UNIVERSIDAD DE GRANADA

TRABAJO FIN DE GRADO

INGENIERÍA INFORMÁTICA

Fundamentos teóricos de las curvas elípticas y su aplicación en criptografía

Un estudio un detalle sobre Curvas Ellípticas aplicadas a la Criptografía

Autor
Leon Elliott Fuller

Directores
Jesús García Miranda

Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación

—

Granada, Abril de 2025

Fundamentos teóricos de las curvas elípticas y su aplicación en criptografía

Un estudio un detalle sobre Curvas Ellípticas aplicadas a la Criptografía.

Autor
Leon Elliott Fuller

Directores
Jesús García Miranda

Fundamentos teóricos de las curvas elípticas y su aplicación en criptografía

Leon Elliott Fuller

Palabras clave: curvas elípticas, criptografía, computación cuántica, ECDH, ECDSA, ECC, RSA

Resumen

En 1985 Neal Koblitz y Victor Miller propusieron de forma independiente el uso de curvas elípticas en criptografía. Hoy en día, los esquemas basados en curvas elípticas (ECC) se han consolidado como una alternativa eficiente a RSA, pues ofrecen niveles de seguridad comparables con tamaños de clave sensiblemente menores. Esto los hace ideales para dispositivos con recursos limitados y para aplicaciones que requieren alto rendimiento.

En este trabajo profundizaremos en los fundamentos de ECC y describiremos los principales protocolos Elliptic Curve Diffie–Hellman (ECDH) y Elliptic Curve Digital Signature Algorithm (ECDSA). A continuación analizaremos tanto los ataques clásicos al problema del logaritmo discreto en curvas elípticas como las amenazas cuánticas, en particular el algoritmo de Shor, que podría comprometer ECC en la era de los ordenadores cuánticos.

Por último, se realizarán comparaciones de tiempo con respecto a la generación de claves entre RSA y ECC.

Theoretical foundations of elliptic curves and their application in cryptography

Leon Elliott Fuller

Keywords: elliptic curves, cryptography, quantum computing, ECDH, ECDSA, ECC, RSA

Abstract

In 1985 Neal Koblitz and Victor Miller independently proposed the use of elliptic curves in cryptography. Today, elliptic curve schemes (ECC) have become a highly efficient alternative to RSA, offering comparable security with significantly smaller key sizes. This makes them ideal for resource-constrained devices and high-performance applications.

In this work we dive into the fundamentals of ECC and describe the main protocols used, such as Elliptic Curve Diffie–Hellman (ECDH) and Elliptic Curve Digital Signature Algorithm (ECDSA). We then analyze both classical attacks on the elliptic curve discrete logarithm problem and quantum threats, in particular Shor's algorithm, which could compromise ECC in the era of quantum computers.

Finally, we present timing comparisons for key generation between RSA and ECC.

Yo, **Leon Elliott Fuller**, alumno de la titulación Ingeniería Informática de la **Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación de la Universidad de Granada**, con NIE ..., autorizo la ubicación de la siguiente copia de mi Trabajo Fin de Grado en la biblioteca del centro para que pueda ser consultada por las personas que lo deseen. Todo el código fuente así como este documento en formato LaTeX se puede encontrar en el siguiente repositorios de GitHub: https://github.com/leonfullxr/ECC-Thesis

Fdo: Leon Elliott Fuller

Granada a X de mayo de 2025.

D. **Jesús García Miranda**, Profesor del Área del Grado en Ingeniería Informática del Departamento de Álgebra de la Universidad de Granada.

Informa:

Que el presente trabajo, titulado ***Fundamentos teóricos de las curvas elípticas y su aplicación en criptografía***, ha sido realizado bajo su supervisión por **Leon Elliott Fuller**, y autorizamos la defensa de dicho trabajo ante el tribunal que corresponda.

Y para que conste, expiden y firman el presente informe en Granada a X de mayo de 2025.

El tutor:

Jesús García Miranda

Agradecimientos

Poner aquí agradecimientos...

Índice general

1. Introducción **1**
- 1.1. Contexto histórico y motivación . . . 1
- 1.2. Objetivos . . . 2
- 1.3. Planificación . . . 2
- 1.4. Estructura . . . 2

2. Introducción a la criptografía **5**
- 2.1. Estructura de un criptosistema . . . 6
- 2.2. Criptografía sin clave o Funciones hash . . . 7
  - 2.2.1. MD5 . . . 8
  - 2.2.2. SHA-1 . . . 8
- 2.3. Criptografía de clave privada o simétrica . . . 9
  - 2.3.1. Cifrado de bloque . . . 10
  - 2.3.2. Cifrado de flujo . . . 14
- 2.4. Criptografía de clave pública o asimétrica . . . 15
  - 2.4.1. Intercambio de claves Diffie–Hellman . . . 16
  - 2.4.2. Firma digital . . . 18
- 2.5. Resumen comparativo . . . 18
- 2.6. Principios de diseño criptográfico . . . 18
- 2.7. Seguridad y modelos de ataque . . . 19
  - 2.7.1. Ataques pasivos y activos . . . 19
  - 2.7.2. Modelos clásicos de criptoanálisis . . . 19
  - 2.7.3. Ataques avanzados . . . 20

3. Campos finitos **21**
- 3.1. Introducción a campos finitos . . . 21
  - 3.1.1. Representación binaria y suma de enteros . . . 22
- 3.2. Aritmética en $\mathbb{F}_p$ . . . 23
  - 3.2.1. Suma en $\mathbb{F}_p$ . . . 24
  - 3.2.2. Multiplicación de enteros . . . 24
  - 3.2.3. Algoritmo de Karatsuba . . . 25
- 3.3. Aritmética en $\mathbb{F}_{2^m}$ . . . 26
  - 3.3.1. Representación polinomial . . . 27

3.3.2. Suma en $\mathbb{F}_{2^m}$ 28
3.3.3. Multiplicación en $\mathbb{F}_{2^m}$ 29
3.4. Aritmética en campos de extensión óptima (OEF) 30
3.5. Selección de campos para ECC: comparativa y recomendaciones 32

4. Teoría de las curvas elípticas 35
4.1. Introducción y motivación 36
4.1.1. Historia y aplicaciones 36
4.1.2. Ventajas frente a otros grupos 36
4.2. Definición de una curva elíptica 36
4.2.1. Forma de Weierstrass general 36
4.2.2. Transformaciones y cambios de coordenadas 36
4.3. Estructura de grupo 36
4.3.1. Punto en el infinito 36
4.3.2. Ley de suma de puntos (geometría proyectiva) 36
4.3.3. Propiedades algebraicas 36
4.4. Curvas sobre cuerpos finitos 36
4.4.1. Curvas sobre $\mathbb{F}_p$ 36
4.4.2. Curvas binarias sobre $\mathbb{F}_{2^m}$ 36
4.5. Coordenadas y representación 36
4.5.1. Coordenadas afines 36
4.5.2. Coordenadas proyectivas y Jacobianas 36
4.5.3. Coordenadas “mixed” y optimizaciones 36
4.6. Selección de parámetros 36
4.6.1. Tamaños de curva y seguridad 36
4.6.2. Cofactores y subgrupos 36
4.6.3. Curvas estandarizadas (secp256k1, Curve25519, . . .) . 36

5. Criptografía con curvas elípticas (ECC) 37
5.1. Fundamentos de ECC 38
5.1.1. Ventajas frente a RSA y DH 38
5.1.2. Comparativa de tamaños de clave 38
5.2. Intercambio de claves: ECDH 38
5.2.1. Protocolo básico 38
5.2.2. Variantes y extensiones (X25519, etc.) 38
5.3. Firmas digitales: ECDSA 38
5.3.1. Generación y verificación de firma 38
5.3.2. Problemas prácticos (k reutilizado, mal RNG) 38
5.4. Otras construcciones criptográficas 38
5.4.1. ECIES: cifrado híbrido 38
5.4.2. EdDSA: firmas deterministas sobre edwards 38
5.5. Aspectos de implementación 38
5.5.1. Resistencia a canal lateral 38
5.5.2. Bibliotecas y hardware acelerado 38

5.6. Estándares y recomendaciones 38
5.6.1. NIST, SECG, RFCs relevantes 38
5.6.2. Buenas prácticas de despliegue 38

Índice de figuras

2.1. Esquema de clasificación dentro de la Teoría de la Información 5
2.2. Clasificación de la criptografía en función del método de cifrado 6
2.3. Esquema de un criptosistema formal 7
2.4. Ejemplo de cifrado y descifrado de bloque ECB 12
2.5. Ejemplo de cifrado y descifrado de bloque CBC 14
2.6. Ejemplo de cifrado asimétrico 16

3.1. Representación de $a \in \mathbb{F}_{2^m}$ como un arreglo A de palabras de W bits. Los $s = Wt - m$ bits de orden más alto de $A[t-1]$ permanecen sin usar. 27

Índice de cuadros

2.1. Resumen del intercambio de claves Diffie–Hellman 17

3.1. Parámetros de ejemplo para OEF. Aquí $p = 2^n - c$ es primo y $f(z) = z^m - \omega$ es irreducible en $\mathbb{F}_p[z]$. 31
3.2. Comparativa aproximada de costes en una CPU de 3 GHz (tamaños de campo 256 bits). 33

Capítulo 1

Introducción

1.1. Contexto histórico y motivación

La comunicación confidencial entre sujetos ha sido una necesidad constante a lo largo de la historia. Cuando los mensajes debían transmitirse de forma no presencial o transportarse físicamente, surgía el reto de asegurar que ninguna entidad externa pudiera leerlos o alterarlos sin ser detectada.

Ya en la Antigüedad, civilizaciones como la espartana emplearon la "Escítala" en el siglo V a. C., un dispositivo de transposición que cifraba mensajes al enrollarlos en un cilindro. Más tarde, los griegos documentaron con Polibio un sistema de sustitución basado en el orden de las letras, y Roma adoptó el famoso Cifrado César, que desplazaba el alfabeto para ocultar información militar y diplomática.

Durante el Renacimiento, el cifrado avanzó hacia métodos más sofisticados, como el de Leon Battista Alberti que introdujo en 1465 la idea del cifrado polialfabético mediante discos concéntricos, lo que supuso un gran avance en esa época. Blaise de Vigenère perfeccionó esta técnica en el siglo XVI con una tabla que combinaba alfabetos de forma periódica, y en Europa se publicaron tratados (como el *Cryptomenytices et Cryptographiae* de Selenus) que difundieron estos métodos entre cortes y ejércitos.

El siglo XX presenció el mayor salto tecnológico, donde se partió de los cifrados manuales de la Primera Guerra Mundial hasta la aparición de máquinas electromecánicas como la Enigma alemana en la Segunda Guerra Mundial, una máquina de rotores que automatizaba los cálculos que era necesario realizar para las operaciones de cifrado y descifrado de mensajes. El descifrado de Enigma por los criptógrafos aliados no solo cambió el curso del conflicto, sino que marcó el inicio de la era del cálculo automatizado al servicio de la criptografía. Tras la guerra, Claude Shannon sentó las bases teóricas de la información y la seguridad, y en los años 70 el NIST publicó el Estándar de Cifrado de Datos (DES), consolidando el cifrado simétrico en la informática moderna.

Sin embargo, todos los criptosistemas anteriores tenían un inconveniente, y es que todos se basaban en el hecho de que tanto la parte que descifraba como la que codificaba debían conocer el método de cifrado y la clave para descifrarlo. ¿Pero cómo se transmite la clave de forma segura? Por supuesto, con cifrado. Pero entonces, ¿cómo se puede enviar el cifrado, para que la entidad que descifra el mensaje pueda conocer dicho cifrado? Como se puede apreciar, dicho problema de la criptografía es un bucle infinito entre el cifrado y la distribución de claves que no se puede resolver con seguridad garantizada para todos los lectores deseados.

Hoy día, RSA y sistemas análogos sustentan protocolos críticos como HTTPS, correo firmado, servicios bancarios,etc. Pero presentan dos limitaciones crecientes:

- **Tamaño de clave versus seguridad.** Para alcanzar un nivel de protección comparable al de cifrados simétricos de 128 bits, RSA requiere claves de varios miles de bits, lo que penaliza el rendimiento en cómputo y ancho de banda.
- **Vulnerabilidad cuántica.** Los algoritmos cuánticos plantean un desafío fundamental para la criptografía moderna. En particular, Shor demostró en 1994 que un ordenador cuántico suficientemente grande podría factorizar enteros y resolver problemas de logaritmo discreto en tiempo polinómico, lo que rompería sistemas como RSA, Diffie–Hellman e incluso ECC [1][2]. Esta perspectiva ha impulsado la búsqueda de algoritmos resistentes a estos ataques cuánticos, así como la adopción de curvas elípticas de mayor seguridad y longitudes de clave incrementadas.

Estas limitaciones han impulsado la adopción de la *Criptografía de Curvas Elípticas (ECC)*. Gracias a la dificultad del problema del logaritmo discreto en curvas elípticas, ECC ofrece un nivel de seguridad equivalente con claves drásticamente más pequeñas (por ejemplo, 256 bits en ECC frente a 3072 bits en RSA), mejorando así el rendimiento y la eficiencia. Además, la resistencia actual de ECC frente a los ataques cuánticos es mayor que la de RSA bajo el mismo modelo cuántico, lo que la posiciona como la opción de próxima generación para sistemas criptográficos modernos.

1.2. Objetivos

1.3. Planificación

1.4. Estructura

Antes de pasar a detalles más técnicos, me gustaría detallar el contenido de este proyecto:

1. **Capítulo 1: Introducción** Se encuentra una breve introducción a nuestro proyecto, así como un contexto histórico y las motivaciones que nos han llevado a realizarla.

2. **Capítulo 2: Introducción a la criptografía** Se presentan los conceptos esenciales, donde se definen las bases necesarias para comprender la seguridad de los sistemas actuales.

3. **Capítulo 3: Introducción a la computación cuántica** Define una base sólida sobre la computación cuántica, para próximamente, detallar los posibles ataques que se podrían realizar sobre las curvas elípticas.

4. **Capítulo 4: Teoría de las curvas elípticas** Se expone la teoría algebraica de las curvas elípticas sobre cuerpos finitos, la ley de grupo y las propiedades matemáticas clave. Incluye ejemplos de parametrización e implementaciones sencillas en código.

5. **Capítulo 5: Criptografía con curvas elípticas (ECC)** Describe los esquemas de clave pública basados en ECC: generación de claves, Elliptic Curve Diffie–Hellman (ECDH) y Elliptic Curve Digital Signature Algorithm (ECDSA). Se comparan su rendimiento y tamaño de clave frente a RSA.

6. **Capítulo 6: Ataques clásicos a ECC** Analiza el problema del logaritmo discreto en curvas elípticas, técnicas de criptoanálisis clásicas (baby-step/giant-step, Pollard's rho) y su complejidad computacional en la práctica.

7. **Capítulo 7: Ataques cuánticos a ECC** Aquí se muestra el impacto del algoritmo de Shor y otros métodos cuánticos sobre ECC, estimando los recursos necesarios y las implicaciones para la seguridad futura. Se introduce la criptografía post-cuántica como posible resistencia.

8. **Capítulo 8: Implementación y experimentos** Detalla las herramientas y librerías empleadas, presenta fragmentos de código para ECC y simulaciones cuánticas, y muestra visualizaciones de curvas y resultados bajo diferentes parámetros.

9. **Capítulo 9: Resultados y pruebas** Recoge métricas de rendimiento, comparaciones entre RSA y ECC, junto con análisis de tiempos de cómputo.

10. **Capítulo 9: Conclusiones** Se pueden encontrar las conclusiones finales así como las recomendaciones para futuros trabajos.

Capítulo 2

Introducción a la criptografía

La criptografía es el arte de encubrir los mensajes con signos convencionales, que sólo pueden cobrar algún sentido a través de una clave secreta que nace en conjunto con la escritura.

En su clasificación dentro de las ciencias, la criptografía proviene de una rama de las matemáticas, que fue iniciada por el matemático Claude Elwood Shannon en 1948, denominada: "Teoría de la Información". Esta rama de las ciencias se divide en: "Teoría de Códigosz en Çriptología". Y a su vez la Criptología se divide en Criptoanálisis y Criptografía, como se muestra en la siguiente figura 2.1:

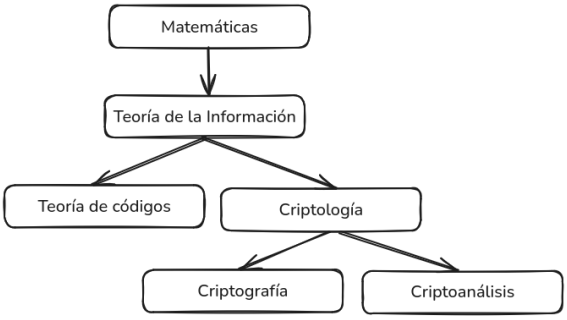

Figura 2.1: Esquema de clasificación dentro de la Teoría de la Información

En un sentido más amplio, la Criptografía es la ciencia encargada de diseñar funciones o dispositivos, capaces de transformar mensajes legibles o en claro a mensajes cifrados de tal manera que esta transformación (cifrar) y su transformación inversa (descifrar) sólo pueden ser factibles con el conocimiento de una o más llaves. En contraparte, el criptoanálisis es la ciencia que estudia los métodos que se utilizan para, a partir de uno o varios mensajes cifrados, recuperar los mensajes en claro en ausencia de la(s) llave(s) y/o encontrar la llave o llaves con las que fueron cifrados dichos mensajes.

La criptografía se puede clasificar históricamente en dos:

- Criptografía Clásica: es aquella que se utilizó desde antes de la época actual hasta la mitad del siglo XX. También puede entenderse como la criptografía no computarizada o mejor dicho no digitalizada. Los métodos utilizados eran variados, algunos muy simples y otros muy complicados de criptoanalizar para su época.
  - **Sustitución:** Se reemplazan elementos del mensaje original por otros.
  - **Transposición:** Se reordenan los caracteres del mensaje siguiendo un patrón definido.
- Criptografía Moderna: se divide actualmente en cifrado sin clave (funciones hash), cifrado de clave privada (cifrado simétrico) y cifrado de clave pública (cifrado asimétrico). En el cifrado de clave privada las claves de cifrado y descifrado son la misma (o bien se deriva de forma directa una de la otra), debiendo mantenerse en secreto dicha clave para evitar el descifrado de un mensaje. Por el contrario, en el cifrado de clave pública las claves de cifrado y descifrado son independientes, no derivándose una de la otra, por lo cual puede hacerse pública la clave de cifrado siempre que se mantenga en secreto la clave de descifrado. Veremos algunos algoritmos de cifrado modernos

Tanto la criptografía clásica como la moderna se clasifican de acuerdo a las técnicas o métodos que se utilizan para cifrar los mensajes. Esta clasificación la podemos ver en la siguiente figura 2.2:

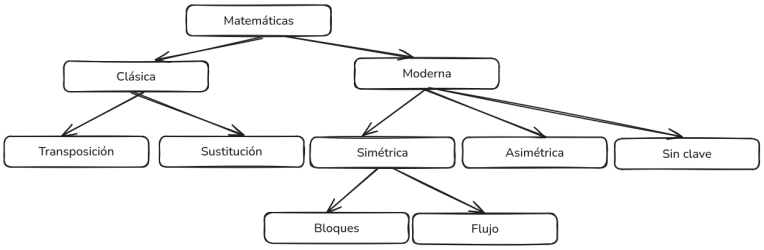

Figura 2.2: Clasificación de la criptografía en función del método de cifrado

2.1. Estructura de un criptosistema

Un criptosistema se define formalmente como la tupla

$$(\mathcal{M}, \mathcal{C}, \mathcal{K}, \mathcal{E}, \mathcal{D}),$$

donde

- $\mathcal{M}$ es el espacio de texto plano.

- $\mathcal{C}$ es el espacio de texto cifrado.
- $\mathcal{K}$ es el espacio de claves.
- $\mathcal{E} = \{E_k : k \in \mathcal{K}\}$ es el conjunto de funciones de cifrado $E_k : \mathcal{M} \to \mathcal{C}$.
- $\mathcal{D} = \{D_k : k \in \mathcal{K}\}$ es el conjunto de funciones de descifrado $D_k : \mathcal{C} \to \mathcal{M}$.

Deben cumplirse las condiciones de corrección:

$$D_{k'}\big(E_k(M)\big) \;=\; M \quad \forall\, M \in \mathcal{M},$$

siendo k' la clave de descifrado asociada a la clave de cifrado k.

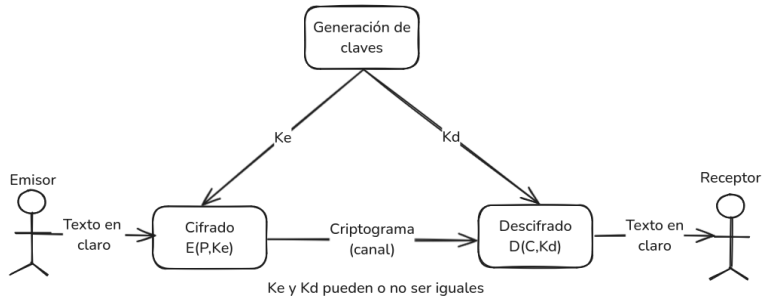

Figura 2.3: Esquema de un criptosistema formal

2.2. Criptografía sin clave o Funciones hash

Las funciones hash son funciones unidireccionales de cálculo sencillo y eficiente, utilizadas para garantizar la integridad de los datos. Se aplican a un mensaje M produciendo un valor de longitud fija:

$$M \;\longmapsto\; H(M),$$

donde $H(M)$ suele tener 128, 160, 256 o 512 bits. Es computacionalmente inviable recuperar M a partir de $H(M)$.

Una función hash criptográfica es una función eficiente que mapea cadenas binarias de longitud arbitraria a cadenas binarias de longitud fija n. Para una función que produce valores de n bits, la probabilidad de que una cadena aleatoria termine en un valor dado es 2^{-n}. El valor de hash actúa como representante compacto de la cadena de entrada.

Para su uso en criptografía, H debe satisfacer al menos:

- **Resistencia a preimagen:** Dado un valor y de n bits, resulta inviable encontrar x tal que $H(x) = y$.

- **Resistencia a segunda preimagen:** Dado un mensaje x, resulta inviable hallar otro $x' \neq x$ con $H(x') = H(x)$.
- **Resistencia a colisiones:** Resulta inviable encontrar dos entradas distintas x, y tales que $H(x) = H(y)$.

Cuando H es público y sin clave, se le llama *código de detección de modificaciones* (MDC). Si se añade una clave secreta al cómputo de hash, obtenemos un *código de autenticación de mensaje* (MAC), que ofrece además autenticación del origen junto con integridad.

2.2.1. MD5

El MD5 (Message-Digest Algorithm 5) procesa el mensaje de entrada en bloques sucesivos de 512 bits y produce un resumen (digest) de 128 bits [3]. Su diseño sigue la construcción de Merkle–Damgard, inicializando cuatro registros de 32 bits y aplicando cuatro rondas de 16 operaciones cada una por bloque [4]. En 2004 se demostró que MD5 ya no es resistente a colisiones: dos entradas diferentes pueden colisionar y producir el mismo digest en tiempo práctico [4]. Por ejemplo, en 2009, Marc Stevens, Arjen Lenstra y Benne de Weger, un grupo de investigadores, construyeron colisiones con prefijos elegidos y generaron un certificado CA malicioso [5]. Actualmente se desaconseja MD5 para cualquier uso criptográfico y se recomienda SHA-2 o SHA-3 como alternativas seguras [6].

2.2.2. SHA-1

El SHA (Secure Hash Algorithm 1), fue publicado en 1995 como FIPS 180-1 por el NIST [7]. Procesa los datos de entrada en bloques de 512 bits al igual que MD5, pero produce un digest de 160 bits con cinco registros de 32 bits y 80 rondas de mezcla [7]. Su algoritmo es el siguiente:

1. Se toma el mensaje original y se rellena hasta alcanzar una longitud de $448 \mod 512$ bits. Es decir, al dividir la longitud total entre 512, el resto debe ser 448.
2. Se añade un campo de 64 bits que indica la longitud del mensaje original (incluyendo el propio relleno). Así, la longitud total del mensaje resultante, denotado P', será un múltiplo de 512 bits.
3. P' se divide en bloques de longitud fija de 512 bits, denotados como $Y_1, Y_2, \ldots, Y_L$.
4. Se inicia el cálculo del resumen. Cada bloque de 512 bits se mezcla con un búfer de 160 bits mediante 80 rondas. Cada 20 rondas se modifican las funciones de mezcla utilizadas. Este proceso continúa hasta procesar todos los bloques de entrada.

Una vez terminado el cálculo, el buffer de 160 bits contiene el valor del compendio de mensaje. El SHA es 2^{32} veces más seguro que el MD5, pero su cálculo es más lento.

En 2005 Xiaoyun Wang y colaboradores redujeron la complejidad para hallar colisiones de 2^{80} a 2^{63} operaciones [8], y en 2017 Google y CWI demostraron la primera colisión real de SHA-1 en el experimento *SHAttered* [9]. El NIST anunció en 2011 la deprecación de SHA-1 y ha planificado retirarla completamente para fines de 2030 [6]. Chrome dejó de aceptar certificados TLS firmados con SHA-1 en enero de 2017 [10], Mozilla Firefox siguió en febrero de 2017 [11] y Microsoft finalizó el soporte para contenidos firmados con SHA-1 en 2021 [12].

2.3. Criptografía de clave privada o simétrica

Un cifrado simétrico utiliza una clave k elegida de un espacio (conjunto) de claves posibles K para cifrar un mensaje m perteneciente al conjunto de mensajes posibles M, generando un texto cifrado $c \in C$.

El cifrado puede verse como una función:

$$e : K \times m \to C$$

cuyo dominio es el conjunto de pares (k, m) y cuyo codominio es el espacio de textos cifrados C.

El descifrado se representa como una función:

$$d : K \times C \to m$$

Queremos que el descifrado revierta el cifrado:

$$d(k, e(k, m)) = m \quad \forall k \in K, \forall m \in M$$

Usando notación de subíndice para la clave, podemos escribir:

$$e_k : m \to C \quad \text{y} \quad d_k : C \to m$$

cumpliendo:

$$d_k(e_k(m)) = m \quad \forall m \in M$$

Esto implica que e_k es una función inyectiva. Es decir, si $e_k(m) = e_k(m')$, entonces:

$$m = d_k(e_k(m)) = d_k(e_k(m')) = m' \quad \forall k \in K, \forall m \in M$$

Propiedades deseables de un cifrado simétrico

Sea (K, m, C, e, d) un sistema de cifrado. Este debe cumplir las siguientes propiedades:

1. Para todo $k \in K$ y $m \in M$, debe ser fácil computar $e_k(m)$.
2. Para todo $k \in K$ y $c \in C$, debe ser fácil computar $d_k(c)$.
3. Dados uno o más textos cifrados $c_1, \ldots, c_n$ producidos con una misma clave k, debe ser computacionalmente difícil recuperar $d_k(c_1), \ldots, d_k(c_n)$ sin conocer k.

Con este tipo de criptografía podemos garantizar la confidencialidad porque únicamente quien posea la llave secreta será capaz de ver el mensaje. El problema con la criptografía simétrica es que si yo quisiera compartir secretos con n personas, para cada persona tendría que generar una nueva llave secreta. Otro problema asociado con este tipo de criptografía es cómo comparto con otra persona de una forma confidencial e integra la llave secreta. Estos problemas se resuelven de cierta manera con criptografía asimétrica.

2.3.1. Cifrado de bloque

Los cifrados de bloque cifran mensajes dividiéndolos en grupos de símbolos del mismo tamaño (bloques) y aplicando sobre cada uno de ellos un mismo algoritmo de cifra.

Llamamos B el tamaño de bloque del cifrado. Un mensaje de texto claro general consiste entonces en una lista de bloques de mensaje escogidos de M, y la función de cifrado transforma estos bloques en una lista de bloques de texto cifrado en C, donde cada bloque es una secuencia de B bits. Si el texto claro termina con un bloque de menos de B bits, rellenamos el final del bloque con ceros.

El cifrado y el descifrado se realizan bloque a bloque, por lo que basta estudiar el proceso para un único bloque de texto claro, es decir, para un solo $m \in M$. Esto, por supuesto, explica la conveniencia de dividir un mensaje en bloques: un mensaje puede ser de longitud arbitraria, y resulta útil centrar el proceso criptográfico en una pieza de longitud fija. El bloque de texto claro m es una cadena de B bits, que para mayor concreción identificamos con el número correspondiente en forma binaria. En otras palabras, identificamos M con el conjunto de enteros m que satisfacen $0 \leq m < 2^B$ mediante

$$\underbrace{m_{B-1}\, m_{B-2} \cdots m_2\, m_1\, m_0}_{\text{lista de } B \text{ bits de } m}$$

$\longleftrightarrow$

$$\underbrace{m_{B-1} \cdot 2^{B-1} + m_{B-2} \cdot 2^{B-2} + \cdots + m_1 \cdot 2 + m_0}_{\text{entero entre 0 y } 2^B - 1}$$

Aquí $m_0, m_1, \ldots, m_{B-1}$ son cada uno 0 o 1.

De manera similar, identificamos el espacio de claves K y el espacio de textos cifrados C con conjuntos de enteros correspondientes a cadenas de bits de un determinado tamaño de bloque. Para mayor simplicidad notacional, denotamos los tamaños de bloque para claves, textos claros y textos cifrados por B_k, B_m y B_c, respectivamente. No tienen por qué coincidir. Así, hemos identificado

$$K = \{\, k \in \mathbb{Z} : 0 \leq k < 2^{B_k} \}$$

$$M = \{\, m \in \mathbb{Z} : 0 \leq m < 2^{B_m} \}$$

$$C = \{\, c \in \mathbb{Z} : 0 \leq c < 2^{B_c} \}$$

Surge de inmediato una pregunta importante: ¿qué tan grande debe ser el conjunto K que se debe de escoger, equivalentemente, qué tamaño de bloque de clave B_k se debería de elegir? Si B_k es demasiado pequeño, se puede probar cada número entre 0 y $2^{B_k} - 1$ hasta dar con la clave generada. Más precisamente, dado que se asume que se conoce el algoritmo de descifrado d (principio de Kerckhoffs), toma cada $k \in K$ y calcula $d_k(c)$. Suponiendo que se es capaz de distinguir entre textos claros válidos e inválidos, se acabará recuperando el mensaje.

Este ataque se conoce como ataque de búsqueda exhaustiva (o ataque de fuerza bruta), puesto que se explora exhaustivamente el espacio de claves. Con la tecnología actual, una búsqueda exhaustiva se considera inviable si el espacio tiene al menos 2^{80} elementos. Por tanto, se debería de escoger $B_k \geq 80$.

Para muchos criptosistemas, existen refinamientos al ataque de búsqueda exhaustiva que sustituyen efectivamente el tamaño del espacio por su raíz cuadrada. Estos métodos se basan en el principio de que es más fácil encontrar objetos coincidentes (colisiones) en un conjunto que localizar un objeto en particular. Describimos algunos de estos ataques de "man-in-the-middle.º de colisiones en la sección 2.6. Si están disponibles ataques de este tipo, se deberá de escoger $B_k \geq 160$.

Existen distintos modos de funcionamiento para los cifrados de bloques. El *Electronic CodeBook* (ECB) y el *Cipher-Block Chanining* (CBC) son dos de los modos más antiguos. Entre los sistemas de cifrado por bloques más conocidos están el Data Encryption Standard, Triple DES (3DES)[13], Advanced Encryption Standard (AES)[14] y Twofish[15][16].

ECB

El *Electronic CodeBook* (ECB) es el modo de funcionamiento más sencillo del cifrado por bloques. Es más fácil porque se cifra directamente cada bloque de texto plano de entrada y la salida es en forma de bloques de texto cifrado. Generalmente, si un mensaje tiene un tamaño superior a b bits, se puede dividir en un montón de bloques y repetir el procedimiento. En la figura se muestra un ejemplo de funcionamiento del mismo 2.4

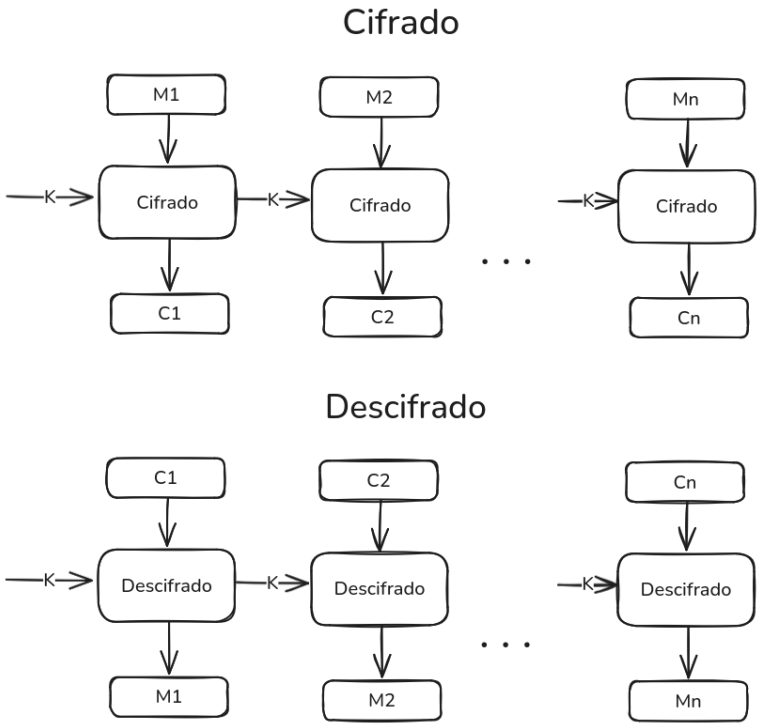

Figura 2.4: Ejemplo de cifrado y descifrado de bloque ECB

CBC

El modo *Cipher Block Chaining* (CBC) mejora la seguridad del modo ECB al encadenar cada bloque cifrado con el siguiente. Antes de entrar en su explicación, en este modo se introduce un nuevo concepto: Un IV es esencialmente otro input (además del texto plano M y la llave K) usado para cifrar el texto. Es un bloque de datos, usado por varios modos de cifrado de bloques para introducir aleatoriedad en el proceso de cifrado para poder generar como resultado distintos textos cifrados incluso con el mismo texto plano cifrado varias veces.

No es necesario que sea secreto, pero si es necesario que no se vuelva a usar. Es recomendable que sea aleatorio, impredicible y de un único uso. De

forma simplificada:

- Para cada bloque M_i de texto claro:

$$C_i = E\big(M_i \oplus C_{i-1}, K\big)$$

  donde C_{i-1} es el bloque cifrado anterior.

- Para el primer bloque, no existe C_0, por lo que se usa un vector de inicialización (IV) aleatorio:

$$C_1 = E\big(M_1 \oplus \text{IV}, K\big).$$

- De este modo, incluso si $M_i = M_j$, normalmente $C_i \neq C_j$ porque $C_{i-1} \neq C_{j-1}$.

Pasos de cifrado

1. Calcular $X_1 = M_1 \oplus \text{IV}$ y luego $C_1 = E(X_1, K)$.
2. Para $i > 1$, calcular $X_i = M_i \oplus C_{i-1}$ y luego $C_i = E(X_i, K)$.
3. Repetir hasta procesar todos los bloques de texto claro.

Pasos de descifrado

1. Descifrar $D_1 = D(C_1, K)$ y obtener $M_1 = D_1 \oplus \text{IV}$.
2. Para $i > 1$, descifrar $D_i = D(C_i, K)$ y obtener $M_i = D_i \oplus C_{i-1}$.
3. Repetir hasta recuperar todos los bloques de texto original.

En la figura se muestra un ejemplo de funcionamiento del mismo 2.5

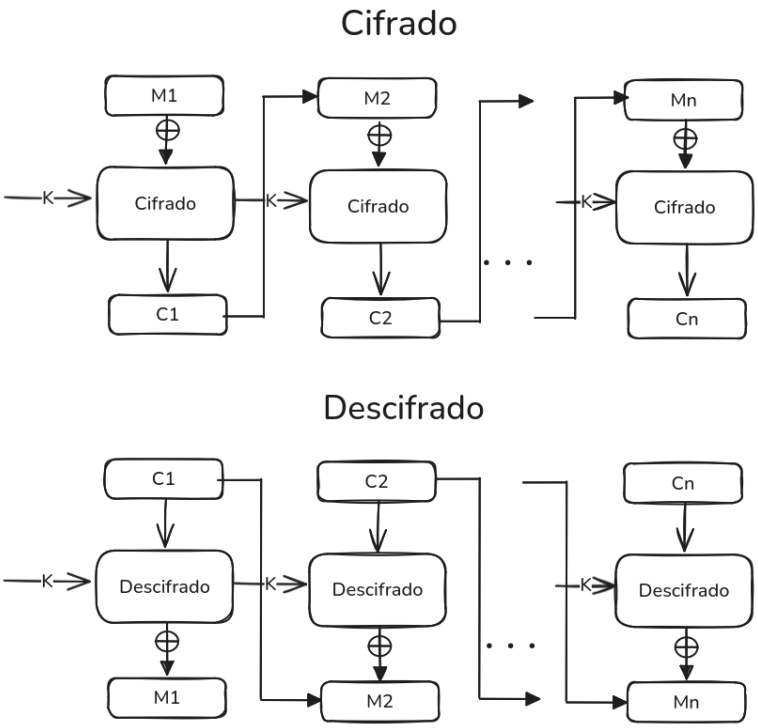

Figura 2.5: Ejemplo de cifrado y descifrado de bloque CBC

Una desventaja del mismo es que si se reutiliza el mismo IV para dos mensajes idénticos, se obtendrán los mismos bloques C_1, lo que compromete seguridad[15].

2.3.2. Cifrado de flujo

Los cifrados de flujo son ciertamente cifrados de bloque con una longitud de bloque igual a uno. Lo que los hace útiles es que la transformación de cifrado puede cambiar para cada símbolo del texto plano que se cifra.También pueden usarse cuando los datos deben procesarse símbolo por símbolo (por ejemplo, si el equipo no dispone de memoria o el almacenamiento intermedio es limitado).

Sea K el espacio de claves. Una secuencia de símbolos $e_1e_2e_3\ldots \in K$ se denomina **flujo de claves** o *keystream*.

Sea A un alfabeto de q símbolos, y sea E_e un cifrado por sustitución simple con longitud de bloque 1, donde $e \in K$. Dado un texto claro $m_1m_2m_3\ldots$ y un flujo de claves $e_1e_2e_3\ldots$, un cifrado de flujo produce un texto cifrado $c_1c_2c_3\ldots$ tal que:

$$c_i = E_{e_i}(m_i)$$

Si d_i denota la función inversa de e_i, entonces:

$$D_{d_i}(c_i) = m_i$$

Este tipo de criptografía se basa en hacer un cifrado bit a bit, esto se logra usando la operación XOR. Se utiliza un algoritmo determinístico que genera una secuencia pseudoaletoria de bits que junto con los bits del mensaje se van cifrando utilizando la operación XOR.

Un ejemplo de ello es el cifrado de vernam [17], definido sobre $A = \{0, 1\}$. Dado un mensaje binario $m_1 m_2 \dots m_t$ y una clave binaria $k_1 k_2 \dots k_t$ de la misma longitud, el texto cifrado es:

$$c_i = m_i \oplus k_i, \quad 1 \leq i \leq t$$

Si la clave es aleatoria y no se reutiliza, se denomina **one-time pad**.

Observación: Hay dos cifrados por sustitución posibles sobre A:

- E_0: $0 \mapsto 0$, $1 \mapsto 1$
- E_1: $0 \mapsto 1$, $1 \mapsto 0$

El flujo de claves determina cuál se aplica a cada símbolo.

Algunos ejemplos de cifrado de flujo son RC4 (usado en redes inalámbricas)[18, 19] y A5 (usado en telefonía celular)[20, 21].

2.4. Criptografía de clave pública o asimétrica

Si Alice y Bob quieren intercambiar mensajes usando un cifrado simétrico, deben primero acordar mutuamente una clave secreta k. Esto es factible si tienen la oportunidad de reunirse en secreto o si pueden comunicarse una vez por un canal seguro. Pero ¿qué ocurre si no disponen de esa oportunidad y cada comunicación entre ellos es vigilada por su adversaria Eva? ¿Es posible que Alice y Bob intercambien una clave secreta en estas condiciones?

La primera reacción de la mayoría es que no es posible, ya que el intruso ve cada fragmento de información que intercambian. Fue la brillante intuición de Diffie y Hellman que, bajo ciertas hipótesis, sí lo es. La búsqueda de soluciones eficientes (y demostrables) a este problema, denominado criptografía de clave pública (o asimétrica), constituye una de las partes más interesantes de la criptografía matemática.

Definimos espacios de claves K, de textos claros M y de textos cifrados C. Sin embargo, un elemento k del espacio de claves es en realidad un par de claves,

$$k = \big(k_{\text{priv}},\, k_{\text{pub}}\big),$$

denominadas clave privada y clave pública, respectivamente. Para cada clave pública k_{pub} existe una función de cifrado correspondiente

$$e_{k_{\text{pub}}} : M \longrightarrow C,$$

y para cada clave privada k_{priv} existe una función de descifrado correspondiente

$$d_{k_{\text{priv}}} : C \longrightarrow M.$$

Estas funciones cumplen que si el par $\left(k_{\text{priv}}, k_{\text{pub}}\right)$ pertenece al espacio de claves K, entonces

$$d_{k_{\text{priv}}}\left(e_{k_{\text{pub}}}(m)\right) = m \quad \forall\, m \in M.$$

Para que un cifrado asimétrico sea seguro, debe resultar difícil para el intruso calcular la función de descifrado $d_{k_{\text{priv}}}(c)$, incluso si conoce la clave pública k_{pub}. Bajo esta suposición, Alice puede enviar k_{pub} a Bob mediante un canal inseguro, y Bob puede devolver el texto cifrado $e_{k_{\text{pub}}}(m)$ sin temor a que el intruso lo descifre. Para poder descifrar, es necesario conocer la clave privada k_{priv}, y, presumiblemente, solo Alice dispone de ella. A esta clave privada a veces se le llama información de *trapdoor* de Alice, porque proporciona una vía directa para calcular la función inversa de $e_{k_{\text{pub}}}$. El hecho de que las claves de cifrado y descifrado (k_{pub} y k_{priv}) sean diferentes confiere al cifrado su carácter asimétrico, de ahí su nombre.

Resulta fascinante que Diffie y Hellman concibieran este concepto sin contar con un par concreto de funciones; no obstante, sí propusieron un método análogo por el cual Alice y Bob pueden intercambiar de forma segura un dato aleatorio cuyo valor no conocen inicialmente. Describimos el método de intercambio de claves de Diffie y Hellman en la Sección 2.4.1 y, a continuación, analizaremos más adelante varios cifrados asimétricos como RSA (Sección) y ECC (Sección), cuya seguridad se fundamenta en la dificultad presunta de distintos problemas matemáticos.

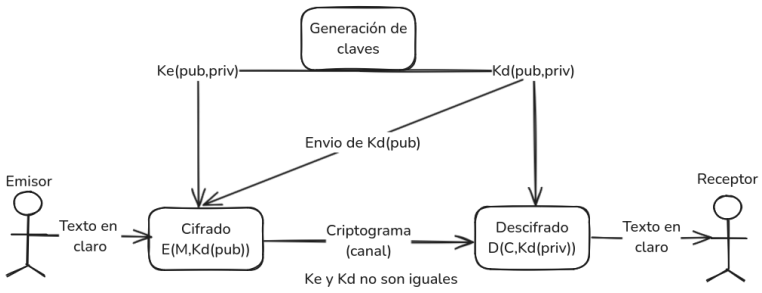

Figura 2.6: Ejemplo de cifrado asimétrico

2.4.1. Intercambio de claves Diffie–Hellman

Como hemos mencionado anteriormente, el protocolo de Diffie-Hellman, publicado en 1976 [22], resuelve el problema de dos entidades que desean compartir una clave secreta para usar en un cifrado simétrico, pero su único canal de comunicación es inseguro y todo lo que intercambian lo observa

el intruso. ¿Cómo pueden establecer una clave que el intruso no pueda deducir? La brillante idea de Diffie y Hellman fue aprovechar la dificultad del problema del logaritmo discreto en $\mathbb{F}_p^*$. A continuación, se muestra el procedimiento del mismo:

1. Alice y Bob acuerdan públicamente un primo grande p y un generador g de $\mathbb{F}_p^*$.
2. Alice elige en secreto un entero a y calcula
$$A \equiv g^a \pmod{p}.$$
3. Bob elige en secreto un entero b y calcula
$$B \equiv g^b \pmod{p}.$$
4. Intercambian A y B por el canal público. Eve conoce p, g, A, B.
5. Alice calcula la clave compartida
$$K = B^a \equiv g^{ba} \pmod{p},$$
y Bob calcula
$$K = A^b \equiv g^{ab} \pmod{p}.$$

Fase	**Operación**
Intercambio: elegir p y g	Un tercero de confianza elige y publica un primo p grande y un generador g de orden primo en $\mathbb{F}_p^*$.
Generación de claves secretas	Alice elige a, calcula $A \equiv g^a \pmod{p}$; Bob elige b, calcula $B \equiv g^b \pmod{p}$.
Intercambio público	Alice envía A a Bob, Bob envía B a Alice.
Cálculos finales	Alice calcula $K \equiv B^a \pmod{p}$; Bob calcula $K \equiv A^b \pmod{p}$.

Cuadro 2.1: Resumen del intercambio de claves Diffie–Hellman

Ejemplo 2.1. *Supongamos que Alice y Bob acuerdan $p = 7$ y $g = 5$. Alice elige $a = 3$ y calcula*
$$A = 5^3 \equiv 125 \equiv 6 \pmod{7},$$
mientras que Bob elige $b = 4$ y obtiene
$$B = 5^4 \equiv 625 \equiv 2 \pmod{7}.$$

Tras intercambiar $A = 6$ y $B = 2$, ambos calculan

$$K = 2^3 \equiv 8 \equiv 1 \pmod 7, \quad K = 6^4 \equiv 1296 \equiv 1 \pmod 7,$$

de modo que la clave compartida es $K = 1$. El intruso ve p, g, A, B pero, para calcular $K = g^{ab}$, tendría que resolver el problema del logaritmo discreto en $\mathbb{F}_7^*$, lo cual es prácticamente imposible cuando p tiene cientos de dígitos.

2.4.2. Firma digital

TODO: preguntar si es necesario esta seccion

2.5. Resumen comparativo

Los sistemas criptográficos actuales suelen combinar ambas: por ejemplo, se utiliza un esquema de clave pública para establecer una clave de sesión y luego un cifrado simétrico para procesar los datos.

Hasta la fecha, la encriptación de clave pública es sustancialmente más lenta que la simétrica; sin embargo, no existe prueba de que deba serlo necesariamente. En la práctica, los puntos clave son:

1. La criptografía de clave pública facilita la firma eficiente, el no repudio) y la gestión de claves.

2. La criptografía de clave simétrica es muy eficiente para cifrar datos y para aplicaciones de integridad.

Observación (tamaños de clave) Para lograr un nivel de seguridad equivalente, las claves privadas en sistemas de clave pública (por ejemplo, 2048 o 3072 bits en RSA) deben ser mucho más largas que las secretas en sistemas simétricos (por ejemplo, 128 bits en AES). Esto se debe a que, mientras que el ataque más eficiente contra la simétrica es la búsqueda exhaustiva, los sistemas de clave pública conocidos admiten algoritmos *atajo* (por ejemplo, factorización) más eficientes que el barrido completo del espacio de claves. Por tanto, para igualar los 128 bits simétricos, se requieren claves RSA de unos 3072 bits, es decir, más de 24 veces más largas, aunque valores del orden de 10 veces mayores se citan frecuentemente en la literatura [23].

2.6. Principios de diseño criptográfico

En 1883, Auguste Kerckhoffs [24] formuló seis principios esenciales para el diseño de sistemas criptográficos:

1. Debe ser, cuanto menos, imposible de romper en la práctica.

2. La seguridad no debe depender del secreto del algoritmo, sino únicamente de la clave.

3. La clave debe poder ser almacenada, pudiendo ser reemplazable.

4. El texto cifrado debe poder transmitirse sin errores por medios estándar.

5. El sistema debe ser portátil y operable por una sola persona.

6. El sistema debe ser fácil de usar, sin requerir un esfuerzo mental excesivo.

2.7. Seguridad y modelos de ataque

La seguridad de un sistema criptográfico se evalúa frente a distintos modelos de ataque, que describen el acceso y las capacidades del adversario. A grandes rasgos, estos modelos se agrupan en:

2.7.1. Ataques pasivos y activos

Ataques pasivos: El atacante únicamente observa el tráfico cifrado (o el sistema) sin modificarlo. Incluyen la intercepción de mensajes y el análisis de tráfico, en los que se extrae información de patrones de comunicación [25][26].

Ataques activos: El atacante altera, inyecta o interfiere el flujo de datos. Ejemplos típicos son el man-in-the-middle (interposición entre emisor y receptor) y la modificación de mensajes en tránsito [27].

2.7.2. Modelos clásicos de criptoanálisis

- *Ataque por texto cifrado únicamente (COA):* El atacante dispone solo del texto cifrado. Es el escenario más realista pero también el más limitado.

- *Fuerza bruta (exhaustiva):* Se calculan y prueban todas las claves posibles hasta dar con la correcta. Su dificultad depende únicamente del tamaño de la clave [28][29].

- *Ataque con texto plano conocido (KPA):* Se accede a uno o varios pares (texto plano, texto cifrado), lo que permite reducir el espacio de búsqueda.

- *Ataque por texto plano elegido (CPA):* El atacante elige ciertos textos para cifrar y obtiene sus correspondientes cifrados. Es fundamental en la seguridad de sistemas de clave pública.

- *Ataque adaptativo por texto plano elegido (CPA2):* Variante de CPA donde el atacante decide nuevos textos a cifrar tras ver resultados anteriores.
- *Ataque por texto cifrado elegido (CCA):* El atacante puede descifrar libremente ciertos cifrados de su elección, obteniendo los planos resultantes.
- *Ataque adaptativo por texto cifrado elegido (CCA2):* Versión más poderosa de CCA, donde cada descifrado elegido se basa en análisis previos.

2.7.3. Ataques avanzados

- *Canal lateral (side-channel):* Se explota información derivada de la implementación (tiempos, consumo eléctrico, emisiones electromagnéticas) para inferir claves [30].
- *Ataque de clave relacionada (related-key):* El atacante conoce cifrados de textos planos bajo claves que guardan alguna relación matemática con la clave objetivo.
- *Sesgo estadístico (bias attacks):* Se buscan patrones no aleatorios en la generación de claves o estados intermedios para acotar posibles valores [31].
- Ataque *Evil Maid*: Consiste en acceso físico al dispositivo para instalar malware o keyloggers y robar claves cuando el usuario no está presente.

Capítulo 3

Campos finitos

La aritmética en campos finitos es fundamental para numerosos algoritmos de clave pública, incluidos los basados en el problema del logaritmo discreto en campos finitos, los esquemas de curvas elípticas y las aplicaciones emergentes de curvas hiperelípticas. El rendimiento de estos protocolos depende en gran medida de nuestra capacidad para realizar rápidamente operaciones básicas en el campo subyacente.

Los campos finitos (también llamados cuerpo finito o cuerpo de Galois) se denotan como GF(p^m), donde p es un número primo y m un entero positivo. Para que los sistemas de curvas elípticas sean eficientes, es imprescindible implementar de manera óptima la suma, la resta, la multiplicación y la inversión en dicho campo. Existen tres familias de campos que resultan especialmente adecuadas para ECC (Criptografía de Curvas Elípticas):

- **Campos primos** ($\mathbb{F}_p$),
- **Campos binarios** ($\mathbb{F}_{2^m}$),
- **Campos de extensión óptima** (OEF).

En las secciones siguientes se presenta una introducción informal a la teoría de campos finitos y se describen en detalle los algoritmos eficientes para cada tipo de campo.

Dado que en muchas aplicaciones los tamaños de p^m son muy grandes, entender la complejidad computacional de estas operaciones es clave para evaluar el rendimiento global de los esquemas criptográficos basados en curvas elípticas.

Nota: $GF(p^m) == \mathbb{F}_{p^m}$.

3.1. Introducción a campos finitos

Los campos son abstracciones de sistemas numéricos familiares (como los racionales $\mathbb{Q}$, los reales $\mathbb{R}$ o los complejos $\mathbb{C}$) que reúnen ciertas propiedades

esenciales. Un campo $\mathbb{F}$ consta de un conjunto $\mathbb{F}$ junto con dos operaciones, suma $(+)$ y producto $(\cdot)$, que satisfacen:

1. $(\mathbb{F}, +)$ es un grupo abeliano con identidad suma denotada por 0.
2. $(\mathbb{F} \setminus \{0\}, \cdot)$ es un grupo abeliano con identidad multiplicativa denotada por 1.
3. La multiplicación distribuye sobre la suma:

$$(a + b) \cdot c = a \cdot c + b \cdot c \quad \text{para todo } a, b, c \in \mathbb{F}.$$

Si el conjunto $\mathbb{F}$ es finito, se dice que el campo es finito.

En un campo, la resta se define mediante la suma de inversos aditivos: para $a, b \in \mathbb{F}$,

$$a - b \;=\; a + (-b),$$

donde $-b$ es el elemento único que satisface $b + (-b) = 0$.

La división se define usando el inverso multiplicativo: para $a, b \in \mathbb{F}$ con $b \neq 0$,

$$\frac{a}{b} \;=\; a \cdot b^{-1},$$

siendo b^{-1} el único elemento que cumple $b \cdot b^{-1} = 1$.

El *orden* de un campo finito es el número de sus elementos. Existe un campo finito de orden q si y solo si q es una potencia prima, es decir, $q = p^m$ con p primo (característica del campo) y m entero positivo.

- Si $m = 1$, el campo se llama *campo primo* y se denota $\mathbb{F}_p$.
- Si $m \geq 2$, se llama *campo de extensión* y se denota $\mathbb{F}_{p^m}$.

Para cada potencia prima q, existe esencialmente un único campo finito de orden q: todos ellos son isomorfos (idénticos en estructura, aunque difiera la representación de sus elementos). Por ello se usa la notación general $\mathbb{F}_q$.

3.1.1. Representación binaria y suma de enteros

Antes de entrar en los campos primos y binarios e Todo entero no negativo a tiene una representación binaria única

$$a \;=\; \sum_{i=0}^{n-1} a_i 2^i, \quad a_i \in \{0, 1\}, \quad a_{n-1} \neq 0.$$

Los dígitos binarios a_i se llaman *bits*, y decimos que a es un entero de n bits. Para representar enteros negativos se añade un bit de signo.

Para sumar dos enteros en binario aplicamos el método enseñado en primaria, sumando uno a uno(bit a bit) y llevando el 'dígito de más' hacia la

siguiente suma cuando sea necesario. Por ejemplo, para calcular $43+37 = 80$ en binario:

$$\begin{array}{r} {\color{red}101111} \\ 101011 \\ +100101 \\ \hline 101000 \end{array}$$

Los 'dígitos de más' aparecen en rojo.

La implementación en hardware suele usar un *sumador de 1 bit* que toma dos bits a_i, b_i y una llevada de entrada c_i, y calcula una suma s_i y una llevada de salida c_{i+1} como:

$$\begin{aligned} c_{i+1} &= (a_i \wedge b_i) \ \vee \ (c_i \wedge a_i) \ \vee \ (c_i \wedge b_i), \\ s_i &= a_i \oplus b_i \oplus c_i, \end{aligned}$$

donde $\wedge$, $\vee$ y $\oplus$ son las funciones booleanas AND, OR y XOR, respectivamente.

Encadenando $n+1$ de estos sumadores de 1 bit podemos sumar dos números de n bits usando $7n + 7 = O(n)$ operaciones booleanas sobre bits individuales.

Observación 3.1. *El encadenamiento de sumadores, o* ripple addition*, impone un cómputo secuencial y ya casi no se utiliza en hardware real. En su lugar se emplean esquemas de* carry-lookahead *que permiten paralelizar la generación de las llevadas. Gracias a esto, los procesadores modernos pueden sumar en un solo ciclo de reloj enteros de 64 o incluso 128 bits, y con instrucciones SIMD (Single Instruction Multiple Data) realizar varias sumas de 64 bits en paralelo por ciclo.*

Otra opción es agrupar los bits en palabras de 64 bits:

$$a = \sum_{i=0}^{k-1} a_i \, 2^{64i}, \quad k = \lceil \tfrac{n}{64} \rceil.$$

Con ello, sumamos dos números de n bits en $O(k) = O(n)$ operaciones de palabra de 64 bits. Cada operación de palabra se ejecuta internamente con circuitos de bits de coste constante, de modo que el coste total sigue siendo $O(n)$ en términos de operaciones a nivel de bit.

3.2. Aritmética en $\mathbb{F}_p$

Sea p un número primo. El conjunto

$$\mathbb{F}_p = \{0, 1, 2, \ldots, p-1\},$$

con las operaciones de suma y multiplicación módulo p, $\mathbb{Z}/p\mathbb{Z}$, constituye un campo finito de orden p. Denotaremos este campo por $\mathbb{F}_p$ y llamaremos a p el *módulo* de $\mathbb{F}_p$. Para cualquier entero a, a mód p denota el único residuo r, $0 \leq r \leq p-1$, obtenido al dividir a por p; esta operación se llama *reducción módulo p*.

Todos los elementos no nulos de $\mathbb{F}_p$ tienen inverso multiplicativo, y forman un grupo cíclico de orden $p-1$ (esto es una consecuencia del pequeño Teorema de Fermat).

Ejemplo 2.1 (campo primo $\mathbb{F}_{31}$) Los elementos de $\mathbb{F}_{31}$ son $\{0, 1, 2, \ldots, 30\}$. A continuación se muestran algunas operaciones en $\mathbb{F}_{31}$:

1. **Suma:** $9 + 25 = 34 \equiv 3 \pmod{31}$.
2. **Resta:** $9 - 25 = -16 \equiv 15 \pmod{31}$.
3. **Multiplicación:** $9 \cdot 25 = 225 \equiv 8 \pmod{31}$.
4. **Inverso multiplicativo:** $9^{-1} = 7$ ya que $9 \cdot 7 = 63 \equiv 1 \pmod{31}$.

3.2.1. Suma en $\mathbb{F}_p$

Para sumar dos elementos de $\mathbb{F}_p \cong \mathbb{Z}/p\mathbb{Z}$, representados como enteros únicos en $[0, p-1]$, basta sumar los enteros y comprobar si el resultado es $\geq p$; en tal caso restamos p para obtener un valor en $[0, p-1]$. De forma análoga, tras restar dos enteros, sumamos p si el resultado es negativo. El coste total sigue siendo $O(n)$ operaciones a nivel de bit, donde $n = \lg p$ es el número de bits necesarios para representar un elemento del campo.

Para sumar o restar dos elementos de $\mathbb{F}_q \cong (\mathbb{Z}/p\mathbb{Z})[x]/(f)$, simplemente sumamos o restamos los coeficientes correspondientes de los polinomios, con un coste total de $O(d \lg p)$ operaciones de bit, donde $d = \deg f$. Nuevamente, si definimos $n = \lg q = d \lg p$, el coste es $O(n)$ operaciones de bit.

3.2.2. Multiplicación de enteros

A diferencia de la suma, este problema se cree ampliamente que $O(n \log n)$ es lo óptimo y bajo ciertas conjeturas incluso se ha demostrado condicionalmente, pero no se conoce de forma incondicional. Sólo muy recientemente se estableció esa cota superior. Dado que no conocemos la complejidad exacta, es habitual usar la notación $M(n)$ para denotar el tiempo de multiplicar dos enteros de n bits. Esto permite expresar límites para algoritmos que dependen de la complejidad de la multiplicación de enteros sin atarse al estado actual del arte. En las últimas dos décadas, las cotas superiores de $M(n)$ han mejorado al menos cuatro veces.

Calculemos $37 \times 43 = 1591$ con el método "de la escuela" en binario:

$$\begin{array}{r} 101011 \\ \times\ 100101 \\ \hline 101011 \\ 101011 \\ +101011 \\ \hline 11000110111 \end{array}$$

Multiplicar bits individuales es sencillo (una compuerta AND), pero requiere n^2 multiplicaciones de bit, seguidas de n sumas de números de n bits (desplazados adecuadamente). La complejidad de este algoritmo es $\Theta(n^2)$, lo que da la cota superior $M(n) = O(n^2)$. La única cota inferior conocida es la trivial $M(n) = \Omega(n)$, por lo que podríamos esperar mejorar $O(n^2)$, y de hecho se han desarrollado algoritmos más rápidos.

3.2.3. Algoritmo de Karatsuba

El algoritmo de Karatsuba se basa en un enfoque "divide y vencerás". En lugar de representar los enteros de n bits con n dígitos en base 2, podemos escribirlos en base $2^{n/2}$ y calcular su producto del modo siguiente:

$$a = a_0 + 2^{n/2} a_1, \qquad b = b_0 + 2^{n/2} b_1,$$

$$ab = a_0 b_0 \ + \ 2^{n/2}\,(a_1 b_0 + a_0 b_1) \ + \ 2^n\, a_1 b_1.$$

Esto reduce la multiplicación de n bits a cuatro multiplicaciones de $\frac{n}{2}$ bits y tres sumas de enteros de $O(n)$ bits (desplazar por potencias de 2 es gratuito, basta con "mover" el resultado binario). Sin embargo, Karatsuba observó que mediante la identidad

$$a_1 b_0 + a_0 b_1 = (a_0 + a_1)(b_0 + b_1) \ - \ a_0 b_0 \ - \ a_1 b_1$$

podemos obtener $a_1 b_0 + a_0 b_1$ usando solo una multiplicación adicional (más las dos ya necesarias para $a_0 b_0$ y $a_1 b_1$), junto con sumas y restas. Así, reutilizando $a_0 b_0$ y $a_1 b_1$, el producto ab se calcula con tres multiplicaciones y seis sumas/restas de enteros de tamaño $O(n)$.

Aplicando recursivamente esta misma idea a cada subproducto, la complejidad de tiempo $T(n)$ satisface

$$T(n) \ = \ 3\,T\big(\tfrac{n}{2}\big) \ + \ O(n) \ = \ O\big(n^{\log_2 3}\big),$$

luego

$$M(n) \ = \ O\big(n^{\log_2 3}\big), \quad \text{donde } \log_2 3 \approx 1.59.$$

Ejemplo del algoritmo de Karatsuba

Multipliquemos dos enteros de 4 bits usando Karatsuba:

$$a = 1011_2 = 11, \quad b = 1101_2 = 13,$$

dividimos en mitades de 2 bits:

$$a_1 = 10_2 = 2, \quad a_0 = 11_2 = 3, \quad b_1 = 11_2 = 3, \quad b_0 = 01_2 = 1.$$

1. Calcular $p = a_0 b_0 = 3 \times 1 = 3 = 0011_2$.
2. Calcular $r = a_1 b_1 = 2 \times 3 = 6 = 0110_2$.
3. Calcular

   $$s = (a_0 + a_1)(b_0 + b_1) - p - r = 5 \times 4 - 3 - 6 = 20 - 9 = 11 = 1011_2.$$

4. Desplazar s dos bits (multiplicar por 2^2) y r cuatro bits (2^4):

   $$p = 00000011_2, \quad s \ll 2 = 00101100_2, \quad r \ll 4 = 01100000_2.$$

5. Sumar en binario (con acarreo habitual):

$$\begin{array}{r} 00000011 \\ +\,00101100 \\ \hline 00101111 \\ +\,01100000 \\ \hline 10001111_2 = 143. \end{array}$$

En efecto, $11 \times 13 = 143$, y hemos usado solo tres multiplicaciones de 2 bits y algunas sumas/restas de tamaño constante.

La eficiencia de la multiplicación en campos finitos es crucial en esquemas de curvas elípticas, ya que, las limitaciones de los multiplicadores enteros y el coste del acarreo pueden convertirse en cuellos de botella en implementaciones directas.

3.3. Aritmética en $\mathbb{F}_{2^m}$

Los campos finitos de orden 2^m, también llamados *campos binarios*, se construyen como extensiones de polinomios:

$$\mathbb{F}_{2^m} \cong \mathbb{F}_2[x]/(f(x)),$$

donde $f(x)$ es un polinomio irreducible de grado m sobre $\mathbb{F}_2$. En tal campo, cada elemento puede representarse como un polinomio de grado ¡m con coeficientes en $0, 1$ (equivalentemente, como un vector binario de m bits).

3.3.1. Representación polinomial

Sea $f(z)$ un polinomio binario irreducible de grado m, y escríbalo como

$$f(z) = z^m + r(z).$$

Los elementos de $\mathbb{F}_{2^m}$ son los polinomios binarios de grado a lo sumo $m-1$. La suma de elementos es la suma habitual de polinomios sobre $\mathbb{F}_2$, y la multiplicación se realiza mód $f(z)$.

A un elemento

$$a(z) = a_{m-1}z^{m-1} + \cdots + a_1 z + a_0$$

se le asocia el vector binario $a = (a_{m-1}, \ldots, a_1, a_0)$ de longitud m. Definamos

$$t = \lceil \tfrac{m}{W} \rceil, \quad s = Wt - m.$$

En software, a puede almacenarse en un arreglo de t palabras de W bits:

$$A = \big(A[t-1], \ldots, A[1], A[0]\big),$$

donde el bit menos significativo de $A[0]$ es a_0, y los s bits más significativos de $A[t-1]$ quedan sin usar (siempre a cero).

$$\underbrace{A[t-1] \;\; A[t-2] \;\; \cdots \;\; A[1] \;\; A[0]}_{t \text{ palabras de } W \text{ bits}}$$

$$\underbrace{a_{m-1} \;\cdots\; a_{tW}}_{s \text{ bits sin usar}} \;\; a_{tW-1} \;\cdots\; a_{2W} \;\; a_{2W-1} \;\cdots\; a_W \;\; a_{W-1} \;\cdots\; a_0$$

Figura 3.1: Representación de $a \in \mathbb{F}_{2^m}$ como un arreglo A de palabras de W bits. Los $s = Wt - m$ bits de orden más alto de $A[t-1]$ permanecen sin usar.

Cada elemento se escribe como

$$a(x) = a_{m-1}x^{m-1} + \cdots + a_1 x + a_0, \quad a_i \in \{0, 1\}.$$

La suma de elementos se realiza como la suma de polinomios coeficiente a coeficiente (operación equivalente a XOR bit a bit, sin acarreo), y la multiplicación se reduce módulo $f(x)$:

$$a(x) \cdot b(x) \text{ mód } f(x).$$

La reducción $p(x)$ mód $f(x)$ es el residuo de grado ¡m tras la división larga.

Ejemplo (campo $\mathbb{F}_{2^3}$) Sea $f(x) = x^3 + x + 1$. Entonces

$$\mathbb{F}_{2^3} = \{0, 1, x, x+1, x^2, x^2+1, x^2+x, x^2+x+1\}.$$

1. Suma: $(x^2 + x + 1) + (x + 1) = x^2$.
2. Multiplicación: $(x^2 + x + 1)(x + 1) = x^3 + x + 1 \equiv x + 1 \pmod{f(x)}$.
3. Inverso: $(x^2 + x + 1)^{-1} = x^2$, pues $(x^2 + x + 1)x^2 = x^4 + x^3 + x^2 \equiv 1 \pmod{f(x)}$.

Ejemplo (isomorfismo de campos) Para $m = 3$ hay dos irreducibles: $f_1(x) = x^3 + x + 1$ y $f_2(x) = x^3 + x^2 + 1$. Los campos $\mathbb{F}_2[x]/(f_1)$ y $\mathbb{F}_2[x]/(f_2)$ tienen idénticos elementos y son isomorfos, pues existe un mapeo que envía la clase de x en uno a una raíz de f_1 en el otro.

3.3.2. Suma en $\mathbb{F}_{2^m}$

La suma de elementos de $\mathbb{F}_{2^m}$ se realiza bit a bit (XOR) en cada palabra, por lo que únicamente requiere t operaciones de palabra.

Algorithm 1 Suma en $\mathbb{F}_{2^m}$

Require: Arreglos $A[0..t-1]$ y $B[0..t-1]$ que representan los coeficientes de $a(z)$ y $b(z)$.
Ensure: Arreglo $C[0..t-1]$ que representa $c(z) = a(z) + b(z)$.

```
for i = 0 to t − 1 do
    C[i] ← A[i] ⊕ B[i]
end for
return C
```

Ejemplo de suma en $\mathbb{F}_{2^m}$ (bit a bit)

Consideremos $m = 4$, de modo que cada elemento se representa con 4 bits. Sean los polinomios

$$a(z) = z^3 + z + 1 \quad \longleftrightarrow \quad A = [1, 0, 1, 1],$$
$$b(z) = z^2 + z \quad \longleftrightarrow \quad B = [0, 1, 1, 0].$$

La suma $c(z) = a(z) + b(z)$ se obtiene bit a bit por XOR:

$$\begin{array}{rl} 1\,0\,1\,1 & (A) \\ \oplus\, 0\,1\,1\,0 & (B) \\ \hline 1\,1\,0\,1 & (C) \end{array}$$

Por tanto,

$$c(z) \longleftrightarrow C = [1, 1, 0, 1],$$

que corresponde al polinomio $z^3 + z^2 + 1$.

3.3.3. Multiplicación en $\mathbb{F}_{2^m}$

La técnica "shift-and-add"para la multiplicación en campos finitos se basa en la observación:

$$a(z) \cdot b(z) = \sum_{i=0}^{m-1} a_i \, z^i \, b(z).$$

En la iteración i, se calcula

$$b \leftarrow b \cdot z \bmod f(z)$$

(mediante un desplazamiento y, si el bit de más alto orden de b era 1, una reducción por $r(z)$) y, si $a_i = 1$, se acumula

$$c \leftarrow c + b.$$

Algorithm 2 Multiplicación "shift-and-add."[en] $\mathbb{F}_{2^m}$

Require: Polinomios binarios $A[0..t-1]$ y $B[0..t-1]$ que representan $a(z)$ y $b(z)$.
Ensure: Polinomio $C[0..t-1]$ que representa $c(z) = a(z) \cdot b(z)$ mód $f(z)$.

```
1: C ← 0
2: if a_0 = 1 then
3:    C ← B
4: end if
5: for i = 1 to m − 1 do
6:    Desplazar B un bit a la izquierda
7:    if el bit desplazado era 1 then
8:       B ← B ⊕ R
9:    end if
10:   if a_i = 1 then
11:      C ← C ⊕ B
12:   end if
13: end for
14: return C
```

Ejemplo de "shift-and-add"

Trabajemos en $\mathbb{F}_{2^4}$ con polinomio de reducción $f(x) = x^4 + x + 1$. Sean

$$a(x) = x^3 + x, \quad b(x) = x^2 + 1.$$

Los representamos como vectores de 4 bits:

$$a = [\,1, 0, 1, 0\,], \quad b = [\,0, 1, 0, 1\,].$$

1. Inicializar $C \leftarrow 0$. Como $a_0 = 0$, C sigue siendo 0.
2. $i = 1$: Desplazamos $b \to x\,b = x^3 + x$. Como $a_1 = 1$,
$$C \leftarrow C \oplus b = x^3 + x.$$
3. $i = 2$: Desplazamos $b \to x^4 + x^2 \equiv (x + 1) + x^2$ (reducción de x^4). Ahora $b = x^2 + x + 1$. Como $a_2 = 0$, C no cambia.
4. $i = 3$: Desplazamos $b \to x^3 + x^2 + x$. Como $a_3 = 1$,
$$C \leftarrow (x^3 + x) \oplus (x^3 + x^2 + x) = x^2.$$

El resultado final es
$$c(x) = x^2.$$

3.4. Aritmética en campos de extensión óptima (OEF)

En las secciones anteriores explicamos la aritmética en $\mathbb{F}_{p^m}$ para el caso $p = 2$ (campos binarios) y $m = 1$ (campos primos). La representación mediante base polinomial en el caso binario se generaliza a cualquier campo de extensión $\mathbb{F}_{p^m}$, con la aritmética de coeficientes realizada en $\mathbb{F}_p$.

La representación por base polinomial para campos binarios se generaliza a todos los campos de extensión de la siguiente manera. Sea p un primo y $m \geq 2$. Denotamos por $\mathbb{F}_p[z]$ el anillo de polinomios en la variable z con coeficientes en $\mathbb{F}_p$. Sea $f(z) \in \mathbb{F}_p[z]$ un polinomio irreducible de grado m—existen para cualquier par (p, m) y pueden hallarse eficientemente. La irreducibilidad significa que $f(z)$ no se factoriza como producto de polinomios en $\mathbb{F}_p[z]$ de grado menor que m. Entonces:

$$\mathbb{F}_{p^m} = \{a_{m-1}z^{m-1} + \cdots + a_1 z + a_0 : a_i \in \mathbb{F}_p\}$$

con suma habitual de polinomios y producto reducido módulo $f(z)$.

Para implementaciones en hardware, los campos binarios son atractivos porque las operaciones solamente requieren desplazamientos y sumas bit a bit (módulo 2). En software, esto puede resultar lento si existe un multiplicador de enteros en hardware. Por otro lado, la aritmética en campos primos exige gestionar la propagación de acarreo, lo que puede ser costoso.

La idea de los *campos de extensión óptima* (OEF) es elegir p, m y el polinomio de reducción de forma que se ajusten mejor al hardware, en específico, escoger p de modo que quepa en una sola palabra hace que se simplifique el manejo de acarreo.

Definición 3.1. *Un* campo de extensión óptima *(OEF) es un campo finito* $\mathbb{F}_{p^m}$ *tal que:*

1. $p = 2^n - c$ *para enteros* n *y* c *con* $\log_2 |c| \leq n/2$.
2. *Existe un polinomio irreducible* $f(z) = z^m - \omega$ *en* $\mathbb{F}_p[z]$.

Si $c \in \{\pm 1\}$ *se dice que el OEF es de* Tipo I *(y* p *es primo de Mersenne si* $c = 1$*); si* $\omega = 2$*, es de* Tipo II.

p	$f(z)$	Parámetros	Tipo
$2^7 + 3$	$z^{13} - 5$	$n = 7$, $c = -3$, $m = 13$, $\omega = 5$	–
$2^{13} - 1$	$z^{13} - 2$	$n = 13$, $c = 1$, $m = 13$, $\omega = 2$	I, II
$2^{31} - 19$	$z^6 - 2$	$n = 31$, $c = 19$, $m = 6$, $\omega = 2$	II
$2^{31} - 1$	$z^6 - 7$	$n = 31$, $c = 1$, $m = 6$, $\omega = 7$	I
$2^{32} - 5$	$z^5 - 2$	$n = 32$, $c = 5$, $m = 5$, $\omega = 2$	II
$2^{57} - 13$	$z^3 - 2$	$n = 57$, $c = 13$, $m = 3$, $\omega = 2$	II
$2^{61} - 1$	$z^3 - 37$	$n = 61$, $c = 1$, $m = 3$, $\omega = 37$	I
$2^{89} - 1$	$z^2 - 3$	$n = 89$, $c = 1$, $m = 2$, $\omega = 3$	I

Cuadro 3.1: Parámetros de ejemplo para OEF. Aquí $p = 2^n - c$ es primo y $f(z) = z^m - \omega$ es irreducible en $\mathbb{F}_p[z]$.

Los siguientes resultados permiten verificar la irreducibilidad de un binomio $z^m - \omega$ en $\mathbb{F}_p[z]$.

Teorema 3.1. *Sea* $m \geq 2$ *un entero y* $\omega \in \mathbb{F}_p^*$*. El binomio* $f(z) = z^m - \omega$ *es irreducible en* $\mathbb{F}_p[z]$ *si y solo si:*

1. *Cada factor primo de* m *divide el orden* e *de* ω *en* $\mathbb{F}_p^*$*, pero no divide* $(p-1)/e$.
2. *Si* $m \equiv 0 \pmod 4$*, entonces* $p \equiv 1 \pmod 4$.

Corolario 3.1. *Si* ω *es un elemento primitivo de* $\mathbb{F}_p^*$ *y* $m \mid (p-1)$*, entonces* $z^m - \omega$ *es irreducible en* $\mathbb{F}_p[z]$.

Los elementos de $\mathbb{F}_{p^m}$ son polinomios

$$a(z) = a_{m-1}z^{m-1} + \cdots + a_1 z + a_0, \quad a_i \in \mathbb{F}_p.$$

Ejemplo 3.4 (campo $\mathbb{F}_{251^5}$) Sea $p = 251$, $m = 5$ y $f(z) = z^5 + z^4 + 12z^3 + 9z^2 + 7$, irreducible en $\mathbb{F}_{251}[z]$. Entonces $\mathbb{F}_{251^5}$ consta de polinomios de grado < 5 con coeficientes en $\{0, 1, \ldots, 250\}$. Por ejemplo, sean:

$$a = 123z^4 + 76z^2 + 7z + 4, \quad b = 196z^4 + 12z^3 + 225z^2 + 76.$$

1. Suma: $a + b = 68z^4 + 12z^3 + 50z^2 + 7z + 80$.
2. Resta: $a - b = 178z^4 + 239z^3 + 102z^2 + 7z + 179$.
3. Multiplicación: $a \cdot b = 117z^4 + 151z^3 + 117z^2 + 182z + 217$.
4. Inverso: $a^{-1} = 109z^4 + 111z^3 + 250z^2 + 98z + 85$,

3.5. Selección de campos para ECC: comparativa y recomendaciones

Como se ha mencionado anteriormente, la elección del campo finito sobre el que definimos una curva elíptica tiene un impacto decisivo en el rendimiento, la seguridad y la facilidad de implementación del sistema criptográfico. A continuación, resumimos los principales criterios y trade-offs:

- **Campos primos $\mathbb{F}_p$:**
  - Muy usados en estándares (p.ej. NIST P-256, SECP256k1) [32].
  - Operaciones de multiplicación y reducción pueden aprovechar multiplicadores de enteros en hardware [33].
  - Más sencillos de proteger frente a canales laterales (menos ramificaciones en tiempo de reducción) [34].
- **Campos binarios $\mathbb{F}_{2^m}$:**
  - Ventajosos en hardware ligero (solo desplazamientos y XOR) [35].
  - Menos populares en software general-propósito porque requieren multiplicación bit-a-bit [33].
  - Históricamente usados en tarjetas inteligentes y estándares ISO/IEC [35].
- **Campos de extensión óptima (OEF):**
  - Diseñados para casar la longitud de palabra del procesador con la aritmética de $\mathbb{F}_p$.
  - Permiten reducir costes de acarreo y de reducción polinómica.
  - Para aplicaciones de alto rendimiento, se emplean de Tipo I o II cuyas operaciones en $\mathbb{F}_p$ caben en una palabra de máquina y el polinomio de reducción es un binomio simple ($z^m - \omega$) [36, 37]. Ejemplos concretos se recogen en la Tabla 3.1.

Criterios de selección

Al diseñar un sistema ECC, se ha de considerar lo siguiente:

1. *Rendimiento*: mediante tiempos de multiplicación e inversión. En CPU modernas, usar primos especiales (Mersenne, pseudo-Mersenne) acelera la reducción[38].
2. *Resistencia a canales laterales*: operaciones condicionales mínimas, tiempo de ejecución constante y operaciones atómicas [34].
3. *Compatibilidad y estandarización*: preferir curvas en $\mathbb{F}_p$ ampliamente auditadas (p.ej. curvas NIST, Brainpool, Curve25519) [39].

4. *Constrain hardware*: en dispositivos con ALU reducida puede convenir $\mathbb{F}_{2^m}$ o OEF para evitar multiplicadores de enteros avanzados [36].

Ejemplo de comparativa de costes

Campo	Multiplicación	Inversión	Uso típico
$\mathbb{F}_p$ (Mersenne)	≈ 1 μs	≈ 3 μs	TLS, SSH, Bitcoin
$\mathbb{F}_{2^m}$ (binario)	≈ 2 μs	≈ 5 μs	Smart cards, RFID
OEF ($p = 2^{32} - 5$)	≈ 0.8 μs	≈ 2.5 μs	Hardware especializado

Cuadro 3.2: Comparativa aproximada de costes en una CPU de 3 GHz (tamaños de campo 256 bits).

Los tiempos de la Tabla 3.2 se basan en benchmarks de OpenSSL y estudios de referencia en la literatura [40, 33]. Veremos y comprobaremos todo esto en la práctica, donde analizaremos la eficiencia en tiempo de cómputo de los distintos campos, junto con parámetros específicos recomendados por el NIST.

Veremos y comprobaremos todo esto en la práctica, donde analizaremos la eficiencia en tiempo de cómputo de los distintos campos, junto con parámetros específicos recomendados por el NIST.

Capítulo 4

Teoría de las curvas elípticas

4.1. Introducción y motivación

4.1.1. Historia y aplicaciones

4.1.2. Ventajas frente a otros grupos

4.2. Definición de una curva elíptica

4.2.1. Forma de Weierstrass general

4.2.2. Transformaciones y cambios de coordenadas

4.3. Estructura de grupo

4.3.1. Punto en el infinito

4.3.2. Ley de suma de puntos (geometría proyectiva)

4.3.3. Propiedades algebraicas

4.4. Curvas sobre cuerpos finitos

4.4.1. Curvas sobre $\mathbb{F}_p$

4.4.2. Curvas binarias sobre $\mathbb{F}_{2^m}$

4.5. Coordenadas y representación

4.5.1. Coordenadas afines

4.5.2. Coordenadas proyectivas y Jacobianas

4.5.3. Coordenadas "mixed" y optimizaciones

4.6. Selección de parámetros

4.6.1. Tamaños de curva y seguridad

4.6.2. Cofactores y subgrupos

4.6.3. Curvas estandarizadas (secp256k1, Curve25519, ...)

Capítulo 5

Criptografía con curvas elípticas (ECC)

5.1. Fundamentos de ECC

5.1.1. Ventajas frente a RSA y DH

5.1.2. Comparativa de tamaños de clave

5.2. Intercambio de claves: ECDH

5.2.1. Protocolo básico

5.2.2. Variantes y extensiones (X25519, etc.)

5.3. Firmas digitales: ECDSA

5.3.1. Generación y verificación de firma

5.3.2. Problemas prácticos (k reutilizado, mal RNG)

5.4. Otras construcciones criptográficas

5.4.1. ECIES: cifrado híbrido

5.4.2. EdDSA: firmas deterministas sobre edwards

5.5. Aspectos de implementación

5.5.1. Resistencia a canal lateral

5.5.2. Bibliotecas y hardware acelerado

5.6. Estándares y recomendaciones

5.6.1. NIST, SECG, RFCs relevantes

5.6.2. Buenas prácticas de despliegue

Bibliografía

[1] Shor's algorithm: A quantum threat to modern cryptography. https://postquantum.com/post-quantum/shors-algorithm-a-quantum-threat/. Accessed: May 2025.

[2] The quantum computing threat. https://docs.paloaltonetworks.com/network-security/quantum-security/administration/quantum-security-concepts/the-quantum-computing-threat. Accessed: May 2025.

[3] Ronald Rivest. Rfc 1321: Md5 message-digest algorithm. IETF RFC 1321, 1992. URL https://www.rfc-editor.org/rfc/rfc1321.html.

[4] Xiaoyun Wang, Dengguo Feng, Xuejia Lai, and Hongbo Yu. Collisions for hash functions md4, md5, haval-128 and ripemd. Cryptology ePrint Archive, Paper 2004/199, 2004. URL https://eprint.iacr.org/2004/199.

[5] Marc Stevens, Arjen K. Lenstra, and Benne de Weger. Chosen-prefix collisions for md5 and the creation of a rogue certification authority certificate. In *Advances in Cryptology – EUROCRYPT 2009*, volume 5479 of *Lecture Notes in Computer Science*, pages 55–72. Springer, 2009. URL https://link.springer.com/chapter/10.1007/978-3-642-01001-9_4.

[6] National Institute of Standards and Technology (NIST). SP 800-131A Rev. 2: Transitions: Recommendation for transitioning the use of cryptographic algorithms and key lengths. NIST, 2019. URL https://csrc.nist.gov/publications/detail/sp/800-131a/rev-2/final.

[7] National Institute of Standards and Technology (NIST). FIPS PUB 180-1: Secure hash standard. NIST, 1995. URL https://csrc.nist.gov/publications/detail/fips/180/1/final.

[8] Xiaoyun Wang, Yiqun Yin, and Hongbo Yu. Finding collisions in the full sha-1. In *Advances in Cryptology – CRYPTO 2005*, pages 17–36, 2005. URL https://www.iacr.org/archive/crypto2005/36210017/36210017.pdf.

[9] Google Security Blog and CWI Amsterdam. Announcing the first sha-1 collision. Google Online Security Blog, 2017. URL https://security.googleblog.com/2017/02/announcing-first-sha1-collision.html.

[10] The Chromium Project. A further update on sha-1 certificates in chrome. Chromium Security, 2016. URL https://www.chromium.org/Home/chromium-security/education/tls/sha-1/.

[11] Mozilla Security Blog. The end of sha-1 on the public web. Mozilla Security Blog, 2017. URL https://blog.mozilla.org/security/2017/02/23/the-end-of-sha-1-on-the-public-web/.

[12] Microsoft. Microsoft security advisory: Sha-1 deprecation for ssl/tls certificates. Microsoft Support, 2017. URL https://support.microsoft.com/en-us/topic/microsoft-security-advisory-sha-1-deprecation-for-ssl-tls-certificates-may-

[13] TechTarget. Expert advice: Encryption 101 – triple des explained. https://www.techtarget.com/searchsecurity/tip/Expert-advice-Encryption-101-Triple-DES-explained. Accessed: 6 May 2025.

[14] National Institute of Standards and Technology (NIST). FIPS 197: Advanced encryption standard (aes). https://csrc.nist.gov/pubs/fips/197/final, 2001. Accessed: 6 May 2025.

[15] Binary Terms. Block cipher. https://binaryterms.com/block-cipher.html. URL https://binaryterms.com/block-cipher.html.

[16] Bruce Schneier. Twofish. https://www.schneier.com/academic/twofish/. Accessed: 6 May 2025.

[17] Crypto Museum. The vernam cipher. https://www.cryptomuseum.com/crypto/vernam.htm, 2012. URL https://www.cryptomuseum.com/crypto/vernam.htm. © Crypto Museum. Creado: Sábado 11 de agosto de 2012. Última modificación: Sábado, 21 de diciembre de 2024 - 21:40 CET.

[18] Joachim Strombergson and Simon Josefsson. Test vectors for the stream cipher rc4. Informational RFC 6229, Internet Engineering Task Force (IETF), May 2011. URL https://www.rfc-editor.org/rfc/rfc6229.html.

[19] GeeksforGeeks contributors. What is rc4 encryption? https://www.geeksforgeeks.org/what-is-rc4-encryption/. Accessed: 6 May 2025.

[20] Thomas Stockinger. Gsm network and its privacy – the a5 stream cipher. Technical report, nop.at, November 2005. URL https://www.nop.at/gsm_a5/GSM_A5.pdf.

[21] Oliver Damsgaard Jensen and Kristoffer Alvern Andersen. A5 encryption in gsm. Technical report, University of California, Santa Barbara, June 2017. URL https://sites.cs.ucsb.edu/~koclab/teaching/cren/project/2017/jensen%2Bandersen.pdf.

[22] W. Diffie and M. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, November 1976. ISSN 1557-9654. doi: 10.1109/TIT.1976.1055638. URL https://ieeexplore.ieee.org/document/1055638/.

[23] NIST. Security strength of rsa in relation with the modulus size. Crypto StackExchange, respuesta destacada, 2012. URL https://crypto.stackexchange.com/questions/8687/security-strength-of-rsa-in-relation-with-the-modulus-size.

[24] Fabien A. P. Petitcolas and Smals. Kerckhoffs principle. In Henk C. A. van Tilborg and Sushil Jajodia, editors, *Encyclopedia of Cryptography and Security*. Springer, 2nd edition, January 2011. doi: 10.1007/978-1-4419-5906-5. URL https://www.researchgate.net/publication/264273789_Kerckhoffs_Principle.

[25] Different types of cryptography attacks. https://www.infosectrain.com/blog/different-types-of-cryptography-attacks/. Accessed: May 2025.

[26] Active and passive attacks in information security. https://www.geeksforgeeks.org/active-and-passive-attacks-in-information-security/. Accessed: May 2025.

[27] What are cryptographic attacks? https://www.goallsecure.com/blog/cryptographic-attacks-complete-guide/. Accessed: May 2025.

[28] Cryptography attacks: 6 types & prevention. https://www.packetlabs.net/posts/cryptography-attacks/, . Accessed: May 2025.

[29] Types of cryptographic attacks to know. https://fiveable.me/lists/types-of-cryptographic-attacks. Accessed: May 2025.

[30] Cryptanalysis and types of attacks. https://www.geeksforgeeks.org/cryptanalysis-and-types-of-attacks/, . Accessed: May 2025.

[31] Modern cryptographic attacks: A guide for the perplexed. https://research.checkpoint.com/2024/modern-cryptographic-attacks-a-guide-for-the-perplexed/. Accessed: May 2025.

[32] NIST. FIPS 186-4: Digital signature standard (dss). Technical report, National Institute of Standards and Technology, 2013. URL https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-4.pdf.

[33] Darrel Hankerson, Scott Vanstone, and Alfred Menezes. Guide to elliptic curve cryptography. Springer, 2004. ISBN 978-0-387-95276-8. doi: 10.1007/11745853_14.

[34] Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In *Advances in Cryptology (CRYPTO)*, page 388–397, 1999.

[35] SECG. SEC 2: Recommended elliptic curve domain parameters. https://www.secg.org/sec2-v2.pdf. Version 2.0, May 2000.

[36] Hermann Militzer and Christof Paar. Efficient implementation of oef arithmetic for ecc. *Cryptographic Hardware and Embedded Systems*, page 123–134, 2004.

[37] Paulo Barreto and Reinaldo Dahab. Efficient multiplication on f_{2^n} and f_p via oef. In *Selected Areas in Cryptography (SAC)*, page 1–13, 2003.

[38] Daniel J. Bernstein. Curve25519: New diffie-hellman speed records. In Moti Yung, Yevgeniy Dodis, Aggelos Kiayias, and Tal Malkin, editors, *Public Key Cryptography - PKC 2006*, pages 207–228, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg. ISBN 978-3-540-33852-9.

[39] IETF CFRG. Elliptic curve cryptography (ecc) and curve generation. https://datatracker.ietf.org/wg/cfrg/documents/. IETF Internet-Draft.

[40] OpenSSL Project. Openssl benchmarking, 2021. URL https://www.openssl.org/docs/manmaster/man1/openssl-speed.html.

Otro material

- Diversas consultas puntuales al sitio *StackOverFlow*.
- Material docente de las asignaturas Álgebra Lineal y Estructuras Matemáticas, Metodología de la Programación, Aprendizaje Automático y Fundamentos de Redes impartidas en el Grado de Ingeniería Informática en la Universidad de Granada.