* las facturas. Aunque de reciente implementación, poca innovación a la forma de proteger los comprobantes fiscales se ha realizado en nuestro país. Este prototipo trata de ser un primer paso en la evolución de la firma digital en México.

El presente trabajo se ha organizado de la siguiente manera. El capítulo uno contiene de forma general los conceptos de criptografía de clave pública empleados en el sello de CFDI, el problema con los recursos computacionales del sello actual y una alternativa con criptografía de curva elíptica. El capítulo dos presenta la evolución del comprobante fiscal en México y los avances en ECC para su puesta en marcha. Es una revisión de la literatura más relevante para esta investigación. En el capítulo tres se expresa la posibilidad de disminuir los recursos computacionales con un sello basado en curva elíptica. El capítulo cuatro plantea como objetivo desarrollar un prototipo de sello digital con el ECDSA. El capítulo cinco contiene los fundamentos matemáticos que soportan el uso de criptografía de curva elíptica en sellos digitales y su seguridad. El capítulo seis presenta el método para generar, verificar y transmitir nuestra propuesta de sello digital. El capítulo siete presenta los datos obtenidos en nuestras pruebas con tablas y gráficas. Así, se facilita compara el prototipo con los sellos RSA. El capítulo ocho contiene las conclusiones a las que se llegó durante el desarrollo de este proyecto. Finalmente, en el capítulo nueve se discute sobre computación cuántica y se recomienda el problema de isogenia entre curvas elípticas para generar sellos resistentes a futuros ataques cuánticos.

Capítulo 1

Introducción

En México, una *factura* o *comprobante fiscal digital por Internet* (CFDI) es auténtico si el emisor lo *sella* con su *certificado de sello digital* (CSD). En realidad el sello es una *firma digital* generada a partir de un *esquema de firma digital*. Un esquema de firma digital se basa en un *criptosistema asimétrico* o *de clave pública*. De todos los criptosistemas de clave pública solo dos se emplean por su eficiencia y seguridad: el criptosistema de Rivest-Shamir-Adleman (RSA) y el criptosistema con curva elíptica (ECC). Razón por la cual los sellos actuales de los comprobantes fiscales se generan con el algoritmo de firma RSA. Pero el alto volumen de emisiones de CFDI y el tamaño de sus sellos demandan considerables recursos computacionales.

La *criptografía de clave pública* o *criptografía asimétrica* fue propuesta por Diffie y Hellman en 1976 como una solución a los problemas de confidencialidad y autentificación en canales inseguros de comunicaciones (Diffie y Hellman, 1976). Los autores proponían el uso de una *clave pública* para cifrar mensajes y una *clave privada* para descifrarlos. La clave pública se colocaba en un directorio de libre acceso mientras que la clave privada no se compartía con nadie. Ambas claves están relacionadas matemáticamente. Sin embargo, la singularidad de este tipo de criptosistema es que la clave privada no puede obtenerse a partir de la clave pública. Más importante aún es la capacidad de producir un análogo a la firma autógrafa para realizar transacciones comerciales por Internet.

El *esquema de firma digital* puede verse en la Figura 1.1, donde Ana envía un documento M firmado digitalmente a Juan. El documento M es público, por lo tanto solo se busca garantizar que Ana lo firmó. A través de una *función de digestión* o *hash* $\mathcal{H}$ se mapea el documento M, representado como una cadena de bits de longitud variable, a una cadena de longitud fija. Se denotará el proceso de cifrado utilizando la clave pública como $\mathcal{E}$ y el descifrado utilizando la clave privada como $\mathcal{D}$. La *firma digital* de un documento M se obtiene al *descifrar* con la clave privada de Ana el resumen del documento $\mathcal{H}(M)$; es decir, la firma $s = \mathcal{D}(\mathcal{H}(M))$. Juan recibe una copia del mensaje M junto con la firma por medio de un canal inseguro de comunicación. Para verificarla, Juan *cifra* la

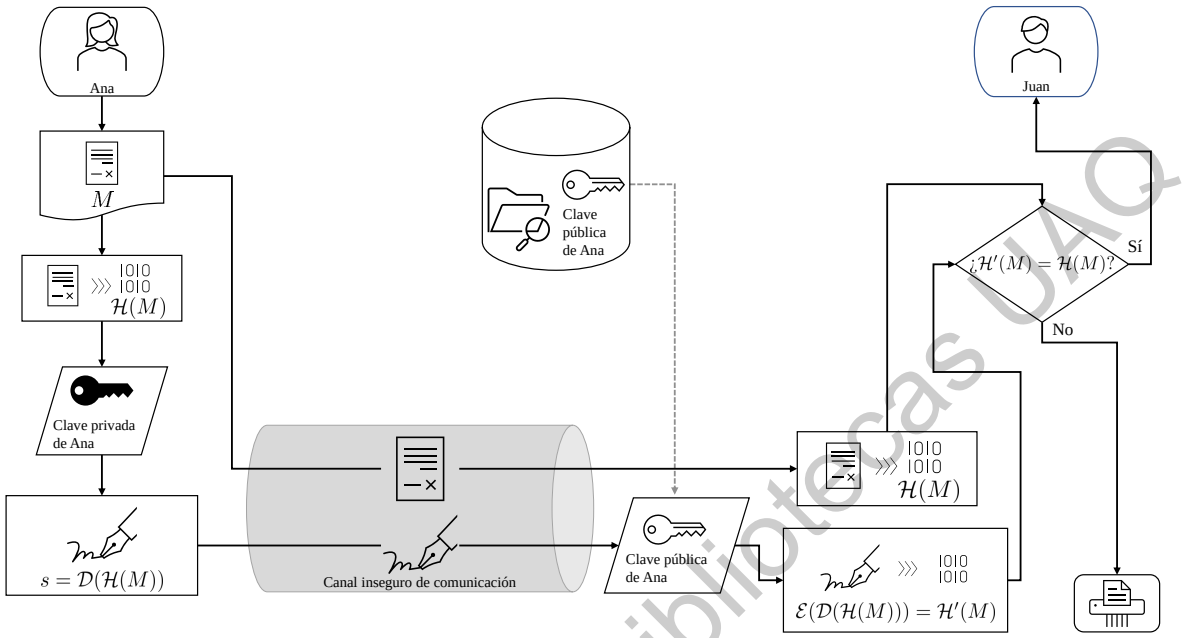

Figura 1.1: Esquema de firma digital

firma con la clave pública de Ana obteniendo $\mathcal{H}'(M) = \mathcal{E}(\mathcal{D}(\mathcal{H}(M)))$. Posteriormente, con el mensaje recibido se obtiene el resumen $\mathcal{H}(M)$. Si ambos resúmenes son iguales, es decir $\mathcal{H}'(M) = \mathcal{H}(M)$, la firma en el documento es valida. En caso de no coincidir, puede ser que el mensaje se alteró o el firmante no es quien afirma ser.

El creciente acceso a Internet sumado con el fenómeno informático de las *firmas digitales* motivaron el cambio de muchos documentos a un formato digital, entre ellos la *factura* o *comprobante fiscal*. Para que este documento sea válido es necesario que sea *timbrado* o *sellado*. Como se puede observar en la Figura 1.2 el sello es una serie de caracteres incomprensibles. Parecería más un secreto que no quiere revelarse a nadie y si se dispusiera del tiempo suficiente se tardaría miles de años en falsificarlo. Aunque un mismo usuario genere múltiples sellos con el mismo CSD cada uno de ellos es distinto. Esto porque el sello generado depende del mensaje, a diferencia de los sellos con tinta. El sello que tienen los comprobantes fiscales son *firmas digitales*. Por lo tanto, en este trabajo el termino de sello hace referencia propiamente a una firma digital a menos que se indique lo contrario.

Estas firmas son más seguras por su componente algorítmico y su fundamento matemático en la *teoría de números*. El resultado de cada llamada al algoritmo de firma varía de acuerdo al mensaje. En cambio una firma autógrafa al ser siempre idéntica se puede recortar y copiar en distintos documentos con un software de edición fotográfica para cometer un fraude.

```
AM0PWKyhvpj1Pf7AJVzAAGjaYU0t6r5hjk0DOj+wISCSdA2LZj7jmnBKivivgU8J5svcto9kABfNm246HG2y8
  Q6YcQJmB6Dw2bUBoZfrPE54yP+S5MfPtCw5QhS948Pc91gJcLPrHmaRXINaEqq0mTGWr4aWSAZxcb9
  Dql9KnvLcXt30KISnbc2+4m9RtpsTPLk2joKFGxf8eejGL69vO8txtmLqioInFDhTPWQcIKMdUutUbREsSsQ
  SfmOuoQdVBCCMY7SUK2ZtGDaCnshQSOVz/GHGfLQT4Qj0hetPtaDi60YPM5Mf3cekonBHb4jc2+FuCJ
  W+JKCsnI7sJ4+iYg==
```

Figura 1.2: Ejemplo de Sello Digital generado con un certificad de 2048 bits.

El CSD es emitido por el Servicio de Administración Tributaria (SAT) y contiene datos que asocian una clave pública a un contribuyente de manera única. Todos los CFDI son sellados con el CSD del emisor. Este contiene la clave pública correspondiente para verificar que los sellos generados por su respectiva clave privada son auténticos. Actualmente se utiliza una clave pública RSA con 2048 bits de longitud. El sello que produce también tiene esta longitud. La Figura 1.3 muestra una gráfica con el crecimiento que ha tenido la emisión de CSD en México. La cantidad es acumulada por año hasta el 30 de Abril del 2018.

Un aspecto importante es el espacio de almacenamiento para la clave pública. Sin considerar los datos personales que contiene el certificado, el directorio público para consultarlas necesitaría 1.854 GB de espacio en disco duro. Al utilizar el ECC las claves ocuparían 231.73 MB. Es decir, una reducción en el espacio de almacenamiento del 87.5 %. Esta nueva clave ofrece la misma seguridad pero con un tamaño de solo 256 bits.

En este trabajo se utilizarán los prefijos Tera, Giga, Mega y Kilo del Sistema Internacional de Unidades (SI, *The International System of Units*). Estos son potencias base 10. Así, por ejemplo 1 MB = 10^6 bits (Newell y Tiesinga, 2019). La mayoría de fabricantes de almacenamiento prefieren esta métrica. También están respaldados por el Centro Nacional de Metrología (CENAM) (Centro Nacional de Metrología, 2003). Por todo esto, se emplean los prefijos del SI para métricas de almacenamiento y ancho de banda.

Desde el año 2014 los ciudadanos con actividades económicas están obligados a emitir *comprobantes fiscales digitales por Internet* (CFDI) sellados con el *algoritmo de firma RSA*. Esto ha permitido al SAT automatizar los procesos contables y la recaudación de impuestos. En la Figura 1.4 se muestra una gráfica con el impacto que ha ocasionado este nuevo esquema de facturación. Se presenta un crecimiento en la cantidad de CFDI acumulados por año. Se recuerda que el tamaño del sello depende de la longitud de la clave pública.

Una desventaja generalizada de los criptosistemas de clave pública es utilizar claves de mayor tamaño para aumentar el nivel de seguridad. Por lo cual, el tiempo de ejecución de los algoritmos de firma y el espacio de almacenamiento se afectan nega-

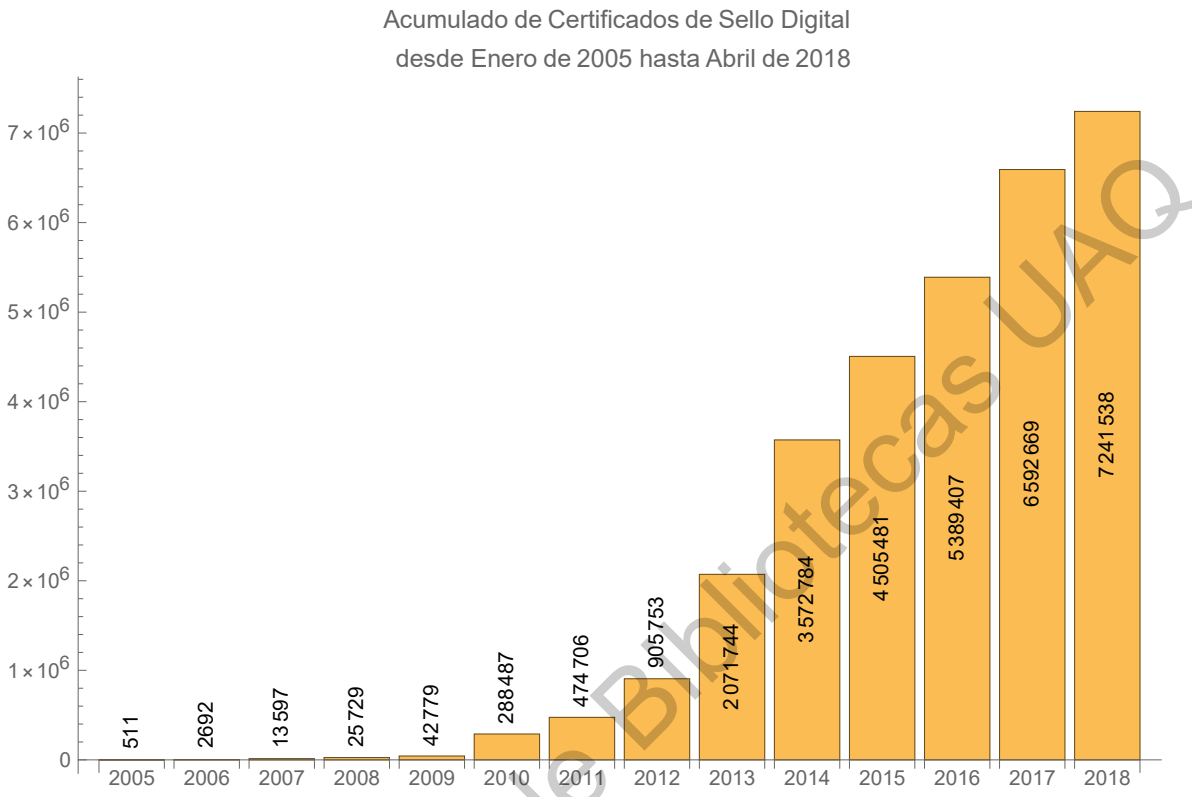

Figura 1.3: Cantidad de Certificados de Sello Digital acumulado por año con corte al 30 de Abril del 2018.

tivamente. Si cada uno de los 35 347 377 408 sellos acumulados al 30 de Abril del 2018 son de 2048 bits, el almacenamiento para el campo del sello sería de 9.049 TB. En cambio, si se utilizaran sellos basados en el ECC el espacio ocupado sería de 1.131 TB. Es importante resaltar que estos espacios son únicamente ocupados por los sellos. El CFDI es un archivo XML con más información. Por lo que en la práctica se requeriría más espacio de almacenamiento del expuesto anteriormente. Con todo, la reducción en el tamaño del sello y las claves con el ECC no debe pasarse por alto.

1.1. Planteamiento del problema

Actualmente el uso del criptosistema de clave pública RSA para sellar el CFDI es importante debido a la seguridad que ofrece ante posibles fraudes. Aún cuando requiere claves y firma con una longitud de 2048 bits. Se puede observar por la tendencia en el crecimiento de la emisión de CSD y CFDI un impacto en los costos de procesamiento y almacenamiento tanto del contribuyente como del SAT y sus Proveedor Autorizado de Certificación (PAC).

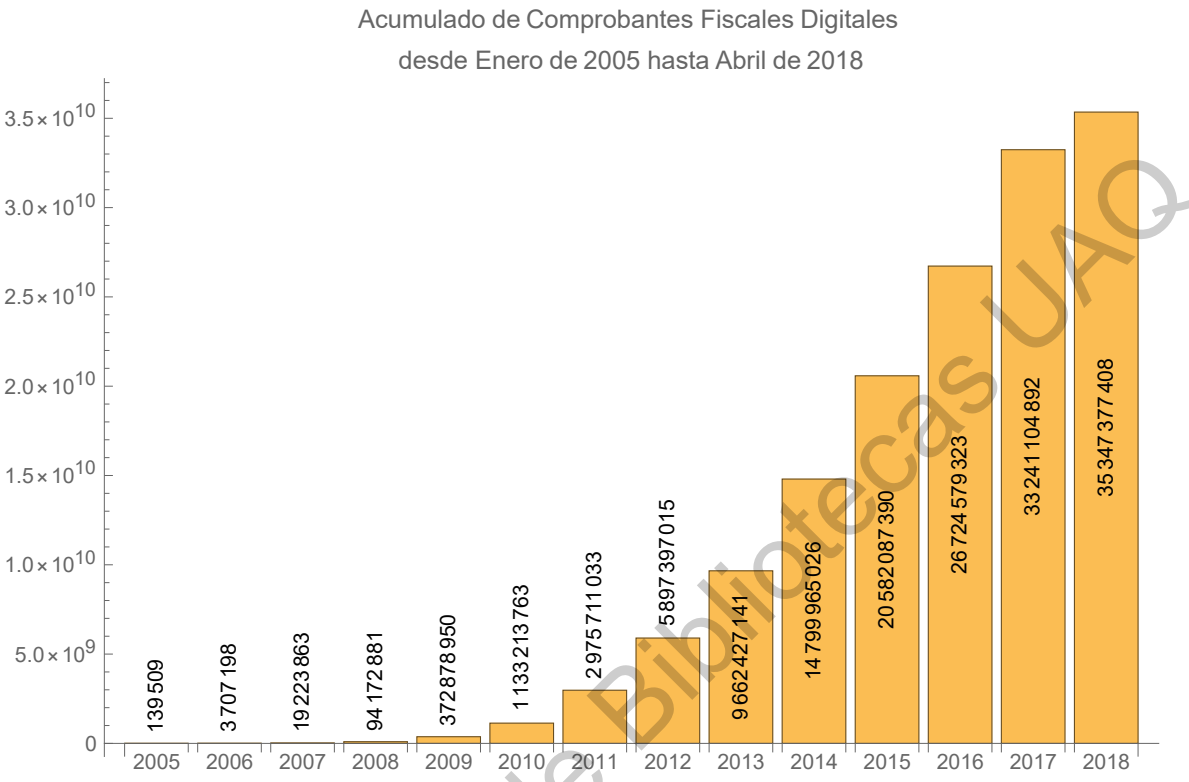

Figura 1.4: Acumulado de facturas digitales con corte al 30 de Abril del 2018.

Considerando lo anterior se propone una alternativa con claves y sellos de menor tamaño manteniendo una seguridad igual o mayor. García, Henríquez y Cortéz (García y col., 2008) así como Henríquez y Chávez (Chavez y Henriquez, 2015) han señalado fallas de seguridad en el CFDI al utilizar claves RSA con tamaños no recomendados por el National Institute of Standards and Technology (NIST, Español: Instituto Nacional de Estándares y Tecnología de EE. UU.) (Barker, 2020). Estos autores han propuesto el uso del criptosistema de curva elíptica (ECC, *Elliptic Curve Cryptography*) por sus claves de menor tamaño que ofrecen una seguridad equivalente a la firma RSA.

1.2. Justificación

Los profesionales de las tecnologías de la información y comunicación poseen los conocimientos matemático para implementar el criptosistema RSA. Sin embargo, implementar el criptosistema con curva elíptica requiere estudios especializados en áreas de las matemáticas como teoría de números, álgebra moderna, criptografía y ciencias de la computación. Por lo tanto, para realizar una implementación segura y eficiente del ECC es necesario realizar estudios de posgrado en matemáticas, computación y criptografía.

En México hay 64.7 millones de contribuyentes que venden productos o servicios y están obligados a expedir CFDI sellados con un CSD. El contribuyente debe almacenar estos comprobantes fiscales durante cinco años en caso de aclaraciones o auditorías del SAT. Un 96 % de los CFDI son timbrados por un *proveedor autorizado de certificación* (PAC). Para ello, el emisor comparte su clave privada y en su lugar el PAC ejecuta el algoritmo de firma RSA . Existen 79 empresas que son PAC. Algunas de ellas también ofrecen servicios de almacenamiento en la nube. Además, una copia de todos los CFDI es almacenada por el SAT para compararse con registros bancarios y detectar defraudación fiscal. Así, los beneficiados de tener un sello reducido serían los contribuyentes, los PAC y el SAT.

Datos de la Asociación Mexicana de Proveedores Autorizados de Certificación (AMEXIPAC) estiman que en México se emiten 17 millones de facturas al día. Así, diariamente estos sellos ocupan 4.35 GB de almacenamiento. De donde resulta que el SAT requiere 130.6 GB en un mes, 1.59 TB en un año y 7.94 TB para cinco años. Con sellos basados en el ECC, se pronostican 992.8 GB de almacenamiento para todos los sellos generados durante cinco años. El espacio ocupado por los sellos más los otros campos del archivo XML que contiene el CFDI es mayor a las cantidades antes mencionadas. Este ahorro es simplemente para el campo del sello digital. En la Tabla 1.1 se presenta una comparación de costos por almacenar sellos digitales durante cinco años en diferentes plataformas de servicios en la nube. Se observa que en cualquiera el costo se reduce 88 % con sellos basados en la ECC. No solo el SAT se beneficiaría de tener claves y sellos de menor tamaño, también los contribuyentes que realizan sus facturas y los PAC.

Plataforma	**Tarifa GB por mes**	**Costo por almacenar sellos RSA 5 años**	**Costo por almacenar sellos ECC 5 años**
Amazon S3	$0.023	$180.23	$22.52
Google Cloud Storage	$0.026	$203.74	$25.46
Microsoft Azure Files	$0.058	$454.49	$56.79

Tabla 1.1: Costos de almacenamiento estimados en las tres principales plataformas de computo en la nube. El precio es en dólar y puede variar según su uso.

El problema se abordará con un enfoque matemático, desarrollando un *prototipo* de sello basado en la ECC. En la medida de lo posible se consideran las características que señalan las leyes y normas mexicanas para el sellado de un CFDI. No se pretende definir algún estándar o protocolo nuevo para su generación. La regulación y legislación de los estándares tecnológicos para sellar un CFDI compete al SAT, la Secretaría de Economía (SE) y la Secretaría de la Función Pública (SFP). Por lo tanto está fuera de nuestro alcance. Además, al referirnos a la *seguridad* del sello estaremos hablando

sobre la *seguridad teórica*. Esta se enfoca en la dificultad de resolver eficientemente el problema matemático en el que está basado el criptosistema y no en los errores o ataques a la implementación.

Muchos avances para la eficiencia de la ECC se han desarrollado en los últimos 30 años. Posiblemente el más importante es el Elliptic Curve Digital Signature Algorithm (ECDSA, Español: Algoritmo de firma digital con curva elíptica). Estudios han encontrado que el ECDSA es más eficiente que el algoritmo de firma RSA (Vanstone, 1997) (Caelli y col., 1999) (Bernstein y col., 2011). Se plantea que la base de la propuesta para el sello de un CFDI con curva elítica sea este algoritmo. Al igual que la ECC, la seguridad del ECDSA está basada en el Elliptic Curve Discrete Logarithm Problem (ECDLP, Español: Problema del logaritmo discreto en curva elíptica). Hasta la fecha en que se redacta este documento es inexistente algún algoritmo que resuelva este problema en tiempo polinomial.

Capítulo 2

Antecedentes

En el año 1978 Rivest, Shamir y Adleman equiparon el sistema de correo electrónico con las propiedades de privacidad y autenticidad mediante un nuevo método conocido como *criptografía de clave pública* (Rivest y col., 1978). El *criptosistema Rivest-Shamir-Adleman* (RSA) permite a cualquier usuario cifra un mensaje M con la clave pública del destinatario. Este último descifra el criptograma utilizando su clave privada. Un análogo a la firma autógrafa para el mensaje M se genera de forma inversa, se descifra el mensaje M para obtener una firma s y la identidad del signatario se comprueba recuperando el mensaje M a partir de la firma s. Las firmas generadas con este método dependen tanto del mensaje como del firmante. La seguridad del criptosistema RSA descansa en la dificultad computacional de factorizar un número entero de 240 cifras o más producto de dos números primos p y q (Boudot y col., 2020) (Garey y Johnson, 1990). Regularmente se nombra este producto como *módulo* y se denota con la letra n. El criptosistema RSA es el estándar tecnológico definido por el SAT para *sellar* los comprobantes fiscales digitales por Internet (Santín, 2017).

Neal Koblitz (Koblitz, 1987) y Victor Miller (Miller, 1985) proponían utilizar la estructura de grupo generada por las curvas elípticas para diseñar otro criptosistema de clave pública. Koblitz se enfocó en los campos de característica dos por su eficiencia en la multiplicación de un punto por un escalar mientras que Miller expuso el acuerdo de clave Diffie-Hellman con curva elíptica sobre un campo primo. Luego, en el año 2000 Solinas diseño un algoritmo para curvas de Koblitz que explota el endomorfismo de Frobenius y acelera un 50 % la multiplicación de un punto por un escalar (Solinas, 2000). De manera semejante, en el año 2001 Gallant, Lambert y Vanstone mejoran la velocidad para realizar esta operación en curvas elípticas definidas sobre campos primos (Gallant y col., 2001). Los Standards for Efficient Cryptogrpahy (SEC, Español: Normas para criptografía eficiente) se publicaron en el año 2010 y extendieron el termino *curva de Koblitz* a curvas definidas en campos de característica prima que emplean endomorfismos para cálculos eficientes (Brown, 2009). Este documento define la curva de Koblitz *secp256k1*, misma que se utilizará en la nueva propuesta de sello digital para comprobantes fiscales digitales.

En 1985 Schoof presenta un algoritmo de complejidad $\mathcal{O}(log^9 p)$ para calcular el número de puntos de una curva elíptica en un campo finito F_p (Schoof, 1985). Una mejora sustancial fue realizada en el año 1995 y dio como resultado el algoritmo Schoof-Elkies-Atkin (SEA) (Schoof, 1995). A causa de esto, la ECC recibió un impulso para llevarse de la teoría a la práctica.

En el año 1991 el NIST publicó la propuesta de un Digital Signature Standard (DSS, Español: Estándar de firma digital). En respuesta, John C. Anderson comunicó la idea de S. Vanstone sobre un ECDSA. Este sustituye el grupo cíclico $(\mathbb{Z}/n\mathbb{Z})^*$ en el esquema de firma digital de ElGamal por el grupo de puntos de una curva elíptica. El ECDSA fue aceptado como un estándar ISO en el año 1998, como un estándar IEEE en el año 2000 y como un estándar ANSI en el año 1999 (Johnson y col., 2001).

En el año 1997 dos investigadores en los laboratorios RSA, Robshaw y Yin, compararon la ECC con el criptosistema RSA (Robshaw y Yin, 1997). Observaron que la ECC implementada sobre un campo binario con 2^{160} elementos tiene la misma seguridad que el RSA si su módulo n es de 1024 bits. Sus datos mostraron que la ECC requiere menor espacio de almacenamiento. Por otra parte, el desempeño de la ECC resultó superior en los procesos de descifrado o firma aunque en el proceso de verificación sobresalió el RSA. Otros investigadores obtuvieron datos similares al comparar ambos criptosistemas pero con diferentes tamaños de clave (Vanstone, 1997), (Jansma y Arrendondo, 2004), (Bernstein y col., 2011), (Mahto y Yadav, 2017), (Toradmalle y col., 2018). Además, Suárez-Albela, Fernández-Caramés, Fraga-Lamas y Castedo señalaron que en redes de Internet de las cosas (IoT) se presenta un rendimiento superior y menor consumo de energía con la ECC (Suárez-Albela y col., 2018). Conviene subrayar que la operación realizada al timbrar una factura es firmar digitalmente la digestión o *hash* del CFDI. Debido a los CSD la verificación raramente se lleva a cabo.

En 1999 los profesores Caelli, Dawson y Rea (Caelli y col., 1999) evaluaron los distintos algoritmos de firma digital. Pusieron especial énfasis en las firmas generadas con ECC. Notaron como la Public Key Infrastructure (PKI, Español: Infaestructura de clave pública), tanto mundial como nacional, debería proveer distintos algoritmos de firma en caso de que alguno fuera comprometido. Así, se tendría un sistema redundante. La demora en adoptar firmas digitales con curva elíptica se debe a la complejidad matemática para crearlas de tal manera que sean seguras y eficientes. Esto debido a que existen pocos profesionales de las tecnologías de la información y comunicación con tal grado de especialización. Los comprobantes fiscales digitales en México son un ejemplo de estos problemas. La PKI del SAT soporta únicamente el algoritmo de firma RSA. Así, se tiene un único punto de falla que podría causar un colapso en el esquema de comprobantes fiscales. Además, a los estudiantes de las carreras en tecnologías de la información (TI) de nuestro país no se les imparten tópicos relacionados con curvas elípticas. Incluso en licenciaturas de matemáticas puras es difícil encontrar esta asigna-

tura.

En el año 2004 una alternativa al comprobante fiscal en papel se desarrolla por el SAT, el comprobante fiscal digital (CFD) (Asociación Mexicana de Proveedores Autorizados de Certificación, 2018). Para emitir un CFD el contribuyente debía contar con su firma electrónica y un certificado de sello digital (CSD). Este último se solicita después de obtener la firma. El CSD permite sellar cada factura y garantizar su autenticidad. Las facturas o comprobantes fiscales digitales son un archivo XML al que se agregan los detalles de la transacción en sus atributos. Tiene un campo exclusivo para ser sellado con el CSD. Debido a la brecha tecnológica en el país apareció un tercer actor para emitir los CFD, el *proveedor autorizado de certificación*. Este se encarga de validar el cumplimiento de los estándares tecnológicos en los CFD de los contribuyentes que contratan sus servicios. Además, coloca el sello de visto bueno en lugar del SAT. Un nuevo esquema para emitir comprobantes fiscales por Internet (CFDI) se implantó en el año 2011 y entre sus características destacan: la estandarización de los archivos XML, el uso de los CSD y el acceso a herramientas gratuitas para generar facturas electrónicas (Asociación Mexicana de Proveedores Autorizados de Certificación, 2018). Fue hasta el año 2012 cuando se publica la Ley Federal de Firma Electrónica Avanzada con el objetivo de regular y homologar su uso en todo el país («Ley de Firma Electrónica Avanzada», 2012). Más aún, en el año 2014 el SAT dejó de reconocer los comprobantes fiscales en papel, obligando a expedir únicamente comprobantes fiscales digitales por Internet. Este mismo año se promulgó el Reglamento de la Ley de Firma Electrónica Avanzada (Cámara de Diputados del H. Congreso de la Unión, 2014). La ley y el reglamento de firma electrónica definen que los encargados de establecer los estándares tecnológicos y publicar las disposiciones generales de la ley de firma electrónica avanzada son la Secretaría de la Función Pública (SFP), la Secretaría de Economía (SE) y el SAT (Secretaría de Economía y col., 2016). De este documento se derivan las especificaciones técnicas del Anexo 20 de la Resolución Miscelánea Fiscal del 2017 (Santín, 2017), donde se definen los algoritmos para sellar los CFDI.

Los primeros investigadores que resaltaron los fallos de seguridad en el CFDI fueron González-García, Rodríguez-Henríquez y Cruz-Cortés. Detectaron el envío de la clave privada por Internet, la fácil alteración de la fecha y hora en los comprobantes, un mal uso y acceso a la lista de revocación, empleo de algoritmos criptográficos obsoletos, tamaños de clave precarios y la omisión de buenas practicas para un almacenamiento seguro (González-Garcıa y col., 2007). Propusieron la creación de una *Notaría Digital*, un tercero de confianza que coloca un sello de tiempo a todos los CFDI emitidos. Esta nueva dependencia digital también se encargaría de guardar una copia de todos los comprobantes estampados. Esto con el fin de evitar modificaciones malintencionadas. También los autores notaron el uso del Message-Digest Algorithm 5 (MD5, Español: Algoritmo de Resumen del Mensaje 5), función *hash* que Wang y Yu demostraron no ser resistente a colisiones (Wang y Yu, 2005). Así mismo, alertaron sobre el imprudente tamaño de claves RSA. Su uso ponía en riesgo la seguridad de los CFDI. Esto de

acuerdo a cálculos con la formula propuesta por Lenstra (Lenstra y Verheul, 2001). Considerando esto, plantearon el uso de ECC para generar las firmas del CFDI. El año siguiente implementaron su Notaría Digital con el protocolo Evidence Record Syntax (ERS, Español: Sintaxis del Registro de Evidencias) y el Long-term Archive Protocol (LTAP, Español: Protocolo para archivar a largo plazo) (García y col., 2008). Sin embargo, en su implementación no se utilizó ECC para los sellos de los comprobantes.

En el año 2015 una nueva auditoría de seguridad a los CFDI se realizó por León y Rodríguez-Henríquez (Chavez y Henriquez, 2015). Por lo que se refiere al algoritmo de *hash*, se sustituía el algoritmo MD5 por el Secure Hash Algorithm 1 (SHA-1, Español: Algoritmo de digestión seguro 1). No obstante, Wang, Yin y Yu demostraron que esta función tenía colisiones (Wang y col., 2005). El tamaño del módulo RSA continuaba incumpliendo las recomendaciones del NIST, en ese entonces módulos de 3072 bits. De igual manera, los investigadores recomendaron ECC para generar los sellos plasmados en el CFDI.

En resumen, estudios previos han desarrollado algoritmos para llevar la ECC a la práctica. Estos encontraron notables diferencias en eficiencia entre el RSA y la ECC. Por otra parte, la falta de alternativas para sellar un CFDI es una amenaza al esquema de facturación electrónica en México. Como varios autores han resaltado, la criptografía con curva elíptica y su posible aplicación a la firma digital de los CFDI parece una solución viable. Con base en estos antecedentes, el capítulo siguiente presenta la hipótesis y supuesto del presente trabajo de investigación.

Capítulo 3

Hipótesis

En este capítulo se presenta el supuesto de utilizar la criptografía de curva elíptica como una alternativa para firmar los CFDI. Además, se considera la hipótesis de reducir las variables computacionales en la creación, almacenamiento y transmisión de los CFDI.

3.1. Supuesto

La criptografía de curva elíptica es una alternativa para timbrar los CFDI y evitar un único punto de fallo en el esquema del comprobantes fiscales del SAT. También, la seguridad de los CFDI firmados con criptografía de curva elíptica es igual o mayor a los firmado por RSA.

3.2. Hipótesis

La firma de los CFDI empleando criptografía con curva elíptica aporta ventajas en tiempos de cómputo reducidos, menor espacio de almacenamiento y mayores tasas de transmisión en redes de computadoras. Además, los usuarios tendrán en su posesión claves con menor tamaño pero que proporciona igual o mayor nivel de seguridad que las actuales claves de la firma RSA. La principal desventaja consiste en la verificación de la firma, el tiempo con curvas elípticas es mayor al de validar firmas RSA.

El capítulo siguiente, *Objetivos*, conducirá los esfuerzos para respaldar el supuesto y comprobar la hipótesis. Es importante notar que la hipótesis es de carácter cuantitativo mientras que el supuesto de carácter cualitativo. En los objetivos se especifica cómo medir las variables de la hipótesis y demostrar el supuesto de utilidad y seguridad.

Capítulo 4

Objetivos

El propósito del capítulo es presentar los objetivos que ayudarán a responder el problema de investigación, una alternativa al sello del CFDI con curva elíptica. Además, permitirán validar el supuesto, la hipótesis y definir el producto a entregar finalizado el trabajo. Se describen varios objetivos específicos para lograr un prototipo de sello digital para CFDI con curva elíptica.

4.1. Objetivo general

El objetivo general de este trabajo es diseñar, desarrollar y validar un prototipo computacional de sello digital para CFDI utilizando criptografía de curva elíptica en el lenguaje de programación de alto nivel Mathematica (Wolfram Research, Inc., s.f.). Está más allá del alcance de este estudio llevar a la práctica esta alternativa de timbrado en la infraestructura de firma digital del SAT. De modo que se debe considerar como una *prueba de concepto* y no como un prototipo totalmente operativo.

4.2. Objetivos específicos

A continuación se listan los objetivos específicos para lograr el prototipo propuesto:

- Aplicar algoritmos para obtener claves públicas y privadas del criptosistema RSA y del criptosistema de curva elíptica en Mathematica®.
- Aplicar el algoritmo de firma digital RSA y el algoritmo de firma digital con curva elíptica a un CFDI en Mathematica®.
- Comparar entre ambos criptosistemas el tamaño de claves, tiempo de cómputo y espacio de almacenamiento con estadística descriptiva.

- Comparar entre ambos métodos el rendimiento de transmisión de los sellos en una red punto a punto formada por dos computadoras *Raspberry Pi* con estadística descriptiva.
- Verificar que los parámetros seleccionados para cada criptosistema resistan ataques conocidos por la comunidad criptográfica y determinar su nivel de seguridad.

En síntesis, se define el producto a entregar con este proyecto, así como una guía para su realización. También las métricas para resaltar sus ventajas y desventajas. En las páginas que siguen se proporcionan las bases necesarias para cumplir con los objetivos. Se abordan temas de ECC y RSA, sobre el lenguaje de alto nivel Mathematica y el contexto del CFDI en México.

Capítulo 5

Fundamentos

En este capítulo se abordará la base teórica de los criptosistemas de curva elíptica y el RSA. El desarrollo del prototipo se alcanzarán con bases solidas en teoría de números, teoría de campos y grupos, criptografía de clave pública y programación basada en el paradigma funcional y simbólico de *Mathematica*©. También se expone el contexto, marco legal y estándares tecnológicos de los CFDI en México.

5.1. Teoría de números

5.1.1. Divisibilidad

Definición 5.1.1 (Divisibilidad)**.** Sean a y b números enteros con a distinto de 0. Se dice que a divide b si existe un entero k tal que $b = k\,a$. Esto se denotado por mj $a \mid b$. (Trappe y Washington, 2006).

Definición 5.1.2 (Número compuesto)**.** Si n puede obtenerse a partir de dos números a, b como el producto $n = a\,b$ tal que $1 < a, b < n$ se dice que n es un *número compuesto*. Es decir, n es divisible por uno o más divisores distintos de 1 y él mismo.

Ejemplo 5.1.1. $2 \mid 12, \quad 3 \mid 15, \quad 7 \nmid 17$. El símbolo $\nmid$ del último enunciado se lee como *no divide*. En otras palabras, siete no divide a diecisiete. Además el número 15 es compuesto porque $15 = 3 \times 5$.

5.1.2. Números primos

Definición 5.1.3 (Número primo)**.** Sea p un número estrictamente mayor a uno y divisible únicamente por 1 y él mismo. Decimos entonces que p es un número primo.

Euclides demostró, por reducción a lo absurdo, que existen infinitos números primos.

Teorema 5.1.1. *Existen infinitos números primos.*

La frecuencia con que aparecen los números primos en la recta numérica no sigue un patrón conocido. Por lo tanto, resulta muy difícil conocer el número exacto de primos en un intervalo. Con todo, el teorema de los números primos permite aproximar cuantos de ellos menores a un número x existen.

Teorema 5.1.2 (Teorema de los números primos)**.** *Sea $\pi(x)$ la función que devuelve el número de primos menores que x. Entonces*

$$\pi(x) \approx \frac{x}{\ln x},$$

de forma que la razón $\pi(x)/(x/\ln x) \to 1$ cuando $x \to \infty$ (Trappe y Washington, 2006).

La demostración de este teorema se realizo en 1896 por Jaques Hadamard y Charles de la Vallée Poussin. Ellos se basaron en el desarrollo de la teoría de funciones de variable compleja de Riemann (Cilleruelo, 2000). En 1949 Paul Erdös y Atle Selberg presentaron una demostración elemental del teorema (Erdös, 1949) (Selberg, 1949). Unas décadas después, en 1980, D. J. Newman propuso otra demostración analítica del teorema (Newman, 1980).

Ejemplo 5.1.2. En criptografía se emplean números primos de 200 cifras o más. Así tenemos que hay

$$\pi(10^{200}) - \pi(10^{199}) \approx \frac{10^{200}}{\ln 10^{200}} - \frac{10^{199}}{\ln 10^{199}} \approx 1{,}9 \times 10^{197}$$

primos con 200 dígitos. Para tener una idea, si apilaramos este número de veces monedas de un peso se alcanzaría el diámetro del universo observable y se podría repetir otras cientos de miles de billones de veces.

Una propiedad interesante de los números primos es que pueden utilizarse para construir cualquier número entero. Este es el teorema fundamental de la aritmética.

Teorema 5.1.3 (Teorema fundamental de la aritmética)**.** *Todos los enteros mayores que 1 se pueden expresar como el producto de potencias de números primos de manera única (salvo permutación de los factores).*

5.1.3. Máximo común divisor

Definición 5.1.4 (Máximo común divisor)**.** El máximo común divisor de dos números a, b, es el mayor entero positivo que los divide. Se denota como $\text{mcd}(a, b)$.

Definición 5.1.5 (Primos relativos)**.** Decimos que a y b son primos relativos si $\text{mcd}(a, b) = 1$.

Función ϕ de Euler

Una función que se emplea en secciones delante es la *función fi de Euler*. Ya que se definió el concepto de primos relativos se puede formular.

Definición 5.1.6 (Función ϕ de Euler)**.** La función fi de Euler se define para enteros positivos n mediante

$$\phi(n) = k$$

donde k es el número de enteros positivos menores o iguales a n que son primos relativos de n (Fraleigh, 1998).

En particular, si $n = p$ donde p es un número primo se tiene

$$\phi(p) = (p - 1).$$

Por ejemplo $\phi(17) = 16$. En el caso que $n \neq p$ simplemente se evalúa cada número entre uno y n. Por ejemplo, $\phi(10) = 4$. Ahora bien, este proceso de cómputo es costoso cuando la lista de números a evaluar es grande. De ahí el siguiente teorema para calcular el valor de la función fi de Euler si se conocen los factores primos de n (Paar y Pelzl, 2009).

Teorema 5.1.4. *Sea n de la forma*

$$n = p_1^{e_1} p_2^{e_2} \dots p_k^{e_k}$$

donde p_i son distintos números primos y e_i son enteros positivos. Entonces

$$\phi(n) = n \prod_{i=1}^{k} \left(1 - \frac{1}{p_i}\right).$$

Ejemplo 5.1.3. Calcule el valor de la función fi de Euler $\phi(100)$.
Solución. La factorización de cien es $100 = 2^2\, 5^2$. Por el teorema anterior

$$\phi(100) = 100 \left[\left(1 - \frac{1}{2}\right)\left(1 - \frac{1}{5}\right)\right] = 40.$$

Existen 40 primos relativos a 100. Estos son: 1, 3, 7, 9, 11, 13, 17, 19, 21, 23, 27, 29, 31, 33, 37, 39, 41, 43, 47, 49, 51, 53, 57, 59, 61, 63, 67, 69, 71, 73, 77, 79, 81, 83, 87, 89, 91, 93, 97, 99.

Note el caso cuando n es el producto de dos primos distintos $n = pq$. Por el teorema 5.1.4 se tiene que

$$\phi(n) = (p - 1)(q - 1).$$

Ejemplo 5.1.4. Calcular $\phi(91)$.
Solución. Dado que $91 = 7 \times 13$ tenemos que

$$\phi(91) = (7 - 1)(13 - 1) = 72.$$

El algoritmo de Euclides

El algoritmo de Euclides calcula el mínimo común divisor de una manera rápida y eficientemente. Su complejidad es polinomial y esta acotada por una función logarítmica. Supongamos que a es mayor que b, en caso contrario se intercambian y r es el residuo de la división entera a/b. Entonces $a = b\,q + r$. Después, se busca un número u que divide a y b. Por lo tanto $a = s\,u$ y $b = t\,u$ y u divide al residuo

$$r = a - b\,q = s\,u - (t\,u)q = u\,(s - t\,q).$$

De manera similar, se busca un número v que divida b y r. Así tenemos que $b = s'\,v$, $r = t'\,v$ y v divide el entero a porque

$$a = b\,q + r = (s'\,v)q + t'\,v = v(s'\,q + t').$$

Por lo tanto un divisor común de a y b es un divisor de b y r. Este proceso puede iterarse de la siguiente manera:

$$\begin{array}{lll}
q_1 = \left\lfloor \dfrac{a}{b} \right\rfloor & a = b\,q_1 + r_1 & r_1 = a - b\,q_1 \\
q_2 = \lfloor \dfrac{b}{r_1} \rfloor & b = q_2\,r_1 + r_2 & r_2 = b - q_2\,r_1 \\
q_3 = \lfloor \dfrac{r_1}{r_2} \rfloor & r_1 = q_3\,r_2 + r_3 & r_3 = r_1 - q_3\,r_2 \\
\vdots & \vdots & \vdots \\
q_n = \lfloor \dfrac{r_{n-2}}{r_{n-1}} \rfloor & r_{n-2} = q_n\,r_{n-1} + r_n & r_n = r_{n-2} - q_n\,r_{n-1} \\
q_{n+1} = \lfloor \dfrac{r_{n-1}}{r_n} \rfloor & r_{n-1} = q_{n+1}\,r_n + 0 & r_n = r_{n-1}/q_{n+1}
\end{array}$$

El algoritmo termina cuando q_{n+1} divide exactamente a r_{n-1}, de donde resulta que r_n es el máximo común divisor de a y b. Es decir $\mathrm{mcd}(a, b) = r_n$ (Weisstein, 2005).

En su libro, Bach y Shallit presentan una implementación eficiente del algoritmo de Euclides (Bach y col., 1996). Se trata de un algoritmo recursivo. Se presenta los pasos en el Algoritmo 1. En el paso 4 se utiliza la operación *módulo* que se define más adelante, en la sección 5.1.5.

Ejemplo 5.1.5. Calcular el $\mathrm{mcd}(210, 947)$.
Solución. Asignamos $a = 947$ y $b = 210$ por la condición $a > b$. La división entre a y b produce el cociente 4 y residuo 107. Después dividimos 210 entre 107 y obtenemos el cociente 1 y residuo 103. Continuamos dividiendo 107 entre 103, lo que resultan en el cociente 1 y residuo 4. Iteramos hasta que el residuo r_{n+1} sea igual a cero. Una

Algoritmo 1: Algoritmo de Euclides

Datos: Los enteros a y b

Resultado: El máximo común divisor de a y b

1 **si** *b es igual a 0* **entonces**

2 devuelve a

3 **en otro caso**

4 ejecuta el Algoritmo de Euclides con las entradas $a \leftarrow b$ y $b \leftarrow a \pmod{b}$

vez terminado el algoritmo se tiene en el penúltimo residuo, es decir el valor de r_n, el máximo común divisor de ambos números.

$$\begin{aligned} 947 &= 4 \times 210 + 107 \\ 210 &= 1 \times 107 + 103 \\ 107 &= 1 \times 103 + 4 \\ 103 &= 25 \times 4 + 3 \\ 4 &= 1 \times 3 + 1 \\ 3 &= 3 \times 1 + 0. \end{aligned}$$

Por tanto el $\text{mcd}(210, 947) = 1$ y se concluye que 210 y 947 son primos relativos.

5.1.4. Solución de ecuaciones $a\,x + b\,y = d$

El algoritmo de Euclides permite demostrar el siguiente teorema.

Teorema 5.1.5. *Sean a y b dos enteros, con alguno diferente de cero, y sea $d = \text{mcd}(a, b)$. Entonces existen enteros x, y tal que $a\,x + b\,y = d$. En particular, existen enteros x, y con $a\,x + b\,y = 1$ si a y b son primos relativos.*

Surge la pregunta de cómo calcular x, y para resolver la ecuación $a\,x + b\,y = d$. Existe un método que para obtenerlos trabaja hacía atrás el algoritmo de Euclides. Se inicia a partir del último paso del algoritmo de Euclides y los residuos permiten calcular estos coeficientes. Se conoce como *el algoritmo de Euclides extendido* y se presenta a continuación.

El algoritmo de Euclides extendido

Supongamos que se empieza dividiendo a entre b, por tanto $a = q_1\,b + r_1$. Se ejecuta el algoritmo de Euclides y obtenemos los cocientes $q_1, q_2, \cdots, q_{n+1}$. Los coeficientes que solucionan la ecuación $a\,x + b\,y = d$ pueden obtenerse de la siguiente secuencia:

$$\begin{aligned} x_0 &= 1, & x_1 &= 0, & x_j &= -q_{j-1}x_{j-1} + x_{j-2}, \\ y_0 &= 0, & y_1 &= 1, & y_j &= -q_{j-1}y_{j-1} + y_{j-2} \end{aligned}$$

hasta obtener x_{n+1}, y_{n+1} que satisfacen

$$a\,x_{n+1} + b\,y_n + 1 = b.$$

Ejemplo 5.1.6. Considere el ejemplo 5.1.5 donde $q_1 = 4, q_2 = 1, q_3 = 1, q_4 = 25, q_5 = 1, q_6 = 3$, el cálculo del coeficientes x es:

$$\begin{aligned}
x_0 &= 1,\\
x_1 &= 0,\\
x_2 &= -4 \times x_1 + x_0 = 1,\\
x_3 &= -1 \times x_2 + x_1 = -1,\\
x_4 &= -1 \times x_3 + x_2 = 2,\\
x_5 &= -25 \times x_4 + x_3 = -51\\
x_6 &= -1 \times x_5 + x_4 = 53
\end{aligned}$$

De manera similar calculamos y:

$$\begin{aligned}
y_0 &= 0,\\
y_1 &= 1,\\
y_2 &= -4 \times y_1 + y_0 = -4,\\
y_3 &= -1 \times y_2 + y_1 = 5,\\
y_4 &= -1 \times y_3 + y_2 = -9,\\
y_5 &= -25 \times y_4 + y_3 = 230\\
y_6 &= -1 \times y_5 + y_4 = -239
\end{aligned}$$

Por lo tanto $947(53) + 210(-239) = 1$

Se puede escribir el algoritmo de Euclides como un sistema de ecuaciones de la forma

$$\begin{aligned}
\begin{bmatrix} a \\ b \end{bmatrix} &= \begin{bmatrix} q_1 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} b \\ r_1 \end{bmatrix}\\
\begin{bmatrix} b \\ r_1 \end{bmatrix} &= \begin{bmatrix} q_2 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} r_1 \\ r_2 \end{bmatrix}\\
\begin{bmatrix} r_1 \\ r_2 \end{bmatrix} &= \begin{bmatrix} q_3 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} r_2 \\ r_3 \end{bmatrix}\\
&\vdots\\
\begin{bmatrix} r_{n-2} \\ r_{n-1} \end{bmatrix} &= \begin{bmatrix} q_n & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} r_{n-1} \\ r_n \end{bmatrix}\\
\begin{bmatrix} r_{n-1} \\ r_n \end{bmatrix} &= \begin{bmatrix} q_{n+1} & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} r_n \\ r_{n+1} \end{bmatrix}
\end{aligned}$$

donde el residuo enésimo n es el máximo común divisor de a y b. Es decir $r_n = \text{mcd}(a, b)$ y $r_{n+1} = 0$.

A continuación se combinan cada una de las ecuaciones anteriores para obtener

$$\begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} q_1 & 1 \\ 1 & 0 \end{bmatrix} \cdots \begin{bmatrix} q_{n+1} & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} d \\ 0 \end{bmatrix}$$

Después definimos la matriz M_k como

$$M_k = \begin{bmatrix} m_k & c_k \\ d_k & e_k \end{bmatrix} = \begin{bmatrix} q_1 & 1 \\ 1 & 0 \end{bmatrix} \cdots \begin{bmatrix} q_k & 1 \\ 1 & 0 \end{bmatrix}$$

Despejando se tiene entonces

$$M_{n-1}^{-1} \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} d \\ 0 \end{bmatrix}$$

El determinante de la matriz M_{n-1} es $\det M_{n-1} = (-1)^n$ y se calcula la matriz inversa

$$M_{n-1}^{-1} = (-1)^n \begin{bmatrix} e_{n-1} & -c_{n-1} \\ -dn - 1 & m_{n-1} \end{bmatrix}$$

Despejando tenemos $(-1)^n (a\, e_{n-1} - b\, c_{n-1}) = d$, en consecuencia $x = (-1)^n e_{n-1}$, $y = (-1)^{n+1} c_{n-1}$.

Este razonamiento puede aplicarse a un algoritmo eficiente para calcular ambos coeficientes. Su orden de complejidad es $O((\log a)(\log b))$. El pseudocódigo para un lenguaje de programación se presenta en el algoritmo 2.

Algoritmo 2: Algoritmo extendido de Euclides

Datos: Dos enteros a, b

Resultado: Máximo común divisor d y coeficientes x, y que satisfacen la ecuación $a\, x + b\, y = \text{mcd}(a, b)$

```
1 M ← [1 0; 0 1] ;
2 n ← 0 ;
3 mientras b ≠ 0 hacer
4 |   calcular q ← ⌊a/b⌋ ;
5 |   M ← M [q 1; 1 0] ;
6 |   (a, b) ← (b, a − q b) ;
7 |   n ← n + 1
8 devolver d = u, x = (−1)^n M_{2,2}, y = (−1)^{n+1} M_{1,2}
```

5.1.5. Aritmética modular

De manera general, en criptografía se realiza aritmética en un conjunto finito de números enteros. Realizar operaciones aquí es muy distinto a realizarlos en conjuntos infinitos; por ejemplo los reales $\mathbb{R}$ (Paar y Pelzl, 2009). En el siguiente ejemplo se notan operaciones que genera resultados fuera del conjunto y como se convierten a elementos dentro del mismo.

Ejemplo 5.1.7. Considere el conjunto de números

$$S = \{0, 1, 2, 3, 4, 5, 6\}$$

y las operaciones

$$3 \times 2 = 6$$
$$4 + 1 = 5$$
$$5 + 5 = 3$$

El resultado de la operación $5 + 5 = 3$ es el residuo de dividir la suma $5 + 5 = 10$ entre 7. Esto genera un cociente 1 y residuo 3. El resultado de la operación es el residuo obtenido y se escribe $3 \pmod 7$.

Definición 5.1.7 (Operación módulo)**.** Sean a, r, m números enteros con $m > 0$. Escribimos

$$a \equiv r \pmod m$$

si m divide a $a - r$. Se conoce a m como el módulo y r es el residuo.

La definición anterior se lee *a es congruente con r módulo m*. Esta expresión matemática se simboliza con $\equiv$. La operación módulo devuelve el resto de una *división entera con residuo*. Es decir

$$a = q\,m + r \quad \text{con } 0 \leq r < m \quad \text{y} \quad q = \lfloor a/m \rfloor.$$

Así, cualquier número a se representa como un entero del intervalo $[0, m)$. De ahí que existan infinitos residuos congruentes con a módulo m.

Proposición. *Sean a, b, c, d, m enteros con $m \neq 0$. Suponga que $a \equiv b \pmod m$ y $c \equiv d \pmod m$ (Trappe y Washington, 2006). Entonces*

$$(a + c) \equiv (b + d) \pmod m, \qquad (a - c) \equiv (b - d) \pmod m, \qquad ac \equiv bd \pmod m.$$

Una noción importante en teoría de números y criptografía es *la congruencia* entre dos números. Una congruencia $\equiv$ se comporta de forma similar a una igualdad $=$; de ahí su parecido gráfico. En la anterior proposición las operaciones de suma, resta y multiplicación se realizan de forma normal a menos que el resultado sea mayor o igual a m. Si esto último sucede, se realiza la operación y después se anota el residuo de dividir entre m. En cada uno de los casos de la proposición anterior se conserva la congruencia. Tanto la operación a la izquierda del símbolo como a la derecha representan el mismo valor.

División módulo m

Se debe tener especial cuidado al dividir módulo m. La división en estos casos es más enredosa que la división común. Como regla general se puede dividir por $a \pmod m$ cuando el $\text{mcd}(a, m) = 1$ (Trappe y Washington, 2006).

Proposición. *Sean a, b, c, m enteros con $m \neq 0$ y $\text{mcd}(a, m) = 1$. Si $a\,b \equiv a\,c \pmod m$ entonces $b \equiv c \pmod m$. Dicho de otra manera, si a y m son primos relativos entonces se puede dividir ambos lados de la congruencia entre a.*

Ejemplo 5.1.8. Resolver $2x + 7 \equiv 3 \pmod{17}$.
La solución se puede obtener despejando. Así, tenemos $2x \equiv 3 - 7 \equiv -4 \pmod{17}$. Finalmente, $x \equiv -2 \equiv 15 \pmod{17}$. La división entre 2 es viable porque $\text{mcd}(2, 17) = 1$ (Trappe y Washington, 2006).

Por una cuestión de notación formal el residuo r se expresan tal que $0 \leq r < m$. Igualmente es correcto utilizar números negativos (Paar y Pelzl, 2009). En el ejemplo anterior se prefiere el valor final de x como $x = 15$ en lugar de $x = -2$.

Proposición. *Suponga que $\text{mcd}(a, m) = 1$. Sean s, t enteros tal que $a\,s + m\,t = 1$. Entonces $a\,s \equiv 1 \pmod m$. Nombramos la variable s como el inverso multiplicativo de $a \pmod m$ y se denota como a^{-1}.*

El inverso multiplicativo es útil cuando tenemos fracciones en las congruencias. Es decir, la división es igual a multiplicar por el inverso del denominador. Es necesario recalcar que solamente se tienen fracciones de la forma $\frac{b}{a}$ que cumplen con la condición que el $\text{mcd}(a, m) = 1$. El inverso s, así como el valor de t, se calculan con el algoritmo de Euclides extendido. Estos valores permiten resolver ecuaciones de la forma $ax \equiv c \pmod m$ cuando $\text{mcd}(a, m) = 1$.

Ejemplo 5.1.9. Desarrollando el ejemplo anterior se llega a $2x \equiv -4 \pmod{17}$. Despejando se tiene la fracción $-4/2$ y puede operarse como la multiplicación por el inverso del denominador $(-4)\,2^{-1}$ pues el $\text{mcd}(2, 1) = 1$. Se aplica el algoritmo de Euclides extendido con $a = 2$, $b = 17$ para obtener $s = 9$ y $t = 16$. Estos valores satisfacen $as + mt \equiv 1 \pmod{17}$. Entonces $(2)s \equiv (2)a^{-1} \equiv (2)9 \equiv 1 \pmod{17}$ y finalmente tenemos $(-4)1/2 \equiv (-4)9 \equiv 15 \pmod{17}$.

5.1.6. Exponenciación modular

En criptografía es común encontrar números de la forma $x^a \pmod m$. Por ejemplo obtener el residuo de $10^{257} \pmod{773}$. En primer lugar se tiene que calcular 10^{257}, un número muy grande con 257 cifras y el método convencional de multiplicaciones sucesivas requiere de $O(a \log m)^2$ (González, 2004). Existe un método más eficiente utilizando la representación base dos con complejidad $O(\log a)$.

Un número positivo n base b se denota $(d_{k-1}d_{k-2}\dots d_1d_0)_b$ donde d son los dígitos entre 0 hasta $b-1$. En este trabajo suele omitirse el paréntesis cuando la base $b = 10$. Entonces cualquier número n se puede representar como una combinación lineal

$$n = d_{k-1}b^{k-1} + d_{k-2}b^{k-2} + \cdots + d_1b^1 + d_0 \text{ (Koblitz, 1994).}$$

Ejemplo 5.1.10. Convertir 10^6 a base $b = 2$ (Koblitz, 1994).
Solución. Para convertir un número n a la base b primero se obtiene la última cifra como el residuo de dividir n entre b. Se sustituye n por el entero del cociente y se repite el anterior procedimiento para calcular la penúltima cifra. Así hasta que el cociente sea igual a cero. Finalmente tenemos que

$$10^6 = (11110100001001000000)_2.$$

Una vez que se convierte un número a base 2 se puede elevar un exponente de forma acelerada. El método de exponenciación modular rápida tiene fundamento en el siguiente teorema (González, 2004).

Teorema 5.1.6. *Sea $x^a \pmod m$ con x, a, m enteros y suponga que la forma binaria de a es*

$$a = d_{k-1}2^{k-1} + d_{k-2}2^{k-2} + \cdots + d_1 2 + d_0$$

entonces tenemos que

$$\begin{aligned} x^a &= x^{d_{k-1}2^{k-1}+d_{k-2}2^{k-2}+\cdots+d_1 2+d_0} \\ &= \prod_{i=0}^{k-1} x^{d_i 2^i} \pmod m \end{aligned}$$

.

Ejemplo 5.1.11. Calcular el residuo $10^{202} \pmod{773}$.

Solución. La representación en base 2 de $202 = (11001010)_2$. Por lo tanto

$$\begin{aligned}
10^{0\times 2^0} &= 1 \\
10^{1\times 2^1} &= 100 \\
10^{0\times 2^2} &= 1 \\
10^{1\times 2^3} &= 10^8 \equiv 82 \pmod{773} \\
10^{0\times 2^4} &= 1 \\
10^{0\times 2^5} &= 1 \\
10^{1\times 2^6} &= 10^{64} \equiv 348 \pmod{773} \\
10^{1\times 2^7} &= 10^{128} \equiv 516 \pmod{773}
\end{aligned}$$

Entonces $1 \times 100 \times 1 \times 82 \times 1 \times 1 \times 348 \times 516 = 1\,472\,457\,600 \equiv 47 \pmod{773}$.

Con esta base teórica se desarrolló el Algoritmo 3 para resolver la exponenciación modular $x^a \pmod m$ (González, 2004). Este puede implementarse en un lenguaje de programación para ser llamado en repetidas ocasiones.

Algoritmo 3: Exponenciación modular rápida

Datos: Valores x, a, m
Resultado: Residuo c que satisface $c = x^a \pmod m$

```
Obtener la representación binaria d_{k-1}2^{k-1} + d_{k-2}2^{k-2} + ··· + d_1 2 + d_0 ;
Sea c ← 1 ;
para i ← d_{k-2} a 0 hacer
    c ← c^2 (mod m) ;
    si d_i = 1 entonces
        c ← cx (mod m)
devolver c
```

5.1.7. Teorema chino del residuo

El *teorema chino del residuo* permite obtener a partir de una congruencia módulo m un sistema de congruencias. Cada una de estas nuevas congruencias tiene como módulo un factor primo de m. De forma inversa permite obtener una sola congruencia a partir de varias. En criptografía se emplea para acelerar cálculos y criptoanálisis.

Teorema 5.1.7 (Teorema chino del residuo)**.** *Sean $m_1, \ldots, m_k$ enteros tal que $\mathrm{mcd}(m_i, m_j) = 1$ con $i \neq j$. Dados los enteros $a_1, \ldots, a_k$ existe exactamente una solución $x \pmod{m_1 \ldots m_k}$ al sistema de congruencias simultáneas*

$$x \equiv a_1 \pmod{m_1}, \quad x \equiv a_2 \pmod{m_2}, \quad \ldots \quad , xa_i \equiv \pmod{m_k}.$$

Una forma de calcular la solución x de un sistema de congruencias es la siguiente:

1. Para $i = 1, \ldots, k$ calcular el producto $z_i = m_1 \ldots m_{i-1} m_{i+1} \ldots m_k$.
2. Para $i = 1, \ldots, k$ hacer $y_i = z_i^{-1} \pmod{m_i}$.
3. Calcular $x = a_1 y_1 z_1 + \cdots + a_k y_k z_k$

Ejemplo 5.1.12. Resolver el sistema de congruencias

$$\begin{aligned} x &\equiv 3 \pmod 7 \\ x &\equiv 5 \pmod{15} \\ x &\equiv 7 \pmod{13}. \end{aligned}$$

Solución. Se tiene que $z_1 = 195$, $z_2 = 91$, $z_3 = 105$. Sus inversos son $y_1 = 6$, $y_2 = 1$, $y_3 = 1$. La solución es $x = (3)(6)(195) + (5)(1)(91) + (7)(1)(105) = 4700 \equiv 605 \pmod{1365}$. Note que el módulo es el producto $7 \times 15 \times 13 = 1365$. Además $x = 605$ satisface el sistema

$$\begin{aligned} 605 &\equiv 3 \pmod 7 \\ 605 &\equiv 5 \pmod{15} \\ 605 &\equiv 7 \pmod{13}. \end{aligned}$$

5.1.8. El pequeño teorema de Fermat y el teorema de Euler

Dos de los teoremas más importantes en teoría de números fueron formulados por Pierre de Fermat en el siglo dieciséis y por Leonhard Euler en el siglo diecisiete. Aunque en su tiempo desconocido, ambos teoremas tienen aplicaciones prácticas en el área de criptografía.

Teorema 5.1.8 (Pequeño teorema de Fermat)**.** *Si p es un número primo y p no divide a, entonces*

$$a^{p-1} \equiv 1 \pmod p.$$

El pequeño teorema de Fermat es verdad para módulos primos. Una generalización se presenta en el teorema de Euler donde se considera para cualquier número n, donde $\phi(n)$ denota la función fi de Euler.

Teorema 5.1.9 (Teorema de Euler)**.** *Si el* $\text{mcd}(a, n) = 1$ *entonces*

$$a^{\phi(n)} \equiv 1 \pmod n.$$

Ejemplo 5.1.13. Calcular los últimos tres dígitos de 7^{803}.
Solución. Conocer los últimos tres dígitos equivale a trabajar módulo 1000. Dado que $\phi(1000) = 1000(1 - \frac{1}{2})(1 - \frac{1}{5}) = 400$ tenemos que $(7^{400})^2 7^3 \equiv 7^3 \equiv 343 \pmod{1000}$. Por lo tanto los últimos tres dígitos son 343. En este ejemplo es posible cambiar el exponente 803 por 3 debido a que $803 \equiv 3 \pmod{\phi(1000)}$. (Trappe y Washington, 2006)

Ejemplo 5.1.14. Calcular $2^{43210} \pmod{101}$.
Solución. El número 101 es primo, puede comprobarse con el pequeño teorema de Fermat $2^{100} \equiv 1 \pmod{101}$. Por el mismo tenemos que $2^{43210} \equiv (2^{100})^{432}2^{10} \equiv 1^{432}2^{10} \equiv 1024 \equiv 14 \pmod{101}$. En este caso es posible cambiar el exponente 43210 por 10 ya que $43210 \equiv 10 \pmod{\phi(101)}$ (Trappe y Washington, 2006).

Se mostró cómo comprobar si un número es primo con el pequeño teorema de Fermat, aunque existen algunas excepciones. Por ejemplo el número compuesto $561 = 3(11)(17)$ lo cumple ya que $2^{560} \equiv 1 \pmod{561}$ pero claramente no es primo. Estos números compuestos se conocen como *números de Carmichael* y es raros encontrarlos. No obstante, hay una alta probabilidad que el número sea primos si cumple con el pequeño teorema de Fermat.

5.2. Grupos y Campos

Los *grupos* y *campos* son estructuras algebraicas fundamentales en criptografía de clave pública. Una estructura algebraica es un conjunto con una o más operaciones entre sus elementos. Dependiendo del número de estas reciben un nombre. La estructura de *grupo* es la base del álgebra abstracta y la estructura más sencilla en esta disciplina. Existen muchos grupos pero solo algunos interesan para lograr implementar sistemas criptográficos. Ya se verá más adelante que los campos contienen grupos. De los campos podemos mencionar: el campo de los número reales $\mathbb{R}$ con las operaciones adición $+$ y multiplicación $\times$, el campo de los números complejos $\mathbb{C}$ o algún campo de elementos finitos F_p. Estos últimos son menos conocidos pero cumplen con ciertas propiedades especiales para emplearse en criptografía. Se presentan algunas definiciones y ejemplos sencillos que ayudarán a entender los conceptos antes de utilizar números grandes. Muchas de estas estructuras servirán para definir los sistemas criptográficos con curvas elípticas.

5.2.1. Relaciones de equivalencia

Considere el conjunto S formado por fracciones de la forma $\frac{m}{n}$ con $m, n \in \mathbb{Z}^+$. Es lógico decir que $\frac{8}{12}$ y $\frac{6}{9}$ son iguales. Es decir, ambos representan al número racional $\frac{2}{3}$. Más aún, las fracciones $\frac{4}{6}$,$\frac{10}{15}$ y $\frac{676}{1014}$ también representan a $\frac{2}{3}$. Esta es la idea central de una *relación de equivalencia*. Aunque todas estas fracciones tienen numerador y denominador distinto identifican el mismo elemento en $\mathbb{Q}$. Podemos decir que existe una *relación*, denotada por símbolo $\sim$, entre ellos. En el ejemplo anterior, cada fracción es un múltiplo de la forma $\frac{(k)m}{(k)n}$ para $k \in \mathbb{Z}^+$ con $m = 2$ y $n = 3$. Además, cada una de estas fracciones pertenece al conjunto S pues $km, kn \in \mathbb{Z}^+$. Por consiguiente se podría dividir el conjunto S en distintos compartimentos que contengan las representaciones de cada número racional $\mathbb{Q}$ y sus múltiplos. De esta manera cada elemento del conjunto pertenecería solo a una de estas particiones.

Teorema 5.2.1. *Una relación $\sim$ entre elementos de un conjunto no vacío satisface las propiedades siguientes:*

1. *Reflexividad, para toda $a \in S$ entonces $a \sim a$.*
2. *Simetría, sea $a, b \in S$ entonces $a \sim b$ equivale a tener $b \sim a$.*
3. *Transitividad, si $a \sim b$ y $b \sim c$ entonces $a \sim c$ para $a, b, c \in S$.*

Entonces la relación $\sim$ parte el conjunto S donde

$$[a] = \{x \in S \mid a \sim b\}$$

es el subconjunto con todos los elementos que se identifican como a. Así cada una de estas particiones $[a]$ es una relación que satisface las propiedades reflexiva, simétrica y transitiva con $a \sim x$ y $x \in [a]$.

Definición 5.2.1 (Relación de equivalencia)**.** Una relación entre dos elementos de un conjunto S que satisface las propiedades reflexiva, simétrica y transitiva del teorema 5.2.1 es una *relación de equivalencia*.

Definición 5.2.2. Una partición del conjunto S formada a partir de una relación de equivalencia es una *clase de equivalencia*. Cada elemento de S pertenece solo a una clase de equivalencia.

Ejemplo 5.2.1. Sea el conjunto $S = \mathbb{Z}$ y $m \in \mathbb{Z}^+$. Se define la relación módulo m entre sus elementos como en la sección 5.1.5 donde si m divide $a - b$ decimos que a es congruente con b módulo m. También, si m divide $b - c$ decimos que b es congruente con c módulo m. Para $a, b, c \in S$ se cumple:

1. La relación $a \sim a$ es reflexiva puesto que $a \equiv a \pmod{m}$.
2. Las relaciones $a \sim b$ y $b \sim a$ son simétricas pues $a \equiv b \pmod{m}$ y $b \equiv a \pmod{m}$.
3. Las relaciones $a \sim b$ y $b \sim c$ son transitivas pues $a \equiv b \pmod{m}$ y $b \equiv c \pmod{m}$ implica $a \equiv c \pmod{m}$.

Así, tenemos que la relación módulo m es una relación de equivalencia y parte el conjunto en $m - 1$ clases de equivalencia. Las clases en que se distribuyen los elementos del conjunto S son:

$$\begin{aligned}
[0] &= \{\ldots, -2m, -m, 0, m, 2m, \ldots\} \\
[1] &= \{\ldots, -2m+1, -m+1, 1, m+1, 2m+1, \ldots\} \\
&\vdots \\
[m-1] &= \{\ldots, -2m+(m-1), -m+(m-1), m-1, m+(m-1), 2m+(m-1), \ldots\}
\end{aligned}$$

Las clases de equivalencia formadas por la operación módulo m se conocen como *clases residuales módulo m*. Cada una de estas clases tiene un número infinito de elementos. Por ejemplo, si $m = 12$ las clases residuales módulo 12 son:

$$\begin{aligned}[0] &= \{\dots, -24, -12, 0, 12, 24, \dots\} \\ [1] &= \{\dots, -23, -11, 1, 13, 25, \dots\} \\ &\vdots \\ [11] &= \{\dots, -13, -1, 11, 23, 35, \dots\}\end{aligned}$$

5.2.2. Operación binaria

Aunque desde el inicio de la sección *Fundamentos* se ha sumado y multiplicado, ambos son casos de una *operación binaria*. Es conveniente definir esta generalización, pues nos ayudará a definir sumas y multiplicaciones poco habituales. En criptografía se trabaja con la operación suma y multiplicación módulo m pero en criptografía con curva elíptica se define una operación llamada suma muy distinta a las anteriores. De manera que se puede establecer una operación entre dos elementos de un conjunto siempre que coincida con la siguiente definición.

Definición 5.2.3 (Operación binaria)**.** Una operación binaria $*$ en un conjunto es una regla que asigna a cada par ordenado de elementos dentro del conjunto algún elemento del mismo conjunto (Herstein, 1980).

Ejemplo 5.2.2. En el conjunto de números enteros positivos $\mathbb{Z}^+$ definimos la operación $a*b$ que es igual al mínimo entre a y b o al valor común $a = b$. Así $2 * 11 = 2$, $15 * 10 = 10$, $3 * 3 = 3$. (Fraleigh, 1998).

Ejemplo 5.2.3. Definimos en el conjunto de enteros positivos $\mathbb{Z}^+$ la operación * como $a * b = a$. Así $2 * 3 = 2$, $25 * 10 = 25$, $5 * 5 = 5$.

Ejemplo 5.2.4. Definimos en el conjunto de enteros positivos $\mathbb{Z}^+$ la operación $*'$ como $a*'b = (a*b)+2$ donde * está definida como en el ejemplo 5.2.2. Así $4*7 = 6$, $25*9 = 11$ y $7 * 7 = 9$ (Fraleigh, 1998).

Es necesario recalcar que intercambiar el orden de los elementos respecto al operador puede alterar el resultado. Note la operación binaria del ejemplo 5.2.3. Es claro que $2 * 3 = 2$ no es igual a $3 * 2 = 3$. A pesar de operar los mismos elementos producen una salida distinta. Es decir, el resultado es afectado por el orden de los elementos respecto al operador. Esto es por como se formuló la operación. En cambio el ejemplo 5.2.2 devuelve el mismo resultado sin importar el orden.

En muchas ocasiones es necesario operar más de dos elementos y la definición se limita únicamente a dos. Pero esto se puede remediar. Suponga que se desean operar tres elementos a, b y c. Delimitamos con paréntesis las combinaciones posibles para realizar las operaciones binarias $(a * b) * c$ o $a * (b * c)$. Si el orden no afecta el resultado de la operación tenemos que

$$(a * b) * c = a * (b * c).$$

Una operación binaria que cumple esta igualdad es dada en el ejemplo 5.2.2. En cambio, si definimos la operación * como en el ejemplo 5.2.4 la igualdad es falsa. Por ejemplo, operar

$$(7 * 25) * 4 = 6 \neq 8 = 7 * (25 * 4)$$

y por lo tanto la igualdad no tendría ningún sentido. Estas propiedades de las operaciones binarias generan algunos grupos aptos para criptografía.

Definición 5.2.4. Una operación binaria $*$ en un conjunto S es *conmutativa* si y solo si $a*b = b*a$ para todo $a, b \in S$. La operación $*$ es *asociativa* si y solo si $(a*b)*c = a*(b*c)$ para todo $a, b, c \in S$ (Fraleigh, 1998).

En cualquier conjunto se puede definir una operación binaria *. Al definirlas se debe tener cuidado que cada elemento a se asigne a un par de elementos del conjunto. También para cada par ordenado de elementos del conjunto se asigna un elemento dentro del mismo conjunto. Una vez definido el concepto de operación binaria se procede a definir una estructura algebraica importante en criptografía de clave pública, los grupos.

5.2.3. Grupos

Grupo

De forma sencilla, un grupo es un conjunto con una operación definida entre sus elementos. Así, por ejemplo la operaciones suma + en los reales $\mathbb{R}$ es un grupo. No solo existe este grupo sino que se puede establecer cualquier operación binaria entre un par de elementos, cualesquiera de un conjunto S, si es consistente con la siguiente definición.

Definición 5.2.5 (Grupo)**.** Un *grupo* $\langle G, * \rangle$ es un conjunto no vacío G junto con una operación binaria * en G tal que:

1. La operación es **cerrada**, es decir $a, b \in G$ implica que $a * b \in G$.
2. La operación es **asociativa**, es decir $a, b, c \in G$ implica que $(a * b) * c = a * (b * c)$.
3. Un **elemento neutro** $e \in G$ existe en el conjunto tal que $a * e = e * a = a$ para todo $a \in G$.
4. Para todo $a \in G$ existe un **elemento inverso** a^{-1} tal que $a * a^{-1} = a^{-1} * a = e$.

Antes de continuar conviene subrayar la notación a usar. Siempre que no cause confusión y para agilizar la lectura se sustituye la notación de grupo $\langle G, * \rangle$ simplemente por G. A menos que se indique lo contrario, se asumirá como operación binaria la multiplicación. Se prefiere la notación multiplicativa ab en lugar de $a \times b$. Las definiciones, teoremas, corolarios y lemas redactados son igualmente validos para operaciones distintas a la multiplicación. Por ejemplo la suma.

Así mismo, al referirnos a la multiplicación $aa = a^2$ podemos utilizar la notación de potencias. Su equivalente con la operación suma es $n + n = 2n$. En general, para

$$\underbrace{n \times n \times \cdots \times n}_{k \text{ multiplicaciones}} = n^k$$

su equivalente en suma es

$$\underbrace{n + n + \cdots + n}_{k \text{ sumas}} = kn.$$

Recuerde que k sumas se puede anotar como una multiplicación de n por el escalar k.

Es importante señalar que el elemento neutro de un grupo es único. También el inverso es único. Estas características de un grupo se anotan en el siguiente lema.

Lema 5.2.2. *Si G es un grupo entonces:*

1. *El elemento identidad de G es único.*
2. *Todo elemento $a \in G$ tiene un inverso único.*
3. *Para todo $a \in G$ se tiene $(a^{-1})^{-1} = a$.*
4. *Para todo $a, b \in G$ se tiene $(ab)^{-1} = b^{-1}a^{-1}$.*

Por otro lado se tiene la *ley de cancelación en grupos*, la cual nos permite despejar variables de una ecuación. Esta ley es similar a lo expuesto en la sección 5.1.5. Solamente se puede dividir a los elementos primos relativos a m. Estos forman un grupo bajo la multiplicación módulo m.

Lema 5.2.3. *Sea G un grupo y $a, b, c \in G$. Si $ac = bc$ entonces $a = b$. En particular si $ac = a$ entonces c es el elemento neutro de G.*

En criptografía de clave pública se tiene especial interés por los grupos finitos y conmutativos. A continuación se definen estos grupos fundamentale para los criptosistemas de clave pública basados en *el problema del logaritmo discreto.*

Definición 5.2.6 (Grupo abeliano)**.** Se dice que un grupo G es *abeliano* o conmutativo si para cualesquiera $a, b \in G$ se tiene que $ab = ba$.

Con esto se quiere decir que el grupo es abeliano si la operación es conmutativa. Como veremos más adelante, estos grupos están estrechamente relacionados con un tipo especial de grupos finitos. Los grupos cíclicos.

El número de elementos es otra característica importante de un grupo. En criptografía, es importante trabajar en grupos finitos con muchos elementos. Aunque el uso de las palabras "grande" o "muchos" podría resultar ambiguo e inapropiado para un texto académico, en criptografía tienen el objetivo de indicar números o elementos de un conjuntos tan inmensurables como el número de partículas en el universo. Si se escriben ocupan entre dos o tres renglones y entorpecen la lectura. Por lo tanto se prefieren estos adjetivos.

Definición 5.2.7 (Orden del grupo)**.** Si el número de elementos de un grupo G es contable decimos que G es un *grupo finito*. El número de elementos de un grupo finito se llama *orden* y se denota por $|G|$. Si el grupo G tiene infinitos elementos decimos que G es un *grupo infinito*.

Definición 5.2.8 (Orden de un elemento)**.** Si G es un grupo y $a \in G$, definimos el *orden* del elemento a como el entero positivo mínimo n tal que $a^n = e$ y se denota por $\text{ord}(a)$.

Ejemplo 5.2.5. Considere el conjunto finito S que contiene solamente al elemento e. Es fácil ver que e con la operación * forma un grupo: existe elemento neutro, es cerrado bajo la operación asociativa * y él mismo actúa como su inverso. Además es abeliano. Este es el *grupo trivial* y su orden es $|G| = 1$.

Ejemplo 5.2.6. Supongamos que G consiste en los números reales 1 y -1 con la multiplicación entre números reales como operación. Entonces $\langle G, \times \rangle$ es un grupo abeliano de orden $|G| = 2$ pues

$$1 \times 1 = 1, \quad 1 \times -1 = -1, \quad -1 \times -1 = 1.$$

Subgrupo

Definición 5.2.9 (Subgrupo)**.** Un subconjunto H de G se dice que es un *subgrupo* de G respecto a la operación * en G. Por lo tanto H es un grupo $\langle H, * \rangle$.

Verificar que un subconjunto de G es un subgrupo requiere comprobar dos condiciones: la propiedad de cerradura bajo la operación del grupo y la existencia de inversos. Más aún, solo la primer condición es necesaria para identificar si un subconjunto finito de un grupo es subgrupo. Ambos se define en los siguientes lemas.

Lema 5.2.4. *Un subconjunto no vacío H del grupo G es subgrupo si y solo si:*

1. *Es cerrado, es decir $a, b \in H$ implica que $ab \in H$.*
2. *Hay inverso para cualquier elemento, es decir $a \in H$ implica que $a^{-1} \in H$.*

El siguiente teorema es de gran relevancia en criptografía de clave pública. Se formuló en 1770 por Lagrange (Herstein, 1980) y es un resultado importante en criptografía tanto para diseñar criptosistemas como para llevar a cabo criptoanálisis. Establece una relación peculiar entre el orden de un grupo finito y el orden de un subgrupo. Los corolarios 5.2.7 al 5.2.9 se siguen de este teorema.

Teorema 5.2.5 (Teorema de Lagrange)**.** *Si G es un grupo finito y H es un subgrupo de G entonces el orden del subgrupo $|H|$ divide al orden del grupo $|G|$.*

La criptografía de clave pública se interesa por grupos finitos, en particular por *grupos cíclicos*. Los grupos cíclicos son una rama de los grupos finitos. Presentan propiedades particulares que permiten emplearlos en criptografía. Aunque existen otro tipo de grupos finitos, las siguientes páginas se limita únicamente a este tipo.

Grupo cíclico

Sea G un grupo finito, el orden del grupo $m = |G|$ y un elemento a distinto al elemento neutro e. Se continua con la notación multiplicativa por fines didácticos pero es válido para cualquier otra operación. Multiplicar a por si mismo, esto es $aa = a^2$, genera el elemento $a^2 \in G$. Este nuevo elemento pertenece al grupo G por la propiedad de cerradura. También $aa^2 = a^3 \in G$, $aa^3 = a^4 \in G$ y así hasta $a^m = e$. En otras palabras, se puede generar el grupo G a partir del elemento a. Al elemento a que operado consigo mismo genera el grupo se conoce como *generador*. Si tal elemento existe se dice que el grupo G es *cíclicamente generado por a*.

Definición 5.2.10 (Grupo cíclico)**.** Un grupo es cíclico si para cualquier elemento $b \in G$ existe un elemento $a \in G$ y $n \in \mathbb{Z}$ tal que $a^n = b$. Tal elemento a genera G y es un generador de G. Lo denotamos como $\langle a \rangle = G$ (Lidl y Niederreiter, 1994).

Teorema 5.2.6. *Todo grupo cíclico es abeliano (Fraleigh, 1998).*

Una peculiaridad de los grupos cíclicos es que su operación es conmutativa. Por lo tanto, todos los grupo cíclicos son abelianos. Esto es lo que define el teorema anterior.

Puede existir más de un elemento generador del grupo G. A excepción del grupo trivial, es obvio que e no puede ser un elemento generador de G. De ahí que G no pueda ser generado por todos sus elementos. Los siguientes corolarios procedcen del teorema de Lagrange e indican posibles candidatos a ser generadores del grupo.

Corolario 5.2.7. *Si G es un grupo finito y $a \in G$ entonces* $\text{ord}(a)$ *divide a* $|G|$.

El corolario anterior nos da una idea sobre todos los ordenes de cada elemento del grupo, son divisores de $|G|$. Una condición importante para ser generador es que deben dividir al $|G|$.

Corolario 5.2.8. *Si G es un grupo finito y $a \in G$ entonces $a^{|G|} = e$.*

Multiplicar cualquier elemento $|G|$ veces es igual al elemento neutro. Por lo tanto el orden de cualquier elemento a no es mayor a $|G|$.

Corolario 5.2.9. *Si G es un grupo finito cuyo orden es un número primo entonces G es un grupo cíclico. Además todos sus elementos con excepción del neutro son generadores de G.*

En criptografía se trabaja con grupos de orden primo. Por el corolario 5.2.9 encontrar un generador del grupo es trivial (Katz y Lindell, 2021). Además, se cree que el problema del logaritmo discreto es difícil en estos grupos.

Definición 5.2.11. Sea $S = \{[0], [1], \ldots, [m-1]\}$ el conjunto de clases residual módulo m con la operación suma módulo m entre sus elementos. Entonces $\langle S, + \rangle$ es un grupo conocido como el *grupo de enteros módulo m* y se denota $\mathbb{Z}_m^+$ (Lidl y Niederreiter, 1994).

Ejemplo 5.2.7. Considere el conjunto S del ejemplo 5.1.7 bajo la operación suma módulo 7. La tabla 5.1 muestra la operación de suma bajo el conjunto S. Esta es conocida como *tabla de Cayley* y contiene todas las operaciones posibles dentro del grupo.

$+$	0	1	2	3	4	5	6
0	0	1	2	3	4	5	6
1	1	2	3	4	5	6	0
2	2	3	4	5	6	0	1
3	3	4	5	6	0	1	2
4	4	5	6	0	1	2	3
5	5	6	0	1	2	3	4
6	6	0	1	2	3	4	5

Tabla 5.1: Tabla de Cayley

Este conjunto con la operación suma módulo siete es el grupo finito abeliano $\mathbb{Z}_7^+$. Este tipo de grupo también se conoce como *grupo aditivo*. Su elemento neutro es el cero. Para encontrar el inverso de un elemento basta con intersecar la fila y columna en cero. Por ejemplo, el inverso del elemento 3 es 4 porque su intersección en la tabla es 0 y $3 + 4 \equiv 0 \pmod 7$. Además, cualquier elemento excepto el 0 es un generador del grupo. En especial 1 ya que $\langle 1 \rangle = \{1, 2, 3, 4, 5, 6, 0\}$.

Definición 5.2.12. Sea $S = \{[1], [2], \ldots, [m-1]\}$ el conjunto de todas las clases residuales módulo m que son primos relativos de m con la operación multiplicación módulo m. Entonces $\langle S, \times \rangle$ es un grupo conocido como *grupo multiplicativo de enteros módulo* m y se denota por $\mathbb{Z}_p^{\times}$.

A diferencia del grupo de enteros módulo m, se excluye la clase residual cero. La razón de su omisión es la operación multiplicación módulo m. En este grupo el elemento cero no tendría inverso ya que $0x \neq 1$.

Teorema 5.2.10. *Si p es primo entonces $\mathbb{Z}_p^{\times}$ es un grupo cíclico de orden $p-1$.*

Ejemplo 5.2.8. Considere del ejemplo 5.2.7 todas las clases residuales que son primos relativos a siete. Se tiene el conjunto $S = \{[1], [2], \ldots, [m-1]\}$ bajo la operación multiplicación módulo siete. Entonces $\langle S, \times \rangle$ se conoce como el *grupo multiplicativo* $\mathbb{Z}_7^{\times}$. La Tabla 5.2 muestra esta operación bajo el conjunto S. El elemento neutro del grupo multiplicativo es 1. En este caso los valores que se intersecan en 1 son inversos. Por ejemplo, el inverso del elemento 6 es él mismo. Observe que 2 no es generador del grupo pues $\langle 2 \rangle = \{2, 4, 1\}$. En cambio 3 es un generador porque $\langle 3 \rangle = \{3, 2, 6, 4, 5, 1\}$.

$\times$	1	2	3	4	5	6
1	1	2	3	4	5	6
2	2	4	6	1	3	5
3	3	6	2	5	1	4
4	4	1	5	2	6	3
5	5	3	1	6	4	2
6	6	5	4	3	2	1

Tabla 5.2: Tabla de Cayley para multiplicación.

Subgrupo cíclico

En criptosistemas que dependen del problema del logaritmo discreto se trabaja con subgrupos finitos. Se conjetura que en ellos el problema de decisión Diffie-Hellman es difícil. Un subgrupo es un subconjunto H de G tal que $H \leq G$ con la misma operación definida en G.

Definición 5.2.13. Sea G un grupo y sea $a \in G$. Entonces

$$H = \{a^n \mid n \in \mathbb{Z}\}$$

es un subgrupo de G y es el menor subgrupo que contiene a. Es decir, si otro subgrupo B contiene a entonces $H \subseteq B$.

Teorema 5.2.11. *Todo subgrupo de un grupo cíclico es cíclico (Fraleigh, 1998).*

En caso de que $p > 3$ todos los grupos $\mathbb{Z}_p^\times$ tienen un número compuesto por orden. Por esta razón se trabaja con subgrupos cíclicos de orden primo. El siguiente teorema asegura la existencia de este subgrupo.

Teorema 5.2.12. *Sea $p = rq + 1$ con p, q números primos. Entonces*

$$\mathbb{G} = \{[h^r \pmod p \mid h \in \mathbb{Z}_p^\times]\}$$

es un subgrupo de $\mathbb{Z}_p^\times$ de orden primo q (Katz y Lindell, 2021).

El subgrupo cíclico $\mathbb{G}$ tiene orden primo. La variable r se conoce como el *cofactor*. En la práctica se tienen estandarizados varios valores de p y q. Se conjetura que resolver el problema del logaritmo discreto para p grande en este subgrupo es difícil.

Ejemplo 5.2.9. Nuevamente se considera el grupo $\mathbb{Z}_7^\times$. El orden del grupo es 6 pero existe un subgrupo de orden primo con $q = 3$ pues $7 = (2)3 + 1$. En realidad hay dos subgrupos de orden 3. Los generadores de ambos son $\langle 2 \rangle = \{2, 4, 1\}$ y $\langle 4 \rangle = \{4, 2, 1\}$.

5.2.4. Campos

Definición 5.2.14. Un *campo* es un conjunto F con dos operaciones binarias, comúnmente llamadas adición y multiplicación, junto con dos elementos distintos: el elemento 0 tal que $a + 0 = a$ para todo $a \in F$ y el elemento unitario 1 tal que $a \times 1 = a$.

Un campo F forma un grupo abeliano respecto a la suma. Si se excluye el cero entonces se tiene un grupo multiplicativo abeliano. En pocas palabras, un campo está formado por un grupo aditivo y un grupo multiplicativo.

Ejemplo 5.2.10. Los números reales $\mathbb{R}$ con la operación suma y multiplicación forman un campo. La identidad multiplicativa es 1 y el neutro aditivo es 0. Ambas operaciones son conmutativas. El único elemento sin inverso multiplicativo es 0. Este campo es infinito.

Campos finitos

Sea p un número primo. Un campo finito con q elementos existe si y solo si q es una potencia prima, es decir $q = p^m$ con m un entero positivo. Al valor p se le denomina característica del campo y m extensión del campo.

En criptografía se trabaja sobre campos finitos donde el número de elementos del conjunto F es contable. Se denota por F_p al campo finito con $p - 1$ elementos con las operaciones suma y multiplicación módulo p. Al igual que los grupos, se estudian los campos finitos de orden primo por sus aplicaciones en criptografía.

Ejemplo 5.2.11. Considere $S = \{[0], [1], [2], [3], [4], [5], [6]\}$ el conjunto de las clases residuales módulo 7 del ejemplo 5.1.7 y las operaciones suma y multiplicación. Hemos visto en el ejemplo 5.2.7 que este conjunto es un grupo aditivo con 0 el elemento neutro. Además, en el ejemplo 5.2.8 se excluyó al 0 para generar un grupo multiplicativo con elemento neutro o unitario 1. Entonces, el conjunto con ambas operaciones es el campo finito F_7.

5.3. Criptosistema RSA

En 1978 se publicó la primer implementación práctica de un criptosistema de clave pública por Rivest, Shamir y Adleman; mejor conocido como criptosistema RSA o simplemente RSA (Rivest y col., 1978). A pesar de tener más de 40 años en uso, el criptosistema RSA continua vigente.

5.3.1. Generación de claves

El primer paso para utilizar RSA es calcular el valor de n. Sean p, q pseudoprimos de 200 dígitos o más. Se calcula el valor del módulo

$$n = p\,q.$$

Con este valor se realizarán operaciones aritméticas modulares $\pmod{n}$.

Después calculamos el exponente público e tal que

$$\text{mcd}(e, \phi(n)) = 1.$$

Usualmente e es la cuarto primo de Fermat $F_4 = 2^{2^4} + 1 = 65\,537$. La clave pública es la dupla (e, n).

Luego se calcula el exponente privado d tal que

$$d\,e \equiv 1 \pmod{\phi(n)}.$$

El valor del exponente privado se obtiene con el algoritmo de Euclides extendido. La clave privada es el par (d, n). La complejidad para obtener las claves es polinomial.

La seguridad del sistema depende de mantener en secreto los valores p, q y d. Se conjetura que calcular $\phi(n)$ o el exponente privado d a partir de la clave pública es difícil; es un problema abierto de complejidad NP $\cap$ co-NP (Garey y Johnson, 1990). Una vez que se obtuvo el módulo n pueden eliminarse los valores de p y q, pero d debe resguardarse y evitar se comparta con alguien.

5.3.2. Esquema de firma digital RSA

Al igual que todos los esquema de firma digital, el esquema de firma RSA consiste de dos procesos: la firma y verificación. Su seguridad depende de resolver el problema de la factorización. Si los parámetros seleccionados producen un módulo de 2048 bits entonces el tiempo para solucionarlo es tan largo que obliga a desistir.

Firma

Considere la Figura 1.1 donde Bob solicita a Alicia firmar un documento M. Ella debe utilizar el esquema de firma digital RSA del Algoritmo 4 para genera la firma s del mensaje M. La complejidad de este algoritmo es polinomial. En general, producir un *hash* tiene complejidad lineal $O(n)$. La exponenciación modular del paso 3 tiene complejidad $O(\log d \log \phi(n)^2)$.

Algoritmo 4: Algoritmo de firma RSA

Datos: Clave privada (d, n) y mensaje M a firmar.
Resultado: Firma digital s del mensaje M
1 Obtener el resumen del mensaje $m \leftarrow \mathcal{H}(M)$ donde $\mathcal{H}$ es una función resumen;
2 Convertir m a un entero tal que $m < \phi(n)$;
3 **devolver** *La firma* $s \leftarrow m^d \pmod{\phi(n)}$

Verificación

La *verificación con anexo de una firma digital* requiere enviar el mensaje M y su respectiva firma digital. El Algoritmo 5 valida que la firma s fue producida a partir del mensaje M y la clave pública e de Alicia. Por lo tanto, Alicia y nadie más firmo el documento solicitado por Bob. De manera similar, la complejidad de este algoritmo es polinomial.

Algoritmo 5: Algoritmo de verificación RSA

Datos: Clave pública (e, n) y mensaje M junto con su firma digital s
Resultado: Aceptar o rechazar firma

```
si 0 < s < φ(n) entonces
    Calcular m' ← s^e (mod φ(n));
    Calcular m ← H(M);
    si m' = m entonces
        Aceptar firma;
    en otro caso
        Rechazar firma;
    fin
en otro caso
    devolver Rechazar firma
fin
```

A continuación se demuestra que el mensaje M fue firmado por la clave privada (d, n) ya que

$$d\,e \equiv 1 \pmod{\phi(n)},$$

por lo tanto

$$s^e \equiv m^{de} \equiv m' \equiv m \pmod{\phi(n)}.$$

Si la función resumen $\mathcal{H}$ es de una sola dirección y además resistente a colisiones, un estafador no puede agregar la firma s a otro mensaje M_1 ya que

$$\mathcal{H}(M) = m \neq m_1 = \mathcal{H}(M_1)$$

y el Algoritmo 5 devuelve rechazar.

La verificación con anexo del esquema de firma digital RSA se define en el estándar PKCS#1 (Moriarty y col., 2016). Este estándar es empleado por el SAT para comprobar que la factura expedida fue realizada por un contribuyente inscrito en el RFC.

5.3.3. Seguridad del criptosistema RSA

El algoritmo de la *criba general de campos numérico* (GNFS, *Genral Number Field Sieve*) es el mejor método para resolver el problema de la factorización entera (Fúster Sabater y col., 2012) (Leyland, 2005). Su complejidad en notación L (Lenstra, 2005) es

$$L[\tfrac{1}{3}, c] = O\left(e^{(c+o(1))(\ln n)^{\frac{1}{3}}(\ln\ln n)^{\frac{2}{3}}}\right)$$

donde la constante $c = (\frac{64}{9})^{\frac{1}{3}} \approx 1{,}923$ y $o(1)$ denota una función que converge a cero cuando $n \to \infty$.

El récord actual del GNFS es la factorización de RSA-250. Este número es de 795 bits de longitud y 250 cifras. Esto fue logrado en el año 2020 por Boudot, Gaudry, Guillevic, Heninger, Thomé y Zimmermann (Boudot y col., 2020). La anterior marca fue RSA-768 con 232 dígitos y 768 bits (Kleinjung y col., 2010). Es por esto que el tamaño del módulo y las claves aumenten para garantizar que el RSA sea seguro.

Sea b el nivel de seguridad en bits para el criptosistema RSA. El GNFS implica elegir el módulo n con un tamaño de al menos $(0{,}5 + o(1))b^2/\log b$ bits (Bernstein, 2009). Si $b = 112$ entonces $n \approx 1330$ bits. No obstante, establecer el tamaño del módulo a partir del nivel de seguridad requiere de un análisis más detallado. Está ampliamente aceptado que el módulo n conste de 2048 bits para $b = 112$ y en este trabajo se define de la misma forma (Barker, 2020).

5.4. Criptosistema de curva elíptica: ECC

5.4.1. Curvas elípticas

Definimos una curva elíptica E sobre un campo F como el conjunto de puntos (x, y) que satisfacen la ecuación

$$y^2 = x^3 + ax^2 + b \tag{5.1}$$

junto con un punto especial denominado *punto al infinito* denotado por $\mathcal{O}$. Esta ecuación se conoce como la *ecuación de Weierstrass*. Tanto a, b y x son elementos del campo F. Para visualizarla se considera la curva E sobre el campo de los reales $\mathbb{R}$. Además, sus coeficientes deben satisfacer el determinante $4a^3 + 27b^2 \neq 0$.

Definición 5.4.1. Sea F un campo y $a, b \in F$. Definimos una curva elíptica sobre el campo F como el conjunto de puntos que satisfacen la Ecuación 5.1 junto con un punto especial denominado *punto al infinito*

$$E(F) = \{\mathcal{O}\} \cup \{(x, y) \mid y^2 = x^3 + ax^2 + b\}.$$

La Figura 5.1 muestra cinco ejemplos de curvas elípticas en el campo de los reales $\mathbb{R}$. Los tres primeros casos son polinomios cúbicos con solución única mientras que los dos restantes tienen tres soluciones. El determinante $4a^3 + 27b^2 \neq 0$ evita el caso con raíces repetidas.

De forma práctica, se puede visualizar el punto al infinito $\mathcal{O}$ en el plano como la unión de la parte superior e inferior del eje y detrás de una hoja. Geométricamente, el punto al infinito $\mathcal{O} = [0, 1, 0]$ es un punto del plano proyectivo (Washington, 2008). No es necesario entrar en los detalles de su representación, más bien se considerará como un elemento que cumple ciertas propiedades; es el neutro aditivo del grupo formado por

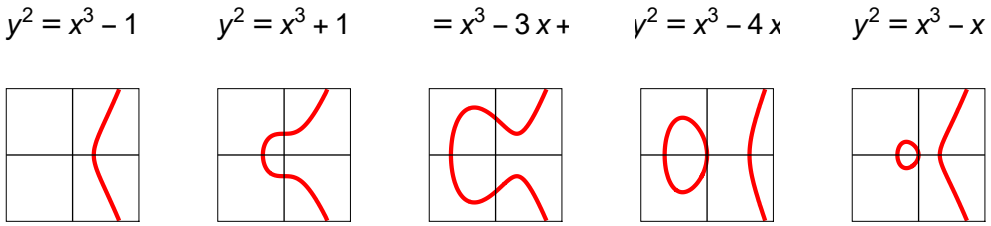

Figura 5.1: Cinco curvas elípticas en $\mathbb{R}$. Fuente: WolframMathWorld (Weisstein, 2009)

una curva elíptica.

La suma de dos puntos de una curva elíptica E definida en $\mathbb{R}$ puede obtenerse gráficamente mediante el *método de la cuerda y la tangente*. Sean $P = (x_P, y_P), Q = (x_Q, y_Q)$ puntos sobre E. Si $x_P \neq x_Q$ entonces el primer paso consiste en trazar una línea recta entre ambos puntos con pendiente

$$m = \frac{y_Q - y_P}{x_Q - x_P}.$$

La recta

$$y = m(x - x_p) + y_p$$

cortará la curva E en un tercer punto que denotamos por $-R$. Este punto se obtiene al sustituir en la ecuación de la curva la ecuación de la recta

$$(m(x - x_p) + y_p)^2 = x^3 + ax + b.$$

Simplificando

$$0 = x^3 - m^2x^2 + \dots.$$

Las tres raíces de este polinomio grado 3 corresponden a los tres puntos de la recta que se interceptan con E. Ya se conocen dos soluciones, las coordenadas x de los puntos P y Q. Hay que mencionar, además, que la suma de las tres raíces es igual al negativo del coeficiente cuadrático

$$x_P + x_Q + x_R = m^2.$$

Por consiguiente podemos calcular

$$x_R = m^2 - x_P - x_Q$$

y

$$-y_R = m(x_R - x_P) + y_P$$

Ahora reflejamos $-R = (x_R, -y_R)$ con respecto al eje x para obtener el punto $R = P + Q = (x_R, y_R)$ tal que

$$x_R = m^2 - x_P - x_Q, \qquad y_R = m(x_P - x_R) - y_P.$$

Lo dicho hasta aquí supone una suma distinta a la usual con números reales. En otras palabras, se ha definido una operación binaria y el símbolo de adición + para denotarla. Un ejemplo gráfico de suma de puntos con el método de la cuerda puede verse en la Figura 5.2.

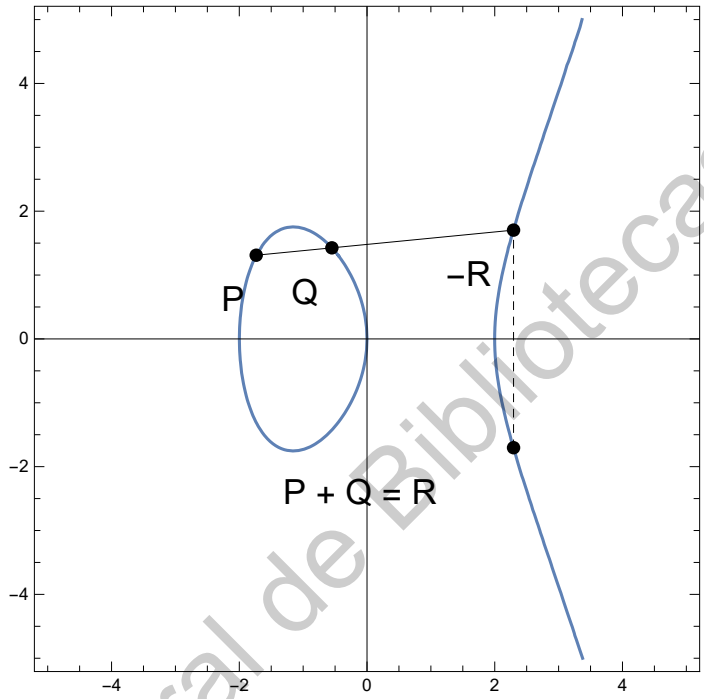

Figura 5.2: Ejemplo de suma de puntos P y Q con el método de la cuerda y la tangente.

Ejemplo 5.4.1. Considere la curva elíptica $y^2 = x(x+1)(2x+1)/6$ definida sobre los reales $\mathbb{R}$. Sume los puntos $(\frac{1}{2}, -\frac{1}{2})$ y $(1, 1)$ sobre la curva.

La pendiente entre ambos puntos es

$$m = \frac{1 + \frac{1}{2}}{1 - \frac{1}{2}} = 3.$$

Por lo tanto la ecuación de la recta entre ambos puntos es $y = 3x - 2$. Sustituyendo en la ecuación de la curva tenemos

$$(3x - 2)^2 = \frac{x(x+1)(2x+1)}{6}$$

Simplificando

$$0 = x^3 - \frac{51}{2}x^2 + \dots$$

Al tener las raíces $\frac{1}{2}$ y 1 se llega a

$$\frac{1}{2} + 1 + x = \frac{51}{2}$$

y se encuentra $x = 24$. Finalmente, la coordenada $y = 3x - 2 = 70$. Reflejando se tiene el punto $R = (24, -70)$.

Sumar dos puntos que tienen la misma coordenada en x pero diferente ordenada en y producirá una recta vertical que, aparentemente, no corta a la curva en un tercer punto. La recta cortará la curva en $\mathcal{O}$. Podemos decir entonces que $Q = -P$. Por lo tanto $P - P = \mathcal{O}$. Esta recta corta la curva en el punto al infinito.

En tercer lugar, si la operación a realizar es el duplicado de un punto, es decir sumar consigo mismo un punto, se traza la tangente de la curva E que pasa por el punto P. La pendiente puede obtenerse derivando la ecuación de la curva

$$2y\frac{dx}{dy} = 3x^2 + a \quad \text{y} \quad m = \frac{dy}{dx} = \frac{3x^2 + a}{2y}.$$

Si la pendiente es cero, se tiene que $P + P = \mathcal{O}$. Salvo este caso, la recta cortará la curva E en un tercer punto denotado por $-R = (x_R, -y_R)$. En este caso únicamente se conoce una raíz, pero es una raíz duplicada. De manera que

$$x_R = m^2 - 2x_P \qquad y_R = m(x_P - x_R) - y_P.$$

Es decir, $P + P = 2P = R$ es el punto simétrico con respecto al eje X de $-R$. Un ejemplo de *doblado de punto* puede verse en la Figura 5.3.

Finalmente se supone que $Q = \mathcal{O}$. La recta entre los puntos P y $Q = \mathcal{O}$ corta la curva en $-P$. Reflejarlo respecto al eje x resulta nuevamente en el punto P. Por lo tanto

$$P + \mathcal{O} = P.$$

En resumen, se define la operación suma de puntos $(+)$ de la siguiente manera:

1. Si $P = Q$ y $y_P = 0$ entonces $P + Q = \mathcal{O}$. Además $P + \mathcal{O} = \mathcal{O} + P = P$, es decir, $\mathcal{O}$ actúa como neutro aditivo.
2. Dado un punto P definimos $-P = (x_P, -y_P)$, de manera que $P - P = \mathcal{O}$.
3. Dados dos puntos P y Q tal que $x_P \neq x_Q$, entonces la suma de dos puntos se define como $R = P + Q = (x_R, y_R)$ siendo

$$\begin{aligned} x_R &= m^2 + a_1 - x_P - x_Q, \\ y_R &= m(x_P - x_R) - y_P, \\ m &= \frac{y_Q - y_P}{x_Q - x_P}. \end{aligned}$$

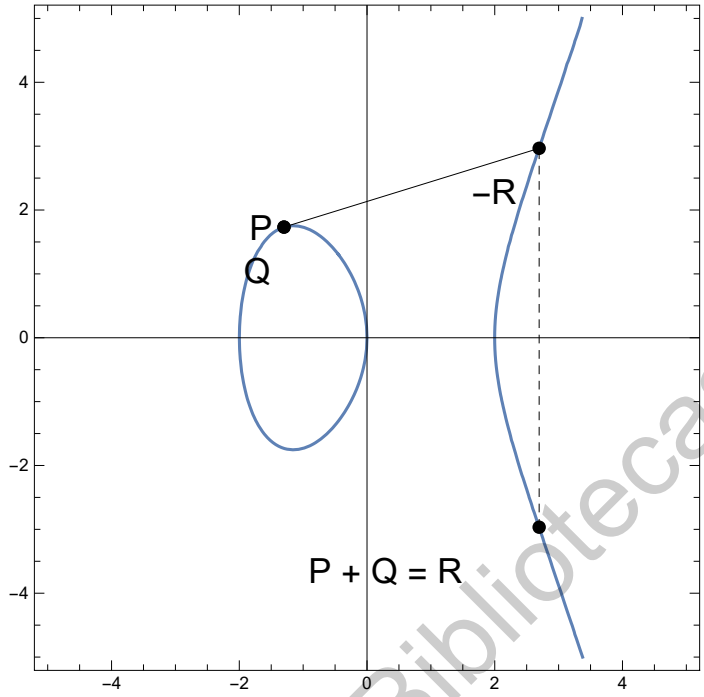

Figura 5.3: Ejemplo de doblado de punto cuando $Q = P$ con el método de la cuerda y la tangente.

4. Dado un punto P, la suma de este punto por sí mismo $R = P + P = 2P$ se denomina *duplicado de un punto*. Los valores de sus coordenadas son

$$\begin{aligned} x_R &= m^2 + a_1 m - 2x_P, \\ y_R &= m(x_P - x_R) - y_P, \\ m &= \frac{3x_P^2 + a}{2y_P}. \end{aligned}$$

Está claro que la suma $P + Q$ produce otro punto R sobre la curva E. De ahí que la operación suma de puntos es *cerrada*. En la siguiente sección comprobaremos que el conjunto de puntos de una curva elíptica E bajo la operación suma es un grupo aditivo abeliano.

De los casos anteriores definimos el *producto de un punto por un escalar*

$$kP = \underbrace{P + \cdots + P}_{k \text{ sumandos}}.$$

Por desgracia sumar k veces el punto P consigo mismo resulta ineficiente. En su lugar se calcula $2P$ y se *duplica* para sumarlo consigo mismo. Por ejemplo, calcular $19P$ por el método de duplicado es calcular

$$2P, \quad 4P = 2P + 2P, \quad 8P = 4P + 4P, \quad 16 = 8P + 8P, \quad 19 = 16P + 2P + P.$$

En la primer iteración se realiza un doblado de punto. Después se repite el método de la tangente y finalmente realizamos una adición de puntos con el método de la cuerda. Existen algoritmos que permiten un calculo eficiente evitando realizar k sumas.

Podemos precisar esta idea con el Algoritmo 6 (Washington, 2008). Se debe tener cuidado en $\mathbb{R}$ cuando k es grande dado que las coordenadas crecen considerablemente cada iteración. Para grupos finitos las coordenadas se reducen módulo p y se evita un desbordamiento de memoria. Su complejidad es $O(\log_2 k)$ operaciones de duplicado y sumas de puntos. En general su complejidad es polinomial. Por esto, los campos finitos se prefieren para criptografía. Además, para grupos finitos grandes encontrar el escalar k dado kP es difícil. Este problema se conoce como *el problema del logaritmo discreto en curvas elípticas*. De este depende la seguridad de los criptosistemas con curvas elípticas.

Algoritmo 6: Multiplicación de un punto por un escalar

Datos: k un entero positivo y P un punto sobre una curva elíptica
Resultado: El punto $B \leftarrow kP$

1 Inicie con $a \leftarrow k, \quad B \leftarrow \mathcal{O}, \quad C \leftarrow P$;
2 **si** *a es par* **entonces**
3 | Asigne $a \leftarrow a/2$, $B \leftarrow B$ y $C \leftarrow 2C$;
4 **si** *a es impar* **entonces**
5 | Asigne $a \leftarrow a - 1$, $B \leftarrow B + C$ y $C \leftarrow C$;
6 **si** $a \neq 0$ **entonces**
7 | Ir al paso 2;
8 **devolver** B

5.4.2. Estructura de grupo aditivo

Se vio en la sección 5.2 la estructura de grupo. Aunque utilizamos la notación multiplicativa, la definición 5.2.5 de grupo aplica para cualquier operación definida entre elementos de un conjunto que satisface las propiedades de cerradura, asociatividad, existencia de inversos y elemento neutro . Algo semejante ocurre con la operación suma de puntos de una curva elíptica sobre los reales $\mathbb{R}$.

Teorema 5.4.1. *La adición de puntos de una curva elíptica E satisface las siguientes propiedades:*

1. *La operación es **cerrada**, es decir $P, Q \in E$ implica que $P + Q \in E$.*
2. *El punto $\mathcal{O}$ es el elemento neutro tal que $P + \mathcal{O} = P$ para todo $P \in E$.*
3. *Todo elemento $P \in E$ tiene **inverso** $-P$ tal que $P + (-P) = \mathcal{O}$.*
4. *La operación es **asociativa**, es decir $(P + Q) + R = P + (Q + R)$ para todo $P, Q, R \in E$.*

5. La operación es ***conmutativa****, es decir* $P + Q = Q + P$ *para todo* $P, Q \in E$.

De modo que los puntos de una curva elíptica junto con el punto al infinito $\mathcal{O}$ *forman un grupo aditivo abeliano.*

Aunque se ha trabajado en el campo de los reales $\mathbb{R}$, las formulas y propiedades expuestas aplican para otros campos. La criptografía con curva elíptica se enfoca en las curvas definidas sobre *campos finitos* y el grupo aditivo que deriva de estos.

5.4.3. Curvas elípticas en campos finitos

En criptografía con curvas elípticas se utilizan dos tipos de campos, aquellos cuyo orden es un número primo impar $q = p$ o potencias de la forma $q = 2^m$. Los *campos finitos primos* se denotan por F_p mientras que los *campos binarios finitos* F_{2^m}. Es importante señalar que este trabajo se enfoca solamente en campos finitos primos.

Si el campo F_p tiene característica $p \neq 2$ o 3 y extensión $m = 1$ la Ecuación 5.1 así como la suma de puntos en curvas elípticas y toda la aritmética se realiza módulo p.

Ejemplo 5.4.2. La figura 5.4 muestra el conjunto de puntos que satisfacen $y^2 = x^3 + ax + b$, definida sobre el campo primo F_{17} con $a = 10$ y $b = 2$.

Todos los puntos se pueden encontrar evaluando los enteros $\mathbb{Z}_{17}^{+}$ en la ecuación de la curva. Pongamos por ejemplo $x = 0$. Evaluando tenemos $y = \sqrt{2} \pmod{17}$. De manera que $6^2 = 36 \equiv 2 \pmod{17}$. También $11^2 = 121 \equiv 2 \pmod{17}$. Estos son los únicos enteros elevados al cuadrado congruentes módulo dos. Por tanto, se encontraron los punto $(0, 6)$ y $(0, 11)$. De forma similar se encuentran los 22 puntos que forman la curva. Así

$$\begin{aligned} E(F_{17}) = \{&\mathcal{O}, (0,6), (0,11), (1,8), (1,9), (2,8), (2,9), (3,5), \\ &(3,12), (4,2), (4,15), (8,4), (8,13), (11,7), (11,10), \\ &(13,0), (14,8), (14,9), (15,5), (15,12), (16,5), (16,12)\}. \end{aligned}$$

Ahora se realiza la suma de los puntos $(3, 5)$ y $(11, 10)$. La pendiente entre ambos es

$$\frac{10-5}{11-3} = 5(8^{-1}) = 5(15) \equiv 7 \pmod{17}.$$

Por lo tanto, la ecuación de la recta es $y = 7(x - 3) + 5 \equiv 7x + 1$. Sustituyendo en la ecuación de Weierstrass y cambiando el orden

$$0 = x^3 - 15x^2 - 4x + 1.$$

La suma de las raíces es igual al coeficiente 15. Ya se conocen las raíces 3 y 11. Luego, la tercer raíz es $x = 1$. Como $y = 7x + 1$ tenemos $y = 8$. Reflejando respecto al

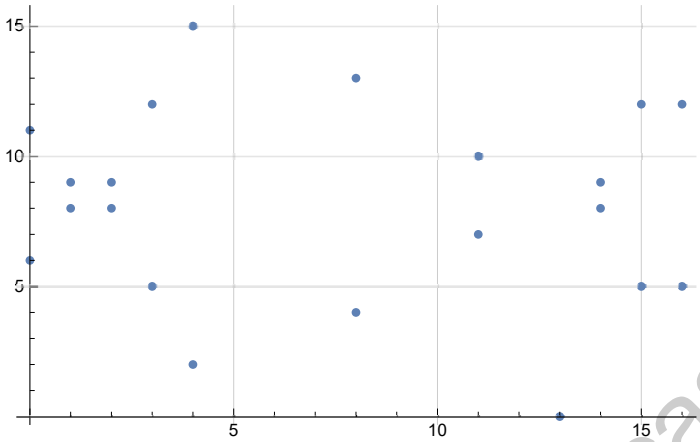

Figura 5.4: Puntos de la curva elíptica $y^2 = x^3 + 10x + 2$ definida sobre F_{17}.

eje x da como resultado el punto $(1, 9)$.

Ahora se multiplica el punto $(0, 11)$ por el escalar $k = 2$. Calculamos la pendiente derivando

$$\frac{dy}{dx} = \frac{10}{5} \equiv 2 \pmod{17}.$$

La ecuación de la recta $y = 2x+11$ se sustituye en la ecuación de la curva. Simplificando

$$0 = x^3 - 4x^2.$$

La suma de las raíces es igual a 4. Ya se conoce la raíz repetida 0. Por lo tanto, la tercer raíz es $x = 4$. Como $y = 2x + 11$ tenemos $y \equiv 2 \pmod{17}$. Reflejando respecto al eje x se obtiene el punto $(4, 15)$ (Gayoso y col., 2018).

Orden de una curva elíptica sobre un campo finito El número de puntos de una curva elíptica sobre un campo finito es también finito; aunque el número de estos puntos no es trivial. Considere el ejemplo anterior, es claro que son finitos. Para obtenerlos se evaluó cada uno de los valores en F_{17}. Si dos de estos valores satisfacen la ecuación de la curva módulo p se ha encontrado un nuevo punto. Sin embargo, cuando p es grande este proceso es impracticable.

Definimos el orden de una curva E sobre un campo F como el número de puntos que tiene dicha curva. Es decir

$$\#E(F_p) = \mid \{(x, y) \in F_p \times F_p \mid f(x, y) = 0\} \cup \mathcal{O} \mid$$

donde $\times$ denota al producto cartesiano y $\#E$ es el orden o número de puntos de la curva. Previamente se definió el orden de un grupo por $|G|$. Aunque es una notación

habitual, la literatura referente a curvas elípticas emplea la notación $\#E(F)$ para referirse al orden del grupo. En conformidad, se utilizará esta última notación en el resto de la sección.

Teorema 5.4.2 (Hasse)**.** *Sea E una curva elíptica sobre un campo finito F_p. Entonces el orden de la curva $\#E(F_p)$ cumple*

$$\#E(F_p) = p + 1 - t, \qquad \mid t \mid \leq 2\sqrt{p}, \tag{5.2}$$

donde t representa la traza de la curva (Washington, 2003).

El teorema de Hasse permite estimar el número de puntos de una curva elíptica sobre un campo finito. Para criptografía es importatnte que el orden de una curva elíptica $E(F_p)$ sea aproximadamente igual al tamaño p del campo.

Ejemplo 5.4.3. Considere la curva elíptica definida en el ejemplo 5.4.2. Por el teorema de Hasse se tiene

$$\#E(F_{17}) = 17 + 1 - t$$

donde $\mid t \mid \leq 2\sqrt{17}$. Es decir, se encuentra entre $-2\sqrt{17} \leq t \leq 2\sqrt{17}$. Note que $2\sqrt{17} \approx 8$. Por tanto, esta curva tendría entre 26 y 10 puntos. En particular $t = -4$ y el orden del grupo $\#E(F_{17}) = 22$. Se cumple así que el número de puntos para esta curva está dentro del intervalo delimitado por el teorema de Hasse.

Los algoritmos más conocidos para calcular el número de puntos de una curva elíptica son el algoritmo de Schoof (Schoof, 1985) con complejidad $O(\log^9 p)$ y el algoritmo de Schoof-Elkies-Atkins con complejidad $O(\log^5 p)$ (Schoof, 1995). Su corto tiempo de procesamiento permitió el avance de la criptografía de curva elíptica a la práctica.

Orden de un punto en la curva El orden de un punto $P \in E(F_p)$ es el valor entero positivo n más pequeño tal que $nP = \mathcal{O}$ y se denota como $\text{ord}(P)$. Por el teorema de Lagrange 5.2.5 , el orden de un punto P divide el orden de la curva $\#E(F_p)$.

El orden de una grupo puede ser primo o compuesto. Si G es un punto capaz de generar un subgrupo cíclico con orden primo n y no existe otro factor primo de $\#E$ mayor que n, entonces se puede definir el *cofactor* como el cociente del orden de la curva entre su mayor factor primo n. Es decir

$$h = \frac{\#E(F_p)}{n}.$$

En criptografía es conveniente utilizar curvas cuyo orden $\#E$ sea un número primo. En caso de un orden compuesto, que sean el producto de un número primo por un cofactor pequeño. Típicamente $2, 3,$ o 4.

5.4.4. Criptosistemas con curvas elípticas

En los últimos años los criptosistemas con curvas elípticas han adquirido gran popularidad por tener el mismo nivel de seguridad que los criptosistemas RSA empleando claves tres veces menores. Además, sus cálculos son eficientes. De ahí su uso en dispositivos de cómputo limitado, por ejemplo en dispositivos *Internet de las cosas* (IoT). También *Blockchain*, una tecnología de registros decentralizada, usa ECC por la seguridad que ofrecen sin incrementar exponencialmente el tamaño de las claves.

La siguiente sección presenta los parámetros y algoritmos necesarios para implementar un criptosistema con curva elíptica. En concreto el Elliptic Curve Digital Signature Algorithm. Esto con el objetivo de desarrollar un prototipo de *timbre digital* para el Comprobante Fiscal Digital por Internet.

Generación de parámetros

Los parámetros de una curva elíptica pueden expresarse como el conjunto $D = \{q, a, b, G, n, h\}$ donde:

1. El tamaño del campo finito se denota por q, con $q = p$ donde p es un número primo impar.
2. Dos elementos del campo a y b en F_q que definen la ecuación de una curva elíptica E sobre F_q.
3. Dos elementos del campo x_G, y_G que definen un punto $G = (x_G, y_G)$ de orden primo en $E(F_q)$.
4. El orden n del punto G, con $n > 2^{160}$ y $n > 4\sqrt{q}$.
5. El cofactor $h = \#E(F_q)/n$.

En este trabajo se estudia la curva *secp256k1* diseñada por Certicom®. La razón de utilizarla es su interoperabilidad con múltiples estándares. Por otro lado, sus parámetros permiten una implementación eficiente en el campo F_p. El tamaño del campo, en hexadecimal, es

$$p = FFFFFFFF\ \ FFFFFFFF\ \ FFFFFFFF\ \ FFFFFFFF\ \ FFFFFFFF\ \ FFFFFFFF\ \ FFFFFFFE\ \ FFFFFC2F = 2^{256} - 2^{32} - 2^9 - 2^8 - 2^7 - 2^6 - 2^4 - 1.$$

Los coeficientes a, b de la Ecuación 5.1 que define la curva $E(F_p)$ son:

$$a = 0,$$
$$b = 7.$$

El punto que genera un subgrupo en formato comprimido, es decir la coordenada x, es

$$G = 02 \quad 79BE667E \quad F9DCBBAC \quad 55A06295 \\ CE870B07 \quad 029BFCDB \quad 2DCE28D9 \quad 59F2815B \quad 1.$$

Sin comprimir el punto $G = (x, y)$ se tiene

$$G = 04 \quad 79BE667E \quad F9DCBBAC \quad 55A06295 \\ CE870B07 \quad 029BFCDB \quad 2DCE28D9 \quad 59F2815B \\ 16F81798 \quad 483ADA77 \quad 26A3C465 \quad 5DA4FBFC \\ 0E1108A8 \quad FD17B448 \quad A6855419 \quad 9C47D08F \quad F$$

Finalmente el orden del punto G y el cofactor son:

$$n = FFFFFFFF \quad FFFFFFFF \quad FFFFFFFF \\ FFFFFFFE \quad BAAEDCE6 \quad AF48A03B \quad BFD25E8C \quad D,$$

$$h = 01.$$

Para asegurar que estos parámetros sean validos y seguros se ejecutan los pasos del Algoritmo 7. Como p es grande se necesita un sistema de álgebra computacional para verificarlos. Así, codificando el Algoritmo 7 se confirma que los parámetros de la curva *secp256k1* son apropiados. Todos los usuarios de un sistema criptográfico definido bajo esta curva deben utilizarlos. Por lo tanto, deben estar disponibles en un archivo público. De esta manera pueden consultarlos para iniciar una comunicación segura.

Algoritmo 7: Validación de los parámetros de una curva elíptica

Datos: El conjunto $D \leftarrow \{p, a, b, G, n, h\}$
Resultado: Aceptar o rechazar los parámetros del conjunto D

1 Verificar p sea un número primo impar;
2 Verificar que $G \neq \mathcal{O}$;
3 Verificar que a, b, x_G y y_G son elementos de F_p, es decir, que estos parámetros son enteros en el intervalo $[0, p-1]$;
4 Verificar que a y b definen una curva elíptica E sobre F_p tal que $4a^3 + 27b^2 \neq 0 \mod p$;
5 Verificar que G sea un punto de la curva E. Es decir $y_G^2 = x_G^3 + ax_G + b$;
6 Verificar que n es primo;
7 Verificar que $n > 2^{160}$ y también $n > 4\sqrt{p}$;
8 Verificar que $nG = \mathcal{O}$;
9 Calcular $h' \leftarrow \lfloor (\sqrt{p}+1)^2/n \rfloor$ y validar que $h = h'$;
10 Verificar que n no divide $p^k - 1$ para cada k, con $1 \leq k \leq 20$;
11 Verificar que $n \neq p$;
12 Si alguno de los puntos anteriores falla, entonces D es invalido, en caso contrario D es valido

Generación de claves

Una vez definido y validado el conjunto de parámetros de una curva elíptica $D = \{p, a, b, G, n, h\}$ el firmante genera su clave pública y privada con el Algoritmo 8. Para ser más precisos la clave pública es el punto Q y la clave privada el entero d. La selección pseudoaleatoria del paso 1 es constante $O(1)$ y depende del algoritmo empleado. El paso 2 multiplica un punto por un escalar. Por lo tanto, su complejidad es polinomial.

Algoritmo 8: Generación de claves para ECDSA

Datos: Una curva E definida en el campo F_p
Resultado: Clave pública Q y privada d

1 Seleccionar de forma aleatoria o pseudoaleatoria d tal que $1 \leq d \leq n-1$.
2 Calcular $Q \leftarrow dG$.
3 Asignar Q como la clave pública y d como la clave privada del usuario.

El Algoritmo 9 verifica que la clave pública sea valida. Una vez valorada, la clave pública se comparte con cualquier usuario para cifrar un mensaje o verificar una firma digital.

Algoritmo 9: Validación de clave pública para ECDSA

Datos: Una clave pública $Q \leftarrow (x_Q, y_Q)$ asociada a un conjunto valido de parámetros $D \leftarrow \{p, a, b, G, n, h\}$ definidos sobre el campo F_p.

Resultado: Clave pública valida o invalida

1 Revisar que $Q \neq \mathcal{O}$.
2 Revisar que x_Q, y y_Q son elementos de F_p.
3 Revisar que el punto Q pertenece a la curva E definida por a y b.
4 Revisar que $nQ = \mathcal{O}$.
5 Si cualquiera de los puntos anteriores falla, entonces Q es invalido. En caso contrario Q es valido.

Esquema de firma digital utilizando curvas elípticas: ECDSA

La mayor utilidad de los criptosistemas con curva elíptica no es el cifrado sino la generación de firmas digitales. De ahí el desarrollo de una versión del Digital Signature Algorithm (DSA, *Digital Signature Algorithm*) sustituyendo el grupo de enteros módulo p por los puntos de una curva elíptica. El Elliptic Curve Digital Signature Algorithm (ECDSA), al igual que DSA, se basa en el esquema de firma de ElGamal pero supera su velocidad de verificación al realizar solo dos multiplicaciones de un punto por un escalar. Desde el año 2000 el ECDSA es un estándar ISO, IEEE y ANSI (Johnson y col., 2001).

Considere la Figura 1.1 donde Juan solicita a Ana firmar un documento M. Una vez generados los parámetros y claves de un ECC se ejecuta un algoritmo de firma y verificación. Estos algoritmos se detallan a continuación.

Generación de firmas y verificación

Sea $\mathcal{H}$ una función de digestión o *hash* que mapea una cadena de bits de longitud variable a una cadena de bits de longitud fija y M un mensaje. El Algoritmo 10 describe como firmar un mensaje M utilizando la clave privada asociada a un conjunto valido de parámetros $D = \{p, a, b, G, n, h\}$. En general, la complejidad de este algoritmo es polinomial $O(n^c)$.

Algoritmo 10: Generación de firma ECDSA

Datos: Un mensaje M, y el par de claves (d, Q) asociados a un conjunto de parámetros $D \leftarrow \{p, a, b, G, n, h\}$ validos.

Resultado: Una firma (r, s) para el mensaje M.

1 Calcular el *hash* del mensaje $e \leftarrow \mathcal{H}(M)$ y representar e como un entero.
2 Generar de forma aleatoria o pseudoaleatoria un entero k tal que $1 \leq k \leq n-1$.
3 Calcular $(x_1, y_1) \leftarrow kG$.
4 Calcular $r \leftarrow x_1 \mod n$. Si $r = 0$ ir al paso 2.
5 Calcular $k^{-1} \mod n$.
6 Calcular $s \leftarrow k^{-1}(e + dr) \mod n$. Si $s = 0$ entonces ir al paso 2.
7 La firma del mensaje m es (r, s).

Suponga que Ana generó la firma con su par de claves (d, Q). Para verificar la firma (r, s) del mensaje M se siguen los pasos del Algoritmo 11. Es importante que el interesado en corroborar la autenticidad de la firma, Juan en nuestro ejemplo, tenga acceso a la clave pública del firmante. La complejidad de este algoritmo es similar al anterior, pero se realiza una suma y una multiplicación de punto por un escalar adicional.

Algoritmo 11: Verificación de firma ECDSA

Datos: Una firma (r, s) del mensaje M, una copia del mensaje M y la clave pública Q asociada al conjunto de parámetros $D \leftarrow \{p, a, b, G, n, h\}$.

Resultado: Aceptar o rechazar la firma.

1 Revisar que r y s son enteros en el intervalo $[1, n-1]$.
2 Calcular $e' \leftarrow \mathcal{H}(M)$ y convertirlo a un entero.
3 Calcular $w \leftarrow s^{-1} \mod n$.
4 Calcular $u_1 \leftarrow e'w \mod n$ y $u_2 = rw \mod n$.
5 Calcular $X \leftarrow u_1 G + u_2 Q$.
6 Si $X = \mathcal{O}$ rechazar la firma. En caso contrario calcular $v \leftarrow x_1 \mod n$.
7 Aceptar la firma si y solo si $v = r$.

Llegado a este punto se demostrará que la verificación de la firma realizada por el Algoritmo 11 funciona. Si la firma (r, s) de un mensaje M fue realmente generada por el usuario en posesión del par de claves d, Q entonces $s = k^{-1}(e + dr)$. Reordenando los términos tenemos

$$k \equiv s^{-1}(e + dr) \equiv s^{-1}e + s^{-1}rd \equiv we' + wrd \equiv u_1 + u_2 d \mod n.$$

Por tanto $u_1 G + u_2 Q = (u_1 + u_2 d)G = kG$ y como solo Ana posee d es la única capaz de generar la firma (r, s) del mensaje M.

5.4.5. Seguridad de un criptosistemas con curvas elípticas

Sea p un número primo y a, b enteros distintos de cero módulo p. Suponga que existe un entero k tal que

$$a^k = b \pmod{p}.$$

El *problema del logaritmo discreto* clásico, en notación multiplicativa, es encontrar $k = \log_a b \pmod{p}$. El algoritmo de *cálculo de índice* tiene complejidad subexponencial y es la mejor herramienta para obtener k (Washington, 2003).

La seguridad de los ECC se basa en la dificultad de resolver su análogo en curvas elípticas, el Elliptic Curve Discrete Logarithm Problem (ECDLP, *Elliptic Curve Discrete Logarithm Problem*). Se conjetura que su complejidad es mayor a la del problema clásico del logaritmo discreto. Esto debido a que no existe algoritmo de complejidad subexponencial para resolverlo (Koblitz y col., 2000) (Costello, 2012). A continuación se enuncia el ECDLP con una notación diferente porque el grupo de una curva elíptica E es aditivo.

Elliptic Curve Discrete Logarithm Problem Dada una curva elíptica E definida sobre el campo finito F_p y los puntos $G, P \in E$ con G un generador de orden n, encontrar el entero k tal que $P = kG$ si tal k existe.

Actualmente no existe un algoritmo que resuelva eficientemente el ECDLP si el orden del generador n es grande. Múltiples algoritmos se han desarrollado para solucionar este problema. A día de hoy, el algoritmo *ro* de Pollard es el mejor método. A continuación se presentan los ataques más comunes al ECDLP.

Algoritmo de busqueda exhaustiva Un ataque de búsqueda exhaustiva o fuerza bruta del punto P consiste en obtener $G,\ 2G,\ 3G$ hasta encontrar el punto $P = xG$. En el peor de los casos se necesitaría realizar n multiplicaciones y de media $n/2$. Para evitarlo es importante escoger un punto generador del grupo G con orden n muy grande para que el ataque sea inviable (Gayoso y col., 2018).

Algoritmo *baby-step giant-step* El algoritmo consiste en calcular una lista con múltiplos pequeños de G, es decir el paso-enano, y después calcular múltiplos grandes de G, el paso-gigante (*baby-step giant-step*). Si se encuentra un punto común en ambas listas es posible calcular x de forma sencilla. En el peor de los casos este algoritmo necesitará almacenar una lista de $\sqrt{n}$ puntos en memoria y procesar $2\sqrt{n}$ multiplicaciones de un punto por un escalar (Johnson y col., 2001).

Algoritmo 12: Baby-step Giant-step

Datos: Un punto P junto con el punto generador G y su orden n
Resultado: El entero k tal que $P \leftarrow kG$

```
Calcular s ← ⌊√n⌋ + 1;
para i ← 0 a s hacer
    Almacenar G_i ← iG;
fin
repetir
    Q_j ← P − jsG con j ← 0, 1, ... s − 1
hasta que Q_j coincida con algún valor G_i de la lista;
devolver k ← i + js (mod n)
```

Ejemplo 5.4.4. Sea E la curva $y^2 = x^3 + 7x + 9$ sobre F_{23}. Por el teorema de Hasse el orden $\#E(F_{23})$ es a lo más 24. Dado el punto generador $G = (8, 5)$ y el punto $P = kG = (19, 3)$ se procede a determinar el escalar k con el algoritmo *baby-step giant-step*. El valor $s = \lfloor\sqrt{23}\rfloor + 1 = 5$. Los seis puntos iG para $0 \leq i \leq 5$ son

$$\mathcal{O},\ (8, 5),\ (15, 19),\ (4, 3),\ (17, 2),\ (16, 13).$$

Ahora se calcula $Q_j = P - jsG$ para $j = 0, 1, 2$ y se obtienen los puntos

$$(19, 3),\ (19, 20),\ (17, 2).$$

Se detiene en $j = 2$ puesto que $G_4 = Q_2$, $(17, 2) = P - (2)5G$. Por lo tanto $k = 4 + (2)5 \pmod{23} = 14$.

Algoritmo de Polling-Hellman El algoritmo de Polling-Hellman resuelve el ECDLP encontrando $k \pmod{p_i^{e_i}}$ en algunos de los subgrupos de orden primo p_i tal que $\#E(F_p) = p_1^{e_1} p_2^{e_2} \ldots p_s^{e_s}$. Después recuperar $k \pmod{p}$ resolviendo el sistema de congruencias

$$\begin{aligned} k &\equiv k_1 \pmod{p_1^{e_1}} \\ k &\equiv k_2 \pmod{p_2^{e_2}} \\ &\vdots \\ k &\equiv k_s \pmod{p_s^{e_s}} \end{aligned}$$

con el Chinese Remainder Theorem (CRT, *Chinese Remainder Theorem*). Para sortear este ataque es necesario escoger una curva cuyo orden sea divisible entre un primo n grande.

Algoritmo 13: Polling-Hellman

Datos: El punto $P \leftarrow kG$ donde G es el punto generador con orden n del grupo $E(F_p)$

Resultado: El entero k

1 Factorizar el orden del punto G como $n \leftarrow p_1^{e_1} p_2^{e_2} \dots p_s^{e_s}$;
2 **para cada** *factor primo* $p_i^{e_i}$ **hacer**
3 $r \leftarrow 0$;
4 Calcular $T \leftarrow \{j(\frac{n}{p_i})G \mid 0 \leq j \leq p_i - 1\}$;
5 Calcular $(\frac{n}{p_i})P$. Este punto es un elemento $k_0'(\frac{n}{p_i})G$ de T;
6 $r \leftarrow r + 1$;
7 **si** $r = e_i$ **entonces**
8 Ir al paso 18
9 **fin**
10 $Q_1 \leftarrow P - k_0'G$;
11 Calcular $(\frac{n}{p_i^2})Q_1$. Este punto es un elemento $k_1'(\frac{n}{p_i}G)$ de T;
12 $r \leftarrow r + 1$;
13 **mientras** $r < e_i$ **hacer**
14 $Q_r \leftarrow Q_{r-1} - k_{r-1}' p_i^{r-1} G$;
15 Determinar k_r' tal que $(\frac{n}{p_i^{r+1}})Q_r = k_r'(\frac{n}{p_i}G)$;
16 $r \leftarrow r + 1$;
17 **fin**
18 Calcular $k_i \leftarrow k_0' + k_1' p_i + \cdots + k_{e_i-1}' p^{e_i-1} \pmod{p^{e_i}}$
19 **fin**
20 Resolver el sistema de congruencias $k_i \equiv k \pmod{p_i^{e_i}}$ para $1 \leq i \leq s$ con el *teorema del residuo chino*;
21 **devolver** *El entero k*

Ejemplo 5.4.5. La curva $y^2 = x^3 + 6x + 22$ definida sobre el campo F_{127} tiene el punto generador $G = (88, 63)$ con orden $n = 24 = 2^3\ 3$. Dado el punto $P = (72, 103)$ se pasa a calcular el valor k con el algoritmo Polling-Hellman.

***k* (mod 8)**. Se calcula el conjunto $T = \{\mathcal{O}, (40, 0)\}$. Puesto que

$$\tfrac{24}{2}P = \mathcal{O} = 0\ (\tfrac{24}{2}G)$$

se infiere que $k_0' = 0$. A continuación

$$Q_1 = P - 0G = P.$$

Dado que $\frac{24}{2^2}Q_1 = (40, 0) = 1\ (\frac{24}{2}G)$ se deduce que $k_1' = 1$. Similarmente

$$Q_2 = Q_1 - 1(2G) = (40, 0).$$

Dado que $(\frac{24}{2^3})Q_2 = (40, 0) = 1(\frac{24}{2}G)$ entonces $k_2' = 1$. Por lo tanto

$$k_1 = 0 + 1(2) + 1(2^2) = 6.$$

***k* (mod 3)**. De manera semejante calculamos $T = \{\mathcal{O}, (116, 99), (116, 28)\}$. Como

$$\tfrac{24}{3}P = (116, 28) = 2(\tfrac{24}{3}G)$$

tenemos que $k_0' = 2$. De modo que $k_2 = 2$.

Finalmente se resuelve el sistema de ecuaciones

$$\begin{aligned} k &\equiv 6 \pmod 8 \\ k &\equiv 2 \pmod 3 \end{aligned}$$

con el CRT para obtener $k = 14$.

Algoritmo ρ de Pollard Se trata de un algoritmo probabilístico dado que tiene alta posibilidad de terminar en el tiempo estimado; aún así puede tardar más de lo esperado. Se construye una secuencia de puntos a partir de una función f pseudoaleatoria. Los puntos se obtienen de forma recursiva $Q_{i+1} = f(Q_i)$ a partir de un punto pseudoaleatorio Q_0 de $E(F_p)$. Puesto que el grupo $E(F_p)$ es cíclico, eventualmente comenzará a repetirse dicha secuencia para un subíndice $i_0 < j_0$ tal que $Q_{i_0} = Q_{j_0}$; lo que gráficamente se asemeja a una letra griega *ro*. Entonces

$$Q_{i_0+1} = f(Q_{i_0}) = f(Q_{j_0}) = Q_{j_0+1}$$

y de igual modo $Q_{i_0+l} = Q_{j_0+l}$ para todo $l \geq 0$. Entonces la secuencia tiene como periodo $j_0 - i_0$ o un divisor del mismo $j_0 - i_0$.

Una implementación ineficiente almacena todos los puntos Q_i hasta encontrar una coincidencia. Ahora bien, se puede mejorar aplicando el *algoritmo de la tortuga y la liebre* para guardar únicamente dos puntos. Un punto avanza un paso mientras el otro dos. Por lo tanto se calcula el par (Q_i, Q_{2i}) para $i = 1,\ 2, \ldots$ hasta encontrar una coincidencia, en cuyo caso los subíndices están apartados por un valor d. Entonces, cualquier otro par de subíndices que difiere por d resulta en dos puntos iguales. El siguiente par de puntos en la secuencia puede calcularse como

$$Q_{i+1} = f(Q_i), \qquad Q_{2(i+1)} = f(f(Q_{2i})).$$

La dificultad es elegir una función f que, además de ser pseudoaleatoria, devuelva información útil para calcular k cuando se da una coincidencia. Se divide $E(F_p)$ en subconjunto disjuntos $S_1, S_2, \ldots, S_s$ aproximadamente del mismo tamaño. Luego, se escojen $2s$ enteros $a_i, b_i \pmod n$ de forma pseudoaleatoria. Ahora

$$M_i = a_i G + b_i P.$$

Después, definimos la función como

$$f(Q_{i+1}) = Q_i + M_i = Q_i + a_iG + b_iP$$

para $Q_i \in S_i$. Finalmente, se seleccionan de forma pseudoaleatoria enteros a_0, b_0 para obtener el punto inicial $Q_0 = a_0G + b_0P$. Al calcular cada punto de la secuencia es importante registrarlo en términos de G y P. Si $Q_i = u_iG + v_iP$ y $Q_{j+1} = Q_i + M_i$ entonces $Q_{j+1} = (u_i + a_i)G + (v_i + b_i)P$. En consecuencia $(u_{j+1}, v_{j+1}) = (u_i, v_i) + (a_i, b_i)$. Cuando se encuentra $Q_{j_0} = Q_{i_0}$ tenemos

$$u_{j_0}G + v_{j_0}P = u_{i_0}G + v_{i_0}P,$$

por lo tanto

$$(u_{i_0} - u_{j_0})G = (v_{j_0} - v_{i_0})P.$$

Si $\text{mcd}(v_{j_0} - v_{i_0}, n) = d$ entonces

$$k = (v_{j_0} - v_{i_0})^{-1}(u_{i_0} - u_{j_0}) \pmod{n/d}.$$

Dicho lo anterior, dado que en aplicaciones criptográficas n es primo tenemos $d = 1$ o n. Otra posibilidad es el caso trivial $d = n$ donde los coeficientes de G y P son múltiplos de n. Si esto ocurre se debe reiniciar (Washington, 2008).

Aunque se requiere procesar el mismo número de operaciones del algoritmo baby-step giant-step, el algoritmo ocupa menor espacio de almacenamiento en memoria. La complejidad del algoritmo es $(\sqrt{\pi n})/2$. Además, este algoritmo puede acelerarse mediante *cómputo paralelo*. Así, al ejecutar paralelamente en r procesadores reduce la complejidad a $(\sqrt{\pi n})/2r$ (Johnson y col., 2001).

Ejemplo 5.4.6. Sea $E(F_{37})$ la curva elíptica definida por $y^2 = x^3 + 15x^2 + 9$. Dado $s = 3$, el generador $G = (19, 7)$ con orden 43 y $P = kG = (20, 13)$ se procede a calcular k con el algoritmo Pollard *ro*. Sean

$$Q_0 = 16G + 27P, \quad M_0 = 2G + 3P, \quad M_1 = 5G + 8P, \quad M_2 = 9G + 7P.$$

Entonces la función f se define como

$$f(Q_i) = f(x, y) = \begin{cases} Q_i + 2G + 3P & \text{si } x \pmod 3 = 0, \\ Q_i + 5G + 8P & \text{si } x \pmod 3 = 1, \\ Q_i + 9G + 7P & \text{si } x \pmod 3 = 2. \end{cases}$$

Al iniciar con el punto Q_0 se tiene la secuencia:

$$Q_0 = (3, 28),\ Q_1 = (31, 31),\ Q_2 = (1, 5),\ Q_3 = (33, 25),\ Q_4 = (26, 17),$$
$$Q_5 = (10, 7),\ Q_6 = (16, 33),\ Q_7 = (26, 20),\ Q_8 = (13, 25),\ Q_9 = (12, 20),$$
$$Q_{10} = (12, 17),\ Q_{11} = (8, 30),\ Q_{12} = (10, 30),\ Q_{13} = (0, 3),\ Q_{14} = (2, 26),$$
$$Q_{15} = (1, 32),\ Q_{16} = (31, 6),\ Q_{17} = (20, 24),\ Q_{18} = (11, 5),\ Q_{19} = (16, 4),$$
$$Q_{20} = (10, 30).$$

De esta se observa que $Q_{12} = Q_{20} = (10, 30)$. Si se mantuvo registro de los coeficientes para G y P podemos expresar este punto de dos formas diferentes:

$$Q_{12} = (10, 30) = Q_{11} + 9G + 7P = 24G + 7P + 9G + 7P = 33G + 14P,$$
$$Q_{20} = (10, 30) = Q_{19} + 5G + 8P = 34G + 19P + 5G + 8P = 39G + 27P.$$

Por lo tanto

$$33G + 14P = 39G + 27P$$
$$-6G = 13P$$

Es así que $\text{mcd}(13, 43) = 1$ y

$$k = 13^{-1}(-6) \equiv 10(37) \equiv 26 \pmod{43}.$$

Ahora bien, en lugar de calcular la lista entera se calcula simplemente la pareja de puntos Q_i, Q_{2i} con el algoritmo de la tortuga y la liebre. Así, se tendría la coincidencia $Q_{16} = (31, 6) = 11G + 40P$ y $Q_{32} = (31, 6) = 23G + 23P$. Resolviendo se llega a $k = 26$.

Algoritmo λ de Pollard El algoritmo *lambda* de Pollard es una variante del algoritmo *ro* de Pollard. La idea central es utilizar dos puntos iniciales en lugar de uno para generar dos secuencias pseudoaleatorias con una función f similar al algoritmo *ro* de Pollard. Estas dos secuencias de puntos pueden verse como dos caminos por los que transitan un par de canguros que, por tratarse de un grupo cíclico, coincidirán en un punto después de varias iteraciones. A partir de esta confluencia los canguros saltaran por el mismo camino; lo que gráficamente se asemeja a la letra griega *lambda*.

La complejidad del algoritmo es a lo más $O(\sqrt{n})$ pero con cómputo paralelo puede acelerarse r veces, donde r es el número de procesadores adicionales. No obstante, el algoritmo paralelado *ro* de Pollard es ligeramente más veloz que el algoritmo *lambda*. Por lo cual, el algoritmo Pollard *ro* se considera el mejor método para resolver el ECDLP.

5.5. Comprobantes fiscales digitales por Internet: CFDI

La firma electrónica en México tiene operando desde el año 2000 cuando se publicaron las Reformas al Código de Comercio. Sin embargo, fue hasta el año 2012 cuando

entró en vigor la Ley Federal de Firma Electrónica («Ley de Firma Electrónica Avanzada», 2012). Este decreto le dio un peso jurídico y reguló los servicios que requieren firma digital. Al día de hoy, su principal aplicación se encuentra en en el SAT. Para ser precisos, en el comprobante fiscal.

En México un Comprobante Fiscal Digital por Internet permite al SAT mantener un registro de los gastos e ingresos de los contribuyentes. Es un mecanismo de control que permite la vigilancia del cumplimiento de las obligaciones fiscales. Además brinda seguridad en las transacciones realizadas entre vendedor y cliente.

Con el fin de intercambiar la información de los CFDI a través de Internet y entre distintas plataformas el SAT decidió valerse de estos dos estándares gratuitos: XSD y XML. Las reglas para expedir los CFDI se establecen en el Anexo 20 de la Segunda Resolución de modificaciones a la Resolución Miscelánea Fiscal para 2017 (Santín, 2017). Plantea que los CFDI se obtengan a partir de una *XML Schema Definition* (XSD, *XML Schema Definition*). El XSD es una plantilla que define la estructura de un archivo XML. Este es el formato que todos los CFDI deben tener. Luego de llenar cada campo delimitado por el XSD se genera el CFDI como un archivo en *eXtensible Markup Language* (XML, *eXtensible Markup Language*) para su distribución y almacenamiento.

Un CFDI debe cumplir con las siguientes características:

- Autenticidad
- Integridad
- No repudio
- Certidumbre de origen

La autenticidad garantiza que el firmante fue quien verdaderamente suscribió. La integridad implica que si el mensaje es alterado es fácil detectarlo. El no repudio consiste en la imposibilidad del firmante para negar su autoría. La certidumbre de origen permite rastrear a la persona física o moral que expidió el documento firmado. Para que un CFDI cumpla con todos estos atributos y sea infalsificable se debe *timbra con un sello digital*.

5.5.1. Timbre o sello digital

Los *timbres* o *sellos digitales* en los CFDI son *firmas digitales* generadas por un criptosistema de clave pública. El termino "sello" es ambiguo ya que, entrando en rigor, se refiere a una firma digital. A no ser que se indique lo contrario, el lector debe considerar los términos *timbre* y *sello* como sinónimos de firma digital en este trabajo.

Hasta la fecha en que se redactó el presente trabajo de investigación, el SAT ocupa el algoritmo de firma RSA para sellar los CFDI. Se asume que el lector tiene conocimiento del algoritmo RSA. Es común en los cursos de matemáticas discretas estudiar este cifrado. Para quien no este familiarizado con este algoritmo recomendamos consultar la sección 5.3.

Para generar un sello digital se requieren los siguientes elementos:

- Cadena Original del CFDI a sellar.
- Certificado de Sello Digital y su correspondiente clave privada.
- Algoritmos de criptografía de clave pública para firma electrónica avanzada.
- Especificaciones de conversión de la firma electrónica avanzada a Base 64.

La Cadena Original se forma con cierta información que contiene el CFDI. Los datos son separados por caracteres pleca (||). El Certificado de Sello Digital, que contiene la clave pública, y la clave privada son entregados por el SAT a cualquier contribuyente que los solicite. Las claves son generadas conforme la sección 5.3.1 y la firma electrónica avanzada con el Algoritmo 4.

La conversión a Base 64 de la firma digital consiste en transformar la representación numérica de la firma digital a una cadena de bits. Esta cadena se segmenta cada 24 bits. Después, cada una de estas se subdivide de seis en seis bits pues $24 = 4 \times 6$. Cada grupo de seis bits se codifica, de izquierda a derecha, en uno de los 64 caracteres que define el estándar (Josefsson, 2006). Se adjunta una tabla de codificación Base64 en el Anexo A.

El Anexo 20 (Santín, 2017) define los siguientes algoritmos para sellar un CFDI:

- SHA-256, que es una función *hash* la cual produce una salida de 256 bits a partir de una entrada con dimensión variable.
- RSAPrivateEncrypt, que utiliza la clave privada del emisor para cifrar un mensaje.
- RSAPublicDecrypt, que utiliza la clave pública para descifrar el mensaje generado por el algoritmo de cifrado antes mencionado.

Con el fin de sellar un CFDI se utilizan estos algoritmos de la siguiente manera:

1. Se aplica la función *hash* SHA-256 a la cadena original por sellar. Esto genera una salida de 256 bits (32 bytes). Esta cadena de bits se representa como un número entero para procesarse en el siguiente paso. La posibilidad de que dos entradas distintas generen el mismo *hash* es pequeña.

2. Con la clave privada correspondiente al certificado digital del firmante se cifra la representación numérica del *hash* utilizando el Algoritmo 4.

3. El resultado se transforma a una cadena binaria que no necesariamente consta de caracteres imprimibles. De ahí que se utilice la codificación Base64 para generar caracteres comprensibles al poseedor de la factura.

En la Figura 1.2 se tiene un ejemplo de un sello digital generado con un certificado de 2048 bits.

Se empleó el lenguaje de programación de alto nivel *Mathematica*® para implementar los algoritmos antes mencionados y generar un CFDI con curva elíptica. Sus ventajas se describen en la siguiente sección.

5.6. Sistema de computación técnica Mathematica®

Mathematica® es un sistema de computación técnica utilizado por científicos, innovadores y académicos. Se basa en *Wolfram Language*, un lenguaje de programación simbólico y funcional fácil de aprender. Es por su carácter simbólico que implementa el *paradigma de programación funcional*. Cuenta con más de 6000 funciones nativas basadas en eficientes algoritmos desarrollados por Wolfram Research. Estas funciones son útiles en áreas como geometría, ciencia de datos, procesamiento de señales, redes neuronales, aprendizaje automático y criptografía (Wolfram Research, Inc., s.f.).

Wolfram Language puede ejecutarse en Windows, Mac y Linux. También se puede utilizar en la *nube* a través de un navegador web. Aunque este sistema necesita la compra de una licencia, se puede obtenerse gratis al instalar el sistema operativo NOOBS en una *Raspberry Pi*.

Las expresiones en Wolfram Language son funciones muy similares a $f(x)$ donde f es la operación que se aplica al parámetro x. La notación difiere por el uso de *paréntesis cuadrados* en lugar de paréntesis. Por ejemplo, la entrada en Mathematica para sumar $2 + 3$ sería `Plus[2, 3]` donde *Plus* es la función adición y los parámetros 2 y 3 los números a sumar. Para evaluar la entrada se escribe el comando `Shift + Enter` (Wolfram, 2017). El resultado mostrado en pantalla es 5. Existen otras funciones nativas, este fue un simple ejemplo, pero es importante subrayar que cada una de ellas acepta distinto número y tipo de parámetros.

El libro *The Mathematica Book* contiene todos los aspectos de Mathematica. La última versión del libro consta alrededor de 1500 páginas (Wolfram, 2003). Es por esto que es imposible tratar a profundidad aquí todo lo relacionado a Mathematica. Dicho lo anterior, la siguiente sección se enfoca únicamente en las funciones nativas criptográficas para desarrollar el prototipo de sello con curva elíptica para CFDI. Para más

información sobre Mahtematica, posibles usos y aplicaciones se recomienda consultar la documentación oficial de Wolfram Language.

5.6.1. Criptografía en Wolfram Language

Wolfram Languaje incorporó funciones nativas criptográficas a partir de la versión 10.1. Después, la versión 12.1 amplió la gama de funciones criptográficas simétricas y asimétricas para garantizar la confidencialidad, integridad y autenticidad de las expresiones cifradas o firmadas digitalmente. Estas funciones se basan en la biblioteca criptográfica de código abierto OpenSSL.

Sobre todo se destacan las funciones nativas para generar pares de claves asimétricas y firmas digitales utilizando criptografía de curva elíptica. Estas son compatibles con distintas criptomonedas. Actualmente la única curva disponible es *secp256k1* pero se planea soportar en el futuro curvas recomendadas por el NIST y el Standards for Efficient Cryptogrpahy (SEC, *Standards for Efficient Cryptogrpahy*). La curva secp256k1 es utilizada por la tecnología *blockchain* de Bitcoin, Ethereum y ARK (Porechna, 2020).

Las funciones de la versión 12.1 de Mathematica® usadas en este trabajo son:

- `GenerateAsymmetricKeyPair[]` genera un par de claves RSA o ECC.
- `Encrypt[]` cifra cualquier expresión de Wolfram Language.
- `Decrypt[]` descifrar objetos generados por `Encrypt[]`.
- `Hash[]` mapea cadenas de bits de longitud arbitraria a cadenas de bits de longitud fija.
- `GenerateDigitalSignature[]` firma digitalmente una expresión de Wolfram Language con una clave privada.
- `VerifyDigitalSignauture[]` verifica que un firmante realizó la firma digital usando su clave pública.

Capítulo 6

Método

En esta sección se describen los materiales y procedimientos para generar el prototipo de un sello digital con el ECDSA para los CFDI. Así mismo la implementación de los algoritmos en Mathematica y la arquitectura de red para medir la tasa de transmisión de la propuesta. Se generaron 206 582 pares de claves y firmas digitales para el criptosistemas RSA como para el ECDSA.

6.1. Materiales

El *hardware* y *software* para llevar a cabo la implementación del prototipo se lista a continuación. Con ellos se logra cumplir los objetivos propuestos en este trabajo. Los materiales son:

- Una laptop Dell Inspiron 7537 con procesador Intel i7-4500U a 1.8 GHz, 8 GB de memoria RAM y sistema operativo Windows 10 Home de 64 bits.
- Una licencia para estudiante de Mathematica® versión 12.1
- Dos Raspberry Pi 4 con procesador Quad-core Cortex-A72 (ARM v8) 64-bit SoC a 1.5 GHz, 4 GB de memoria RAM, interfaz de red Gigabit Ethernet, sistema operativo NOOBS y sus respectivos dispositivos periféricos y fuentes de energía.
- Un cable Ethernet categoría 6.
- Comprobante fiscal digital por Internet
- Aplicación *xlstproc*
- Aplicación *iperf3*
- Par de memorias SD.
- Un comprobante fiscal digital por Internet (CFDI)

6.2. Enfoque metodológico

El presente trabajo es de carácter cuantitativo. Teniendo en cuenta que se comparan los tiempos de cómputo en la generación de las claves y firmas digitales, el espacio de almacenamiento y la tasa de transmisión se trata de una metodología cuantitativa. Además, el algoritmo RSA y el ECDSA modelan una firma con números. En la siguiente sección se define la cantidad de firmas a generar y se obtiene un promedio de estas métricas con estadística descriptiva.

Se estima la seguridad de los criptosistemas de forma cuantitativa y se mide con la complejidad del mejor algoritmo para resolver el problema matemático subyacente. Usualmente se aproxima a una función exponencial base 2. Sin embargo, la perdida de la clave privada pondría en jaque la autenticidad y repudio de un comprobante. Este tipo de vulnerabilidades son prácticas y no se consideran en este trabajo. Se asume un escenario ideal donde ningún usuario puede ser víctima del robo o perdida de su clave privada.

Por lo que se refiere a la propuesta para sellar los CFDI con curva elíptica se decidió utilizar el ECDSA. Como se ha dicho, la elección de este criptosistema es por su tamaño de claves reducido, eficiencia computacional y nivel de seguridad. Para realizar el prototipo se empleó el lenguaje de programación de alto nivel Mathematica en el ordenador portátil Dell y la documentación oficial de sus funciones criptográficas nativas. En cada función se especifican parámetros como el tamaño del claves, el criptosistema y el nombre de la curva. Es necesario recalcar que existe una limitación en la selección de curva dado que la versión 12.1 de Mathematica solo admite la curva *secp256k1*.

Con respecto al sello especificado por el Anexo 20 (Santín, 2017), se implementa con las funciones criptográficas nativas de Mathematica especificando los parámetros del criptosistema RSA, por ejemplo el esquema de rellenado. Considerando que comparar métodos programados en distintas plataformas puede introducir variabilidad, ambos se ejecutan en Mathematica.

6.3. Muestra y unidad de análisis

6.3.1. El sello digital de un CFDI

La unidad de análisis es la firma digital o un sello digital generado a partir de un CFDI. La firma digital es un fenómeno que permite tener un análogo a la firma autógrafa en documentos electrónicos, la cual depende tanto del mensaje como de las claves utilizadas.

Obtención del comprobante fiscal digital por Internet

En cuanto al comprobante fiscal, el SAT proporciona un ejemplo de CFDI a PAC y público en general. Así, a partir de este documento es posible generar la cadena original. De esta se obtiene la unidad de análisis aplicando el algoritmo de digestión SHA-256. Posteriormente se ingresa al algoritmo de firma RSA o el ECDSA para generar un sello digital. Un ejemplo de la versión 3.0 de un CFDI se obtuvo en el sitio web del SAT con extensión `xml`. Se incluye el CFDI empleado en el Anexo B.

Selección de la curva elíptica secp256k1

La curva elíptica secp256k1 es utilizada por tecnologías *blockchain* como Bitcoin, Ethereum y ARK. Fue propuesta por el SEC y actualmente se considera segura. Su conjunto de parámetros D se detalló en la sección de fundamentos. Además, esta curva es la única soportada por las funciones criptográficas de la versión 12.1 de Mathematica®.

Al ser una curva aceptada por múltiples protocolos y estándares tiene una gran interoperabilidad. Además, es constantemente atacada. Este persistente escrutinio da confianza para utilizarla. Así, en caso de un ataque exitoso o una nueva vulnerabilidad la gran comunidad de usuarios alertaría inmediatamente. Esto permite actuar rápido y prevenir intrusiones o fraudes. Por estas razones se eligió la curva secp256k1 para generar los sellos digitales con ECDSA.

Otro rasgo de importancia es su seguridad. Estándares como el IEEE (IEEE, 2000) y NIST (Barker, 2020) consideran que secp256k1 tiene un nivel de seguridad ligeramente superior al RSA módulo 2048. Este tamaño de módulo actualmente se utiliza por el SAT para generar sellos en los CFDI. Por lo tanto, se contrastan ambos criptosistemas teniendo un nivel de seguridad comparable y se evita el peligro de una propuesta vulnerable.

6.3.2. ¿Por qué 206 582 sellos?

Número de firmas digitales

En lo que toca a la muestra, se generaron 206 582 pares de claves y firmas digitales con el algoritmo RSA y el ECDSA. De acuerdo a datos de la AMEXIPAC (Asociación Mexicana de Proveedores Autorizados de Certificación, 2018), en México se expiden 17 millones de facturas al día. De este total el 96 % son procesadas por un PAC. Hasta el año 2018 el SAT tiene registrados 79 PAC. Si distribuimos equitativamente el número de comprobantes entre los PAC entonces cada uno procesaría $(0{,}96 \times 17\,000\,000)/79 = 206\,582$ CFDI al día. Para observar los beneficios que tendría un PAC, uno de los mayores generadores de CFDI, se generaron 206 582 sellos digitales.

Obtener una muestra representativa de CFDI resultó inviable por la burocracia, la limitación de cómputo para procesar los 17 millones de datos generados en un día y el cumplimiento de la ley de protección de datos personales. Si aceptamos que, en su forma última, los algoritmos de firma digital representan al mensaje como un número entonces poco importa el documento. Con todo, se procura utilizar el CFDI del Anexo B.

Se podría objetar que las 206 582 sellos son idénticos por obtenerse del mismo documento. Sin embargo, al generarse con distintas claves se tiene que cada uno es distinto. Más aún, se podría argumentar que la longitud de la cadena original varia. Ahora bien, se recuerda que la cadena original es procesada por la función de digestión SHA-256. Esta devuelve para cualquier entrada una cadena de 256 bits. Es por esto que firmar el mismo documento con diferentes claves vasta para generar múltiples sellos.

En Mathemtatica se utilizó la función `GenerateAsymmetricKeyPair[]` para crear diferentes pares de claves. De acuerdo a la documentación, la función genera de forma pseudoaleatoria un par de claves criptográficas. Se repitió esta función con cada criptosistema hasta tener 205682 claves públicas y sus correspondientes claves privadas. El procedimiento se detalla más adelante.

6.4. Procedimiento

6.4.1. Producir la cadena original

Para generar la *cadena original* el SAT dispone de un archivo `xslt` para transformar un CFDI en una cadena original. El archivo `xslt` se obtuvo a través de la página web oficial del SAT y se incluye una copia en el Anexo C. Los archivos `xlst` transforman documentos `xml`, en este caso el CFDI, en otros archivos `xml`, la cadena original.

Utilizando la línea de comandos de una *Raspberry Pi* se instala el programa `xlstproc`. La herramienta permite aplicar el formato definido por un `xslt` a un documento `xml`. Como se mencionó arriba, se descargó el CFDI y el archivo de transformación `xlst` en el Raspberry Pi. A continuación se ejecutó en una terminal el siguiente comando para generar la cadena original de la Tabla 6.1:

```
$ xlstproc cadenaoriginal_3_0.xlst ejemplocfdv3.xml
```

Una vez que se formó la cadena original es necesario procesarla por una función de digestión o *hash* $\mathcal{H}$. Se utiliza la función SHA-256 que produce una cadena de 256 bits. Para ser más específicos, se copio la cadena original a un cuaderno de Mathematica y se ejecutó la función nativa `Hash[cadenaOriginal,"SHA256"]`. Esto transforma la cadena

||3.0|2010-03-06T20:38:12|ing|reso|PAGO EN UNA SOLA EXHIBICION|488.50|488.50|PPL961114GZ1|PHARMA PLUS SA DE CV|AV. RIO MIXCOAC|No. 140|ACACIAS—BENITO JUAREZ|MEXICO, D.F.|Mexico|03240|AV. UNIVERSIDAD|1858|OXTOPULCO|DISTRITO FEDERAL|Mexico|03910|PEPJ8001019Q8|JUAN PEREZ PEREZ|AV UNIVERSIDAD|16 EDF 3|DPTO 101|COPILCO UNIVERSIDAD|COYOACAN|DISTRITO FEDERAL|Mexico|04360|1.0|CAPSULAS|VIBRAMICINA 100MG 10|244.00|244.00|1.0|BOTELLA|CLORUTO 500M|137.93|137.93|1.0|TABLETAS|SEDEPRON 250MG 10|84.50|84.50|IVA|0.00|0.00|IVA|16.00|22.07|22.07||

Tabla 6.1: Ejemplo de Cadena Original a partir de un CFDI versión 3.0.

original en un número entero de 256 bits para posteriores operaciones criptográficas. El resultado de convertir la cadena original de la Tabla 6.1 a un entero es

21159586336713373872137846405861447245017863645276105873016368938895716757977.

Como se ha dicho, se utilizó únicamente un CFDI y su respectiva cadena original. Ante la objeción de inexactitud por emplear un CFDI cabe señalar que la entrada de los algoritmos de firma es siempre constante debido a la función de digestión SHA-256. Es necesario recalcar que la entrada de los Algoritmos 4 y 10 no es la cadena original sino la digestión de 256 bits producida por la función SHA-256. Por tanto nunca se alteran las entradas de los algoritmos y tampoco su complejidad. Sin embargo, en caso de aceptarse la propuesta se debe obtener la cadena original de cada CFDI utilizando la versión más actualizada del archivo `xlst` proporcionado por el SAT y procesarse con la función digestión SHA-256.

6.4.2. Generación de claves pública y privada

La función `GenerateAsymmetricKeyPair` de Mathematica® genera pseudoaleatoriamente una clave privada y su correspondiente clave pública. Estas claves se utilizan como parámetros en otras funciones criptográficas nativas de Mathematica®. Todas se ejecutan en el ordenador portátil Dell. Conviene subrayar que los algoritmos implementados por Mathematica® son secretos y sus funciones cajas negras para el usuario. A pesar de esto, `GenerateAsymmetricKeyPair` debería de ser similar al Algoritmo 8 de la sección *Fundamentos*. En las siguientes secciones se especifican los pasos para generar las claves de los criptosistemas por comparar.

Claves RSA

La clave pública y privada se generaron con la función `GenerateAsymmetricKeyPair`. Se ejecutó esta función y se especificó el argumento del criptosistema RSA.

Para generar 206 582 pares de claves se utilizó la función nativa `Table`. Esta función genera una lista de n copias. Así, se obtuvo $n = 206\,582$ claves criptográficas RSA. La entrada ingresada en Wolfram Language es

```
Table[GenerateAsymmetricKeyPair["RSA"],n]
```

El tiempo de cada iteración se midió con la función `AbsoluteTiming`. La función `AbsoluteTiming[]` devuelve el número de segundos, en tiempo real, que tarda el computador en evaluar la instrucción. En concreto, el comando ingresado fue

```
Table[AbsoluteTiming[GenerateAsymmetricKeyPair["RSA"]],n]
```

Después se relacionó cada par de claves con su tiempo en un conjunto de datos o *dataset*. Esta base de datos se almacenó en el archivo `RSA206582keysDataset.m`.

Claves ECDSA

Consideremos ahora la generación de las claves para el ECDSA. Se precisó el parámetro criptosistema como `EllipticCurve` en la función `GenerateAsymmetricKeyPair`. Así mismo, se generaron 206 582 pares de claves con la función `Table[]` de Mathematica®. La orden completa fue

```
Table[GenerateAsymmetricKeyPair["EllipticCurve"],n]
```

con $n = 206\,582$.

Finalmente, el tiempo de cada iteración se registro con `AbsoluteTiming`. Se relaciono con su respectivo par de claves en una base de datos nombrada `ECC206582keysDataset.m`. La entrada final en Mathematica fue

```
Table[AbsoluteTiming[GenerateAsymmetricKeyPair["EllipticCurve"]],n]
```

6.4.3. Generación de sello digital para el comprobante fiscal digital por Internet

El siguiente punto trata la firma o sello de un CFDI con las claves previamente producidas. Para realizar una comparación justa de ambos métodos se crean, a partir de un CFDI, sellos digitales con el algoritmo de firma RSA y sellos con el ECDSA en Mathematica®. Esto en el mismo ordenador portátil Dell. El proceso general de ambos criptosistemas se muestra en el diagrama de flujo de la Figura 6.1. Con el fin de presentar el tiempo de computo, espacio de almacenamiento y tasa de transmisión en la sección de resultados, se guardan las 206 582 firmas RSA en un archivo con extensión `.m` y también las 206 582 firmas ECDSA en un archivo con extensión `.m`.

Sello con el algoritmo de firma RSA

Mathematica no cuenta con una función nativa para firmar expresiones con el algoritmo RSA. No obstante, a partir de la función nativa `Encrypt` y el Algoritmo 4 se codifica una nueva función llamada `sing`. Así tenemos que

```
sign[key_PrivateKey, expr_]:= Encrypt[key, Hash[expr,"SHA256"]]
```

De la expresión anterior tenemos el parámetro `key_PrivateKey` que es una de las 206 582 claves privadas almacenadas en `RSA206582keysDataset.m`. La variable `expr_` es la cadena original. Luego, estos parámetros son procesados por la función `Encrypt` que acepta como argumentos una clave privada `key` y un objeto a cifrar. Para ser preciso, el objeto a cifrar es la digestión SHA-256 de la cadena original y fue obtenida por la función nativa `Hash` de Mathematica.

Se utilizó la función `Table[]` para generar los 206 582 sellos digitales. En cada iteración se tomó el cuidado de cambiar la clave privada en `sing`. Así, cada uno de los sellos RSA es distinto. La entrada en Wolfram Language fue

```
Table[AbsoluteTiming[
sign[keysDataset[i]["PrivateKey"], cadenaOriginal]], {i, 1, 206582}]
```

El tiempo real para generar cada uno de los sellos se midió con la función `AbsoluteTiming`. Una vez registrado, se integró el sello y el tiempo a la base de datos que contiene las respectivas claves pública y privada. El archivo que las almacena se nombró `RSAsignaturesDataset.m`.

Sello con el ECDSA

Para generar sellos con el ECDSA, Mathematica cuenta con la función nativa `GenerateDigitalSignature[expr,key]`. El parámetro `expr` es la digestión SHA-256 de la cadena original mientras que `key` es una de las claves privadas generadas por `GenerateAsymmetricKeyPair["EllipticCurve"]`.

De igual manera se utilizó la función `Table[]` para iterar cada una de las claves privadas y generar un total de 206 582 sellos digitales con el ECDSA. Como se ha dicho, se prestó especial atención en cada repetición para intercambiar la clave privada y generar diversos sellos. El conjunto de claves del archivo `ECC206282keysDataset.m` se almacenó en la variable `keysDataset`.

Igualmente se midió el tiempo real de cada copia con `AbsoluteTiming`. Los resultados se unieron a sus respectivas claves pública y privada en la base de datos `ECCsignaturesDataset.m`. Así, la entrada completa en Mathematica para generar los sellos y su tiempo de cómputo es

```
  Table[AbsoluteTiming[
GenerateDigitalSignature[cadenaOriginal,
 keysDataset[i]["PrivateKey"],
 Method -> <|"Type" -> "EllipticCurve",
   "HashingMethod" -> "SHA256"|>]], {i, 1, 206582}]
```

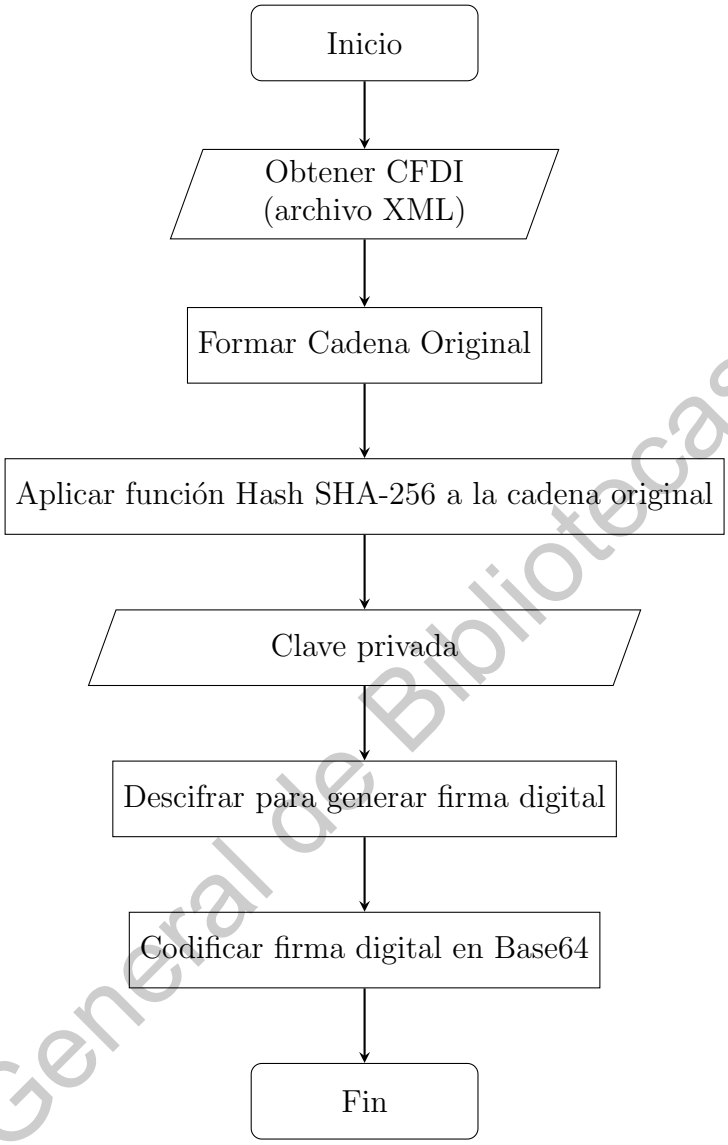

Figura 6.1: Diagrama de flujo para crear sellos digitales.

6.4.4. Medición de la tasa de transmisión de sellos digitales para comprobantes fiscales digitales por Internet

La tasa de transmisión de 206 582 sellos RSA y ECDSA se midió con una red punto a punto entre dos Raspberry Pi. Se conectó un cable Ethernet categoría 6 entre sus interfaces Gigabit Ethernet como lo muestra la Figura 6.2. Así mismo se configuró la dirección IP y máscara de red indicadas.

A continuación se instaló `iPerf3` en ambos Raspberry, una herramienta que reporta el uso real del ancho de banda en redes IP; también conocido como *network*

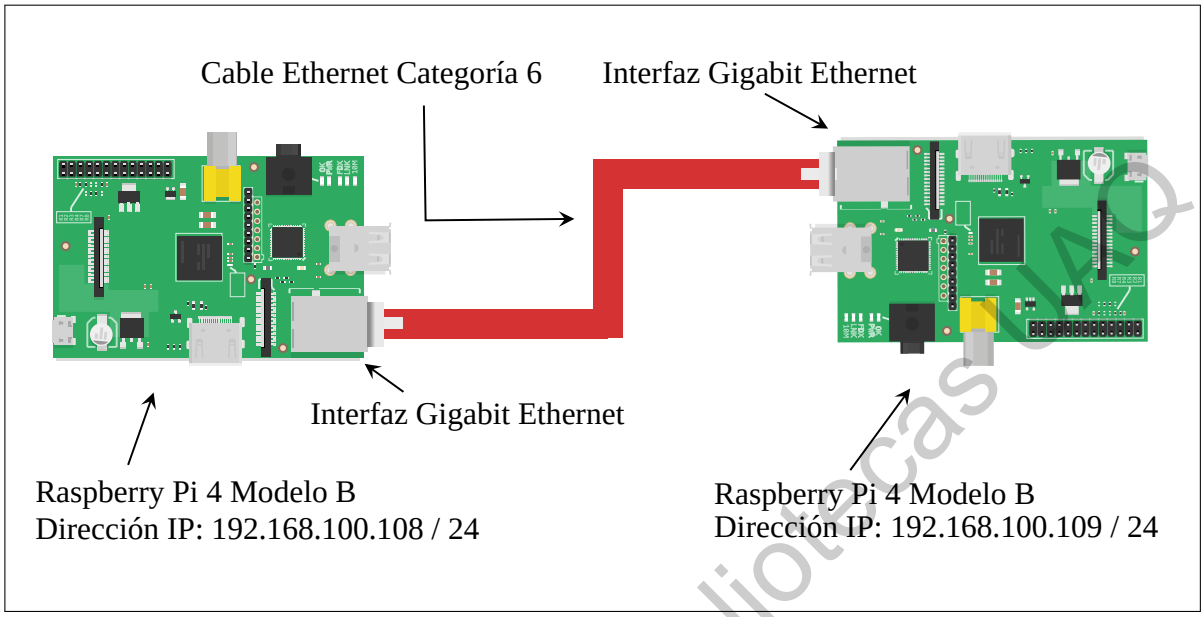

Figura 6.2: Conexión entre dos Raspberry Pi 4 Modelo B para medir la tasa de transmisión (*network throughput*) de los sellos digitales para CFDI.

throughput. Funciona estableciendo un cliente que envía datos a un servidor. Entre las opciones del cliente se puede definir la interfaz de red a utilizar y el archivo a transmitir.

Con respecto al Raspberry Pi que actuó como cliente y servidor, se alternó dependiendo el tipo de sello a transmitir. Para los sellos RSA el Raspberry Pi con dirección IP 192.168.100.109 se configuro como servidor mientras que el Raspberry Pi con la dirección 192.168.100.108 como cliente. Por otro lado, para los sellos ECDSA el Raspberry Pi con dirección IP 192.168.100.108 se definió como servidor y el Raspberry 192.168.100.109 como cliente. En los siguientes apartados se especifican los comandos utilizados en cada caso.

Medición de la tasa de transmisión de sellos RSA con iperf3

Como se ha dicho, el Raspberry Pi con dirección IP 192.168.100.109 se configuró como servidor `iPerf3` ejecutando el siguiente comando

```
iperf3 -s -B eth0
```

La opción `-s` inicia el dispositivo como servidor y la opción `-B` fuerza al Raspberry a utilizar la interfaz Gigabit Ethernet para contestar las peticiones del cliente.

Después, en el cliente se ejecutó el siguiente comando para iniciar la transmisión de los sellos basados en el algoritmo de firma RSA

```
iperf3 -c 192.168.100.109 -F RSAsignaturesDataset.m -J | tee
  iperf-test1-rsa.json
```

La opción `-c` permite conectarse como cliente al servidor 192.168.100.109. Se recuerda que el cliente tiene la dirección IP 192.168.100.108. La opción `-F` permite transmitir el archivo `RSAsignatureDataset.m` en lugar de datos pseudoaleatorios. La opción `-J` muestra la salida en formato `JSON`. Usamos el carácter pleca para dirigir la salida al programa `tee` que guarda los resultados en un archivo `JSON` para su posterior análisis.

Medición de la tasa de transmisión de sellos ECDSA con iperf3

De igual manera, para medir el ancho de banda al trasmitir los 206 582 sellos basados en el ECDSA se configuró como servidor `iPerf3` el Raspberry Pi con dirección IP 192.168.100.109. Esto se logró con el comando

```
iperf3 -s -B eth0
```

Así mismo, se configuró como cliente `iPerf3` el Raspberry Pi con dirección IP 192.168.100.108. El comando para realizarlo es

```
iperf3 -c 192.168.100.108 -F ECCsignaturesDataset.m -J | tee
  iperf-test1-ecc.json
```

La opción `-c` establece el Raspberry en modo cliente y la opción `-F` transmite los sellos almacenados en el archivo `ECCsignatureDataset.m`. Igualmente, la salida se dirigió a la aplicación `tee` para guardarse en un archivo `JSON` nombrado `ipref3-test1-ecc.json`. Los resultados se presentan en el siguiente capítulo *Análisis de resultados*.

6.5. Validez

En lo que sigue se defiende la validez de la propuesta de sello digital para CFDI con el ECDSA. Aunque el sello parece una cadena de caracteres aleatorios, existen algoritmos implementados en Mathematica que permiten verificar su autenticidad. No solo identifican al emisor del sello por su clave pública sino también detectan errores en la transmisión. Razón por la cual esta propuesta es tan efectiva como el actual sello basado en el algoritmo RSA.

6.5.1. Verificación del sello digital

Es imprescindible verificar que el sello de un CFDI fue generado con la respectiva clave privada del emisor. Para ello se utiliza la clave pública del firmante. El diagrama de flujo de la Figura 6.3 muestra, de forma general para ambos criptosistemas, el proceso para verificar un sello digital. Es necesario recalcar que se necesita la cadena original del CFDI, la clave pública y el sello digital para realizarse.

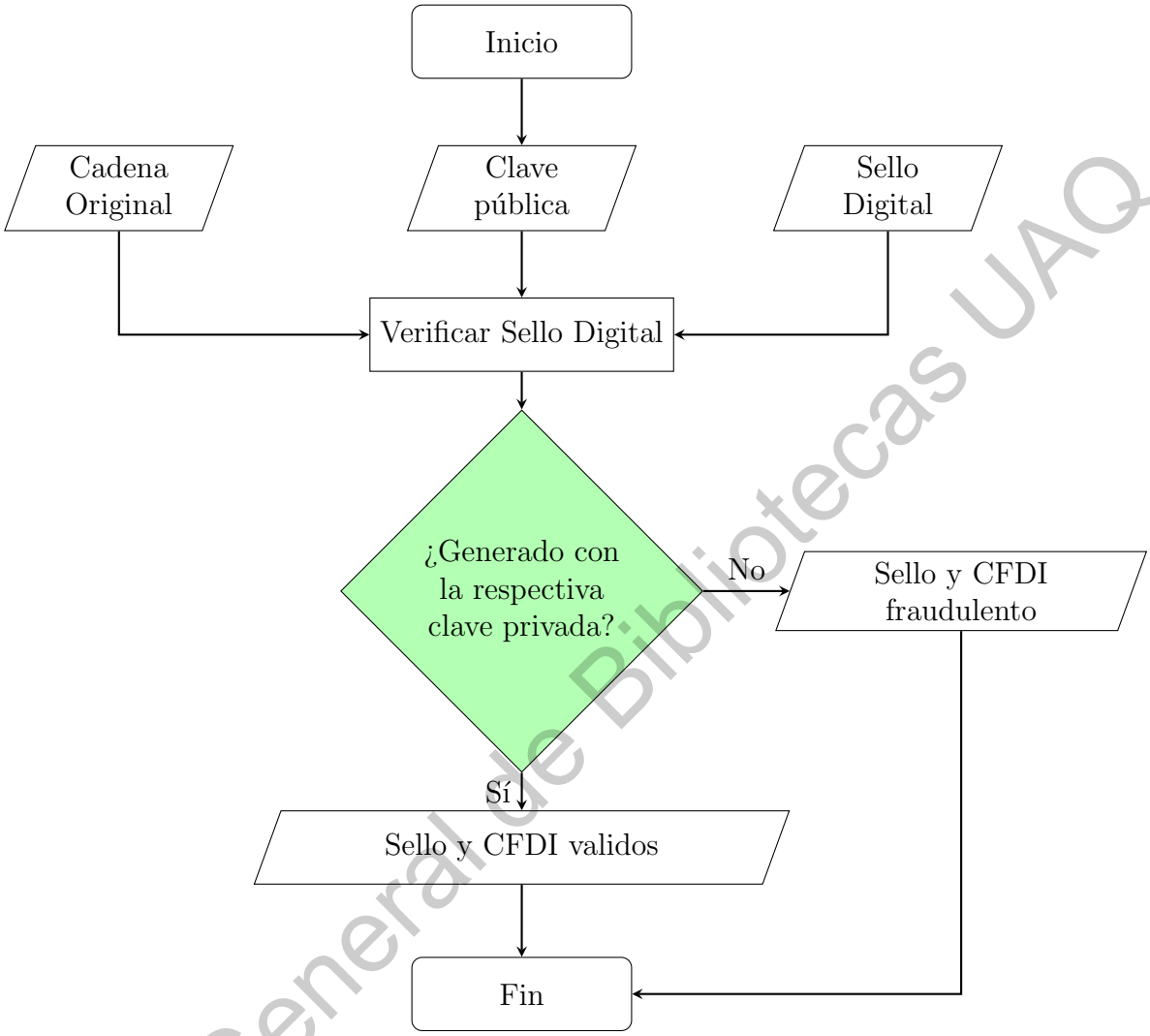

Figura 6.3: Diagrama de flujo para verificar sellos digitales

Verificación del sello basado en el algoritmo de firma digital RSA

De manera semejante, Mathematica no cuenta con una función nativa para verificar firmas digitales generadas por el algoritmo RSA. Para ello se definió la siguiente función basada en el Algoritmo 5

```
verify[key_PublicKey, expr_, signature_EncryptedObject]:=
  Decrypt[key, signature] == Hash[expr, "SHA256"]
```

El parámetro `key` es la clave pública asociada a la clave privada que generó la firma `signature`. Además, se proporciona la cadena original en `expr` para validar que su *hash* coincida con el descifrado de la firma. Esta función devuelve `True` si la digestión obtenida de la firma y la digestión SHA-256 calculada con la cadena original es igual. De lo contrario la función devuelve `False`.

Verificación del sello basado en el algoritmo de firma digital con curva elíptica

La función nativa `VerifyDigitalSignature[{expr, sig}, key]` verifica la autenticidad de los sellos creados con el ECDSA. Al igual que otras funciones nativas de Mathematica es una caja negra pero se basa en el Algoritmo 11. Recibe como parámetros una lista que contiene la cadena original `expr` junto con la firma `sig`. El parámetro `key` contiene la clave pública generada por la función `GenerateAsymmetricKeyPair["EllipticCurve"]`. Esta función devuelve `True` si la firma es valida y `False` en caso contrario.

Arquitectura de red

Con el propósito de comparar el ancho de banda al transmitir sellos generados con el algoritmo RSA y con el ECDSA se diseñó una arquitectura *punto a punto*. Conviene subrayar que estas redes aprovechan todo el ancho de banda disponible en la capa física. El diseño de la red se muestra en la Figura 6.2 y consiste en dos Raspberry Pi conectados por un cable Ethernet categoría 6. El cable soporta transmisiones de 10 000 megabits por segundo. En contraste, la interfaz de red en los Raspberry Pi modelo 4B es *Gigabit Ethernet*. Es decir, soporta velocidades de 1000 megabits por segundo. Por consiguiente un cuello de botella es improbable y se evita registrar una medición errónea.

6.6. Apoyos para el procesamiento de la información

Por lo que se refiere a programas de cómputo para analizar y graficar los datos se apoyó totalmente en Mathematica. Los tiempos de cómputo se obtuvieron con una función nativa. Además, los promedios se calculan con funciones estadísticas nativas. La manipulación de archivos `XML` y `JSON` es sencilla en este software.

6.6.1. Tiempo real de cómputo

Como se ha dicho Mathematica es el sistema de cómputo elegido para implementar los algoritmos de firma digital. Se justifica su uso por incluirse de forma gratuita en el sistema operativo de los Raspberry Pi. Es ideal para implementar prototipos computacionales pues cuenta con miles de funciones nativas que facilitan su diseño.

Sobre la medición al generar las claves de los criptosistemas y firmas digitales se utilizó la función nativa de Mathematica `AbsoluteTiming` que devuelve el *tiempo real* al evaluar una entrada. A diferencia de `Timing`, que solo reporta el tiempo en CPU,

`AbsoluteTiming` cronometra el tiempo total para evaluar una expresión. A fin de obtener una medida realista del tiempo total que requiere el equipo para firmar un CFDI se eligió `AbsoluteTiming`.

6.6.2. Aplicación para medir tasa de transmisión en red

Una vez definida la arquitectura de red se eligió la utilidad `iPerf3` para transmitir los sellos digitales. `iPerf3` es una herramienta para medir la velocidad de transmisión de paquetes TCP, UDP y SCTP; todos parte de la *familia de protocolos de Internet*. Su elección responde a ser una aplicación multiplataforma compatible con el sistema operativo NOOBS de Raspberry. Simula la transmisión de los sellos por Internet, exportar los resultados en archivos JSON y es gratuito. Además permite seleccionar el archivo a transmitir. En este caso `RSAsignaturesDataset` y `ECCsignaturesDataset`.

6.6.3. Importando y exportando datos

Las funciones nativas `Import` y `Export` de Mathematica permiten manipular datos de distintos formatos, desde archivos `XML` a `JSON`. `Import` introduce los datos como arreglos en Mathematica. Después se pueden aplicar funciones nativas a cada valor para su análisis.

En el caso de los datos producidos por `iPerf3` y almacenados en archivos `JSON`, se empleó la función nativa `Key` para extraer los valores de tiempo y bits por segundo. Para convertir los bits por segundo a *megabits por segundo* se divide entre 10^6. Esos resultados permitirán comparar de forma gráfica la transmisión en red de ambos sellos.

Los tiempos de cómputo, la clave pública, la clave privada, las firmas digitales y un campo lógico de verificación se concentraron en una base de datos o *dataset*. Como se ha dicho, estos se guardaron en un archivo con extensión `.m`. Es decir, se utilizó la función `Export[]` para producir un archivo de almacenamiento del criptosistema RSA y otro para el ECDSA.

6.6.4. El formato impreso del sello ECDSA

Ahora bien, la firma digital del ECDSA se conforma por dos valores, r y s. A diferencia del sello RSA, que solo genera un valor, ambas variables se concatenan para formar una cadena de bits que se codifica en Base64. Esto permite imprimir el sello en los CFDI de forma similar. Lo anterior se logra en Mathematica con la función nativa `BaseEncode`.

6.6.5. Gráficas y tablas

Para visualizar los datos se utilizaron las funciones nativas de Mathematica para graficar. Mathematica cuenta con muchas funciones para visualizar datos. Principalmente se usó la función `Histogram`, `ContourPlot` y `StackedListPlot`.

6.7. Estadística descriptiva

Los estadísticos de media para los tiempos de cómputo se calcularon con los datos de `iPerf3` y `AbsoluteTiming` en Mathematica. Se usó la función nativa para estadística descriptiva `Mean`. El tamaño de la entrada para generar un sello es invariante y se esperaría que el tiempo también lo fuera. Sin embargo, en la práctica existen ligeras variaciones. Por lo tanto, se calculan los tiempos promedios para comparar ambos tipos de sellos digitales.

6.8. Plan de presentación de los resultados

Los sellos RSA son codificados en Base64 para mostrarse como caracteres imprimibles. De la misma forma, el sello generado por el ECDSA se codifica en Base64 con el propósito de comparar visualmente su longitud.

El tiempo de cómputo, espacio de almacenamiento y nivel de seguridad se exponen en tablas. Conviene subrayar que el tiempo de cómputo se mide en segundos (s), el espacio de almacenamiento se mide en bits, bytes o megabytes. Finalmente, la seguridad es un valor que representa el exponente de una función exponencial base 2.

El rendimiento de red o *throughput* se presenta en una tabla. También en un gráfico de área apilada para su comparación. Esta variable se mide en Megabits por segundo (Mbits/s).

Resumiendo, en este capítulo se presentó la serie de pasos para obtener, procesar y mostrar los sellos digitales basados en el ECDSA. Además, se aclara el procesamiento de los datos obtenidos por `iPerf3` y `AbsoluteTiming`. Definitivamente la principal herramienta de cómputo para todo esto fue Mathematica. También se justifican varias cuestiones que los críticos podrían objetar. Los resultados obtenidos se presentan en el siguiente capítulo.

Capítulo 7

Análisis de Resultados

En este capítulo se muestra la propuesta de sello digital para CFDI codificado en Base 64. También se incluyen tablas y gráficas para comparar el tiempo de computo, espacio de almacenamiento y uso del ancho de banda entre los sellos basados en ECDSA y los sellos RSA. Finalmente se discuten las bondades de utilizar nuestra propuesta de sellos en CFDI.

La Tabla 7.1 muestra el nivel de seguridad y tamaño de los sellos basados en el algoritmo de firma RSA módulo 2048 y en el ECDSA usando la curva *secp256k1*. El valor que contiene la columna *Nivel de seguridad* es una aproximación al número de operaciones de un ataque con el mejor algoritmo general para resolver el problema matemático subyacente. En el caso del RSA es la *la criba general de campos numérico* y para el ECDLP el Algoritmo *baby-step giant-step*. Para comprender mejor, se espera que un ataque para resolver el problema RSA realice 2^{116} operaciones mientras que para el ECDLP serían 2^{128}. Es decir, una diferencia de 2^{12} veces 2^{116} operaciones para romper la nueva propuesta con curva elíptica. Esta diferencia se puede apreciar en la Figura 7.1. El eje y, el número de operaciones realizadas en una unidad de tiempo, está en escala logarítmica.

A pesar de un leve incremento en la clave pública del ECDSA, una reducción en la clave privada se aprecia en la Tabla 7.1. Si sumamos el tamaño de ambas claves y las comparamos con el criptosistema RSA el almacenamiento del par se reduce aproximadamente un 63 %.

Así mismo, esta Tabla 7.1 proporciona el espacio de almacenamiento ocupado por las firmas o *sellos* digitales. El tamaño del nuevo sello con el ECDSA es significativamente menor en comparación con el actual sello RSA. Se aprecia, en la columna *Firma digital*, una reducción en tamaño del 25 %. Como se ha dicho, la firma que genera el ECDSA está formada por la pareja de valores (r, s) mientras que la firma generada por RSA únicamente por el valor s.

Figura 7.1: Diferencia de seguridad entre la firma RSA y la propuesta con el ECDSA.

Tabla 7.1: Nivel de seguridad y espacios de almacenamiento de los criptosistemas RSA y ECC usados.

Criptosistema	**Nivel de seguridad** (2^n bits)	**Clave Pública** (bits)	**Clave Privada** (bits)	**Firma Digital** (bytes)
RSA-2048	116	17	2048	s=256
ECC(secp256k1)	128	256	512	r=32, s=32

Se puede ver en la Tabla 7.2 el tiempo promedio para generar el par de claves del criptosistema RSA y el ECDSA. Así mismo el tiempo promedio para producir y verificar sus sello digitales. Cada columna representa el tiempo en segundos.

Los tiempos en la columna *Generación de claves* se obtuvieron al evaluar la función nativa `GenerateAsymmectricKey` con `AbsoluteTiming`. La demora en las claves RSA ocurrió debido a la búsqueda de los factores primos del módulo n congruente con el exponente público $e = 2^{16} + 1$. En contraste, una vez definidos los parámetros del ECDSA es más sencillo encontrar un múltiplo del punto generador de la curva G como clave privada y su producto como clave pública.

La columna *Generación de firma digital* contiene el tiempo promedio en producir un sello con el algoritmo de firma RSA y el ECDSA. Los resultados, como se muestran en la Tabla 7.2, indican que un sello para un CFDI con el ECDSA es más eficiente que con el algoritmo de firma RSA.

Contrario a lo esperado, la verificación de los sellos ECDSA fue superior a los sellos RSA. La función nativa de Mathematica para verificar las firmas del ECDSA es una

implementación muy eficiente pues tardó en promedio décimas de segundo. La verificación de firmas RSA tarda más de un segundo. Aunque en la práctica podría ser imperceptible, se trata de una diferencia en orden de magnitud 10^1.

Tabla 7.2: Tiempo promedio de computo en Mathematica® para generar claves, firmas digitales y verificarlas.

Criptosistema	**Generación de claves** (s)	**Generación de firma digital** (s)	**Verificación de firma digital** (s)
RSA-2048	2.0844	1.2525	1.1380
ECC(secp256k1)	0.0648	0.1286	0.0465

Ejemplos del sello digital para CFDI se muestran en la Tabla A.1. Comparando los datos de la Tabla 7.1 y 7.3 se comprueba una diferencia en tamaño. Por otra parte, ambas firmas se codificaron en Base64 para imprimirse en un CFDI. Teniendo en cuenta que el ECDSA devuelve dos variables, se concatenó el valor r y el valor s para producir una cadena de 64 bits. Después se codificó esta con los lineamientos del estándar Base64. A simple vista el sello codificado del ECDSA es menor al del RSA.

Tabla 7.3: Firma Digital codificada en caracteres imprimibles Base64 para ser usada como sello digital en CFDI

RSA-2048	**ECC (spec256k1)**
mOrx99ocpD9dSCiFforIbM7M9CHUbp6 SNA8q+SvRjviK0q/PXscUcBklkSEk11 1sXkyKup1JkG7qvf0T41ngLNcnCln5US kwe2AKAojbvrYYR1q5JY+F/n9kbb5+ xu/Ajy7VngX1VaswQwfGolNR/9drm4lk ZklrUXBsjYLx191iA/CzY2r15MZw2Hn V/0WGOPucMHC1H3+ootqLAYVUg/l ajdIfBbe0QVf6HF5aY2Q8Fg3hWuwctPs EUlyWRpOe5nOjiQivDYeMfGumJsoBF Ta4WjPfJaGNFGhwYaYwaMxTdK1TJ wNSB01BvPOWbT23XgF9BmxDvHUC lVlJzC4Vlw==	BVah0V2AsAIRsHHT2qZqoPBO4MbEF AuxgxgxaDZHvLI=HRqakkgYCViqT2K Xba7xN6CannQidJrfEsGMk4WL/mU=

Finalmente, el uso del ancho de banda registrado por `iPerf3` se presenta en la Tabla 7.4. La tabla muestra que el archivo con los 206 582 sellos basados en el ECDSA tiene un tamaño de 62 MB mientras que el archivo con los sellos RSA un tamaño de 92 MB. El resultado más llamativo que emerge de los datos es la diferencia de solo 30 milisegundos en el tiempo de transmisión de ambos documentos. En teoría las interfaces Gigabit Ethernet de los Raspberrys junto con el cable Ethernet categoría 6 soportan velocidades de 1000 Mbits/s. Por lo tanto, ambos archivos tendrían que demorar lo mismo. Sin embargo, notamos que se aprovecho de forma eficiente el ancho de banda del canal al enviar los sellos del ECDSA. En consecuencia, se registró un menor tiempo

Tabla 7.4: Resultados de la medición del uso real del ancho de banda entre dos Raspberry Pi con iPerf.

Criptosistema	**Número de Firmas**	**Tamaño del archivo (MB)**	**Segundos de transmisión (s)**	**Tasa de bits (Mbits/s)**
ECC	206 582	62	0.5386	920.57
RSA	206 582	92	0.8580	854.45

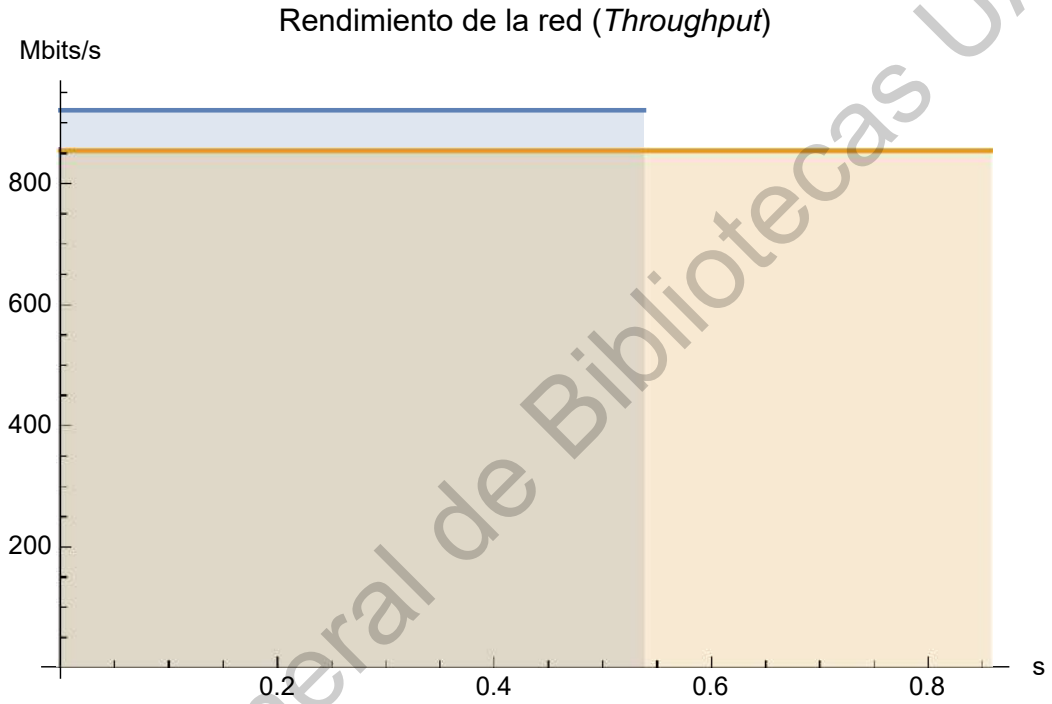

Figura 7.2: Uso real del ancho de banda entre dos Raspberry Pi conectados a través de un cable Ethernet categoría 6 al trasmitir 206 582 sellos digitales RSA y ECDSA.

de transmisión. Hasta ahora se desconoce por qué se presenta este comportamiento.

Una forma alternativa de mostrar el uso del ancho de banda es con la gráfica de la Figura 7.2. Las áreas ensombrecidas representan el caudal de los sellos a través del cable Ethernet categoría 6. La tasa de transferencia del ECDSA se muestra superpuesta sobre la tasa del RSA en color azul. En general estos datos indican que el sello basado en el ECDSA es superior al actual sello RSA.

Capítulo 8

Conclusiones

En México, los CFDI son importantes para el cumplimiento de las obligaciones fiscales. El mecanismo para garantizar su autenticidad son los sellos digitales basados en el algoritmo de firma RSA. Aunque este criptosistema sigue siendo seguro, el ECC presenta mayor eficiencia en los tiempos de procesamiento, espacios de almacenamiento y uso del ancho de banda.

El objetivo inicial de este trabajo era desarrollar una alternativa de sello digital para CFDI con el ECDSA en Mathematica y se logró. La posibilidad de sellar un CFDI con el ECDSA corrobora la hipótesis planteada al inicio. En los casos que ameritan, se implementaron algoritmos criptográficos descritos en el capítulo 5 *Fundamentos*. Otros se encontraron como funciones nativas de Mathematica. Así mismo se comparó el tiempo de procesamiento, espacio de almacenamiento y uso real del ancho de banda entre ambos métodos. Esta investigación halló la propuesta de sello con el ECDSA mejor en todos los aspectos.

La literatura existente establece que el proceso de verificación de una firma RSA es más rápido que el del ECDSA. Por el contrario, los resultados de este estudio muestran notablemente un menor tiempo respecto al método RSA. Esta diferencia podría deberse al tamaño de las claves. Para el tamaño de 2048 bits en RSA la exponenciación modular en la verificación es más costosa que la aritmética de puntos en una curva elíptica con un nivel de seguridad similar. Este es un punto a tener en cuenta ya que incrementar la seguridad implica aumentar el tamaño de las claves.

Aunque ninguna arquitectura de red o diseño se han descrito para comparar la tasa de transmisión de ambos métodos, estudios previos han sugerido un rendimiento de red superior con el ECDSA. Sorprendentemente, con una arquitectura punto a punto entre dos Raspberry Pi se encontró que la velocidad de transmisión para los sellos basados en el ECDSA fue mayor al de los sellos RSA. Aún teniendo ambos un tamaño menor al ancho de banda disponible en un enlace Gigabit Ethernet. Esta diferencia aparente podría explicarse por errores en alguna interfaz Gigabit Ethernet de los Raspberrys. Es

por esto que los datos deben ser interpretados con cautela pues se carece de otro par para realizar una replica y confirmarlo.

En cuanto al tiempo de generación de claves y firmas digitales, resultados similares han sido reportados en otras publicaciones. Se registró un leve incremento en los tiempos, consecuencia del sistema de cómputo Mathematica y el procesador utilizado. Aún así, los datos parecen apoyar la superioridad en la generación de claves y firmas digitales del ECDSA respecto al algoritmo RSA.

Aunque es inusual utilizar Mathematica para comparar criptosistemas, la facilidad que ofrece para desarrollar prototipos computacionales y evaluarlos motivaron su elección. Habitualmente se emplean códigos o bibliotecas en lenguaje C/C++. La librería criptográfica más popular es OpenSSL. Esta misma es la que implementan las funciones nativas de Mathematica. Esto asegura que resultados similares se obtengan al implementarse en otro lenguaje de programación.

Como prueba de concepto, la curva *secp256k1* demostró ser útil y segura para sellar los CFDI. Sin embargo, al implementarse un sistema real de timbrado de CFDI basado en el ECDSA se deben analizar con cuidado los parámetros y la seguridad de la curva a emplear. Aún si se encuentra en un estándar. Todo esto con el fin de poner en marcha un servicio infalible y eficiente.

La propuesta aquí presentada cumple con los requisitos por ley de autenticidad, no repudio, integridad y certidumbre de origen. El ECDSA es aceptado por los certificados del estándar X.509; mismos que el SAT utiliza. Es decir, un cambio abrupto en la PKI es innecesario para su puesta en marcha. De ahí que la transición a este algoritmo por parte del SAT y los PAC no debería suponer complicación alguna. Los resultados de este estudio tienen una serie de implicaciones importantes para el futuro de la firma digital en México, sobre todo en el sello del CFDI.

Capítulo 9

Recomendaciones

Sería interesante tomar de forma aleatoria una muestra de CFDI entre todos los tributarios del RFC, aplicar ambos métodos de timbrado y compararlos. Esto sería de gran ayuda para robustecer el diseño experimental y confirmar los resultados presentados. El desafío es obtener la información del SAT. De conseguirse, se necesitaría diseñar un sistema de protección de datos personales para el contenido de los comprobantes. A menos de contar con respaldo institucional, este experimento representa un verdadero desafío.

Al igual que todos los criptosistemas a lo largo de la historia, el algoritmo de firma ECDSA será roto en el futuro. Esto al encontrar un algoritmo eficiente para resolver el ECDLP o construir una computadora cuántica capaz de ejecutar el *algoritmo de Shor* (Shor, 1997). Este último parece la mayor amenaza en el mediano y largo plazo. Por ello se sugieren líneas de investigación en criptografía postcuántica y específicamente en sellos digitales para CFDI resistentes a esta amenaza. De manera fortuita, una de las alternativas es la criptografía con isogenia entre curvas elípticas supersingulares.

En teoría, el algoritmo de Shor es capaz de resolver el ECDLP y el problema de factorización entera para números primos grandes en tiempo polinomial (Shor, 1997). Si se busca factorizar un número de n bits se necesita una computadora cuántica con $3n$ qubits (Monz y col., 2016). Aunque tal dispositivo es inexistente, la eficiencia y seguridad del ECC y del criptosistema RSA se ha puesto en duda (Bernstein, 2009). El NIST lanzó en el año 2016 un concurso para encontrar sucesores de los criptosistemas de clave pública resistente a ataques de una computadora cuántica. En julio de 2020 se anunciaron los tres finalistas para algoritmos de firma digital junto con otras tres alternativas. Casi todos basados en *la reducción de rejillas* y *criptografía multivariable*. Hasta la fecha no se tiene un ganador para remplazar el RSA o el ECC. Al tratarse de una tecnología de difícil acceso, pocos criptoanálisis se han llevado a cabo. Por lo que más investigación necesita ser realizada.

Google demostró en 2019 la supremacía cuántica al lograr que su procesador cuánti-

co *Sycamore* venciera a una supercomputador convencional resolviendo el problema del muestreo de un circuito cuántico pseudoaleatorio (Arute y col., 2019). Construyeron un procesador de 53 qubits programables para esta tarea. Aunque la existencia en el corto plazo de una computadora cuántica con el número de qubits necesarios para discontinuar RSA y ECDSA parece poco factible, se recomienda realizar investigación sobre algoritmos de firma digital resistentes a computadores cuánticos. Además de implementaciones para generar sellos de CFDI seguros.

Aunque ineficiente en comparación con otros algoritmo de firma digital postcuánticos, es interesante notar una propuesta basada en otro problema con curvas elípticas. Para ser precisos, encontrar un camino dentro del grafo de isogenias de curvas supersingulares (De Feo y col., 2011). Esto ha permitido desarrollar firmas digitales con tamaños de clave pequeños (Yoo y col., 2017) (Galbraith y col., 2017) (De Feo y col., 2020). Se conjetura que este problema es difícil para computadoras clásicas y cuánticas. Por ello, como trabajo a futuro se plantea desarrollar sellos digitales postcuánticos para CFDI basados en el problema de buscar caminos en grafos de isogenias. Sin embargo, es difícil que estos métodos lleguen a ser tan eficientes como la criptografía de curva elíptica.

Referencias

Arute, F., Arya, K., Babbush, R., Bacon, D., Bardin, J. C., Barends, R., Biswas, R., Boixo, S., Brandao, F. G., Buell, D. A. Y col. (2019). [Supremacía cuántica usando un procesador superconductor programable] Quantum supremacy using a programmable superconducting processor. *Nature*, *574*(7779), 505-510.

Asociación Mexicana de Proveedores Autorizados de Certificación. (2018). *Servicios fiscales digitales en México: Evolución* (inf. téc.). AMEXIPAC. https://www.amexipac.org/assets/serviciosfiscalesdigitalesen-mexico-evolucion-2018.pdf

Bach, E., Shallit, J. O., Shallit, J. & Jeffrey, S. (1996). *[Algorítmica teoría de números: algoritmos eficientes] Algorithmic number theory: Efficient algorithms* (Vol. 1). MIT press.

Barker, E. (2020). *National Institute of Standards and Technology Special Publication 800-57 Part 1, Revision 5* (inf. téc.). National Institute of Standards y Technology.

Bernstein, D. J. (2009). [Introducción a la criptografía postcuántica] Introduction to post-quantum cryptography (D. J. Bernstein, J. Buchmann & E. Dahmen, Eds.; 1.ª ed.). En D. J. Bernstein, J. Buchmann & E. Dahmen (Eds.), *[Criptografía postcuántica] Post-Quantum Cryptography* (1.ª ed.). Berlin, Heidelberg, Springer Berlin Heidelberg.

Bernstein, D. J., Duif, N., Lange, T., Schwabe, P. & Yang, B. Y. (2011). [Firmas rápidas y de máxima seguridad] High-speed high-security signatures, En *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, Springer, Berlin, Heidelberg. http://link.springer.com/10.1007/978-3-642-23951-9_9

Boudot, F., Gaudry, P., Guillevic, A., Heninger, N., Thomé, E. & Zimmermann, P. (2020). [Comparando la dificultad entre factorizar y el logaritmo discreto: un experimento de 240 dígitos] Comparing the difficulty of factorization and discrete logarithm: a 240-digit experiment [https://eprint.iacr.org/2020/697].

Brown, D. (2009). *[Estándares de criptografía eficiente, SEC1: criptografía de curva elíptica] Standards for Efficient Cryptography, SEC 1: Elliptic Curve Cryptography* (inf. téc.). Certicom Corp.

Caelli, W. J., Dawson, E. P. & Rea, S. A. (1999). [PKI, criptografía de curva elíptica y firmas digitales] PKI, elliptic curve cryptography, and digital signatures. *Computers & Security*, *18*(1), 47-66.

Cámara de Diputados del H. Congreso de la Unión. (2014). Reglamento de la Ley de Firma Electrónica Avanzada.

Centro Nacional de Metrología. (2003). *Sistema internacional de unidades (SI)* (inf. téc.). Centro Nacional de Metrología.

Chavez, M. A. L. & Henriquez, F. R. (2015). [Vulnerabilidades de seguridad en la factura digital por Internet de México] Security vulnerabilities of the Mexican digital invoices by internet, En *2015 International Conference on Computing Systems and Telematics (ICCSAT)*. IEEE.

Cilleruelo, J. M. (2000). La demostración elemental del teorema de los números primos, En *Las matemáticas del siglo XX*. España, Sociedad Canaria Isaac Newton de Profesores de Matemáticas.

Costello, C. (2012). [Criptografía basada en emparejamientos para principiantes] Pairings for beginners.

De Feo, L., Jao, D. & Plût, J. (2011). [Hacia criptosistemas cuántico resistentes con isogenia de curva elíptica supersingular] Towards quantum-resistant cryptosystems from supersingular elliptic curve isogenies [https://eprint.iacr.org/2011/506].

De Feo, L., Kohel, D., Leroux, A., Petit, C. & Wesolowski, B. (2020). [SQISign: firmas postcuánticas compactas con cuaterniones e isogenias] SQISign: Compact Post-quantum Signatures from Quaternions and Isogenies, En *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, Springer Science; Business Media Deutschland GmbH.

Diffie, W. & Hellman, M. (1976). [Nuevas direcciónes en criptografía] New directions in cryptography. *IEEE transactions on Information Theory*, *22*(6), 644-654.

Erdös, P. (1949). [Sobre un nuevo método en teoría elemental de numeros que lleva a una demostración elemental del teorema de los números primos] On a new method in elementary number theory which leads to an elementary proof of the prime number theorem. *Proceedings of the National Academy of Sciences of the United States of America*, *35*(7), 374.

Fraleigh, J. B. (1998). *Álgebra Abstracta*. México, Addison-Wesley Iberoamericana.

Fúster Sabater, A., Hernández Encinas, L., Martín Muñoz, A., Montoya Vitini, F. & Muñoz Masqué, J. (2012). *Criptografía, protección de datos y aplicaciones*. RA-MA, Madrid.

Galbraith, S. D., Petit, C. & Silva, J. (2017). [Protocolos de identificación y esquemas de firma basados en el problema de isogenia supersingular] Identification Protocols and Signature Schemes Based on Supersingular Isogeny Problems. *Journal of Cryptology*, *33*(1), 130-175.

Gallant, R. P., Lambert, R. J. & Vanstone, S. A. (2001). [Multiplicación de punto rápida en curvas elípticas con endomorfisos eficientes] Faster point multiplication on elliptic curves with efficient endomorphisms, En *Annual International Cryptology Conference*. Springer.

García, V. G., Henríquez, F. R. & Cortés, N. C. (2008). [Seguridad de los documentos fiscales digitales mexicanos] On the security of mexican digital fiscal documents. *Computación y Sistemas*, *12*(1), 25-39.

Garey, M. R. & Johnson, D. S. (1990). *[Computadoras e intratabilidad; una guía a la teoría NP-completo] Computers and Intractability; A Guide to the Theory of NP-Completeness*. USA, W. H. Freeman & Co.

Gayoso, M. V., Hernández, E. L. & Martín, M. A. (2018). *Criptografía con curvas elípticas*. Madrid, Consejo Superior de Investigaciónes Científicas.

González, G. F. (2004). *Factorización de polinomios en campos finitos y su aplicación a la criptografía* (Tesis de maestría). Universidad Autónoma de Querétaro.

González-Garcıa, V., Rodrıguez-Henrıquez, F. & Cruz-Cortés, N. (2007). [Análisis de seguridad a los documentos fiscales digitales en el Servicio de Administración Tributaria mexicano] Security Analysis of the Digital Fiscal Documents in the Mexican Tributary Administration System, En *Magno Congreso Internacional de Computación CIC-IPN, Noviembre del*.

Herstein, I. N. (1980). *Álgebra moderna*. DF, México, Trillas.

IEEE. (2000). [IEEE estándar de especificaciones para criptografía de clave pública] IEEE Standard Specifications for Public-Key Cryptography. *IEEE Std 1363-2000*, 1-228.

Jansma, N. & Arrendondo, B. (2004). [Comparando el rendimiento de firmas digitales con curva elíptica y RSA] Performance comparison of elliptic curve and RSA digital signatures.

Johnson, D., Menezes, A. & Vanstone, S. (2001). [El algoritmo de firma digital con curva elíptica (ECDSA)] The elliptic curve digital signature algorithm (ECDSA). *International journal of information security*, *1*(1), 36-63.

Josefsson, S. (2006). *[La codificación de datos en Base16 Base32 y Base64] The Base16, Base32, and Base64 Data Encodings* (RFC N.º 4648). RFC Editor. RFC Editor. https://www.hjp.at/doc/rfc/rfc4648.html#sec_4

Katz, J. & Lindell, Y. (2021). *[Introducción a la criptografía moderna]Introduction to Modern Cryptography* (3.ª ed.). Boca Raton, Florida, CRC Press.

Kleinjung, T., Aoki, K., Franke, J., Lenstra, A. K., Thomé, E., Bos, J. W., Gaudry, P., Kruppa, A., Montgomery, P. L., Osvik, D. A., Te Riele, H., Timofeev, A. & Zimmermann, P. (2010). [Factorizando un módulo RSA de 768 bits] Factorization of a 768-Bit RSA modulus, En *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*.

Koblitz, N. (1987). [Criptosistemas de curva elíptica] Elliptic curve cryptosystems. *Mathematics of computation*, *48*(177), 203-209.

Koblitz, N. (1994). *[Un curso en teoría de numeros y criptografía] A course in number theory and cryptography* (Vol. 114). Springer Science & Business Media.

Koblitz, N., Menezes, A. & Vanstone, S. (2000). [Situación de la criptografía con curva elíptica] The state of elliptic curve cryptography. *Designs, codes and cryptography*, *19*(2), 173-193.

Lenstra, A. K. (2005). [Notación L] L-Notation. En H. C. A. van Tilborg (Ed.), *[Enciclopedia de criptografía y seguridad] Encyclopedia of Cryptography and Security* (pp. 358-358). Boston, MA, Springer US. https://doi.org/10.1007/0-387-23483-7_237

Lenstra, A. K. & Verheul, E. R. (2001). [Seleccionando tamaños para claves criptográficas] Selecting cryptographic key sizes. *Journal of cryptology*, *14*(4), 255-293.

Ley de Firma Electrónica Avanzada [https://www.dof.gob.mx/nota_detalle.php?codigo=5228864&fecha=11/01/2012]. (2012).

Leyland, P. (2005). [Criba de cuerpos numérico] Number Field Sieve. En H. C. A. van Tilborg (Ed.), *[Enciclopedia de criptografía y seguridad] Encyclopedia of Cryptography and Security* (pp. 430-433). Boston, MA, Springer US. https://doi.org/10.1007/0-387-23483-7_255

Lidl, R. & Niederreiter, H. (1994). *[Introducción a Campos Finitos y sus Aplicaciones] Introduction to Finite Fields and their Applications*. Cambridge, United Kingdom, Cambridge University Press.

Mahto, D. & Yadav, D. K. (2017). [RSA y ECC: un análisis comparativo] RSA and ECC: a comparative analysis. *International journal of applied engineering research*, *12*(19), 9053-9061.

Miller, V. S. (1985). [Uso de curva elíptica en criptografía] Use of elliptic curves in cryptography, En *Conference on the theory and application of cryptographic techniques*. Springer.

Monz, T., Nigg, D., Martinez, E. A., Brandl, M. F., Schindler, P., Rines, R., Wang, S. X., Chuang, I. L. & Blatt, R. (2016). [Ejecución de un algoritmo de Shor escalable] Realization of a scalable Shor algorithm. *Science*, *351*(6277), 1068-1070.

Moriarty, K., Kaliski, B., Jonsson, J. & Rusch, A. (2016). *[PKCS #1: Especificaciones criptográficas RSA versión 2.2.] PKCS #1: RSA Cryptography Specifications Version 2.2* (RFC N.º 8017).

Newell, D. B. & Tiesinga, E. (2019). *[Sistema Internacional de Unidades (SI)] The international system of units (SI)* (inf. téc. N.º 272). National Institute of Standards and Technology. Gaithersburg, MD. https://doi.org/10.6028/NIST.SP.330-2019%20https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.330-2019.pdf

Newman, D. J. (1980). [Demostración analítica simple del teorema de los números primos] Simple analytic proof of the prime number theorem. *The American Mathematical Monthly*, *87*(9), 693-696.

Paar, C. & Pelzl, J. (2009). *[Comprendiendo la criptografía: un libro de texto para estudiantes y practicantes] Understanding cryptography: a textbook for students and practitioners*. Springer Science & Business Media.

Porechna, D. (2020). [Dos líneas de código para un cifrado a prueba de balas: avances en criptografía para Wolfram Language] Two Lines of Code to Bulletproof Encryption: Advancements in Cryptography Development in the Wolfram Language.

Rivest, R. L., Shamir, A. & Adleman, L. (1978). [Método para obtener firmas digitales y criptosistemas d clave pública] A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, *21*(2), 120-126.

Robshaw, M. & Yin, Y. L. (1997). *[Criptosistema de curva elíptica] Elliptic curve cryptosystems* (inf. téc.). RSA Laboratories.

Santín, Q. O. A. (2017). ANEXO 20 de la Resolución Miscelánea Fiscal para 2017. *Ciudad de México*, Diario Oficial de la Federación. https://www.dof.gob.mx/nota_detalle.php?codigo=5468849&fecha=10/01/2017

Schoof, R. (1985). [Curvas elípticas sobre campos finitos y cálculo de raices cuadradas módulo p] Elliptic curves over finite fields and the computation of square roots mod p. *Mathematics of computation*, *44*(170), 483-494.

Schoof, R. (1995). [Contando puntos de una curva elíptica sobre campos finitos] Counting points on elliptic curves over finite fields. *Journal de théorie des nombres de Bordeaux*, *7*(1), 219-254.

Secretaría de Economía, Secretaría de la Función Pública & Servicio de Administración Tributaria. (2016). Disposiciones Generales de la Ley de Firma Electrónica Avanzada.

Selberg, A. (1949). [Demostración elemental del teorema de los números primos] An elementary proof of the prime-number theorem. *Annals of Mathematics*, 305-313.

Shor, P. W. (1997). [Algoritmos de tiempo polinomial para factorización entera y logaritmos discretos con una computadora cuántica] Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM Journal on Computing*, *26*(5), 1484-1509.

Solinas, J. A. (2000). [Aritmética eficiente en curvas de Koblitz] Efficient arithmetic on Koblitz curves, En *[Hacia un cuarto de siglo con criptografía de clave pública] Towards a Quarter-Century of Public Key Cryptography*. Springer.

Suárez-Albela, M., Fernández-Caramés, T. M., Fraga-Lamas, P. & Castedo, L. (2018). A practical performance comparison of ECC and RSA for resource-constrained IoT devices, En *2018 Global Internet of Things Summit (GIoTS)*. IEEE.

Toradmalle, D., Singh, R., Shastri, H., Naik, N. & Panchidi, V. (2018). [Prominencia del ECDSA sobre el algoritmo de firma digital RSA] Prominence Of ECDSA Over RSA Digital Signature Algorithm, En *2018 2nd International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud)(I-SMAC) I-SMAC (IoT in Social, Mobile, Analytics and Cloud)(I-SMAC), 2018 2nd International Conference on*. IEEE.

Trappe, W. & Washington, L. C. (2006). *[Introducción a la criptografía: con teoría de códigos] Introduction to cryptography: with coding theory*. Pearson Prentice Hall.

Vanstone, S. A. (1997). [Criptosistema de curva elíptica, la respuesta a criptografía de clave pública rápida y segura para plataformas restringidas] Elliptic curve cryptosystem—the answer to strong, fast public-key cryptography for securing constrained environments. *Information Security Technical Report*, *2*(2), 78-87.

Wang, X., Yin, Y. L. & Yu, H. (2005). [Buscando colisiones en SHA-1] Finding collisions in the full SHA-1, En *Annual international cryptology conference*. Springer.

Wang, X. & Yu, H. (2005). [Cómo romper MD5 y otras funciones hash] How to break MD5 and other hash functions, En *Annual international conference on the theory and applications of cryptographic techniques*. Springer.

Washington, L. C. (2003). *[Curvas elípticas: teoría de números y criptografía] Elliptic curves: number theory and cryptography*. CRC press.

Washington, L. C. (2008). *[Curvas elípticas: teoría de números y criptografía] Elliptic curves: number theory and cryptography* (2.ª ed.). CRC press.

Weisstein, E. W. (2005). [Algoritmo de Euclides] Euclidean Algorithm. https://mathworld.wolfram.com/EuclideanAlgorithm.html

Weisstein, E. W. (2009). [Curva elíptica] Elliptic Curve. https://mathworld.wolfram.com/EllipticCurve.html

Wolfram Research, Inc. (s.f.). Mathematica, Version 12.1 [Champaign, IL, 2020]. https://www.wolfram.com/mathematica

Wolfram, S. (2003). *[El libro de Mathematica] The Mathematica Book*. Champaign, Illinois, Wolfram Media, Inc.

Wolfram, S. (2017). *[Una Introducción Elemental a Wolfram Language] An Elementary Introduction to the Wolfram Language*. Champaign, Illinois, Wolfram Media, Inc.

Yoo, Y., Azarderakhsh, R., Jalali, A., Jao, D. & Soukharev, V. (2017). [Esquema de firma digital postcuántica basada en isogenias supersingular] A post-quantum digital signature scheme based on supersingular isogenies, En *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, Springer, Cham.

Anexos

Anexo A

Tabla de conversión Base64

Valor	Bits	Carácter	Valor	Bits	Carácter	Valor	Bits	Carácter	Valor	Bits	Carácter
0	000000	A	16	010000	Q	32	100000	g	48	110000	w
1	000001	B	17	010001	R	33	100001	h	49	110001	x
2	000010	C	18	010010	S	34	100010	i	50	110010	y
3	000011	D	19	010011	T	35	100011	j	51	110011	z
4	000100	E	20	010100	U	36	100100	k	52	110100	0
5	000101	F	21	010101	V	37	100101	l	53	110101	1
6	000110	G	22	010110	W	38	100110	m	54	110110	2
7	000111	H	23	010111	X	39	100111	n	55	110111	3
8	001000	I	24	011000	Y	40	101000	o	56	111000	4
9	001001	J	25	011001	Z	41	101001	p	57	111001	5
10	001010	K	26	011010	a	42	101010	q	58	111010	6
11	001011	L	27	011011	b	43	101011	r	59	111011	7
12	001100	M	28	011100	c	44	101100	s	60	111100	8
13	001101	N	29	011101	d	45	101101	t	61	111101	9
14	001110	O	30	011110	e	46	101110	u	62	111110	+
15	001111	P	31	011111	f	47	101111	v	63	111111	/

Tabla A.1: Tabla de conversión Base64.

Anexo B

Ejemplo de CFDI

Listing B.1: Archivo ejemplocfdv3.xml utilizado para generar los sellos

```
<?xml version="1.0" encoding="UTF-8"?>
<cfdi:Comprobante xsi:schemaLocation="http://www.sat.gob.mx/cfd/3
  cfdv3.xsd" xmlns:cfdi="http://www.sat.gob.mx/cfd/3"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  version="3.0" fecha="2010-03-06T20:38:12" sello="tOSe+Ex/wvn33YlGwtf
  mrJwQ31Crd7lI9VcH63TGjHfxk5vfb3q9uSbDUGk9TXvo70ydOpikRVw+9B2Six0mbu3P
  joPpO909oAYITrRyomdeUGJ4vmA2/12L86EJLWpU7vIt4cL8HpkEw7TOFhSdpzb/890+j
  P+C1adBsHU1VHc=" total="488.50" subTotal="488.50"
  certificado="MIIE/TCCA+WgAwIBAgIUMzAwMDEwMDAwMDAxMDAwMDA4MDAwDQ
  YJKoZIhvcNAQEFBQAwggFvMRgwFgYDVQQDDA9BLkMuIGRlIHBydWViYXMxLzAtBgNVBA
  oMJlNlcnZpY2lvIGRlIEFkbWluaXN0cmFjacOzbiBUcmlidXRhcmlhMTgwNgYDVQQLDC
  9BZG1pbmlzdHJhY2nDs24gZGUgU2VndXJpZGFkIGRlIGxhIEluZm9ybWFjacOzbjEpMC
  cGCSqGSIb3DQEJARYaYXNpc25ldEBwcnVlYmFzLnNhdC5nb2IubXgxJjAkBgNVBAkMH
  UF2LiBIaWRhbGdvIDc3LCBDb2wuIEd1ZXJyZXJvMQ4wDAYDVQQRDAUwNjMwMDELMAkGA
  1UEBhMCTVgxGTAXBgNVBAgMEERpc3RyaXRvIEZlZGVyYWwxEjAQBgNVBAcMCUNveW9h
  Y8OhbjEVMBMGA1UELRMMU0FUOTcwNzAxTk4zMTIwMAYJKoZIhvcNAQkCDCNSZXNwb25z
  YWJsZTogSMOpY3RvciBPcm5lbGFzIEFyY2lnYTAeFw0xMDA3MzAxNjU4NDBaFw0xMjA3
  MjkxNjU4NDBaMIGWMRIwEAYDVQQDDAlNYXRyaXogU0ExEjAQBgNVBCkMCU1hdHJpeiBT
  QTESMBAGA1UECgwJTWF0cml6IFNBMSUwIwYDVQQtExxBQUEwMTAxMDFBQUEgLyBBQUF
  BMDEwMTAxQUFBMR4wHAYDVQQFExUgLyBBQUFBMDEwMTAxSERGUlhYMDExETAPBgNVBA
  sMCFVuaWRhZCAxMIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQDD0ltQNthUNUfzq
  0t1GpIyapjzOn1W5fGM5G/pQyMluCzP9Y1VAgBjGgzwYp9Z0J9gadg3y2ZrYDwvv8b72
  goyRnhnv3bkjVRKlus6LDc00K7Jl23UYzNGlXn5+i0HxxuWonc2GYKFGsN4rFWKVy3Fn
  pv8Z2D7dNqsVyT5HapEqwIDAQABo4HqMIHnMAwGA1UdEwEB/wQCMAAwCwYDVR0PBAQDA
  gbAMB0GA1UdDgQWBBSYodSwRczzj5H7mcO3+mAyXz+y0DAuBgNVHR8EJzAlMCOgIaAfh
  h1odHRwOi8vcGtpLnNhdC5nb2IubXgvc2F0LmNybDAzBggrBgEFBQcBAQQnMCUwIwYIK
  wYBBQUHMAGGF2h0dHA6Ly9vY3NwLnNhdC5nb2IubXgvMB8GA1UdIwQYMBaAFOtZfQQim
  lONnnEaoFiWKfU54KDFMBAGA1UdIAQJMAcwBQYDKgMEMBMGA1UdJQQMMAoGCCsGAQUFB
  wMCMA0GCSqGSIb3DQEBBQUAA4IBAQArHQEorApwqumSn5EqDOAjbezi8fLco1cYES/PD
  +LQRM1Vb1g7VLE3hR4S5NNBv0bMwwWAr0WfL9lRRj0PMKLorO8y4TJjRU8MiYXfzSuKY
  L5Z16kW8zlVHw7CtmjhfjoIMwjQo3prifWxFv7VpfIBstKKShU0qB6KzUUNwg2Ola4t4
  gg2JJcBmyIAIInHSGoeinR2V1tQ10aRqJdXkGin4WZ75yMbQH4L0NfotqY6bpF2CqIY3
  aogQyJGhUJji4gYnS2DvHcyoICwgawshjSaX8Y0Xlwnuh6EusqhqlhTgwPNAPrKIXCmO
  WtqjlDhho/lhkHJMzuTn8AoVapbBUn"
```

```
35   formaDePago="PAGO EN UNA SOLA EXHIBICION"
36   noCertificado="30001000000100000800" tipoDeComprobante="ingreso">
37    <cfdi:Emisor rfc="PPL961114GZ1" nombre="PHARMA PLUS SA DE CV">
38       <cfdi:DomicilioFiscal pais="Mexico" calle="AV. RIO MIXCOAC"
39         estado="MEXICO, D.F." colonia="ACACIAS"
40         municipio="BENITO JUAREZ" noExterior="No. 140"
41         codigoPostal="03240"/>
42       <cfdi:ExpedidoEn pais="Mexico" calle="AV. UNIVERSIDAD"
43       estado="DISTRITO FEDERAL" colonia="OXTOPULCO" noExterior="1858"
44       codigoPostal="03910"/>
45    </cfdi:Emisor>
46    <cfdi:Receptor rfc="PEPJ8001019Q8" nombre="JUAN PEREZ PEREZ">
47       <cfdi:Domicilio pais="Mexico" calle="AV UNIVERSIDAD"
48       estado="DISTRITO FEDERAL" colonia="COPILCO UNIVERSIDAD"
49       municipio="COYOACAN" noExterior="16 EDF 3"
50       noInterior="DPTO 101" codigoPostal="04360"/>
51    </cfdi:Receptor>
52    <cfdi:Conceptos>
53       <cfdi:Concepto unidad="CAPSULAS" importe="244.00" cantidad="1.0"
54       descripcion="VIBRAMICINA 100MG 10" valorUnitario="244.00"/>
55       <cfdi:Concepto unidad="BOTELLA" importe="137.93" cantidad="1.0"
56       descripcion="CLORUTO 500M" valorUnitario="137.93"/>
57       <cfdi:Concepto unidad="TABLETAS" importe="84.50" cantidad="1.0"
58       descripcion="SEDEPRON 250MG 10" valorUnitario="84.50"/>
59    </cfdi:Conceptos>
60    <cfdi:Impuestos totalImpuestosTrasladados="22.07">
61       <cfdi:Traslados>
62          <cfdi:Traslado tasa="0.00" importe="0.00" impuesto="IVA"/>
63          <cfdi:Traslado tasa="16.00" importe="22.07" impuesto="IVA"/>
64       </cfdi:Traslados>
65    </cfdi:Impuestos>
66    <cfdi:Complemento>
67       <tfd:TimbreFiscalDigital xmlns:tfd=
68       "http://www.sat.gob.mx/TimbreFiscalDigital" xsi:schemaLocation=
69       "http://www.sat.gob.mx/TimbreFiscalDigital TimbreFiscalDigital.
70       xsd" selloCFD="tOSe+Ex/wvn33YlGwtfmrJwQ31Crd7lI9VcH63TGjHfxk5vf
71       b3q9uSbDUGk9TXvo70ydOpikRVw+9B2Six0mbu3PjoPpO909oAYITrRyomdeUGJ
72       4vmA2/12L86EJLWpU7vIt4cL8HpkEw7TOFhSdpzb/890+jP+C1adBsHU1VHc="
73       FechaTimbrado="2010-03-06T20:40:10"
74       UUID="ad662d33-6934-459c-a128-bdf0393e0f44"
75       noCertificadoSAT="30001000000100000801" version="1.0"
76       selloSAT="j5bSpqM3w0+shGtImqOwqqy6+d659O78ckfstu5vTSFa+2CVMj6Aw
77        fr18x4yMLGBwk6ruYbjBlVURodEIl6nJIhTTUtYQV1cbRDG9kvvhaNAakxqaSOn
78        Ox79nHxqFPRVoqh10CsjocS9PZkSM2jz1uwLgaF0knf1g8pjDkLYwlk="/>
79    </cfdi:Complemento>
80    <cfdi:Addenda/>
81 </cfdi:Comprobante>
```

Anexo C

Archivo xslt para generar Cadena Original

Listing C.1: Archivo ejemplocfdv3.xml utilizado para generar los sellos

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <xsl:stylesheet version="2.0" xmlns:xsl="http://www.w3.org/1999/XSL/
    ↪ Transform" xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:fn=
    ↪ "http://www.w3.org/2005/xpath-functions" xmlns:cfd="http://www.
    ↪ sat.gob.mx/cfd/2" xmlns:ecc="http://www.sat.gob.mx/ecc"
    ↪ xmlns:donat="http://www.sat.gob.mx/donat" xmlns:psgecfd="http://
    ↪ www.sat.gob.mx/psgecfd" xmlns:divisas="http://www.sat.gob.mx/
    ↪ divisas" xmlns:detallista="http://www.sat.gob.mx/detallista"
    ↪ xmlns:ecb="http://www.sat.gob.mx/ecb" xmlns:implocal="http://www
    ↪ .sat.gob.mx/implocal" xmlns:terceros="http://www.sat.gob.mx/
    ↪ terceros" xmlns:iedu="http://www.sat.gob.mx/iedu" xmlns:psgcfdsp
    ↪ ="http://www.sat.gob.mx/psgcfdsp">
3   <!-- Con el siguiente método se establece que la salida deberá ser
      ↪ en texto -->
4   <!-- <xsl:output method="text" version="1.0" encoding="UTF-8"
      ↪ indent="no"/> -->
5   <xsl:output method="text" version="1.0" encoding="UTF-8" indent="no
      ↪ "/>
6   <!--
7     En esta sección se define la inclusión de las plantillas de
        ↪ utilería
8   -->
9   <xsl:include href="http://www.sat.gob.mx/sitio_internet/cfd/2/
      ↪ cadenaoriginal_2_0/utilerias.xslt"/>
10  <!--
11    En esta sección se define la inclusión de las demás plantillas
        ↪ de transformación para
12    la generación de las cadenas originales de los complementos
        ↪ fiscales
13  -->
14  <xsl:include href="http://www.sat.gob.mx/sitio_internet/cfd/ecc/ecc
      ↪ .xslt"/>
```

```
15    <xsl:include href="http://www.sat.gob.mx/sitio_internet/cfd/psgecfd
          ↪ /psgecfd.xslt"/>
16    <xsl:include href="http://www.sat.gob.mx/sitio_internet/cfd/donat/
          ↪ donat.xslt"/>
17    <xsl:include href="http://www.sat.gob.mx/sitio_internet/cfd/divisas
          ↪ /divisas.xslt"/>
18    <xsl:include href="http://www.sat.gob.mx/sitio_internet/cfd/ecb/ecb
          ↪ .xslt"/>
19    <xsl:include href="http://www.sat.gob.mx/sitio_internet/cfd/
          ↪ detallista/detallista.xslt"/>
20    <xsl:include href="http://www.sat.gob.mx/sitio_internet/cfd/
          ↪ implocal/implocal.xslt"/>
21    <xsl:include href="http://www.sat.gob.mx/sitio_internet/cfd/
          ↪ terceros/terceros.xslt"/>
22    <xsl:include href="http://www.sat.gob.mx/sitio_internet/cfd/iedu/
          ↪ iedu.xslt"/>
23    <xsl:include href="http://www.sat.gob.mx/sitio_internet/cfd/
          ↪ psgcfdsp/psgcfdsp.xslt"/>
24    <!-- Aquí iniciamos el procesamiento de la cadena original con su |
          ↪  inicial y el terminador || -->
25    <xsl:template match="/">|<xsl:apply-templates select="/
          ↪ cfd:Comprobante"/>||</xsl:template>
26    <!--  Aquí iniciamos el procesamiento de los datos incluidos en el
          ↪ comprobante -->
27    <xsl:template match="cfd:Comprobante">
28       <!-- Iniciamos el tratamiento de los atributos de comprobante --
             ↪ >
29       <xsl:call-template name="Requerido">
30          <xsl:with-param name="valor" select="./@version"/>
31       </xsl:call-template>
32       <xsl:call-template name="Opcional">
33          <xsl:with-param name="valor" select="./@serie"/>
34       </xsl:call-template>
35       <xsl:call-template name="Requerido">
36          <xsl:with-param name="valor" select="./@folio"/>
37       </xsl:call-template>
38       <xsl:call-template name="Requerido">
39          <xsl:with-param name="valor" select="./@fecha"/>
40       </xsl:call-template>
41       <xsl:call-template name="Requerido">
42          <xsl:with-param name="valor" select="./@noAprobacion"/>
43       </xsl:call-template>
44       <xsl:call-template name="Requerido">
45          <xsl:with-param name="valor" select="./@anoAprobacion"/>
46       </xsl:call-template>
47       <xsl:call-template name="Requerido">
48          <xsl:with-param name="valor" select="./@tipoDeComprobante"/>
49       </xsl:call-template>
50       <xsl:call-template name="Requerido">
51          <xsl:with-param name="valor" select="./@formaDePago"/>
52       </xsl:call-template>
53       <xsl:call-template name="Opcional">
```

```
54              <xsl:with-param name="valor" select="./@condicionesDePago"/>
55          </xsl:call-template>
56          <xsl:call-template name="Requerido">
57              <xsl:with-param name="valor" select="./@subTotal"/>
58          </xsl:call-template>
59          <xsl:call-template name="Opcional">
60              <xsl:with-param name="valor" select="./@descuento"/>
61          </xsl:call-template>
62          <xsl:call-template name="Requerido">
63              <xsl:with-param name="valor" select="./@total"/>
64          </xsl:call-template>
65          <!--
66              Llamadas para procesar al los sub nodos del comprobante
67          -->
68          <xsl:apply-templates select="./cfd:Emisor"/>
69          <xsl:apply-templates select="./cfd:Receptor"/>
70          <xsl:apply-templates select="./cfd:Conceptos"/>
71          <xsl:apply-templates select="./cfd:Impuestos"/>
72          <xsl:apply-templates select="./cfd:Complemento"/>
73      </xsl:template>
74      <!-- Manejador de nodos tipo Emisor -->
75      <xsl:template match="cfd:Emisor">
76          <!-- Iniciamos el tratamiento de los atributos del Emisor -->
77          <xsl:call-template name="Requerido">
78              <xsl:with-param name="valor" select="./@rfc"/>
79          </xsl:call-template>
80          <xsl:call-template name="Requerido">
81              <xsl:with-param name="valor" select="./@nombre"/>
82          </xsl:call-template>
83          <!--
84              Llamadas para procesar al los sub nodos del comprobante
85          -->
86          <xsl:apply-templates select="./cfd:DomicilioFiscal"/>
87          <xsl:if test="./cfd:ExpedidoEn">
88              <xsl:call-template name="Domicilio">
89                  <xsl:with-param name="Nodo" select="./cfd:ExpedidoEn"/>
90              </xsl:call-template>
91          </xsl:if>
92      </xsl:template>
93      <!-- Manejador de nodos tipo Receptor -->
94      <xsl:template match="cfd:Receptor">
95          <!-- Iniciamos el tratamiento de los atributos del Receptor -->
96          <xsl:call-template name="Requerido">
97              <xsl:with-param name="valor" select="./@rfc"/>
98          </xsl:call-template>
99          <xsl:call-template name="Opcional">
100             <xsl:with-param name="valor" select="./@nombre"/>
101         </xsl:call-template>
102         <!--
103             Llamadas para procesar al los sub nodos del Receptor
104         -->
105         <xsl:call-template name="Domicilio">
```

```
106             <xsl:with-param name="Nodo" select="./cfd:Domicilio"/>
107          </xsl:call-template>
108       </xsl:template>
109       <!-- Manejador de nodos tipo Conceptos -->
110       <xsl:template match="cfd:Conceptos">
111          <!-- Llamada para procesar los distintos nodos tipo Concepto -->
112          <xsl:for-each select="./cfd:Concepto">
113             <xsl:apply-templates select="."/>
114          </xsl:for-each>
115       </xsl:template>
116       <!-- Manejador de nodos tipo Impuestos -->
117       <xsl:template match="cfd:Impuestos">
118          <xsl:for-each select="./cfd:Retenciones/cfd:Retencion">
119             <xsl:apply-templates select="."/>
120          </xsl:for-each>
121          <xsl:call-template name="Opcional">
122             <xsl:with-param name="valor" select="./
                  ↪ @totalImpuestosRetenidos"/>
123          </xsl:call-template>
124          <xsl:for-each select="./cfd:Traslados/cfd:Traslado">
125             <xsl:apply-templates select="."/>
126          </xsl:for-each>
127          <xsl:call-template name="Opcional">
128             <xsl:with-param name="valor" select="./
                  ↪ @totalImpuestosTrasladados"/>
129          </xsl:call-template>
130       </xsl:template>
131       <!-- Manejador de nodos tipo Retencion -->
132       <xsl:template match="cfd:Retencion">
133          <xsl:call-template name="Requerido">
134             <xsl:with-param name="valor" select="./@impuesto"/>
135          </xsl:call-template>
136          <xsl:call-template name="Requerido">
137             <xsl:with-param name="valor" select="./@importe"/>
138          </xsl:call-template>
139       </xsl:template>
140       <!-- Manejador de nodos tipo Traslado -->
141       <xsl:template match="cfd:Traslado">
142          <xsl:call-template name="Requerido">
143             <xsl:with-param name="valor" select="./@impuesto"/>
144          </xsl:call-template>
145          <xsl:call-template name="Requerido">
146             <xsl:with-param name="valor" select="./@tasa"/>
147          </xsl:call-template>
148          <xsl:call-template name="Requerido">
149             <xsl:with-param name="valor" select="./@importe"/>
150          </xsl:call-template>
151       </xsl:template>
152       <!-- Manejador de nodos tipo Complemento -->
153       <xsl:template match="cfd:Complemento">
154          <xsl:for-each select="./*">
155             <xsl:apply-templates select="."/>
```

```
156         </xsl:for-each>
157     </xsl:template>
158     <!--
159         Manejador de nodos tipo Concepto
160     -->
161     <xsl:template match="cfd:Concepto">
162         <!-- Iniciamos el tratamiento de los atributos del Concepto -->
163         <xsl:call-template name="Requerido">
164             <xsl:with-param name="valor" select="./@cantidad"/>
165         </xsl:call-template>
166         <xsl:call-template name="Opcional">
167             <xsl:with-param name="valor" select="./@unidad"/>
168         </xsl:call-template>
169         <xsl:call-template name="Opcional">
170             <xsl:with-param name="valor" select="./@noIdentificacion"/>
171         </xsl:call-template>
172         <xsl:call-template name="Requerido">
173             <xsl:with-param name="valor" select="./@descripcion"/>
174         </xsl:call-template>
175         <xsl:call-template name="Requerido">
176             <xsl:with-param name="valor" select="./@valorUnitario"/>
177         </xsl:call-template>
178         <xsl:call-template name="Requerido">
179             <xsl:with-param name="valor" select="./@importe"/>
180         </xsl:call-template>
181         <!--
182             Manejo de los distintos sub nodos de información aduanera de
                ↪ forma indistinta
183             a su grado de dependencia
184         -->
185         <xsl:for-each select=".//cfd:InformacionAduanera">
186             <xsl:apply-templates select="."/>
187         </xsl:for-each>
188         <!-- Llamada al manejador de nodos de Cuenta Predial en caso de
                ↪ existir -->
189         <xsl:if test="./cfd:CuentaPredial">
190             <xsl:apply-templates select="./cfd:CuentaPredial"/>
191         </xsl:if>
192         <!-- Llamada al manejador de nodos de ComplementoConcepto en
                ↪ caso de existir -->
193         <xsl:if test="./cfd:ComplementoConcepto">
194             <xsl:apply-templates select="./cfd:ComplementoConcepto"/>
195         </xsl:if>
196     </xsl:template>
197     <!-- Manejador de nodos tipo Información Aduanera -->
198     <xsl:template match="cfd:InformacionAduanera">
199         <!-- Manejo de los atributos de la información aduanera -->
200         <xsl:call-template name="Requerido">
201             <xsl:with-param name="valor" select="./@numero"/>
202         </xsl:call-template>
203         <xsl:call-template name="Requerido">
204             <xsl:with-param name="valor" select="./@fecha"/>
```

```
205         </xsl:call-template>
206         <xsl:call-template name="Requerido">
207             <xsl:with-param name="valor" select="./@aduana"/>
208         </xsl:call-template>
209     </xsl:template>
210     <!-- Manejador de nodos tipo Información CuentaPredial -->
211     <xsl:template match="cfd:CuentaPredial">
212         <xsl:call-template name="Requerido">
213             <xsl:with-param name="valor" select="./@numero"/>
214         </xsl:call-template>
215     </xsl:template>
216     <!-- Manejador de nodos tipo ComplementoConcepto -->
217     <xsl:template match="cfd:ComplementoConcepto">
218         <xsl:for-each select="./*">
219             <xsl:apply-templates select="."/>
220         </xsl:for-each>
221     </xsl:template>
222     <!-- Manejador de nodos tipo domicilio fiscal -->
223     <xsl:template match="cfd:DomicilioFiscal">
224         <!-- Iniciamos el tratamiento de los atributos del Domicilio
                ↪ Fiscal -->
225         <xsl:call-template name="Requerido">
226             <xsl:with-param name="valor" select="./@calle"/>
227         </xsl:call-template>
228         <xsl:call-template name="Opcional">
229             <xsl:with-param name="valor" select="./@noExterior"/>
230         </xsl:call-template>
231         <xsl:call-template name="Opcional">
232             <xsl:with-param name="valor" select="./@noInterior"/>
233         </xsl:call-template>
234         <xsl:call-template name="Opcional">
235             <xsl:with-param name="valor" select="./@colonia"/>
236         </xsl:call-template>
237         <xsl:call-template name="Opcional">
238             <xsl:with-param name="valor" select="./@localidad"/>
239         </xsl:call-template>
240         <xsl:call-template name="Opcional">
241             <xsl:with-param name="valor" select="./@referencia"/>
242         </xsl:call-template>
243         <xsl:call-template name="Requerido">
244             <xsl:with-param name="valor" select="./@municipio"/>
245         </xsl:call-template>
246         <xsl:call-template name="Requerido">
247             <xsl:with-param name="valor" select="./@estado"/>
248         </xsl:call-template>
249         <xsl:call-template name="Requerido">
250             <xsl:with-param name="valor" select="./@pais"/>
251         </xsl:call-template>
252         <xsl:call-template name="Requerido">
253             <xsl:with-param name="valor" select="./@codigoPostal"/>
254         </xsl:call-template>
255     </xsl:template>
```

```
256     <!-- Manejador de nodos tipo Domicilio -->
257     <xsl:template name="Domicilio">
258         <xsl:param name="Nodo" />
259         <!-- Iniciamos el tratamiento de los atributos del Domicilio --
                ↪ >
260         <xsl:call-template name="Opcional">
261             <xsl:with-param name="valor" select="$Nodo/@calle" />
262         </xsl:call-template>
263         <xsl:call-template name="Opcional">
264             <xsl:with-param name="valor" select="$Nodo/@noExterior" />
265         </xsl:call-template>
266         <xsl:call-template name="Opcional">
267             <xsl:with-param name="valor" select="$Nodo/@noInterior" />
268         </xsl:call-template>
269         <xsl:call-template name="Opcional">
270             <xsl:with-param name="valor" select="$Nodo/@colonia" />
271         </xsl:call-template>
272         <xsl:call-template name="Opcional">
273             <xsl:with-param name="valor" select="$Nodo/@localidad" />
274         </xsl:call-template>
275         <xsl:call-template name="Opcional">
276             <xsl:with-param name="valor" select="$Nodo/@referencia" />
277         </xsl:call-template>
278         <xsl:call-template name="Opcional">
279             <xsl:with-param name="valor" select="$Nodo/@municipio" />
280         </xsl:call-template>
281         <xsl:call-template name="Opcional">
282             <xsl:with-param name="valor" select="$Nodo/@estado" />
283         </xsl:call-template>
284         <xsl:call-template name="Requerido">
285             <xsl:with-param name="valor" select="$Nodo/@pais" />
286         </xsl:call-template>
287         <xsl:call-template name="Opcional">
288             <xsl:with-param name="valor" select="$Nodo/@codigoPostal" />
289         </xsl:call-template>
290     </xsl:template>
291 </xsl:stylesheet>
```

Anexo D

Difusión

CONGRESO INTERNACIONAL DE INVESTIGACIÓN ACADEMIA JOURNALS, CELAYA, 2019

OTORGAN EL PRESENTE

CERTIFICADO

A

DIEGO ABRAHAM OLVERA MENDOZA
FIDEL GONZÁLEZ GUTIÉRREZ
ARTURO GONZÁLEZ GUTIÉRREZ

POR SU PARTICIPACIÓN CON LA PONENCIA TITULADA

IMPLEMENTACIÓN EN MATHEMATICA DEL ESTÁNDAR PKCS#1 PARA LA GENERACIÓN DE CLAVES Y FIRMAS DIGITALES USADOS EN CRIPTOSISTEMAS RSA

VOLUMEN ONLINE CON **ISSN 1946-5351 ONLINE,** VOL. 11, NO. 9 E INDEXACIÓN EN FUENTE ACADÉMICA PLUS (EBSCO) Y LIBRO DIGITAL EBOOK ONLINE INTITULADO *DISEMINACIÓN DE LA INVESTIGACIÓN EN LA EDUCACIÓN SUPERIOR: CELAYA 2019*, CON **ISBN 978-1-939982-50-6 ONLINE.**

LA CUAL FUE PRESENTADA EN EL
TECNOLÓGICO NACIONAL DE MÉXICO EN CELAYA
LOS DÍAS 6, 7 y 8 DE NOVIEMBRE DE 2019, CELAYA, GUANAJUATO, MÉXICO.

DR. RAFAEL MORAS
EDITOR, ACADEMIAJOURNALS.COM
PROFESOR DE INGENIERÍA INDUSTRIAL Y ADMINISTRATIVA
ST. MARY'S UNIVERSITY, SAN ANTONIO, TX. EEUU.

MC. MOISES TAPIA ESQUIVIAS
COORDINADOR GENERAL DEL
CONGRESO INTERNACIONAL DE INVESTIGACIÓN
ACADEMIA JOURNALS, CELAYA, 2019.

No. 1595

C0813

Firma digital:
http://aj.itcelaya.edu.mx/folio.html?f=d88518acbcc3e0cf1f4cd87ba

La Sociedad Mexicana de Ciencias de la Computación
y la Universidad Autónoma de Coahuila a través del
Centro de Investigación en Matemáticas Aplicadas y la Facultad de Sistemas

otorgan la presente

CONSTANCIA

a

Diego Abraham Olvera Mendoza, Fidel González Gutiérrez y Arturo González Gutiérrez

Por su destacada participación en el Coloquio de estudiantes con el tema “Una propuesta de sello digital para los CFDI utilizando criptografía de curvas elípticas”
en el Encuentro Nacional en Computación 2020
efectuado del 24 al 26 de agosto de 2020

Dra. María Lucía Barrón Estrada
Presidenta de la SMCC

Dra. Valería Soto Mendoza
Organizador General Local

Valida el reconocimiento en nuestro sitio escaneando el QR.
O ingresa en https://enc-2020.web.app/validacion/ENC-0202