Ciberseguridad en Servicios y Aplicaciones.

Sergio Escalante Presa

0. Introducción

Estos son los apuntes de la asignatura *Ciberseguridad en Servicios y Aplicaciones*, impartida por la Universidad de Málaga.

En esta página se proporcionará un breve resumen del contenido de cada tema.

---

1. Fundamentos de Seguridad.

En este tema se repasan los **conceptos básicos** que rodean la ciberseguridad. El índice de este tema:

- Introducción.
  - Seguridad de la información: Conceptos.
  - Ciclo de vida de la Seguridad.
  - Modelo básico de **escenario de Seguridad**.
    - Ataques básicos.
- Servicios y mecanismos de Seguridad.
  - Las **cinco categorías fundamentales**.
    - Confidencialidad.
    - Integridad.
    - Autenticidad
    - Autenticación (Control de Acceso).
    - No repudio.
  - Jerarquía (Servicios, Protocolos, Mecanismos, Técnicas, etc.)

---

2. Técnicas Criptográficas Básicas y servicios de seguridad asociados.

En este tema se estudian las diferentes técnicas criptográficas básicas (**Clásicas, Simétricas y Asimétricas**), además de otros servicios asociados, como las **funciones HASH** o las funciones MAC. El índice es el siguiente:

- Criptografía clásica.
  - Cifrado por sustitución y transposición. Ejemplos.
  - Cifrado Producto.
  - Cifrado Vernman (**one-time-pad**).
- Algoritmos simétricos (clave **privada**).
  - Fundamentos.
  - Algoritmo DES.
  - Algoritmo triple-DES.
  - Otros algoritmos simétricos.
  - Modos de operación para algoritmos simétricos.
  - Ventajas y desventajas de los algoritmos simétricos.
- Algoritmos asimétricos (clave **pública**)
  - Cifrado / Descifrado.
  - Firma digital.
  - Intercambio de claves.
  - Algoritmo de Diffie-Hellman.
  - Algoritmo RSA.
- Otras **primitivas criptográficas**.
  - Funciones HASH.
  - Códigos de autenticación de mensajes.

---

3. Esquemas, Protocolos y Mecanismos de Soporte.

1. **Primera Parte: Gestión de claves y protocolos de soporte.**

- El problema de la distribución de claves simétricas.
- Mecanismos contra ataques de repetición.
  - Timestamps.
  - Nonces (núnicos).
  - Combinación de ambas estrategias.
- Modelos de KDC: Simple, PULL y PUSH.
  - Modelo simple — "La Rana de la Boca Grande".
  - Modelo PULL.
  - Modelo PUSH.
- Protocolo Needham-Schroeder.
  - Amended Needham-Schroeder.
- Protocolo Otway-Rees.
- Protocolo Kerberos — Single Sign-On (SSO).
- Protocolos avanzados (combinaciones PULL + PUSH).
  - Yahalom.
  - Neuman-Stubblebine.
  - Kao-Chow.

2. **Segunda Parte: PKI y Certificados.**

- El problema de la autenticidad de la clave pública.
- Estructura del certificado X.509.
  - Uso de claves: X.509v3 Key Usage.
- Infraestructura de Clave Pública (PKI) y cadenas de confianza.
- Revocación de certificados — CRL y OCSP.

3. **Tercera Parte: Tarjetas inteligentes y DNI electrónico.**

- Tarjetas inteligentes (smartcards) — clasificación
- El DNI electrónico (DNI-e)
  - Hardware — chip ST19WL34 y acelerador criptográfico
  - Zonas de memoria — pública, privada y de seguridad
  - Generación y custodia de claves privadas
  - Revocación y validación — OCSP en el DNI-e

4. **Cuarta Parte: Mecanismos de autenticación**

- Mecanismos de autenticación de usuarios
  - Clasificación de factores de autenticación
  - Factor CONOZCO — Autenticación por contraseña
    - Salt: almacenamiento seguro de contraseñas
    - `bcrypt` — función hash adaptativa
    - Principios del salt (resumen)
  - Factor TENGO — Tokens físicos
    - Ejemplo: DNI electrónico (DNI-e)
  - Factor SOY — Autenticación biométrica
  - Comparativa de factores de autenticación
  - Autenticación de doble factor (2FA)
  - Autenticación basada en códigos QR
  - Single Sign-On (SSO)
- Mecanismos de control de acceso
  - Fundamentos del control de acceso
  - Elementos básicos del control de acceso
  - Categorías de mecanismos de control de acceso
  - DAC — Control de acceso discrecional
    - Matriz de acceso
    - Access Control List (ACL) — descomposición por columnas
    - Ticket de capacidades (Capability List) — descomposición por filas
  - MAC — Control de acceso obligatorio
  - RBAC — Control de acceso basado en roles
    - Entidades del modelo RBAC
    - Restricciones en RBAC
    - Extensiones del modelo NIST
  - ABAC — Control de acceso basado en atributos
  - Modelos adicionales de control de acceso

        - CapBAC — Control de acceso basado en capacidades
        - Risk-Based Access Control
        - OrBAC — Control de acceso basado en la organización
    - Comparativa global de modelos de control de acceso

5. **Quinta parte: Protocolos criptográficos avanzados**

- Introducción a los protocolos criptográficos
- Protocolo de división de secretos
    - Concepto
    - Ejemplo con 2 personas (XOR)
    - Extensión a 4 personas
- Protocolo de compartición de secretos (Secret Sharing)
    - Fundamento matemático: interpolación polinómica
    - Ejemplo — esquema umbral (3, 5)
    - Comparativa: división vs. compartición
- Protocolos de bit-commitment
    - Solución con criptografía simétrica
    - Solución con funciones hash
- Protocolos de lanzamiento de moneda
    - Concepto
    - Solución con criptografía asimétrica
- Protocolo de póker mental

---

4. Seguridad y Privacidad en Aplicaciones Telemáticas.

1. **Primera parte: Herramientas de seguridad y comunicación segura**

- Red Team y Blue Team
    - Red Team
    - Blue Team
- Herramientas Red Team
    - Sistemas operativos especializados
        - Kali Linux
        - Parrot Security OS
    - Principales herramientas en Linux
    - Descripción detallada de herramientas clave
        - Hydra — fuerza bruta
        - Nmap y Zenmap — rastreo y *fingerprinting*
        - Wireshark — captura de tráfico (*sniffing*)
        - Scapy — manipulación de paquetes
        - Ettercap — ataques *Man-in-the-Middle*
        - MITRE ATT&CK

- Seguridad en email: PGP y S/MIME
  - PGP (*Pretty Good Privacy*)
    - Servicios de PGP
    - Esquema de transmisión PGP
    - Gestión de claves: anillos de claves
    - OpenPGP y aplicaciones
  - S/MIME (*Secure/Multipurpose Internet Mail Extension*)
    - Modelo de confianza S/MIME
    - Algoritmos criptográficos S/MIME
- Conexión remota segura
  - Telnet y FTP — protocolos inseguros
  - SSH: *Secure Shell* v2
    - Arquitectura de capas de SSH
    - Protección contra ataques
    - Clientes SSH
  - SFTP y FTPS — transferencia segura de ficheros
    - SFTP (*SSH File Transfer Protocol*)
    - FTPS (*FTP Secure*)
    - Comparativa SSH / SFTP / FTPS vs Telnet / FTP
- Herramientas de cifrado
  - Herramientas *open-source*
    - TrueCrypt
    - DiskCryptor
    - VeraCrypt
    - Herramientas de esteganografía
  - Herramientas propietarias
    - BitLocker

2. **Segunda parte: Seguridad en pagos electrónicos**

- Conceptos generales
  - Introducción
    - Participantes en un pago electrónico
  - Clasificación de los sistemas de pago
    - Por el momento en que el vendedor contacta con el banco
    - Por el momento en que se retira el dinero de la cuenta del comprador
    - Por la cantidad implicada en la transacción
  - Problemas de seguridad y soluciones
  - Breve historia: el papel de SS
- Protocolo SET (*Secure Electronic Transaction*)
  - Servicios de seguridad que proporciona
  - Participantes en SET
  - Pasos del protocolo SET
  - Firma dual (innovación técnica de SET)
  - Ventajas y desventajas de SET
- Protocolo CyberCash

  - Pasos del protocolo CyberCash
  - Problemas de CyberCash
- Protocolo iKP (*i-Key Protocol*)
  - Elementos intercambiados en iKP
  - Protocolo 1KP
  - Protocolo 2KP
- Micropagos y protocolo Millicent
  - El problema de los micropagos
  - Protocolo Millicent

3. **Tercera parte: Privacidad de los usuarios en aplicaciones**

- Conceptos generales de privacidad
  - Definición de privacidad
  - Privacidad confidencialidad
  - Formas de violar la privacidad
    - Rastreo de actividad en la red
    - Análisis de tráfico
  - Enfoques para proteger la privacidad
  - Propiedades de la privacidad
  - Tipos de anonimato
  - Técnicas para conseguir privacidad
- Privacidad basada en esquemas avanzados de firma digital
  - Firma ciega (*Blind Signature*)
  - Firma de grupo
  - Firma de anillo
- Privacidad basada en protocolos criptográficos y de enrutado
  - Proxy
  - *Mix Networks* (mezcladores)
  - *Onion Routing* y Tor
    - *Onion Routing*
    - Tor — segunda generación de *Onion Routing*
  - Crowds y Hordes
    - Crowds
    - Hordes
  - Comparativa de protocolos de privacidad

---

5. Seguridad en redes TCP/IP (SSL/TLS)

- Introducción y motivación
  - Amenazas y consecuencias
  - Dos enfoques iniciales: SHTTP frente a SSL
  - Evolución histórica de SSL/TLS
- Servicios de seguridad y arquitectura de SSL

  - Servicios proporcionados
  - Independencia del protocolo de aplicación
  - Criptografía híbrida
  - Sesión y conexión SSL
  - Arquitectura en dos subcapas
- SSL Record Protocol
  - Formato del SSL record
  - Procesamiento del fragmento
- SSL Handshake Protocol
  - Formato de los mensajes de handshake
  - Las cuatro fases del handshake
  - Tabla de mensajes del handshake
  - Ejemplo: formato de `ClientHello` y `ClientKeyExchange`
- Establecimiento de la clave de sesión
  - Parámetros que intervienen en la derivación
  - Intercambio de claves con RSA
  - Intercambio de claves con Diffie-Hellman
  - DHE — Diffie-Hellman efímero y *Perfect Forward Secrecy*
  - DHE-RSA: combinando autenticación e intercambio efímero
- Subprotocolos auxiliares
  - SSL Change Cipher Spec Protocol
  - SSL Alert Protocol
  - SSL Record Protocol
- TLS 1.2 y TLS 1.3
  - TLS 1.2
  - TLS 1.3
  - Comparativa TLS 1.2 vs TLS 1.3
- DTLS — *Datagram Transport Layer Security*
  - Motivación
  - Adaptaciones respecto a TLS
- Síntesis y modelo mental

1. Fundamentos de Seguridad.

1. Introducción.

1.1. Seguridad de la información: Conceptos.

La **seguridad de la información** es un campo multidisciplinar que abarca las tecnologías, los procesos y las prácticas necesarias para proteger la información frente a un amplio espectro de amenazas. Existen varias definiciones aceptadas en la industria y la academia:

Definiciones formales.

- **ISO/IEC 17799:** "La protección de la información frente a un amplio rango de amenazas, con el fin de garantizar la continuidad del negocio, minimizar el riesgo empresarial y maximizar el retorno de la inversión y las oportunidades de negocio."
- **Microsoft Security Glossary:** "La protección de los activos de información mediante el uso de tecnología, procesos y formación."
- **Software Engineering Institute (Carnegie Mellon):** "La capacidad de un sistema para gestionar, proteger y distribuir información sensible."

Aunque las definiciones varían en el detalle, todas comparten el mismo núcleo: **proteger la información** de accesos, modificaciones o interrupciones no autorizadas.

Un concepto clave relacionado es el de **vulnerabilidad**: un error en cualquiera de las fases del ciclo de vida del software (análisis, diseño, desarrollo o implementación) puede producir, a posteriori, un fallo de seguridad. Este fallo es lo que se denomina vulnerabilidad. Como consecuencia de su explotación, se viola la **política de seguridad** del sistema, dejándolo en peligro.

i Escala del problema.

En una red como Internet, con sus dimensiones actuales (miles de millones de dispositivos y usuarios), el efecto de una vulnerabilidad explotada tiene un alcance potencialmente **exponencial**: un único fallo puede comprometer infraestructuras globales o afectar a millones de usuarios en cuestión de horas.

1.2. Ciclo de vida de la Seguridad.

La seguridad no es un estado puntual, sino un **proceso continuo**. La norma ISO 7498-2, que define la arquitectura de seguridad para sistemas de interconexión abiertos, establece que la seguridad debe aplicarse de forma transversal a todo el ciclo de vida de la información y los sistemas.

El ciclo de vida de la seguridad (tal como lo describe el *Handbook of Applied Cryptography*) contempla las siguientes fases, que se retroalimentan entre sí:

1. **Análisis de amenazas y riesgos:** Identificar qué activos existen, qué amenazas los afectan y con qué probabilidad e impacto.
2. **Diseño de políticas y objetivos de seguridad:** Definir qué se desea proteger y bajo qué condiciones, plasmado en una **política de seguridad** formal.
3. **Implementación de mecanismos de seguridad:** Seleccionar y desplegar los controles técnicos y organizativos que satisfacen la política diseñada.
4. **Monitorización y auditoría:** Verificar de forma continua que los mecanismos funcionan correctamente y detectar desviaciones.
5. **Revisión y mejora continua:** Actualizar el sistema de seguridad en función de nuevas amenazas, vulnerabilidades descubiertas o cambios en el entorno.

i Analogía del ciclo de vida.

Piensa en la seguridad como el mantenimiento de un edificio: no basta con construirlo bien (implementación); también hay que inspeccionar periódicamente las instalaciones (monitorización), reparar los desperfectos que aparezcan (mejora continua) y revisar el plano original si el uso del edificio cambia (revisión de políticas).

1.3. Modelo básico de escenario de Seguridad.

Para razonar sobre seguridad de forma sistemática, se recurre a un **modelo de escenario básico** que formaliza los actores y elementos presentes en cualquier comunicación susceptible de ser atacada.

Los elementos fundamentales del modelo son:

- **Emisor (Alice):** La entidad que desea enviar información a otra parte.
- **Receptor (Bob):** La entidad que desea recibir la información del emisor de forma segura.
- **Canal de comunicación:** El medio por el que viaja la información. En Internet, este canal es públicamente accesible y, por tanto, potencialmente hostil.
- **Atacante (Mallory o Eve):** Una entidad adversaria que puede interactuar con el canal. Convencionalmente se distingue entre:
  - **Eve** (*eavesdropper*, fisgón): solo escucha pasivamente el canal.
  - **Mallory** (*malicious*, malicioso): puede interceptar, modificar, eliminar o reinyectar mensajes de forma activa.
- **Tercera parte de confianza (Trent):** Una entidad en quien ambos, Alice y Bob, confían. Puede desempeñar funciones de árbitro, distribuidor de claves, o notario, según el protocolo.

> Convención de nombres.
>
> El uso de nombres propios (Alice, Bob, Mallory, Trent, etc.) es una convención universalmente extendida en la literatura de criptografía y seguridad para clarificar los roles de cada actor. Es mucho más intuitivo que denominarlos "entidad A", "entidad B" y "entidad adversaria".

Un escenario realista añade complejidad: Alice y Bob pueden ser personas, servidores, aplicaciones o dispositivos. El canal puede ser una red LAN, Internet, una comunicación inalámbrica o incluso un canal físico. La presencia de Mallory no significa que esté siempre activo; puede que solo monitorice pasivamente y actúe más tarde.

Este modelo es la base sobre la que se construyen los protocolos de seguridad: cada protocolo define **qué garantías** ofrece a Alice y Bob frente a las posibles acciones de Mallory.

1.4. Ataques básicos.

Dentro del modelo de escenario, los ataques se clasifican según la naturaleza de la intervención del adversario sobre el canal de comunicación:

Ataques pasivos: El atacante únicamente observa el tráfico, sin alterarlo. Son difíciles de detectar porque no dejan rastro en el canal.

- **Escucha (*eavesdropping*):** Interceptar el contenido de los mensajes para obtener información confidencial.
- **Análisis de tráfico (*traffic analysis*):** Aunque el contenido esté cifrado, analizar patrones (quién habla con quién, cuándo, con qué frecuencia, qué volumen de datos) puede revelar información valiosa.

Ataques activos: El atacante manipula el canal de alguna forma. Son más difíciles de ejecutar, pero también más devastadores.

- **Suplantación de identidad (*masquerade* o *spoofing*):** El atacante se hace pasar por Alice o Bob.
- **Modificación de mensajes (*message tampering*):** El atacante altera el contenido de uno o varios mensajes en tránsito.
- **Repetición (*replay attack*):** El atacante captura un mensaje legítimo y lo reenvía más tarde, sin necesidad de conocer su contenido.
- **Denegación de Servicio (*Denial of Service, DoS*):** El atacante impide que Alice y Bob puedan comunicarse, por ejemplo inundando el canal o eliminando mensajes.
- **Hombre en el medio (*Man-in-the-Middle, MitM*):** El atacante se sitúa entre Alice y Bob, retransmitiendo (y posiblemente modificando) todos los mensajes. Cada parte cree estar comunicándose directamente con la otra.

Analogía del ataque MitM.

Imagina que Alice y Bob se escriben cartas físicas. Un cartero malicioso (Mallory) las intercepta, las abre, las lee y/o modifica, y luego las sella de nuevo y las entrega. Ni Alice ni Bob saben que sus cartas han pasado por manos de Mallory. Un ataque MitM en una red funciona de la misma manera, pero a la velocidad de la luz.

---

2. Servicios y mecanismos de Seguridad.

Un **servicio de seguridad** es un servicio de procesamiento o comunicación que proporciona un tipo específico de protección a los recursos del sistema. Las definiciones más aceptadas son:

Definiciones formales.

- **RFC 2828 (Internet Security Glossary):** "Un servicio de procesamiento o comunicación que proporciona un tipo específico de protección a los recursos del sistema."
- **ISO 7498-2 / ITU X.800:** "Un servicio, proporcionado por una

> capa de sistemas abiertos en comunicación, que garantiza una seguridad adecuada de los sistemas o de las transferencias de datos."

Los estándares ISO 7498-2 e ITU X.800 son los marcos de referencia internacionales que estructuran y definen los servicios y mecanismos de seguridad para sistemas de interconexión abiertos. Su estructura es la que se sigue en estas notas.

2.1. Las cinco categorías fundamentales.

Los estándares ISO 7498-2 e ITU X.800 dividen los servicios de seguridad en **cinco grandes categorías**, de las que se derivan catorce servicios específicos en total.

2.1.1. Confidencialidad.

Definición: Protección de los datos frente a su divulgación no autorizada. Garantiza que solo las entidades autorizadas puedan acceder al contenido de la información.

> **i** Analogía.
>
> La confidencialidad es el equivalente digital de un sobre sellado: solo el destinatario legítimo puede abrirlo y leer el contenido. El mecanismo que la implementa habitualmente es el **cifrado**.

Los servicios específicos bajo esta categoría son:

1. **Connection Confidentiality:** Protección de todos los datos de usuario a lo largo de una conexión establecida (orientada a conexión).
2. **Connectionless Confidentiality:** Protección de todos los datos de usuario en un único bloque de datos (sin conexión previa, p. ej., un datagrama UDP).
3. **Selective-Field Confidentiality:** Confidencialidad solo sobre **campos seleccionados** dentro del bloque de datos, dejando el resto en claro. Útil cuando es necesario enrutar o inspeccionar partes del mensaje.
4. **Traffic-Flow Confidentiality:** Protección de la información que podría inferirse de la observación de los flujos de tráfico (frecuencia, volumen, origen/destino). Se implementa mediante técnicas como el *traffic padding* (relleno de tráfico con datos ficticios).

⚠ Advertencia sobre el análisis de tráfico.

Incluso si el contenido de todos los mensajes está perfectamente cifrado, un adversario que observe que Alice y Bob intercambian grandes volúmenes de datos justo antes de una OPA empresarial puede inferir información valiosa. La Traffic-Flow Confidentiality está diseñada precisamente para mitigar este riesgo.

2.1.2. Integridad.

Definición: Garantía de que los datos recibidos son exactamente los que envió una entidad autorizada; es decir, que no han sufrido modificación, inserción, borrado ni repetición no autorizados.

ℹ Analogía.

La integridad es como un sello de lacre en un sobre con el escudo del remitente: si alguien lo ha abierto y vuelto a cerrar, el sello quedará roto o falsificado. En digital, el mecanismo equivalente son los **códigos de autenticación de mensaje (MAC)** o las **firmas digitales**.

Los servicios específicos son:

1. **Connection Integrity with Recovery:** Garantiza la integridad de todos los datos de usuario en una conexión y detecta cualquier modificación, inserción, borrado o repetición, **intentando además recuperar** el estado correcto.
2. **Connection Integrity without Recovery:** Igual al anterior, pero solo **detecta** la violación, sin intentar recuperarse.
3. **Selective-Field Connection Integrity:** Garantiza la integridad de campos seleccionados dentro de los datos de usuario de una conexión.
4. **Connectionless Integrity:** Garantiza la integridad de un único bloque de datos sin conexión. Puede incluir una forma limitada de detección de repetición.
5. **Selective-Field Connectionless Integrity:** Garantiza la integridad de campos seleccionados dentro de un único bloque de datos sin conexión.

2.1.3. Autenticidad.

La **autenticidad** se ocupa de corroborar el **origen** de la información: garantiza que los datos proceden de quien se afirma que proceden. También se conoce como **autenticación de origen de datos** (*data origin authentication*).

Distinción clave: Autenticación de entidad vs. Autenticación de origen.

Es importante no confundir los dos servicios de autenticación definidos en X.800:

- **Autenticación de entidad (*Peer Entity Authentication*):** Confirma la identidad de una entidad **durante el establecimiento de una conexión** (en tiempo real, de forma interactiva). Responde a la pregunta: *¿Estoy hablando con quien creo que estoy hablando ahora mismo?*
- **Autenticación de origen de datos (*Data-Origin Authentication*):** Confirma que **el bloque de datos recibido** proviene de la entidad declarada como origen, sin requerir una conexión en curso. Responde a la pregunta: *¿Quién creó este mensaje?* Se aplica típicamente en transferencias sin conexión (correo electrónico firmado, documentos digitales, etc.).

Peer Entity Authentication: Se utiliza junto con una conexión lógica para proporcionar confianza en la identidad de las entidades conectadas. Previene la suplantación de identidad y los ataques de repetición durante la sesión.

Data-Origin Authentication: En una transferencia sin conexión, proporciona garantía de que la fuente de los datos recibidos es la declarada. No proporciona protección contra la repetición o modificación de unidades de datos.

2.1.4. Autenticación (Control de Acceso).

El **Control de Acceso** previene el uso no autorizado de recursos. Define quién puede acceder a un recurso, bajo qué condiciones y qué operaciones puede realizar sobre él.

Definición (X.800).

"La prevención del uso no autorizado de un recurso (es decir, este servicio controla quién puede acceder a un recurso, bajo qué condiciones puede producirse el acceso, y qué se permite hacer a quienes acceden al recurso)."

El control de acceso se implementa mediante mecanismos como:

- **Listas de control de acceso (ACL):** Tablas que especifican qué entidades tienen permiso sobre qué recursos y con qué nivel.
- **Capacidades (*capabilities*):** Tokens que el sujeto presenta para demostrar su derecho de acceso a un objeto.
- **Atributos de seguridad y etiquetas:** Metadatos asociados a sujetos y objetos que el sistema compara para conceder o denegar el acceso (común en sistemas de seguridad multinivel como los militares).

Relación con la autenticación de entidad.

El control de acceso **depende directamente** de la autenticación de entidad: antes de decidir si alguien puede acceder a un recurso, el sistema necesita saber con certeza quién es esa entidad. Sin autenticación previa, el control de acceso carece de sentido. Por eso estos dos servicios aparecen frecuentemente juntos y su implementación está íntimamente ligada.

2.1.5. No repudio.

El **no repudio** proporciona protección frente a la negación, por parte de una de las entidades involucradas en una comunicación, de haber participado en toda o parte de dicha comunicación.

Es el servicio que responde a la pregunta: *¿Puedo demostrar ante un tercero (por ejemplo, un juez) que esta transacción ocurrió?*

Los dos servicios específicos son:

1. **Nonrepudiation, Origin:** Prueba que el mensaje fue enviado por la parte especificada. El receptor puede demostrar a un árbitro externo que el emisor envió el mensaje.
2. **Nonrepudiation, Destination:** Prueba que el mensaje fue recibido por la parte especificada. El emisor puede demostrar a un árbitro externo que el receptor recibió el mensaje.

Ejemplo cotidiano.

El no repudio es el equivalente digital de un **acuse de recibo notariado**: cuando recibes un burofax, la agencia postal certifica que lo recibiste en una fecha determinada. Si más tarde niegas haberlo recibido, ese certificado es la prueba en tu contra. En digital, el mecanismo que implementa el no repudio son las **firmas digitales**, complementadas opcionalmente con un **tercero de confianza (notario digital)**.

No repudio Autenticidad.

La autenticidad garantiza que los datos *parecen* provenir de un origen legítimo. El no repudio va más allá: garantiza que ese origen **no puede negarlo ante un tercero**. La firma digital es el único mecanismo criptográfico que proporciona ambas propiedades simultáneamente, siempre que la clave privada del firmante esté bajo su exclusivo control.

2.2. Jerarquía (Servicios, Protocolos, Mecanismos, Técnicas, etc.)

La arquitectura de seguridad se organiza en una **jerarquía de niveles de abstracción**, desde los objetivos de alto nivel (los servicios) hasta las herramientas matemáticas de bajo nivel (las técnicas criptográficas):

```
Servicios de seguridad
      Protocolos de seguridad
              Mecanismos de seguridad
                      Técnicas criptográficas
```

Cada nivel depende del inmediatamente inferior para su implementación. Los estándares ISO 7498-2 e ITU X.800 distinguen dos tipos de **mecanismos de seguridad**:

> Definición de mecanismo de seguridad (RFC 2828).
>
> "Un proceso (o un dispositivo que incorpora dicho proceso) que puede utilizarse en un sistema para implementar un servicio de seguridad proporcionado por o dentro del sistema."

Mecanismos específicos: Están directamente relacionados con un servicio de seguridad concreto.

Mecanismo	Descripción
Cifrado (*Encipherment*)	Transforma datos en un formato ininteligible para entidades no autorizadas.
Firma digital	Permite verificar origen e integridad de un mensaje, y proporciona no repudio.
Control de acceso	Mecanismos que implementan políticas de acceso (ACL, *capabilities*, etiquetas).
Integridad del dato	MACs, hashes, checksums; garantizan que los datos no han sido alterados.
Autenticación	Mecanismos de intercambio para verificar la identidad (contraseñas, certificados, etc.).
Padding de tráfico	Relleno de canales con tráfico ficticio para dificultar el análisis de tráfico.
Control de enrutamiento	Selección de rutas seguras o alternativas en la red.
Tercera parte de confianza (*Trusted Third Party*)	Una entidad de confianza mutua que actúa como árbitro o notario (p. ej., una CA).

Mecanismos ubícuos: No están ligados a ningún servicio específico, sino que se aplican de forma transversal a todos los sistemas de seguridad.

Mecanismo	Descripción
Controles de seguridad	Políticas, procedimientos y salvaguardas organizativas.
Etiquetas de seguridad	Marcas asociadas a recursos que indican su nivel de clasificación.
Certificación, validación y simulación	Pruebas formales de que un sistema cumple sus especificaciones de seguridad.
Detección y prevención de eventos	IDS/IPS y sistemas de monitorización continua.
Auditoría y *accountability*	Registro de eventos para su análisis posterior y para responsabilizar a los actores.
Recuperación de la seguridad	Procedimientos para restaurar el sistema a un estado seguro tras un incidente.
Análisis forense	Técnicas para investigar incidentes y recolectar evidencias digitales.
Gestión de la confianza y reputación	Sistemas para evaluar y gestionar la confianza entre entidades.

Relación entre Servicios y Mecanismos (ISO 7498-2 / X.800):

Los mecanismos no tienen una relación uno-a-uno con los servicios: un mismo mecanismo puede implementar varios servicios, y un servicio puede requerir varios mecanismos. La siguiente tabla resume las relaciones principales:

Servicio / Mecanismo	Cifrado	Firma digital	Control acceso	Integridad dato	Autent. intercambio	Padding tráfico	Control enrutamiento	Notarización
Autenticación de entidad	Y	Y	·	·	Y	·	·	·

Servicio / Mecanismo	Cifrado	Firma digital	Control acceso	Integridad dato	Aut-ent. intercambio	Padding tráfico	Control enrutamiento	Notarización
Autenticación de origen	Y	Y	·	·	·	·	·	·
Servicio de control de acceso	·	·	Y	·	·	·	·	·
Confidencialidad en conexión	Y	·	·	·	·	·	Y	·
Confidencialidad sin conexión	Y	·	·	·	·	·	Y	·

Servicio / Mecanismo	Cifrado	Firma digital	Control acceso	Integridad dato	Aut-ent. inter-cambio	Padding tráfico	Control en-rutamiento	Nota-rización
Confidencialidad de campo sel.	Y	·	·	·	·	·	·	·
Confidencialidad de flujo	Y	·	·	·	·	Y	Y	·
Integridad en conexión con rec.	Y	·	·	Y	·	·	·	·
Integridad en conexión sin rec.	Y	·	·	Y	·	·	·	·

Servicio / Mecanismo	Cifrado	Firma digital	Control acceso	Integridad dato	Autent. intercambio	Padding tráfico	Control enrutamiento	Notarización
Integridad campo sel. en conex.	Y	·	·	Y	·	·	·	·
Integridad sin conexión	Y	Y	·	Y	·	·	·	·
Integridad campo sel. sin conex.	Y	Y	·	Y	·	·	·	·
No repudio de origen	·	Y	·	Y	·	·	·	Y
No repudio de destino	·	Y	·	Y	·	·	·	Y

(Y = el mecanismo es relevante para el servicio; · = no aplicable de forma

directa)

Implementación por capas OSI:

Un aspecto clave de los estándares X.800 e ISO 7498-2 es que los servicios de seguridad **pueden implementarse en distintas capas del modelo de referencia OSI**, y no todos los servicios están disponibles en todas las capas:

Servicio	Capa 1	Capa 2	Capa 3	Capa 4	Capas 5/6	Capa 7
Autenticación de entidad			Y	Y		Y
Autenticación de origen			Y	Y		Y
Control de acceso			Y	Y		Y
Confidencialidad en conexión	Y	Y	Y	Y		Y
Confidencialidad sin conexión		Y	Y	Y		Y
Confidencialidad de campo seleccionado						Y
Confidencialidad de flujo de tráfico	Y		Y			Y
Integridad en conexión con recuperación				Y		Y
Integridad en conexión sin recuperación			Y	Y		Y
Integridad de campo selec. en conexión						Y
Integridad sin conexión			Y	Y		Y
Integridad de campo selec. sin conexión						Y
No repudio de origen						Y
No repudio de destino						Y

> **i** Lectura de la tabla.
>
> La tabla muestra que el **no repudio** solo puede proporcionarse en la **capa de aplicación (capa 7)**, lo cual tiene sentido: el no repudio requiere contexto semántico (quién firmó qué documento con qué significado), algo que las capas inferiores no pueden proporcionar. Por el contrario, la **confidencialidad en conexión** puede implementarse desde la capa 1 (cifrado del canal físico) hasta la capa 7 (cifrado de la aplicación), ofreciendo una gran flexibilidad de diseño.

Resumen de la jerarquía:

En definitiva, la arquitectura de seguridad puede resumirse en la siguiente pila conceptual, donde cada nivel se apoya en el siguiente:

```
        SERVICIOS DE SEGURIDAD                ← ¿Qué protejo?
  (Confidencialidad, Integridad, Auth...)

        PROTOCOLOS DE SEGURIDAD               ← ¿Cómo lo orquesto?
    (TLS, IPsec, Kerberos, PGP, SSH...)
```

```
       MECANISMOS DE SEGURIDAD              ← ¿Qué herramientas uso?
(Cifrado, MACs, firmas, ACL, nonces...)

      TÉCNICAS CRIPTOGRÁFICAS               ← ¿Qué matemáticas subyacen?
(AES, RSA, SHA-256, curvas elípticas...)
```

Esta jerarquía es fundamental para entender cómo se diseña y evalúa la seguridad de un sistema: un protocolo como TLS implementa servicios de confidencialidad, integridad y autenticación, haciendo uso de mecanismos como el cifrado simétrico y las firmas digitales, que a su vez se construyen sobre técnicas matemáticas como AES o RSA.

2. Técnicas Criptográficas Básicas.

0. Introducción.

Como se vio en el Tema 1, un **algoritmo de cifrado** es uno de los mecanismos **fundamentales** para implementar servicios de seguridad.

> Definición.
>
> **Criptografía:** Ciencia que estudia cómo mantener los **mensajes seguros**, usando entre otras cosas los algoritmos de cifrado.

> Definición.
>
> **Criptoanálisis:** Ciencia que estudia cómo **romper** los mensajes cifrados.

> Definición.
>
> **Criptología:** Criptografía + Criptoanálisis

Un algoritmo de **cifrado** que transforma un mensaje en algo **ininteligible**. Busca garantizar la ***confidencialidad***.

A su vez, un algoritmo de **descifrado** hace lo inverso, transforma un texto cifrado en texto en **claro**.

> ! Notación.
>
> La función de cifrado se denota como $\boldsymbol{E}$, la de descifrado como $\boldsymbol{D}$; el mensaje en claro se representa con $\boldsymbol{M}$, mientras que el texto cifrado es C.

Además, se cumple lo siguiente:

$E(M) = C$

$D(C) = M$

$D[E(M)] = M$

---

1. Criptografía clásica.

Antiguamente, la criptografía se basaba en algoritmos basados en **caracteres**. Concretamente, estos algoritmos **sustituían** o **trasponían** caracteres.

1.1. Cifrado por sustitución y trasposición. Ejemplos.

Definición.

Cifrado por sustitución: Cada carácter del mensaje se sustituye por otro carácter en el texto cifrado.

Definición.

Cifrado por trasposición: Se *permutan* las posiciones que ocupan los símbolos del mensaje.

a) Cifrado por sustitución Caesar.

Una única transformación. Se desplaza el alfabeto para obtener el alfabeto cifrado. Por ejemplo, si desplazamos el alfabeto 3 posiciones, la A original será una D, la B una E, y así.

Ejemplo: Si movemos el alfabeto 3 posiciones, la palabra `HOLA` pasará a ser `KRÑD`.

Problema: Le damos **ventaja al criptoanalista**. La frecuencia de aparición de las letras no es la misma para todas.

b) Cifrado por sustitución homofónico.

Asignamos a un símbolo del alfabeto del mensaje, varios símbolos del alfabeto cifrado (proporcional a la frecuencia que dicho símbolo aparezca en el mensaje). Al haber una correspondencia **uno a muchos**, solventamos el problema de la frecuencia de las letras.

Problema: Varios criptogramas pueden ser, en realidad, el mismo mensaje.

c) Cifrado por sustitución polialfabética.

Usas **varios alfabetos** para cifrar el mensaje, y vas cambiando entre ellos según ciertas *reglas*.

Ejemplo: tienes un alfabeto para las posiciones pares del mensaje y otro alfabeto para las posiciones impares del texto.

d) Cifrado sencillo por transposición.

Es la forma más simple de transposición: se divide el texto en claro en **filas**, y **se leen como columnas**.

e) Cifrado por transposición con clave.

Una manera de **complicar** la técnica anterior. El tamaño de la **clave** nos va a indicar el **número de columnas** que ha de tener cada fila.

Así, a la hora de **cifrar**, podemos hacerlo en base a ciertas **condiciones.** Por ejemplo, si la clave es *secreto*, cogemos las columnas por orden alfabético (primero la *c*, después la *e*, etc.).

f) Cifrado por transposición Railfence.

Consiste en:

1. Escribir el texto **diagonalmente** con una profundidad p.
2. Se escribe el criptograma leyendo las **filas**.

Ejemplo: De *HOLA A TODOS* (y profundidad 4), obtendríamos:

```
H     O
 O   T D
  L A   O
   A     S
```

Y el mensaje cifrado sería: *HOOTDLAOAS*

g) Métodos polialfabéticos y nomenclátores.

Para complicar más el cifrado, se puede usar el **disco de Alberti** junto con nomenclátores.

> Definición.
>
> **Nomenclátores:** Asociar códigos específicos a ciertas palabras.

Cada **diez** letras descifradas, el disco **exterior** se gira dos posiciones en **sentido horario**.

El disco de Alberti no tiene *u*, así que se identifica con una *v*.

Para poder descifrar, necesitamos saber el **estado inicial del disco** con el que se cifró el mensaje original.

1.2. Cifrado Producto.

Combina tanto sustitución como transposición. En realidad, se puede ver como la aplicación sucesiva de varios cifrados. Es decir:

$$E = E_1 \cdot E_2 \cdot ... \cdot E_r$$

$$E(M) = E_1(E_2(\cdots(E_r(M))\cdots))$$

Luego el descifrado sería:

$$D = D_r \cdot D_{r-1} \cdot ... \cdot D_1$$

$$M = D(C) = D_r(D_{r-1}(\cdots(D_1(C))\cdots))$$

Gracias a esto, obtenemos sistemas de cifrado complejos, **seguros**, difíciles de atacar y fácilmente trasladables a un ordenador, a partir de **sistemas bastante sencillos**.

1.3. Cifrado Vernam (one-time-pad).

Es una variante del cifrado **OTP** (one-time-pad)

> Definición.
>
> Un **one-time-pad** es un conjunto **infinito y no repetitivo** de letras **aleatorias**.

Cada letra del *pad* se usa para cifrar una única letra del texto en claro, aplicando también módulo n (longitud del alfabeto).

En los **ordenadores**, se hace un *XOR* entre el mensaje en claro y la clave.

El cifrado Vernam presenta algunos inconvenientes:

- Hay que generar las letras del OTP de una manera **aleatoria**.
- El OTP es de **un solo uso**.

2. Algoritmos simétricos (clave privada)

2.1. Fundamentos

Definición.

Los **algoritmos simétricos** (o de clave secreta) son aquellos en los que se utiliza la misma clave (K) tanto para el proceso de cifrado como para el de descifrado, denominándose a esta clave **clave de sesión**.

Notación.

- M: Mensaje en claro (Plaintext).
- C: Criptograma o texto cifrado (Ciphertext).
- K: Clave secreta compartida (donde $K = K^*$).
- $E(M, K) = C$: Algoritmo de cifrado público.
- $D(C, K) = M$: Algoritmo de descifrado público.

La base de estos sistemas recae en el **segundo principio de Kerckhoffs**: *"El sistema no debe requerir ser secreto y debe poder caer en manos del enemigo sin inconvenientes"*. Por tanto, la seguridad depende estrictamente de que el emisor y el receptor mantengan en secreto la clave K.

Existen dos maneras fundamentales de aplicar el cifrado:

1. **Cifrado en bloques**: El algoritmo necesita procesar el mensaje M fragmentándolo en bloques de tamaño fijo (n bits). El cifrado se produce cuando el bloque de datos se entremezcla con los datos de la clave mediante operaciones matemáticas (comúnmente la operación lógica XOR) y permutaciones.
2. **Cifrado en flujo**: El mensaje se procesa bit a bit. Cada bit del mensaje se somete a una operación XOR con el bit correspondiente de un flujo de clave (keystream) generado a partir de una clave inicial K (semilla).

Para generar estos elementos pseudoaleatorios, se usan:

- **TRNG (True Random Number Generator)**: Toma valores del entorno físico del hardware.
- **PRNG (Pseudorandom Number Generator)**: Produce una secuencia abierta desde una semilla determinista.
- **PRF (Pseudorandom Function)**: Produce una cadena pseudoaleatoria de bits de longitud fija (usado para generar subclaves).

El efecto avalancha (basado en la teoría de Shannon de 1949): Para que un algoritmo sea seguro, debe cumplir con las propiedades de **difusión** (cada carácter cifrado depende de múltiples partes de la clave) y **confusión** (relación compleja entre criptograma y clave, logrado mediante permutaciones). Esto

genera el *efecto avalancha*: un pequeño cambio en el texto claro o en la clave produce un cambio masivo (estadísticamente significativo) en el criptograma, imposibilitando reducir el espacio de búsqueda.

2.2. Algoritmo DES

Definición.

El **Data Encryption Standard (DES)** es un algoritmo histórico simétrico de cifrado en bloque desarrollado por IBM (basado en el algoritmo Lucifer) estandarizado por el NIST. Utiliza bloques de 64 bits y una técnica interna conocida como **red de Feistel**.

- **Estructura**: La red de Feistel en DES somete al bloque de datos a **16 etapas o rondas** de operaciones iterativas (expansión, permutación, sustitución y operación XOR).
- **Clave**: Recibe una clave de 64 bits, pero el último bit de cada octeto se usa como paridad. Por lo tanto, la longitud efectiva es de **56 bits**.
- **Subclaves**: En cada una de las 16 rondas, se aplica una subclave de 48 bits generada por un algoritmo interno a partir de la semilla original de 56 bits.
- **Seguridad**: Su clave de 56 bits (espacio de 2^{56} claves alternativas) es demasiado corta frente al poder computacional actual, haciéndolo vulnerable a un ataque exhaustivo (fuerza bruta).

2.3. Algoritmo triple-DES

Definición.

El **Triple-DES (3DES)** es un modo de utilizar el algoritmo DES que consiste en aplicar una secuencia de tres operaciones DES consecutivas por cada bloque de datos, contrarrestando la escasa longitud de clave del DES original.

Existen dos variantes:

- **Con 3 claves distintas**: Se usan K_1, K_2, K_3, logrando una longitud combinada de **168 bits**. Es la más segura y recomendada.
- **Con 2 claves distintas**: Se utiliza $K_1 = K_3$. Ofrece 112 bits de longitud efectiva, pero ha sido objeto de diversos ataques teóricos.

2.4. Otros algoritmos simétricos

- **AES (Advanced Encryption Standard)**: Creado para sustituir a DES. Diseñado por Rijmen y Daemen (originalmente *Rijndael*). Opera

con bloques de **128 bits** y longitudes de clave de **128, 192 o 256 bits**. No usa una red de Feistel, sino una red de **sustitución-permutación-operaciones aritméticas** en un campo finito. Aplica 10, 12 o 14 rondas según el tamaño de la clave. Actualmente es el estándar indiscutible.
- **Blowfish**: Cifrado con clave variable (32 a 448 bits), aunque se recomienda usar más de 80 bits. Trabaja con bloques de 64 bits. Por su bloque pequeño, sólo se aconseja para uso heredado (*legacy*).
- **Kasumi**: Clave de 128 bits y bloques de 64 bits. Se utiliza fundamentalmente en redes móviles (UMTS como UIA1, GSM como A5/3).
- **Camellia**: Soporta claves de 128, 192 y 256 bits y bloques de 128 bits. Presenta un rendimiento y robustez similar a AES. Es usado como alternativa segura en TLS.

2.5. Modos de operación para algoritmos simétricos

Los modos de operación definen cómo aplicar el cifrado a los bloques de datos múltiples:

- **ECB (Electronic Codebook)**: Cada bloque se cifra independientemente con la misma clave. Deja expuestos patrones de datos (ej. al cifrar una imagen, el contorno sigue siendo visible).
- **CBC (Cipher Block Chaining)**: Cada bloque en claro se combina mediante XOR con el criptograma del bloque anterior antes de ser cifrado, requiriendo un vector de inicialización (IV) para el primero. Destruye los patrones visuales.
- **CFB (Cipher Feedback)** y **OFB (Output Feedback)**: Modos que permiten que un cifrador por bloques actúe como un cifrador en flujo.
- **CTR (Counter)**: Transforma un cifrado de bloque en un cifrado de flujo combinando mediante XOR un contador incremental cifrado con el texto claro.
- **GCM (Galois-CTR)**: Variante moderna de CTR muy eficiente. Usa MAC de Carter-Wegman en un campo de Galois para proporcionar confidencialidad y, además, **integridad**. Es ampliamente soportado en TLS 1.3.

Regla general recomendada: Longitud mínima de clave y de bloque de ≥ 128 bits, y cambiar la clave (*rekeying*) tras cifrar $2^{n/2}$ bloques.

2.6. Ventajas y desventajas de los algoritmos simétricos

Ventajas:

- Alto rendimiento y bajo coste computacional (cientos de MB/s en hardware, decenas de MB/s en software).
- Los algoritmos son más simples matemáticamente y requieren recursos menores.

- Pueden componerse para generar cifrados más fuertes o usarse para construir generadores pseudoaleatorios y funciones Hash.

Desventajas:

- Claves inherentemente más cortas y susceptibles a ataques de fuerza bruta si no se actualizan.
- Requieren de **acuerdo previo de la clave**: Alice y Bob deben intercambiar K por un canal seguro físicamente, lo cual es ineficiente si están lejanos.
- **Mala escalabilidad**: En un sistema de n usuarios, se requieren $\frac{n\times(n-1)}{2}$ claves separadas.
- Necesitan de una tercera parte de confianza (administrador de claves) en grandes infraestructuras.

---

3. Algoritmos asimétricos (clave pública)

3.1. Cifrado / Descifrado

Definición.

Los **algoritmos asimétricos** (inventados por Diffie, Hellman y Merkle en 1976) se basan en el uso de pares de claves matemáticamente relacionadas para cada usuario: una **clave pública** (K_{pub}) conocida por todos, y una **clave privada** (K_{priv}) custodiada celosamente por el propietario.

La asimetría radica en que lo que se cifra con una clave **sólo puede descifrarse con la otra**.

Si Bob quiere asegurar la **confidencialidad** enviando un mensaje a Alice:

1. Bob obtiene la clave pública de Alice (desde un directorio o *key-ring*).
2. Bob cifra el mensaje con la K_{pub_alice}.
3. Sólo Alice, usando su K_{priv_alice}, podrá recuperar el mensaje original.

Ventajas frente a la simétrica:

- No requiere el acuerdo físico de claves secretas a priori.
- Escala de forma óptima: en una comunidad de n usuarios, sólo existen $2n$ claves.
- Es computacionalmente imposible derivar la clave privada a partir de la pública.

Desventajas:

- Exigen gran longitud de claves (ej. un mínimo recomendado de 2048 bits para RSA o 224 para Curva Elíptica).

- Se basan en funciones matemáticas complejas (factorización, logaritmos discretos), lo que los hace **aproximadamente 1000 veces más lentos** que los simétricos.

3.2. Firma digital

La dualidad de los algoritmos asimétricos permite el servicio de **autenticación** e integridad (no repudio). Si Bob cifra un mensaje con su *propia* clave privada (K_{priv_bob}), cualquiera podrá descifrarlo usando la K_{pub_bob}. Sin embargo, dado que **solo Bob** posee la K_{priv_bob}, cuando Alice descifra exitosamente el mensaje, queda matemáticamente garantizado que el mensaje proviene legítimamente de Bob y no de un impostor.

3.3. Intercambio de claves

Debido al paupérrimo rendimiento en procesamiento de datos extensos del criptosistema asimétrico, la solución moderna es la **criptografía híbrida**:

1. Se utiliza el algoritmo asimétrico (ej. RSA o Diffie-Hellman) exclusivamente para transmitir de forma segura una clave de sesión simétrica generada aleatoriamente (K_{AB}).
2. Se utiliza un algoritmo simétrico (ej. AES) cifrando con la clave K_{AB} para proteger todo el mensaje.

3.4. Algoritmo de Diffie-Hellman

Definición.

El algoritmo de **Diffie-Hellman** (1976) es un protocolo diseñado específicamente para permitir a dos partes acordar e intercambiar una clave secreta a través de un canal de comunicaciones inseguro, basando su seguridad en la dificultad de computar **logaritmos discretos**. No sirve para cifrar mensajes ni firmar, sólo para acordar claves.

Notación.

- q: Un número primo muy grande (elemento público global).
- α: Raíz primitiva de q, donde $\alpha < q$ (elemento público global).
- X_A, X_B: Claves privadas de los usuarios A y B, donde $X < q$.
- Y_A, Y_B: Valores calculados y enviados públicamente.

El proceso matemático de intercambio es:

1. Alice calcula su valor público: $Y_A = \alpha^{X_A} \bmod q$ y se lo envía a Bob.
2. Bob calcula su valor público: $Y_B = \alpha^{X_B} \bmod q$ y se lo envía a Alice.
3. Alice computa la clave secreta combinada: $K = (Y_B)^{X_A} \bmod q$.

4. Bob computa la clave secreta combinada: $K = (Y_A)^{X_B} \bmod q$.

Ambos obtienen exactamente el mismo valor matemático:

$$K = \alpha^{X_A \times X_B} \bmod q$$

3.5. Algoritmo RSA

Definición.

Desarrollado en el MIT en 1977 por Rivest, Shamir y Adleman. Es el criptosistema asimétrico más versátil, ya que provee Cifrado, Firma Digital e Intercambio de claves. Basa su robustez en la extrema dificultad matemática de hallar la factorización en primos de números enteros gigantescos.

Generación de las claves RSA:

1. Seleccionar dos números primos grandes aleatorios p y q.
2. Calcular el módulo $n = p \times q$.
3. Calcular la función indicatriz de Euler: $\varphi(n) = (p-1) \times (q-1)$.
4. Elegir una clave pública e (exponente) que sea primo relativo con $\varphi(n)$ (es decir, $mcd(e, \varphi(n)) = 1$) tal que $e < \varphi(n)$.
5. Determinar la clave privada d (inverso multiplicativo), que debe cumplir que $e \times d \equiv 1 \pmod{\varphi(n)}$.

Para operar, los textos (M) se convierten a equivalentes numéricos decimales (ej. ASCII) divididos en trozos menores al módulo ($M < n$).

- **Para cifrar:**
$$C = M^e \bmod n$$
- **Para descifrar:**
$$M = C^d \bmod n$$

---

4. Otras primitivas criptográficas

4.1. Funciones HASH

Definición.

Una **función hash criptográfica** es una función unidireccional que acepta como entrada un bloque de información M (la preimagen) de longitud variable y produce como salida un bloque o huella digital (h, *digest*) de longitud fija.

Para ser criptográficamente robusta, debe cumplir que sea computacionalmente imposible:

1. Encontrar M a partir de h (**unidireccionalidad**). Las funciones Hash NO pueden ser revertidas, por tanto no sirven para "cifrar/descifrar".
2. Encontrar dos mensajes distintos M y M' que produzcan exactamente la misma salida h (**resistencia a colisiones**).

Al más mínimo cambio en el mensaje original (incluso 1 bit), como promedio, se modificará el 50% de los bits de salida (*efecto avalancha*).

La principal utilidad de una función Hash es aportar el **servicio de Integridad de Datos**. Si combinamos esto con la **Firma Digital** (Criptografía de Clave Pública), el emisor únicamente cifra el $H(M)$ en lugar de todo el documento, haciéndolo infinitamente más eficiente.

Algoritmos conocidos:

- **MD-5**: Produce 128 bits. Actualmente **insegura**; puede sufrir ataques de colisión en apenas segundos.
- **SHA-1**: Produce 160 bits. Ampliamente usada en el pasado pero **rota y considerada insegura** (En 2017, un equipo de Google y CWI Amsterdam demostraron colisiones completas manipulando PDFs bajo el mismo Hash).
- **SHA-2**: Familia actual estándar. Sus variantes principales procesan salidas de **224, 256, 384 y 512 bits** soportando mensajes enormes con bloques internos de procesamiento de 512 o 1024 bits. Segura para sistemas actuales, aunque requiere el truncamiento en algunas variantes más cortas.
- **SHA-3**: Basado en el algoritmo *Keccak*. Creado no por una debilidad de SHA-2, sino como un respaldo estructural distinto (diseño tipo *esponja* en vez de familia MD-X). Genera salidas de 224 a 512 bits. Sin colisiones ni amenazas actuales.
- **Whirlpool**: Función europea que produce 512 bits. Se diferencia por basarse en métodos de diseño parecidos al algoritmo AES, garantizando diversidad algorítmica.

4.2. Códigos de autenticación de mensajes

Las funciones Hash puras carecen de autenticación (cualquiera podría modificar el archivo en tránsito y recalcular su Hash). Para resolver esto existen los MAC.

Definición.

Un **MAC (Message Authentication Code)**, también llamado checksum criptográfico, es una primitiva que toma como entrada el mensaje original M e incorpora una clave secreta compartida K para producir una huella de autenticación.

! Notación.

$$MAC = F_{MAC}(M, K)$$

Alice enviará M junto con el MAC. Al llegar a Bob, éste recalcula el MAC empleando la clave simétrica K que comparte con Alice. Si el MAC coincide, se garantiza no solo la **integridad de los datos**, sino la **autenticación** en el origen del mensaje (Bob está seguro que fue Alice pues sólo ella tiene K).

Pueden desarrollarse basándose en dos categorías:

1. Basadas en funciones Hash: Ej. **HMAC** y UMAC.
2. Basadas en algoritmos de cifrado en bloque: Ej. EMAC, CMAC, AMAC.

3. Seguridad de la Información: Esquemas, Protocolos y Certificados.

Primera Parte: Gestión de las "claves" y Protocolos de Soporte.

1. El problema de la distribución de claves simétricas.

En criptografía simétrica, dos entidades (por ejemplo, Alice y Bob) necesitan compartir una misma clave secreta (K_{AB}) para comunicarse de forma segura. En redes pequeñas esto es manejable, pero hay escenarios donde la utilización de la criptografía de clave pública (o asimétrica) para el intercambio de una clave de sesión K_{AB} puede ser **NO conveniente**: por ejemplo, en redes LAN grandes sin conexión a la red WAN. En este caso, Alice y Bob necesitan igualmente alguna solución que les permita, aun estando geográficamente (moderadamente) lejanos, acordar esa clave de sesión K_{AB}.

La solución estándar es introducir una tercera parte confiable, conocida como **Centro de Distribución de Claves (KDC - Key Distribution Center)**. En el modus operandi general de este tipo de protocolos, cada usuario del sistema comparte, de inicio, una clave secreta con el KDC mediante algún proceso de registro o inscripción previo.

Definición.

KDC (Key Distribution Center): Es un servidor central de confianza. Cada usuario del sistema comparte una clave maestra secreta a largo plazo con el KDC (mediante un registro previo). El KDC utiliza estas claves maestras para generar y distribuir de forma segura claves de sesión temporales entre los usuarios. Su uso se basa en el empleo de **claves jerárquicas**: se requieren al menos dos niveles de claves (las claves maestras

K_{AT}, K_{BT} con el KDC, y la clave de sesión K_{AB} entre los usuarios).

Ventajas del KDC: Permite que usuarios que no se conocen previamente puedan establecer una clave de sesión segura de forma automática, sin necesidad de haber contactado antes.

Inconvenientes del KDC:

1. **Punto único de fallo:** Si el KDC se cae, nadie puede establecer nuevas comunicaciones seguras.
2. **Cuello de botella:** Todo el tráfico de establecimiento de claves pasa por él, ya que todos los usuarios necesitan comunicar con el KDC de forma frecuente para obtener claves. Esto puede saturar la red.
3. **Punto crítico de seguridad:** El KDC posee suficiente información para suplantar a cualquier usuario. Si un atacante lo compromete, toda la red queda vulnerable.

2. Mecanismos contra ataques de repetición.

El mayor riesgo en la distribución de claves son los **ataques de repetición (Replay Attacks)**, donde un atacante (habitualmente denominado *Mallory*) intercepta un mensaje cifrado legítimo y lo reenvía más tarde para causar estragos (ej. Denegación de Servicio o suplantación de identidad). El atacante no necesita conocer el contenido del mensaje para causar daño: le basta con reenviarlo.

Para evitar que se procesen mensajes antiguos, los protocolos utilizan mecanismos que garantizan la **"frescura" (*freshness*)** del mensaje:

1. **Marcas de tiempo (*Timestamps*):** Se incluye la hora exacta en el mensaje. Si el receptor recibe un mensaje con una marca de tiempo muy antigua, lo descarta. *Inconveniente:* Requiere que todos los relojes de la red estén perfectamente sincronizados. Por fallos del sistema o por sabotaje, los relojes pueden desincronizarse, lo que puede derivar en un ataque de denegación de servicio.
2. **Nonces (*Núnicos*):** Un número aleatorio de un solo uso. Cada parte de la comunicación debe recordar todos los nonces enviados o recibidos, y rechazar cualquier mensaje que contenga un nonce previamente usado. *Inconveniente:* Si una de las partes pierde la lista de nonces, es susceptible a ataques de repetición.
3. **Combinación de ambas estrategias:** Se pueden combinar timestamps y nonces para limitar el tiempo durante el cual hay que recordar los nonces, aunque el protocolo resultante se vuelve más complejo.

ℹ Modelo mental.

- **Timestamp** responde a: *¿Es este mensaje reciente?* (Basado en el reloj del sistema).
- **Nonce** responde a: *¿He visto exactamente este mensaje antes?* (Basado en memoria de mensajes procesados).

3. Modelos y Protocolos de KDC.

La mayoría de las técnicas de distribución de claves se adaptan a situaciones, escenarios y aplicaciones específicas. Hay muchos modelos de distribución de claves: **simples** y **genéricos**. Dentro de los genéricos encontramos los modelos **PULL**, **PUSH**, o sus combinaciones.

3.1. Modelo Simple ("La Rana de la Boca Grande").

Alice genera la clave de sesión (K_{AB}) y se la envía al KDC (cifrada con su clave maestra K_{AT}). El KDC la descifra y se la reenvía a Bob (cifrada con la clave maestra de Bob K_{BT}).

Como se puede observar, existe validación de identidad: las claves con el KDC son secretas, por lo que nadie más habría sido capaz de cifrar la clave secreta K_{AB}; además, existe autenticación de cada parte involucrada.

- **Problema:** Es vulnerable a ataques de repetición. Si Mallory intercepta el canal y captura todos los mensajes del KDC a Bob, puede reenviarlos continuamente causando un ataque de Denegación de Servicio (DoS) **sin necesidad de derivar** K_{AB} **ni** K_{BT}. La solución pasa por incorporar nonces o timestamps, tal y como se describió en la sección anterior.

3.2. Modelo PULL.

La entidad A desea tener comunicación segura con B, por lo que **contacta con el KDC primero** para solicitar la clave. El KDC le responde a Alice con la clave de sesión y un "ticket" cifrado que Alice debe entregar a Bob. Generalmente incorpora un mecanismo de desafío-respuesta (*challenge-response*) entre Alice y Bob para confirmar que ambos poseen la clave correcta.

3.3. Modelo PUSH.

La entidad A contacta primero con la entidad B a fin de que **ésta última sea la encargada de solicitar al KDC** la clave correspondiente. Es decir, es Bob quien "empuja" (*push*) la solicitud al KDC en nombre de Alice.

---

4. Protocolo Needham-Schroeder.

Es el protocolo PULL más representativo. Está basado en **nonces** y en un proceso final de desafío-respuesta (*challenge-response*) entre Alice y Bob para confirmar que ambos tienen la clave. Se denomina formalmente:

$$A \to S : A, B, N_A$$
$$S \to A : E_{K_{AT}}\{N_A, K_{AB}, B, E_{K_{BT}}\{K_{AB}, A\}\}$$
$$A \to B : E_{K_{BT}}\{K_{AB}, A\}$$
$$B \to A : E_{K_{AB}}\{N_B\}$$
$$A \to B : E_{K_{AB}}\{f(N_B)\}$$

Donde S es el KDC (Trent), N_A y N_B son nonces, y $f(n) = n - 1$ es una función muy obvia pero suficiente para el desafío-respuesta.

Descripción de los pasos:

1. Alice pide al KDC (S) hablar con Bob enviando su identidad, la de Bob y un nonce N_A.
2. El KDC envía a Alice la clave K_{AB} y un ticket cifrado para Bob, todo ello cifrado con K_{AT}. El nonce N_A garantiza la frescura de la respuesta del KDC.
3. Alice envía el ticket a Bob (no puede descifrarlo, solo reenviarlo).
4. Bob descifra el ticket, obtiene K_{AB}, y envía un nonce N_B cifrado con K_{AB} a Alice (Desafío).
5. Alice responde enviando $f(N_B) = N_B - 1$ cifrado con K_{AB} (Respuesta), demostrando que posee la clave.

⚠ Vulnerabilidades de Needham-Schroeder.

Este protocolo presenta **tres ataques** que fueron descubiertos años después de su publicación:

- **Ataque 1:** Mallory puede suplantar la identidad de Alice si consigue derivar la clave K_{AT}. A partir de ese punto, todos los mensajes quedan comprometidos.
- **Ataque 2:** Mallory puede suplantar la identidad de Bob si consigue derivar la clave K_{BT}.
- **Ataque 3 (el más sutil):** Mallory puede producir un ataque de DoS mediante repetición, especialmente en las últimas fases del protocolo. El problema es que la frescura de los mensajes solo se garantiza en los mensajes 1 y 2, pero **no en el resto** de la transacción. El ticket que Alice envía a Bob en el paso 3 no contiene ningún mecanismo de frescura generado por Bob, por lo que Mallory puede almacenar ese ticket y reutilizarlo más tarde.

La **solución** consiste en extender el uso del nonce al resto de transacciones,

incluyendo un nonce de Bob desde el principio.

4.1. Protocolo Amended Needham-Schroeder.

Soluciona el fallo del protocolo anterior en relación a los ataques de repetición. En esta versión, Bob participa desde el principio aportando un nonce propio antes de que Alice contacte con el KDC, garantizando así la frescura en todos los pasos del protocolo.

---

5. Protocolo Otway-Rees.

Diseñado también para solucionar el fallo de frescura de Needham-Schroeder, aunque con un diseño diferente. El protocolo formalizado es:

$$A \to B : I, A, B, E_{K_{AT}}\{N_A, I, A, B\}$$

$$B \to T : I, A, B, E_{K_{AT}}\{N_A, I, A, B\}, E_{K_{BT}}\{N_B, I, A, B\}$$

$$T \to B : I, E_{K_{AT}}\{K_{AB}, N_A\}, E_{K_{BT}}\{K_{AB}, N_B\}$$

$$B \to A : I, E_{K_{AT}}\{K_{AB}, N_A\}$$

Donde I es un identificador de sesión y N_A, N_B son nonces de Alice y Bob respectivamente. La presencia de los nonces de ambas partes en los mensajes al KDC intenta solucionar el problema del *freshness*.

⚠ Vulnerabilidades de Otway-Rees.

Sin embargo, el protocolo presenta **dos agujeros de seguridad** importantes:

- **Primer agujero:** Mallory puede interceptar y reenviar los mismos mensajes cifrados ($E_{AT}(R_A, I, Alice, Bob)$ y $E_{BT}(R_B, I, Alice, Bob)$) al KDC, haciendo que las partes acepten una clave predecible o inválida.
- **Segundo agujero:** Mallory puede reenviar exactamente los mismos mensajes sin modificarlos, de forma que Alice y Bob piensan que el canal es seguro cuando en realidad la clave de sesión K_{AB} puede ser conocida o controlada por el atacante (por ejemplo, $K_{AB} = (I \| Alice \| Bob)$).

---

6. Protocolo Kerberos.

Protocolo ampliamente utilizado en redes corporativas (y el mecanismo de autenticación por defecto en entornos Windows Active Directory). Se basa en **timestamps** y tiempos de vida de la sesión (L). Su flujo básico es:

$$A \to T : Alice, Bob$$

$$T \to A : E_{K_{BT}}\{Time, L, K_{AB}, Alice\}, E_{K_{AT}}\{Time, L, K_{AB}, Bob\}$$

$$A \to B : E_{K_{AB}}\{Alice, Time\}, E_{K_{BT}}\{Time, L, K_{AB}, Alice\}$$

$$B \to A : E_{K_{AB}}\{Time + 1\}$$

Donde $Time$ es el timestamp de frescura y L es el tiempo de vida del token (caducidad).

Características clave de Kerberos:

- Proporciona un mecanismo de ***Single Sign-On (SSO)***: Alice se autentica una sola vez ante el servidor Kerberos y obtiene tickets que le permiten acceder a múltiples servicios (por ejemplo, el servidor de Contabilidad) sin necesidad de tener una cuenta en cada uno de ellos ni volver a autenticarse.
- Su principal debilidad: **por fallos del sistema o por sabotaje, los relojes pueden desincronizarse**, lo que puede derivar en un ataque de denegación de servicio, ya que los tickets con timestamps desfasados son rechazados.

> **i** Ejemplo de uso de Kerberos.
>
> Alice desea acceder al Servidor del Departamento de Contabilidad sin tener una cuenta en ese servidor. Alice se autentica ante el KDC Kerberos. El KDC le entrega un ticket de servicio para el servidor de Contabilidad, firmado con la clave que comparte con dicho servidor. Alice presenta el ticket al servidor, que lo verifica sin necesidad de contactar de nuevo con el KDC.

---

7. Protocolos Avanzados: Combinaciones PULL-PUSH.

Existen protocolos que combinan múltiples tipos de estrategias (tanto a nivel de modelos como de mecanismos de seguridad):

- **PUSH con PULL → PUSH extendido.**
- **PULL con PUSH → PULL extendido.**

7.1. Protocolo Yahalom.

Objetivo: Permitir a Trent (el KDC) generar la clave de sesión K_{AB} y enviarla directamente a Alice, e indirectamente a Bob (a través de Alice).

$$A \to B : A, N_A$$
$$B \to T : B, E_{K_{BT}}\{A, N_A, N_B\}$$
$$T \to A : E_{K_{AT}}\{B, K_{AB}, N_A, N_B\}, E_{K_{BT}}\{A, K_{AB}\}$$
$$A \to B : E_{K_{BT}}\{A, K_{AB}\}, E_{K_{AB}}\{N_B\}$$

Alice inicia el protocolo, Bob aporta un nonce propio (N_B) que va cifrado hasta el KDC, lo que garantiza la frescura en toda la cadena.

7.2. Protocolo Neuman-Stubblebine.

Objetivo: Combinar timestamps y nonces para verificar la frescura de las transacciones, aprovechando las ventajas de ambos mecanismos y mitigando sus inconvenientes individuales.

$$A \to B : A, N_A$$
$$B \to T : B, E_{K_{BT}}\{A, N_A, timestamp_B\}, N_B$$
$$T \to A : E_{K_{AT}}\{B, K_{AB}, N_A, timestamp_B\}, E_{K_{BT}}\{A, K_{AB}, timestamp_B\}, N_B$$
$$A \to B : E_{K_{BT}}\{A, K_{AB}, timestamp_B\}, E_{K_{AB}}\{N_B\}$$

En fases posteriores, si Alice y Bob desean renovar la sesión sin pasar de nuevo por el KDC:

$$A \to B : N'_A, E_{K_{BT}}\{A, K_{AB}, timestamp_B\}$$
$$B \to A : N'_B, E_{K_{AB}}\{N'_A\}$$
$$A \to B : E_{K_{AB}}\{N'_B\}$$

7.3. Protocolo Kao-Chow.

$$A \to T : A, B, N_A$$
$$T \to B : E_{K_{AT}}\{N_A, K_{AB}, A, B\}, E_{K_{BT}}\{N_A, K_{AB}, A, B\}$$
$$B \to A : E_{K_{AT}}\{N_A, K_{AB}, A, B\}\ ,\ E_{K_{AB}}\{N_A\}\ ,\ N_B$$
$$A \to B : E_{K_{AB}}\{N_B\}$$

! Resumen: ¿Qué protocolo elegir?

Hemos visto que existen **diversos protocolos** para solucionar el problema de la administración e intercambio de claves. El protocolo a elegir depende del escenario y de lo que se quiere proteger. Como regla general:

- Si la red tiene relojes sincronizados y se busca SSO: **Kerberos**.
- Si se necesita frescura sin depender de relojes: **Amended Needham-Schroeder** o **Yahalom**.
- Si se quieren combinar ambas estrategias de frescura: **Neuman-Stubblebine**.

Recuerda además que usar un KDC no está exento de riesgos: el KDC posee suficiente información para suplantar a cualquier usuario, representa un único punto de fallo, y puede convertirse en un cuello de botella de rendimiento.

Segunda Parte: Infraestructuras de Clave Pública (PKI) y Certificados.

1. El problema de la Autenticidad de la Clave Pública.

En sistemas criptográficos asimétricos, las claves públicas son visibles para todos. El riesgo aquí no es que roben la clave pública, sino la **suplantación**: ¿Cómo sabe Bob que la clave pública que afirma ser de Alice realmente le pertenece a ella y no a un atacante que se hace pasar por ella? Y recíprocamente, ¿cómo sabe Alice que la clave pública de Bob es genuina?

Bob necesita algún documento digital con algún "sello de garantía", es decir, algo equivalente a lo que en papel sería un documento oficial certificado por una autoridad.

Para resolver esto, se utilizan los **Certificados Digitales**.

Definición.

Certificado Digital (o de clave pública): Es un documento electrónico que vincula de manera criptográfica e inequívoca la información de identificación de un usuario con su clave pública. Esta garantía es proporcionada por la **firma digital de una tercera parte confiable (TTP - Trusted Third Party)**, denominada **Autoridad de Certificación (CA)**. La CA garantiza que una clave pública pertenece a cierto usuario inequívocamente identificado.
Nótese que la identidad del usuario por sí sola **no es suficiente** para este propósito, ya que un usuario puede tener más de un par <clave pública,

clave privada>. El certificado vincula un par de claves concreto a una identidad.

i Analogía.

Un certificado digital es como el Documento Nacional de Identidad (DNI) físico. Tu nombre es tu identidad, tu fotografía es tu clave pública, y el sello del Ministerio del Interior es la firma que garantiza que ambos te pertenecen.

2. Estructura de un Certificado (X.509).

La ITU-T ha definido una estructura estándar de certificado digital que ha sido adoptada internacionalmente: el **certificado X.509**. Contiene los siguientes campos fundamentales:

Campo	Descripción
Versión	Indica el número de versión de X.509 (1, 2 ó 3).
Número de Serie	Número de identificación único para este certificado digital, asignado por la CA.
Algoritmo de Firma	Algoritmo usado por la CA para firmar (ej. SHA-1 con RSA), para que el receptor sepa cómo validar la firma.
Emisor (*Issuer*)	Nombre X.500 de la CA emisora.
Período de Validez	Fecha y hora de inicio y de expiración (*No válido antes de* / *No válido después de*).
Sujeto (*Subject*)	Nombre en formato X.500 del usuario cuya clave pública se está certificando.
Algoritmo de Clave Pública	Identifica el algoritmo de la clave pública del sujeto (ej. RSA).
Clave Pública	El valor matemático de la clave pública del Sujeto (ej. RSA 1024 bits).
Identificador único de emisor	Cadena opcional para que el nombre de la CA no sea ambiguo (ej. dirección de email, dirección postal).
Identificador único de usuario	Cadena opcional para que el nombre del usuario no sea ambiguo (ej. dirección de email).

Extensiones (v3)	Campo opcional para almacenar información adicional, como el **Uso de la clave** (ver sección 2.1).

! La Firma de la CA.

Al final del documento se añade la **Firma de la CA (Autoridad de Certificación)**. Esta firma es el resultado de:

1. Aplicar una función **Hash** (un resumen matemático) a todos los campos anteriores del certificado.
2. Cifrar ese hash con la **clave privada de la CA**: $SELLO\_DIGITAL = E_{K_{priv\_CA}}(H(documento))$.

Quien verifique el certificado usará la **clave pública de la CA** para descifrar la firma y comparar el hash resultante con el hash que calcule él mismo sobre los campos del certificado. Si coinciden, el certificado es auténtico e íntegro.

2.1. Uso de Claves en los Certificados Digitales.

Cuando una CA crea un certificado, debe indicar expresamente **para qué se van a utilizar las claves** asociadas a dicho certificado. Esto es fundamental: no todas las claves deben poder usarse para todo. El campo **X.509v3 Key Usage** (Uso de Clave) permite especificar usos como:

- **Digital Signature:** Necesario para firmar (ej. firmar documentos PDF).
- **Non Repudiation:** Para garantizar el no repudio.
- **Key Encipherment:** Necesario para cifrar.
- **Key Agreement:** Para protocolos de acuerdo de clave (ej. Diffie-Hellman).
- **Certificate Sign:** Permite a una entidad firmar otros certificados. **Este es el uso que debe asignarse a una CA cuando se la está certificando.**
- **CRL Sign:** Permite firmar Listas de Revocación de Certificados.

El campo **X.509v3 Extended Key Usage** permite especificar usos más específicos, como firma de código (*Code Signing*), autenticación de servidor TLS, OCSP Signing, acceso con Smartcard (*Microsoft Smartcard Login*), etc.

Los certificados se exportan habitualmente en formato **PKCS#12**, que empaqueta el certificado junto con su clave privada en un único fichero protegido por contraseña.

i Pregunta de reflexión.

¿Qué tipo de uso de clave se le debería asignar a una nueva CA por defecto? La respuesta es **Certificate Sign** y **CRL Sign**: la CA necesita

poder firmar certificados de otros y firmar las listas de revocación, pero no necesariamente cifrar o firmar documentos de usuario final.

3. Infraestructura de Clave Pública (PKI).

Es imposible que una sola Autoridad de Certificación (CA) gestione todos los certificados del mundo. Por ello, las CAs se agrupan en una jerarquía llamada **PKI (Public Key Infrastructure)**.

La PKI gestiona todo el **ciclo de vida** de los certificados: los emite, distribuye, salvaguarda, renueva y revoca.

Cadenas de Confianza (*Certification Paths*):

Entre las CAs se utiliza de forma recursiva el esquema de certificación, creándose las **cadenas de confianza** (o caminos de certificación). La jerarquía establece dos tipos de certificados:

- **Certificado raíz (*Root CA*):** Una CA Raíz **se autofirma su propio certificado** (es el ancla de confianza). Tu ordenador o navegador incluye de fábrica una lista de CAs Raíz en las que confía.
- **Certificados intermedios:** Una CA Raíz firma el certificado de una CA Intermedia, la cual a su vez firma el certificado del usuario final. Al validar, tu ordenador escala por esta cadena hasta llegar a una CA Raíz en la que confía.

ℹ Ejemplos de CAs públicas.

Existen CAs gestionadas por la comunidad y gratuitas, como **CAcert.org**, aunque no están incluidas por defecto en todos los navegadores. La CA más utilizada en la actualidad para certificados de servidores web es **Let's Encrypt** (https://letsencrypt.org/), que emite certificados gratuitos y automatiza su renovación, siendo un proyecto de la Linux Foundation.

4. Revocación de Certificados.

Si la clave privada de un usuario es comprometida, o si el usuario deja de trabajar en una organización, su certificado debe ser invalidado inmediatamente, **antes de que llegue su fecha natural de expiración**. La CA se encarga de realizar la revocación bajo petición del usuario, y tiene la obligación de publicar esa información para que el resto de usuarios puedan comprobarlo antes de usar el certificado.

Escenario típico: Para que Bob verifique la firma de Alice sobre un documento digital, no solo ha de verificar el certificado de Alice y su validez temporal; además, **ha de comprobar que ese certificado no ha sido revocado**.

Existen dos mecanismos principales:

1. **CRL (Certificate Revocation List):** Una lista publicada periódicamente (con *timestamping*) por la CA, firmada por la propia CA, que contiene los números de serie de los certificados revocados. Quien verifica debe descargar la lista y comprobar que el número de serie del certificado de Alice no aparece en ella. El estándar X.509 define la estructura de una CRL versión 2.

2. **OCSP (Online Certificate Status Protocol):** Un protocolo (RFC 6960) para consultar el estado de un certificado específico *en tiempo real* a un servidor (OCSP Responder). Define un formato estándar para mensajes de peticiones y respuestas. Es más ágil que la CRL ya que evita descargar listas completas, y permite una respuesta inmediata sobre si un certificado está válido, revocado o se desconoce su estado.

Tercera Parte: Tarjetas Inteligentes y DNI Electrónico.

1. Tarjetas Inteligentes (*Smartcards*).

Una tarjeta inteligente o *smartcard* es una tarjeta que incluye un chip cuya función puede ser variada. Su uso se extiende hoy a muchos sectores (banca, salud, transporte, identidad, etc.).

Clasificación según el método de comunicación o interfaz:

- **Contacto:** Requieren inserción física en un lector.
- **Sin contacto / NFC (*Near-Field Communication*):** Comunicación por radiofrecuencia a corta distancia.

Clasificación según las capacidades del chip:

- **Tarjetas de memoria:** Solo almacenan datos y no albergan aplicaciones. Uso: identificación y control de acceso sin altos requisitos de seguridad (ej. tarjetas de fidelización).
- **Tarjetas microprocesadas:** Albergan datos y aplicaciones. Uso: pago con monederos electrónicos.
- **Tarjetas criptográficas:** Tarjetas microprocesadas avanzadas que incluyen **módulos hardware** para la ejecución de cifrados y firmas digitales en su interior de forma segura. Son el tipo más relevante para ciberseguridad.

2. El DNI Electrónico (DNI-e).

El DNI-e español es un claro ejemplo de **tarjeta criptográfica**. A través de las capacidades criptográficas que aporta, permite identificar al ciudadano en medios telemáticos y realizar firmas electrónicas con plena validez legal.

Hardware del DNI-e:

El DNI-e está dotado con el chip **ST19WL34** (STMicroelectronics), que incorpora, entre otros componentes relevantes:

- CPU de 8 bits de la familia ST19X.
- **Acelerador hardware para algoritmos criptográficos**: DES, Triple DES (con modos CBC), RSA 1024 bits (con y sin CRT), AES-128. Esto es crucial porque las **operaciones criptográficas pesadas se realizan físicamente dentro de la tarjeta**, sin exponer las claves al exterior.
- Soporte hardware para aritmética de gran precisión (hasta 2176 bits de operando), lo que permite implementar RSA con mayor eficiencia.
- **Generador de números aleatorios** compatible con FIPS 140-2, con dos fuentes de entropía.
- Bloque de cálculo de CRC ISO 3309.
- Retención de datos de al menos 10 años y resistencia de 500.000 ciclos de borrado y escritura.

El sistema operativo que gestiona el chip se denomina **DNIe v3.0**, desarrollado por la FNMT (Fábrica Nacional de Moneda y Timbre / CERES) a partir de las especificaciones funcionales de la Dirección General de Policía. Este SO ha sido sometido a los perfiles de protección de la **certificación Common Criteria**.

Interfaces de acceso:

El DNI-e 3.0 ofrece dos opciones para acceder al chip: mediante contacto físico (lector de tarjetas) o mediante **NFC**. En el acceso por NFC se utiliza el **CAN (*Card Access Number*)**, un número que aparece en la parte inferior del DNI 3.0 y corresponde con el número de la tarjeta, como medida de seguridad adicional para establecer el canal cifrado con el chip.

Zonas de Memoria del DNI-e:

La memoria del chip está dividida en tres áreas protegidas por cortafuegos (*firewalls*) lógicos:

1. **Zona Pública:** Accesible sin restricciones. Contiene el certificado de la CA emisora y las claves públicas genéricas.
2. **Zona Privada:** Accesible únicamente tras introducir el **PIN** del ciudadano. Contiene los certificados de uso del titular:
   - **Certificado de Autenticación:** Asegura que la comunicación electrónica se realiza con el titular del DNI, pero no demuestra voluntad de firma. Está restringido a operaciones de confirmación de identidad y acceso seguro a sistemas remotos.

- **Certificado de Firma Digital:** Para la firma de documentos, garantizando la integridad del documento y el **no repudio de origen**.

3. **Zona de Seguridad:** Solo accesible en los **puntos de actualización del DNI-e** (en las comisarías). Contiene datos biométricos (huella dactilar), fotografía y firma manuscrita.

El DNI-e cuenta además con un **certificado de componente**, emitido para autenticar al propio chip y cifrar la comunicación con él, de forma similar a como se utiliza un certificado TLS en un servidor web.

> **!** Generación y Custodia de Claves.
>
> El par de claves (pública y privada) de los certificados del DNI-e **se genera en el interior del chip** gracias al generador de números aleatorios integrado, **en presencia del ciudadano**. Esto garantiza que solo existirá una copia de cada clave privada, y que ésta residirá siempre en el interior del chip.
> La clave privada **jamás sale al exterior**: cuando se firma un documento, el documento (o su hash) entra a la tarjeta, la tarjeta aplica la clave privada internamente, y devuelve únicamente la firma resultante. Esto hace al DNI-e resistente a ataques de extracción de claves desde el exterior.

Revocación y Validación en el DNI-e:

En el ámbito del DNI-e se usa el protocolo **OCSP** para las revocaciones, aislando la carga de trabajo de forma eficiente. En la PKI adoptada para el DNI-e se ha optado por asignar las funciones de **Autoridad de Validación** a entidades **diferentes** de la Autoridad de Certificación:

- La **Autoridad de Certificación** es la Policía Nacional / FNMT-CERES.
- Las **Autoridades de Validación** son entidades independientes (como el Ministerio de Administraciones Públicas o el Ministerio de Industria), que responden a las consultas OCSP sin sobrecargar a la CA.

Marco legal del DNI-e:

El DNI-e está regulado por un marco legal básico que le otorga plena validez jurídica a los documentos firmados con él, equiparando la firma electrónica cualificada a la firma manuscrita ante la ley.

Cuarta Parte: Mecanismos de Autenticación y Control de Acceso.

1. Mecanismos de Autenticación de Usuarios

La autenticación es el proceso de **verificar que una entidad es quien dice ser**. Es el primer eslabón de la cadena de seguridad, anterior a la autorización y al control de acceso.

> ! Importante: autenticación autorización.
>
> - **Autenticación:** ¿Quién eres? → Verifica identidad.
> - **Autorización:** ¿Qué puedes hacer? → Verifica permisos.
>
> Una entidad puede autenticarse correctamente y aun así no tener permiso para acceder a un recurso concreto. Son servicios independientes aunque relacionados.

```
Usuario presenta credencial


   AUTENTICACIÓN        ← ¿Eres quien dices ser?

            (identidad verificada)


   AUTORIZACIÓN         ← ¿Tienes permiso para esto?


    Acceso al recurso
```

1.1 Clasificación de Factores de Autenticación

Existen tres categorías fundamentales, basadas en lo que el usuario aporta como evidencia:

Factor	Categoría	Ejemplos
Conocimiento	Algo que solo yo **CONOZCO**	Contraseñas, PINs
Posesión	Algo que solo yo **TENGO**	Tokens físicos, tarjetas inteligentes, certificados
Inherencia	Algo que yo **SOY**	Huella dactilar, iris, geometría facial

---

1.2 Factor CONOZCO — Autenticación por Contraseña

En los sistemas basados en contraseñas, el usuario conoce cierta información que se supone que nadie más conoce (acceso a S.O., servicios web, etc.).

Características principales:

- La contraseña **puede transferirse** de un usuario a otro (riesgo).
- **Más de un usuario puede usarla simultáneamente** (no garantiza exclusividad).

⚠ Inconvenientes del uso de contraseñas.

- Requieren ser **robustas** (longitud, complejidad) y **no reutilizarse** entre servicios.
- La proliferación de servicios lleva a demasiadas contraseñas → tendencia a reutilizarlas o simplificarlas.
- Las políticas de seguridad no siempre se cumplen:
  - Tamaño mínimo y carácter alfanumérico.
  - Renovación periódica (*rekeying*).

Salt: Almacenamiento Seguro de Contraseñas

Las contraseñas **nunca deben almacenarse en claro**. El mecanismo estándar es almacenar un **hash** de la contraseña:

```
Usuario introduce contraseña


   H(contraseña)   →  comparar con hash almacenado
```

Sin embargo, el hash simple presenta una vulnerabilidad crítica: los ataques de **Rainbow Table**.

💡 Definición: Rainbow Table.

Una Rainbow Table es una tabla precalculada que relaciona valores hash con las contraseñas originales. Permite a un atacante, una vez obtenida la base de datos de hashes, derivar contraseñas de forma extremadamente rápida sin necesidad de calcular hashes en tiempo real.

La solución: el Salt (Sal)

¿Qué es el Salt?

Salt es un valor aleatorio que se añade a la contraseña **antes** de aplicar la función hash. El resultado es:

$$H(\text{Salt} \| \text{Contraseña})$$

El Salt se almacena junto al hash en la base de datos (no es secreto).

¿Por qué el Salt frustra las Rainbow Tables?

- **CASO 1 — El atacante tiene una tabla de hashes precalculados:** No puede comparar directamente $H(1234)$ con $H(\text{Salt} \| 1234)$. La tabla queda obsoleta.
- **CASO 2 — El atacante tiene una lista de contraseñas robadas:** Puede calcular $H(\text{Salt} \| \text{ContraseñaRobada})$ para cada usuario. Sin embargo, **no puede saber si dos usuarios tienen la misma contraseña**, ya que sus Salts distintos producen hashes distintos.

Importante: el Salt NO es secreto.

El Salt **no necesita ser secreto**. Su utilidad radica en ser **único por usuario**, no en ser confidencial. Almacenarlo junto al hash es correcto y esperado.

---

bcrypt — Función Hash Adaptativa

¿Qué es bcrypt?

bcrypt es una función hash diseñada específicamente para contraseñas. Combina Salt y contraseña, y está basada en el algoritmo de cifrado **Blowfish**.

- **¿Para qué sirve?** Almacenar contraseñas de forma resistente a ataques de fuerza bruta.
- **¿Cuándo se usa?** Siempre que se almacenen contraseñas en un sistema de autenticación.

Su resistencia se debe a que incorpora un **factor de coste (número de iteraciones)**: a mayor número de iteraciones, más lento es el cómputo, dificultando los ataques.

```
Almacenamiento en BD:

    Cristina    Salt    BCrypt(Salt   1234)
```

Principios del Salt (Resumen)

Principio	Descripción	Ejemplo
No reutilizar el Salt	Un Salt único por cada contraseña	Generado con `os.urandom()`
Salt suficientemente largo	Al menos 57 bits de entropía	`/etc/shadow`: 10 chars `[a-z|A-Z|0-9]`
Usar funciones hash lentas	Aumentan el coste de ataques de fuerza bruta	`Argon2id`, `PBKDF2`, `bcrypt`

1.3 Factor TENGO — Tokens Físicos

Existe un secreto guardado en un dispositivo físico (token). El usuario posee ese objeto y lo usa para demostrar su identidad.

Tipos más comunes: tarjetas inteligentes (*smartcards*), llaves hardware (USBs criptográficos), certificados digitales.

Características principales:

- Basados principalmente en **criptografía asimétrica** ("tengo una clave privada en el token").
- La transferencia del token es posible físicamente, pero depende del diseño del sistema.
- **Solo un usuario puede usarlo simultáneamente** si el factor es físico.

⚠ Inconvenientes de los tokens físicos.

- No siempre garantiza la identidad del poseedor (cualquiera con el token pasa la autenticación).
- En caso de **pérdida o daño**, el usuario legítimo queda sin acceso.
- Algunos tokens pueden **falsificarse o clonarse**.

Ejemplo: DNI Electrónico (DNI-e)

El DNI electrónico es un ejemplo paradigmático de autenticación basada en posesión combinada con criptografía asimétrica.

Capacidades del DNI-e:

- **Identificación telemática:** acredita la identidad del titular ante servicios electrónicos.
- **Firma electrónica:** garantiza la integridad del documento y el no repudio de origen.

Certificados incluidos:

Certificado	Función	Garantía
Autenticación	Confirma que la comunicación es con el titular	No implica voluntad de firma
Firma	Firma de documentos electrónicos	Integridad + no repudio

Seguridad de las claves: el par de claves se genera dentro del chip, garantizando que **solo existe una copia de cada clave privada**, que nunca abandona el interior del chip.

1.4 Factor SOY — Autenticación Biométrica

Los sistemas biométricos extraen características biológicas o de comportamiento del usuario:

- Huella dactilar
- Imagen del iris
- Geometría facial
- Tamaño y forma de la oreja

Características principales:

- **No transferible:** los datos biométricos no pueden pasarse de un usuario a otro.
- **Exclusivo:** solo el usuario puede usarlos en un momento determinado.

⚠ Inconvenientes de la autenticación biométrica.

- El perfil biométrico debe **almacenarse previamente** en el sistema (requiere protección especial).
- Son sistemas **más costosos** que contraseñas o tokens.
- **Si un dato biométrico se compromete, se pierde para siempre** (ej.: huella destruida por quemadura → no se puede regenerar, a diferencia de una contraseña).

1.5 Comparativa de Factores de Autenticación

Característica	CONOZCO (Contraseña)	TENGO (Token)	SOY (Biométrica)
Transferible	Sí	Sí (físico)	No
Uso simultáneo	Sí	No	No
Revocable	Sí	Sí	No
Coste	Bajo	Medio	Alto
Principal riesgo	Robo/adivinanza	Pérdida/clonación	Compromiso permanente

---

1.6 Autenticación de Doble Factor (2FA)

La autenticación de doble factor combina **dos categorías distintas** de los factores anteriores para aumentar la seguridad.

Ejemplos: - Contraseña (CONOZCO) + Token físico (TENGO) - Dato biométrico (SOY) + Token (TENGO)

> **i** Ejemplo legal: Directiva Europea PSD2.
>
> La **Segunda Directiva de Servicios de Pago (PSD2)** obliga a usar autenticación de doble factor para pagos online: código de la tarjeta + aplicación móvil / SMS.
> El uso de **SMS como segundo factor es considerado débil** debido al riesgo de duplicado de SIM (*SIM swapping*).

1.7 Autenticación Basada en Códigos QR

Los códigos QR (*Quick Response*) permiten autenticar y autorizar el acceso usando dispositivos móviles.

Proceso:

1. El servidor genera un código QR con una contraseña OTP (*One Time Password*) codificada en él.
2. El usuario escanea el código para autenticarse.

Combinación recomendada: QR + OTP

- El QR contiene la OTP (de un solo uso), lo que añade resistencia frente a ataques de repetición (*replay attacks*).

⚠ Inconvenientes de la autenticación por QR.

- La información del usuario debe estar en el servidor remoto.
- El servidor necesita software para generar los códigos QR.
- El cliente necesita software para escanearlos.

1.8 Single Sign-On (SSO)

¿Qué es el SSO?

Single Sign-On (SSO) es un mecanismo que permite a un usuario **autenticarse una sola vez** para acceder a múltiples sistemas independientes pero relacionados, sin necesidad de volver a autenticarse.

- **¿Para qué sirve?** Evitar la gestión y memorización de múltiples credenciales.
- **¿Cuándo se usa?** Entornos corporativos con múltiples servicios internos, portales web federados.

```
Usuario se autentica


  [Servidor SSO]

        → Sistema A  (sin nueva autenticación)
        → Sistema B  (sin nueva autenticación)
        → Sistema C  (sin nueva autenticación)
```

Ventajas del SSO:

Dimensión	Beneficio
Usabilidad	Un solo password / token / certificado
Seguridad	Reduce el riesgo de ataques de intercepción por reducir el número de transmisiones de credenciales
Productividad	Reduce el tiempo invertido en autenticación

⚠ Desventaja crítica del SSO: punto único de fallo.

El servidor SSO es un **único punto de ataque**. Si el atacante compromete el SSO, **obtiene acceso a todos los sistemas** protegidos por él.

---

2. Mecanismos de Control de Acceso

2.1 Fundamentos del Control de Acceso

El control de acceso es uno de los servicios esenciales de la seguridad en ordenadores. El RFC-2828 define la seguridad en ordenadores como las medidas que aseguran especialmente el **servicio de control de acceso**.

Objetivos del control de acceso:

- Prevenir accesos a recursos por parte de usuarios **no autorizados**.
- Prevenir que usuarios legítimos accedan a recursos de forma **no autorizada**.
- Permitir a usuarios legítimos acceder a recursos de forma **autorizada**.

Relación con otros servicios de seguridad (modelo AAA):

```
Autenticación  →  Autorización  →  Auditoría (Accounting)
   (¿Quién?)       (¿Qué puede?)    (¿Qué hizo?)
```

- **Autorización:** concesión de un derecho o permiso para acceder a un recurso.
- **Auditoría:** revisión de registros y actividades del sistema para garantizar el cumplimiento de la política, detectar problemas y recomendar mejoras.

2.2 Elementos Básicos del Control de Acceso

Elemento	Definición	Ejemplos
Objeto	Recurso al que se controla el acceso	Ficheros, registros, puertos, programas, bases de datos
Sujeto	Entidad que potencialmente accede a los objetos	Usuario, proceso, aplicación
Derecho de acceso	Forma en que el sujeto puede acceder al objeto	`read`, `write`, `execute`, `delete`

> **i** Refuerzo conceptual: el sujeto es un proceso.
>
> El concepto de **sujeto** se asimila al concepto de **proceso**: cualquier usuario o aplicación accede a un objeto a través de un proceso que lo representa. No es el usuario directamente quien lee un fichero, sino el proceso que actúa en su nombre.

El mecanismo de control de acceso actúa como **mediador** entre el sujeto y el objeto, aplicándose sobre:

- Aplicaciones, Sistemas operativos, Firewalls, Routers
- Ficheros, Bases de datos
- Dispositivos: servidores, sensores, dispositivos móviles

2.3 Categorías de Mecanismos de Control de Acceso

Las principales categorías de control de acceso **no son mutuamente excluyentes**: un sistema puede combinar varios de ellos para cubrir distintos tipos de recursos.

Modelo	Base del control	Quién decide
DAC	Identidad del solicitante + reglas	El propietario del recurso
MAC	Etiquetas de seguridad vs. autorizaciones	El sistema / administrador central
RBAC	Rol del usuario	El administrador de roles
ABAC	Atributos del sujeto	Evaluación de condiciones

2.4 DAC — Control de Acceso Discrecional

¿Qué es DAC?

Discretionary Access Control (DAC) basa el control de acceso en la **identidad del solicitante** y en las **reglas de acceso** (autorizaciones) que indican qué solicitantes están o no autorizados a realizar determinadas acciones.
El término "discrecional" implica que el **propietario del recurso** puede, a su discreción, conceder o revocar accesos a otros sujetos.

Matriz de Acceso

La herramienta fundamental del DAC es la **matriz de acceso**, donde:

- **Filas → Sujetos** (usuarios, grupos, hosts, aplicaciones)
- **Columnas → Objetos** (ficheros, campos de datos, registros)
- **Celda → Derechos** del sujeto sobre el objeto

	File 1	File 2	File 3	File 4
User A	Own, Read, Write	—	Own, Read, Write	—
User B	Read	Own, Read, Write	Write	Read
User C	Read, Write	Read	—	Own, Read, Write

En la práctica, la matriz de acceso se descompone en dos estructuras más manejables:

Access Control List (ACL) — Descomposición por columnas

Para cada **objeto**, la ACL lista los sujetos y sus derechos de acceso.

> *Analogía: una ACL es como la lista de invitados en la puerta de un local. Cada puerta tiene su lista.*

Ejemplo:

$$L_{bar.txt} = \{(pepe, \{r\}),\ (paco, -),\ (luis, \{r, d\})\}$$
$$L_{foo.exe} = \{(pepe, -),\ (paco, \{x, d\}),\ (luis, \{x\})\}$$

Ventajas de las ACL:

- Fácil ver todos los permisos sobre un objeto determinado.
- Fácil revocar todos los permisos sobre un objeto: $L_{ob} = \{\}$.
- Fácil eliminar los permisos al eliminar el objeto: eliminar L_{ob}.

Desventaja: comprobar todos los permisos de un sujeto concreto es costoso (hay que recorrer todas las ACL).

Uso típico: sistemas orientados a la gestión de recursos, como **sistemas operativos** (ej. permisos UNIX).

Ticket de Capacidades (Capability List) — Descomposición por filas

Para cada **sujeto**, el ticket (perfil de acceso) lista los objetos autorizados y las operaciones permitidas.

Ejemplo:

$$L_{pepe} = \{(bar.txt, \{r\}),\ (foo.exe, -)\}$$
$$L_{paco} = \{(bar.txt, -),\ (foo.exe, \{x, d\})\}$$
$$L_{luis} = \{(bar.txt, \{r, d\}),\ (foo.exe, \{x\})\}$$

Ventajas:

- Fácil ver todos los permisos de un sujeto determinado.
- Fácil revocar todos los permisos de un sujeto: $L_{sj} = \{\}$.

Desventaja: comprobar todos los sujetos con acceso a un objeto concreto es costoso.

Uso típico: sistemas orientados al usuario, como **bases de datos** o **sistemas distribuidos**.

> Tabla de Autorización: tercera alternativa.
>
> La **Tabla de Autorización** almacena una fila por cada derecho de acceso de un sujeto sobre un recurso. Es más eficiente para consultas ad-hoc, aunque puede resultar más voluminosa.

Subject	Access Mode	Object
A	Own	File 1
A	Read	File 1
B	Read	File 1
B	Own	File 2
C	Write	File 4
...	...	...

2.5 MAC — Control de Acceso Obligatorio

> ¿Qué es MAC?
>
> **Mandatory Access Control (MAC)** basa el control de acceso en la **comparación de etiquetas de seguridad** (que indican la criticidad de los recursos) con las **autorizaciones de seguridad** (que indican qué entidades pueden acceder a recursos de cierto nivel).
> A diferencia de DAC, el propietario del recurso **no puede modificar las políticas de acceso**; lo hace el sistema de forma centralizada.

Etiquetas de clasificación (de mayor a menor sensibilidad):

```
Top Secret → Secret → Confidential → Restricted → Unmarked → Unclassified
```

Lógica de acceso MAC: un sujeto con autorización de nivel N puede acceder a recursos con etiqueta $\leq N$.

Uso típico: entornos militares, gubernamentales y sistemas de alta seguridad donde la política de acceso no puede ser discrecional.

2.6 RBAC — Control de Acceso Basado en Roles

¿Qué es RBAC?

Role-Based Access Control (RBAC) no basa el control de acceso en la identidad del usuario, sino en los **roles** que asume dentro de la organización. Un rol es una función o tarea específica (ej. *administrador*, *auditor*, *operador*).

- **¿Para qué sirve?** Simplificar la gestión de permisos en organizaciones con muchos usuarios y recursos.
- **¿Cuándo se usa?** Cuando el conjunto de roles es relativamente estable aunque los usuarios cambien.

Analogía: RBAC desacopla usuarios y permisos mediante roles intermedios. En lugar de asignar permisos a cada persona, se asignan a su cargo, y el cargo a la persona.

Modelo de asignación:

```
Usuario

    → Rol

            → Permisos

                    → Recursos (Objetos)
```

El modelo RBAC asigna:

1. Los **derechos de acceso a los roles**.
2. Los **roles a los usuarios**.

Fundamento: tanto el conjunto de roles como los permisos asociados a cada rol **cambian con poca frecuencia**, a diferencia de los usuarios.

Estandarización: RBAC ha sido estandarizado por el NIST (FIPS PUB 140-2, 2001). Está ampliamente adoptado en: Oracle DBMS, PostgreSQL, SAP R/3, Microsoft Active Directory, SELinux, Wikipedia, entre otros.

Entidades del Modelo RBAC

El modelo RBAC consta de **4 tipos de entidades**:

Entidad	Descripción
Usuario	Persona o proceso que accede al sistema
Rol	Función o tarea dentro de la organización

Entidad	Descripción
Permiso	Autorización para realizar una operación sobre un objeto
Sesión	Instancia activa de un usuario con roles activados

Las relaciones **muchos-a-muchos** entre usuarios roles y roles permisos proporcionan flexibilidad y granularidad que no es factible con DAC.

Restricciones en RBAC

RBAC permite definir restricciones avanzadas:

- **Roles mutuamente exclusivos:** un usuario solo puede asumir uno de los roles de un conjunto dado (restricción estática o dinámica por sesión).
- **Cardinalidad:** límites sobre el número de usuarios por rol, roles por usuario, o roles por sesión.
- **Prerrequisitos:** un usuario solo puede asumir un rol si previamente tiene otro rol asignado (ej. jerarquía de ascenso funcional en una organización).

Extensiones del Modelo NIST

Extensión	Descripción
SSD (*Static Separation of Duties*)	Define roles mutuamente excluyentes de forma permanente
DSD (*Dynamic Separation of Duties*)	Define restricciones sobre los roles que un usuario puede activar en una sesión concreta

---

2.7 ABAC — Control de Acceso Basado en Atributos

¿Qué es ABAC?

Attribute-Based Access Control (ABAC) autoriza el acceso no en función de la identidad del usuario ni de su rol, sino de los **atributos** que posee (características del sujeto, el objeto o el contexto).

- **¿Para qué sirve?** Políticas de acceso muy granulares y expresivas.
- **¿Cuándo se usa?** Cuando las condiciones de acceso son complejas y variables (ej. “solo usuarios mayores de 18 años y con residencia

en España").

Ejemplo:
"Cualquier usuario con más de 18 años y de piel morena tiene acceso al sistema" → esto permite incluso el acceso **anónimo** si la identificación no es estrictamente necesaria.

$$\text{Atributo del sujeto} = \text{Condición de acceso}$$

2.8 Modelos Adicionales de Control de Acceso

CapBAC — Control de Acceso Basado en Capacidades

- Combina **roles** (funciones) con **atributos** (capacidades).
- El acceso solo es posible si el usuario recibe del proveedor del recurso un **token de autorización** (*capability*) que demuestra su capacidad para realizar determinadas acciones.
- **Principal desventaja:** hay que mantener y gestionar todos los certificados de autorización.

Risk-Based Access Control

Diseñado para escenarios heterogéneos donde no es posible predecir el número de usuarios y recursos.

El acceso depende del **riesgo evaluado en tiempo real**:

$$\text{Risk} = V \times P$$

Donde: - V = valor de la información (criticidad/confidencialidad/integridad del recurso). - P = probabilidad del acceso (determinada por el análisis de escenarios amenazantes y políticas de seguridad).

OrBAC — Control de Acceso Basado en la Organización

Extiende y mejora RBAC, añadiendo tres dimensiones:

Dimensión	Descripción
Sujetos	Entidades con roles predefinidos y permisos específicos
Actividades	Conjunto de acciones bajo una misma política de seguridad
Vistas	Relación con los objetos y su acceso según políticas organizativas

Depende de: (1) las políticas de seguridad de la organización, y (2) el nivel de confianza de cada entidad.

2.9 Comparativa Global de Modelos de Control de Acceso

Modelo	Base del control	Flexibilidad	Escalabilidad	Uso típico
DAC	Identidad + reglas del propietario	Alta	Baja (muchos-a-muchos)	S.O., BD
MAC	Etiquetas de clasificación	Baja	Alta	Militar, gobierno
RBAC	Roles organizativos	Alta	Alta	Empresas, ERPs, BBDD
ABAC	Atributos del sujeto/objeto	Muy alta	Media	IoT, cloud, acceso anónimo
Cap-BAC	Tokens de capacidad	Media	Media	IoT, servicios distribuidos

---

Quinta Parte: Protocolos Criptográficos Avanzados

---

1. Introducción a los Protocolos Criptográficos

¿Qué es un protocolo criptográfico?

Un **protocolo criptográfico** es un algoritmo distribuido definido para alcanzar un objetivo específico de seguridad. Consta de una **secuencia precisa de pasos** que especifica las acciones a llevar a cabo por dos o más entidades.
Las **primitivas criptográficas** que los sustentan son: algoritmos de cifrado, firmas digitales, funciones hash, MACs, generadores pseudoaleatorios, etc.

Ejemplos de protocolos criptográficos avanzados (más allá del intercambio de claves y autenticación de entidades):

- División y compartición de secretos
- *Timestamping* (sellado de tiempo)
- *Bit-commitment* (compromiso de bit)
- Lanzamiento de moneda
- Póker mental
- Demostraciones de conocimiento cero (*Zero-Knowledge*)
- Canal subliminal

---

2. Protocolo de División de Secretos

2.1 Concepto

El protocolo de división de secretos se usa cuando hay que **dividir un mensaje** M **en** n **trozos** de forma que:

- Cada trozo por sí mismo no tiene valor.
- Cuando se combinan todos, se recupera el mensaje original.

El esquema más simple divide un mensaje entre **dos personas** y requiere la intervención de una **Tercera Parte Confiable (TTP)**, habitualmente denominada Trent.

2.2 Ejemplo con 2 personas (XOR)

División (realizada por Trent):

1. Trent genera una cadena aleatoria de bits R de la misma longitud que M.
2. Trent computa $S = M \oplus R$.
3. Trent entrega R a Alice y S a Bob.

Reconstrucción:

4. Alice y Bob computan: $R \oplus S = M$

> **i** Analogía criptográfica.
>
> En esencia, Trent está cifrando el mensaje con un *one-time pad*: entrega el texto cifrado a una persona y el pad a otra. El sistema es perfectamente seguro si ninguno de los dos coopera con el adversario.

2.3 Extensión a 4 personas

1. Trent genera tres cadenas aleatorias R, S, T de la misma longitud que M.
2. Computa: $M \oplus R \oplus S \oplus T = U$
3. Distribuye: $R \rightarrow$ Alice, $S \rightarrow$ Bob, $T \rightarrow$ Carol, $U \rightarrow$ Dave.

Reconstrucción:

$$R \oplus S \oplus T \oplus U = M$$

⚠ Limitación crítica.

Si **una sola parte pierde su trozo**, el mensaje **no puede recuperarse**. Todos los trozos son necesarios sin excepción. Esto lo diferencia del protocolo de *compartición* de secretos.

---

3. Protocolo de Compartición de Secretos (Secret Sharing)

¿Qué es el Secret Sharing?

El protocolo de **compartición de secretos** se basa en el concepto de **esquema umbral** (k, n): se divide un mensaje M en n trozos (llamados **sombras**), de forma que **cualquier subconjunto de k sombras** es suficiente para reconstruir el mensaje original.

- **¿Para qué sirve?** Compartir secretos críticos (claves criptográficas, contraseñas maestras) con tolerancia a pérdidas.
- **¿Cuándo se usa?** Custodias de claves, sistemas de recuperación ante desastres, acceso a bóvedas.

Ventaja clave: el conocimiento de $k-1$ sombras **no revela ninguna información** sobre el secreto.

3.1 Fundamento Matemático: Interpolación Polinómica

El esquema umbral (k, n) se basa en la propiedad de que **dados k puntos de un polinomio de grado $k-1$, hay uno y solo un polinomio que los interpola**:

$$q(x_i) = y_i, \quad \forall i$$

Se elige aleatoriamente un polinomio de grado $k-1$:

$$q(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_{k-1} x^{k-1}$$

Donde $a_0 = D$ (el secreto a compartir), y los demás coeficientes se eligen aleatoriamente.

Las sombras son los valores evaluados en el polinomio:

$$D_1 = q(1), \quad D_2 = q(2), \quad \ldots, \quad D_n = q(n)$$

Con k sombras cualesquiera se puede reconstruir el polinomio por interpolación y obtener $D = q(0) = a_0$.

3.2 Ejemplo — Esquema Umbral (3, 5)

Datos del problema: - Secreto: $D = 11$ - Número total de sombras: $n = 5$ - Umbral de reconstrucción: $k = 3$

Construcción del polinomio ($k - 1 = 2 \rightarrow$ grado 2):

$$q(x) = a_0 + a_1 x + a_2 x^2$$

Con $a_0 = 11$ (el secreto), $a_1 = 2$, $a_2 = 1$ (aleatorios):

$$q(x) = 11 + 2x + x^2$$

Distribución de sombras:

Usuario	Evaluación	Sombra
Alice	$q(1)$	**14**
Bob	$q(2)$	**19**
Carol	$q(3)$	**26**
Dave	$q(4)$	**35**
Eve	$q(5)$	**46**

Ningún usuario posee el secreto original (11), pero cualquier trío puede cooperar para recuperarlo.

Reconstrucción con Alice, Bob y Dave:

$$q(1) = 14 = a_0 + a_1 \cdot 1 + a_2 \cdot 1^2$$
$$q(2) = 19 = a_0 + a_1 \cdot 2 + a_2 \cdot 2^2$$
$$q(4) = 35 = a_0 + a_1 \cdot 4 + a_2 \cdot 4^2$$

Sistema de 3 ecuaciones con 3 incógnitas $\rightarrow$ se obtiene $a_0 = D = 11$.

3.3 Comparativa: División vs. Compartición

Característica	División de Secretos	Compartición (k, n)
Trozos necesarios	**Todos** los n	**Cualquier** k de los n
Tolerancia a pérdidas	Ninguna	Sí (hasta $n - k$ pérdidas)

Característica	División de Secretos	Compartición (k, n)
Seguridad con $k-1$ trozos	No garantizada	Información cero revelada
Complejidad matemática	XOR simple	Interpolación polinómica

---

4. Protocolos de Bit-Commitment

¿Qué es el Bit-Commitment?

El **bit-commitment** es un protocolo que permite a Alice **comprometerse con un valor (bit)** sin revelarlo a Bob, garantizando que tampoco podrá modificarlo más tarde.

- **¿Para qué sirve?** Garantizar que una predicción o apuesta es irrevocable, incluso antes de revelarse.
- **¿Cuándo se usa?** Protocolos de negociación, apuestas digitales, licitaciones ciegas, bases de protocolos más complejos.

Propiedades requeridas: - **Ocultamiento (*hiding*):** Bob no puede conocer el bit comprometido. - **Vinculación (*binding*):** Alice no puede cambiar el bit después de comprometerse.

Escenario: Alice quiere hacer una predicción sin revelarla hasta después del evento. Bob quiere asegurarse de que Alice no puede cambiar la predicción *a posteriori*.

4.1 Solución con Criptografía Simétrica

1. Bob genera aleatoriamente una cadena R de bits y la envía a Alice.
   Bob $\rightarrow$ Alice : R

2. Alice crea un mensaje con el bit b y la cadena aleatoria de Bob, lo cifra con una clave aleatoria K y envía el resultado a Bob.
   Alice $\rightarrow$ Bob : $E_K(R, b)$
   (Alice queda comprometida; Bob no puede descifrar sin K)

3. *(Cuando llega el momento de revelar)* Alice envía la clave a Bob.
   Alice $\rightarrow$ Bob : K

4. Bob descifra y verifica que su cadena aleatoria coincide.
   Bob : $D_K(E_K(R, b))$
   (La presencia de R en el mensaje impide que Alice haya enviado un mensaje distinto al original)

4.2 Solución con Funciones Hash

1. Alice genera dos cadenas aleatorias R_1 y R_2, y crea el mensaje (R_1, R_2, b).
2. Computa el hash del mensaje y lo envía a Bob junto a R_1.
   Alice $\rightarrow$ Bob : $H(R_1, R_2, b),\ R_1$
   (El hash previene que Bob pueda invertirlo para obtener b; R_2 garantiza que Alice no pueda encontrar colisiones fácilmente)
3. *(Cuando llega el momento de revelar)* Alice envía el mensaje completo.
   Alice $\rightarrow$ Bob : R_1, R_2, b
4. Bob computa el hash y compara.
   Bob : $H(R_1, R_2, b) \rightarrow$ verifica que coincide con el recibido en el paso 2.

---

5. Protocolos de Lanzamiento de Moneda

5.1 Concepto

Problema: Alice y Bob quieren decidir el resultado de un lanzamiento de moneda (cara o cruz) de forma justa, **sin estar físicamente presentes** y sin que ninguno pueda hacer trampa.

Propiedades requeridas del protocolo:

- Alice debe lanzar la moneda antes de que Bob se pronuncie.
- Alice no puede re-lanzar la moneda después del pronunciamiento de Bob.
- Bob no puede conocer el resultado antes de pronunciarse.

Soluciones posibles:

- Usando el protocolo de bit-commitment.
- Usando criptografía de clave pública.

5.2 Solución con Criptografía Asimétrica

Requisito técnico clave: el sistema de cifrado debe satisfacer la propiedad conmutativa:

$$D_{K1}(E_{K2}(E_{K1}(M))) = E_{K2}(M)$$

Protocolo:

1. Alice y Bob generan, cada uno, un par de claves pública/privada:

$$\text{Alice} : K_{Apu},\ K_{Apr} \qquad \text{Bob} : K_{Bpu},\ K_{Bpr}$$

2. Alice genera dos mensajes M_1 ("cara") y M_2 ("cruz"), con cadenas aleatorias para verificar autenticidad. Los cifra con su clave pública y los envía a Bob en orden aleatorio:

$$\text{Alice} \to \text{Bob} : E_{Apu}(M_1),\ E_{Apu}(M_2)$$

3. Bob, sin poder leer los mensajes, elige uno al azar, lo cifra con su clave pública y se lo devuelve a Alice:

$$\text{Bob} \to \text{Alice} : E_{Bpu}(E_{Apu}(M))$$

4. Alice, sin poder leer el mensaje de Bob, lo descifra con su clave privada (gracias a la propiedad conmutativa) y lo devuelve:

$$D_{Apr}(E_{Bpu}(E_{Apu}(M))) = E_{Bpu}(M)$$

$$\text{Alice} \to \text{Bob} : E_{Bpu}(M)$$

5. Bob descifra con su clave privada y obtiene el resultado del lanzamiento; envía M a Alice:

$$\text{Bob} : D_{Bpr}(E_{Bpu}(M)) = M \qquad \text{Bob} \to \text{Alice} : M$$

6. Alice lee el resultado y verifica la cadena aleatoria para confirmar autenticidad.

> **i** Refuerzo conceptual: ¿por qué esto es justo?
>
> - Ni Alice ni Bob conocen el resultado hasta el paso 5.
> - Alice no puede elegir qué mensaje descifra Bob (lo elige Bob en el paso 3 sobre mensajes que no puede leer).
> - La cadena aleatoria en los mensajes previene que Alice prepare mensajes trampa.
> - **No se necesita una tercera parte de confianza**.

---

6. Protocolo de Póker Mental

El protocolo de **póker mental** es una generalización del protocolo de lanzamiento de moneda, aplicado al reparto justo de cartas en una partida de póker **sin baraja física ni árbitro**.

Protocolo:

1. Alice y Bob generan sus pares de claves:

$$\text{Alice} : K_{Apu},\ K_{Apr} \qquad \text{Bob} : K_{Bpu},\ K_{Bpr}$$

2. Alice genera **52 mensajes** (uno por carta), con cadenas aleatorias. Los cifra con su clave pública y los envía a Bob en orden aleatorio:

$$\text{Alice} \rightarrow \text{Bob} : E_{Apu}(M_i), \quad i = 1 \dots 52$$

3. Bob elige aleatoriamente **5 mensajes** para su mano, los cifra con su clave pública y los devuelve a Alice:

$$E_{Bpu}(E_{Apu}(M_j^B)), \quad j = 1 \dots 5$$

4. Alice descifra con su clave privada y devuelve los mensajes a Bob:

$$D_{Apr}(E_{Bpu}(E_{Apu}(M_j^B))) = E_{Bpu}(M_j^B) \qquad \text{Alice} \rightarrow \text{Bob} : E_{Bpu}(M_j^B)$$

5. Bob descifra con su clave privada y obtiene su mano:

$$D_{Bpr}(E_{Bpu}(M_j^B)) = M_j^B$$

6. De los 47 mensajes restantes, Bob elige 5 para Alice y se los envía:

$$\text{Bob} \rightarrow \text{Alice} : E_{Apu}(M_k^A), \quad k = 1 \dots 5$$

7. Alice descifra con su clave privada y obtiene su mano:

$$D_{Apr}(E_{Apu}(M_k^A)) = M_k^A$$

> **i** Propiedad clave del protocolo.
>
> En ningún momento ninguno de los dos puede conocer las cartas del otro antes de que el protocolo lo permita, y ninguno puede manipular el reparto sin que el otro lo detecte mediante la verificación de las cadenas aleatorias.

Referencias Bibliográficas

Bibliografía básica:

- Schneier, B. *Applied Cryptography: Protocols, Algorithms, and Source Code in C*. Wiley, 1996 (2ª edición).
- Stallings, W. *Cryptography and Network Security: Principles and Practice*. Prentice Hall, 2010 (5ª edición).
- Stallings, W. y Brown, L. *Computer Security: Principles and Practice*. Prentice-Hall, 2011 (2ª edición).

Estándares y RFCs:

- RFC 5280 — *Internet X.509 PKI Certificate and CRL Profile*, 2008.
- RFC 6818 — *Updates to RFC 5280*, 2013.
- RFC 6960 — *X.509 OCSP*, 2013.
- RFC 2828 — *Internet Security Glossary*, 2000.
- ITU-T X.509 — *Public-key and attribute certificate frameworks*, 2012.
- FIPS PUB 140-2 — *Security Requirements for Cryptographic Modules* (NIST, 2001).

Otras referencias:

- Guía de Referencia del DNIe con NFC, 2015.

4. Seguridad y Privacidad en Aplicaciones Telemáticas

Primera Parte: Herramientas de Seguridad y Comunicación Segura

---

1. Red Team y Blue Team

El modelo Red Team / Blue Team es una metodología de seguridad ofensiva-defensiva que simula ataques reales para evaluar y mejorar las defensas de un sistema.

Modelo mental: flujo Red Team → Blue Team.

```
[Red Team] → Diseña y ejecuta ataques
      ↓
[Informe de vulnerabilidades]
      ↓
[Blue Team] → Analiza, mitiga y monitoriza
      ↓
[Sistema más robusto]
```

1.1 Red Team

El **Red Team** es un equipo especializado en actividades **ofensivas**. Atacan su propio sistema de forma controlada con el objetivo de:

- Probar la eficacia y robustez de las soluciones de seguridad frente a ataques reales.
- Analizar posibles fugas de información, conexiones o redirecciones remotas.
- Determinar posibles comportamientos y técnicas de futuros atacantes.

Metodología de trabajo:

1. Diseñan ataques específicos adaptados a la arquitectura del sistema objetivo.
2. Identifican herramientas de hacking apropiadas para explotar vulnerabilidades encontradas.
3. Ejecutan los ataques prediseñados de forma controlada.
4. Extraen debilidades y vulnerabilidades para reportarlas al equipo Blue.

1.2 Blue Team

El **Blue Team** es un equipo especializado en actividades **defensivas**. Tratan, evitan o mitigan vulnerabilidades con el objetivo de:

- Proporcionar herramientas, soluciones y estrategias defensivas.
- Analizar los informes detallados del Red Team para identificar y mitigar vulnerabilidades.
- Monitorizar las actividades del sistema y elaborar planes de actuación.

Metodología de trabajo:

1. Recopilan información del sistema y evalúan riesgos.
2. Identifican soluciones de seguridad existentes o diseñan nuevas para evitar riesgos.
3. Monitorizan constantemente la seguridad del sistema evaluando los eventos producidos.
4. Trabajan en la mejora continua de la seguridad del sistema.

> Refuerzo conceptual: Red Team vs Blue Team.
>
> - **Red Team** = el atacante controlado interno. Simula al adversario.
> - **Blue Team** = el defensor. Reacciona, monitoriza y endurece el sistema.
> - Ambos equipos colaboran: sin informe del Red Team, el Blue Team no sabe qué corregir.

---

2. Herramientas Red Team

2.1 Sistemas Operativos Especializados

Kali Linux

Distribución **Debian GNU/Linux** diseñada principalmente para auditoría y seguridad informática. Ofrece un conjunto relevante de herramientas de seguridad preinstaladas y actualizadas.

Parrot Security OS

Distribución **Debian GNU/Linux** diseñada específicamente para:

- Pruebas de penetración (*pentesting*).
- Evaluación y análisis de vulnerabilidades.
- Análisis forense de sistemas.
- Preservación del anonimato.
- Análisis y prueba de criptografía.

!(images/6064463a7ac28bd156ca279e607c28bca3dceebc8063db2af78d4182a28c0690.jpg)

2.2 Principales Herramientas en Linux

Herramienta	Función principal	Capa/Ámbito
Nmap / Zenmap	Escaneo de puertos, fingerprinting, detección de SO	Red
Wireshark	Captura y análisis de tráfico de red (sniffing)	Red / Transporte
Hydra	Ataque de fuerza bruta a credenciales de acceso	Aplicación
Scapy	Creación, modificación, envío y captura de paquetes	Red
Ettercap	Intercepción de comunicaciones mediante ARP spoofing (MitM)	Red / Enlace
Metasploit	Suite flexible para pruebas de penetración	Transversal
Burpsuite	Pruebas de seguridad en aplicaciones web	Aplicación
Aircrack-ng	Descifrado de claves WPA-PSK y WEP	Inalámbrico
John the Ripper	Derivación y crackeo de contraseñas	Aplicación
Maltego	Inteligencia de fuentes abiertas (OSINT) y análisis forense	OSINT
OWASP-ZAP	Pruebas de seguridad en aplicaciones web	Aplicación

2.3 Descripción Detallada de Herramientas Clave

Hydra — Fuerza Bruta

Definición.

Hydra es un cracker de inicio de sesión que lanza ataques de fuerza bruta usando un diccionario de posibles contraseñas contra múltiples servicios de red (SSH, FTP, HTTP, SMB, etc.).

!(images/d8d795da8755dce3a22787fe4ed44383494b28265e929ea6d215a5ca341a9214.jpg)

Nmap y Zenmap — Rastreo y Fingerprinting

Definición.

Nmap es una herramienta de escaneo de redes que realiza fingerprinting: recolecta información de un sistema (puertos abiertos, SO, servicios y versiones) para conocer su superficie de ataque. El fingerprinting suele dejar rastro en los logs del sistema objetivo.

- Compatible con Windows, Linux, macOS y Solaris.
- Análisis en línea de comandos o mediante interfaz gráfica (**Zenmap**).

Opciones principales de Nmap:

```
# Puertos abiertos y servicios (escaneo básico)
nmap IP/localhost

# Ping simple (sin escaneo de puertos)
nmap -sP IP/localhost

# Rango de puertos en modo agresivo
nmap -p T:1-65535 -T4 IP/localhost

# Puertos concretos
nmap -p T:X-Y,Z -T4 IP/localhost

# Puertos, servicios y versiones de los servicios
nmap -sV -T4 IP/localhost
```

Zenmap es la interfaz gráfica open-source que permite ejecutar estas instrucciones mediante una GUI.

Wireshark — Captura de Tráfico (Sniffing)

Definición.

Wireshark es un disector (analizador) de red multiplataforma que captura tráfico en vivo y permite su análisis fuera de línea. Es la herramienta de referencia para el análisis de protocolos de red.

Características principales:

- Compatible con Windows, Linux, macOS, Solaris, FreeBSD, NetBSD y otros.
- Soporte de descifrado para múltiples protocolos: IPsec, ISAKMP, Kerberos, SNMPv3, SSL/TLS, WEP, WPA/WPA2.
- Exportación en XML, PostScript, CSV o texto plano.

Scapy — Manipulación de Paquetes

Definición.

Scapy es una librería desarrollada en Python con su propio intérprete de línea de comandos. Permite crear, modificar, enviar y capturar paquetes de red de forma programática.

- Compatible con Linux, macOS y Windows.
- Ideal para pruebas de protocolo y construcción de exploits a nivel de red.

Ettercap — Ataques Man-in-the-Middle

Definición.

Ettercap es una herramienta que intercepta comunicaciones mediante **ARP spoofing** (envenenamiento ARP), creando ataques de tipo *Man-in-the-Middle* (MitM).

Funcionamiento:

- La máquina del atacante anuncia que su IP es la del router real, pero su dirección MAC es la del atacante.
- Todo el tráfico de red de la víctima es atraído hacia la máquina del atacante en lugar del router legítimo.
- El atacante puede capturar, analizar e incluso modificar el tráfico en tránsito.

```
[Víctima] → (cree hablar con el router) → [Ettercap / Atacante] → [Router real]
                                                ↑ intercepta todo el tráfico
```

MITRE ATT&CK

Definición.

MITRE ATT&CK es un repositorio de conocimiento ampliamente reconocido que documenta tácticas, técnicas y procedimientos (TTPs) utilizados por atacantes reales, clasificados por ámbito: entornos empresariales, sistemas móviles y sistemas de control industrial (ICS).

- Disponible públicamente en https://attack.mitre.org.
- Usado por equipos Red y Blue para modelar amenazas y planificar defensas.
- Permite mapear ataques a técnicas conocidas, facilitando la detección y respuesta.

---

3. Seguridad en Email: PGP y S/MIME

El correo electrónico es la aplicación más ampliamente utilizada en entornos distribuidos. Su crecimiento ha generado una mayor necesidad de integrar servicios de **autenticación** y **confidencialidad**.

Las dos soluciones de seguridad en email más extendidas son:

- **PGP** (Pretty Good Privacy)
- **S/MIME** (Secure/Multipurpose Internet Mail Extension)

Refuerzo conceptual: PGP vs S/MIME.

Aspecto	PGP	S/MIME
Modelo de confianza	Web of Trust (distribuida, entre usuarios)	Jerarquía de CAs (centralizada, PKI)
Certificados	Formato propio (no estándar X.509)	X.509v3
Uso típico	Personal, técnico, privado	Corporativo, comercial
Gestión de claves	Descentralizada (anillos de claves)	Centralizada (CAs y PKI)

Estándar	OpenPGP (RFC 4880)	RFC 5751

⚠ Confusión habitual.

PGP y S/MIME ofrecen servicios equivalentes (firma y cifrado de correo), pero su modelo de confianza es radicalmente distinto. PGP usa una *web of trust* donde los propios usuarios avalan las claves ajenas; S/MIME usa una jerarquía de Autoridades de Certificación como en HTTPS.

3.1 PGP (Pretty Good Privacy)

Definición.

PGP es una solución criptográfica diseñada por Phil Zimmerman en 1991. Integra algoritmos criptográficos ya existentes en una única aplicación independiente del sistema operativo, proporcionando servicios de autenticidad y confidencialidad para correo electrónico y almacenamiento de ficheros.

Algoritmos integrados: RSA, DSS, Diffie-Hellman (ElGamal), CAST-128, IDEA, 3DES, SHA-1.

Disponible para Windows, UNIX y Mac (versiones libres y comerciales). El IETF ha estandarizado el formato en la **RFC 3156** (MIME Security with OpenPGP).

Servicios de PGP

Función	Algoritmos	Descripción
Firma digital	DSS/SHA o RSA/SHA	Hash SHA-1 del mensaje, cifrado con clave privada del remitente
Cifrado del mensaje	CAST / IDEA / 3DES + DH o RSA	Cifrado con clave de sesión de un solo uso; la clave de sesión se cifra con la clave pública del destinatario
Compresión	ZIP	El mensaje puede comprimirse antes de cifrar
Compatibilidad email	Radix-64 (Base64)	Conversión a ASCII para transparencia en clientes de correo

Esquema de Transmisión PGP

```
EMISOR (A)
   ↓
1. Firma: H(Mensaje) → cifrado con PR_a → Firma digital
2. Cifrado: Mensaje comprimido → cifrado con K_s (sesión)
3. K_s → cifrado con PU_b (clave pública del destinatario)
4. Codificación Radix-64
   ↓
[Correo electrónico]
   ↓
RECEPTOR (B)
   ↓
1. Decodificación Radix-64
2. Descifrar K_s con PR_b → obtiene K_s
3. Descifrar mensaje con K_s
4. Verificar firma con PU_a
```

Notación utilizada en los diagramas:

- K_s: clave de sesión (simétrica, de un solo uso)
- PR_a/PU_a: clave privada/pública del usuario A
- H: función hash (SHA-1)
- Z: compresión ZIP
- $R64$: conversión a Radix-64

Gestión de Claves: Anillos de Claves

PGP proporciona, para cada usuario U, dos estructuras de datos:

- **Private-Key Ring:** almacena los pares `<clave pública, clave privada>` propios del usuario U. La clave privada se almacena cifrada: $E(H(P_i), PR_i)$.
- **Public-Key Ring:** almacena las claves públicas de los otros usuarios con los que U se comunica.

Private-Key Ring:

Timestamp	Key ID (PU_i mod 2^{64})	Public Key	Encrypted Private Key	User ID
T_i	PU_i mod 2^{64}	PU_i	$E(H(P_i), PR_i)$	User i

Public-Key Ring:

Timestamp	Key ID	Public Key	Owner Trust	User ID	Key Legitimacy	Signature(s)	Signature Trust
T_i	$PU_i \bmod 2^{64}$	PU_i	trust_flag$_i$	User i	trust_flag$_i$		

> **i** Refuerzo conceptual: certificados PGP.
>
> Cada fila del Public-Key Ring actúa como un certificado digital no estándar. **PGP no usa el formato X.509**, sino un formato propio. La legitimidad de una clave pública se determina a través de la **web of trust**: cuantas más firmas de usuarios de confianza tiene una clave, más legítima se considera.

OpenPGP y Aplicaciones

OpenPGP (RFC 4880) es el estándar abierto derivado del PGP original. Permite cifrar emails usando criptografía de clave pública con un formato interoperable.

Aplicaciones que soportan OpenPGP:

Plataforma	Cliente de correo	Plugin/Herramienta
Windows	Outlook	gpg4o, Gpg4win, p=p
Windows/Linux/Mac	Thunderbird	Enigmail
Mac	Apple Mail	GPGTools
Android	K-9 Mail	OpenKeychain
iOS	—	iPGMail
Linux	Evolution	Seahorse
Linux	Kmail	Kleopatra

GnuPG es la implementación de referencia del estándar OpenPGP. Permite:

- Gestionar y generar claves públicas y privadas.
- Visualizar y distribuir claves públicas (exportar/importar).
- Cifrar y descifrar documentos.
- Firmar y validar documentos.

3.2 S/MIME (Secure/Multipurpose Internet Mail Extension)

Definición.

S/MIME es una mejora de seguridad del formato MIME para correo electrónico (que a su vez mejora SMTP). Proporciona firma digital y cifrado de mensajes basándose en una infraestructura PKI con certificados X.509v3.

Documentos de referencia:

- RFC 5652: Cryptographic Message Syntax (CMS)
- RFC 5750: Certificate Handling (versión 3.2)
- RFC 5751: Message Specification (versión 3.2)

Modelo de Confianza S/MIME

S/MIME sigue un **modelo PKI híbrido** entre la jerarquía estricta de Autoridades de Certificación (CA) y el modelo en malla. Utiliza certificados de clave pública con formato **X.509v3**.

Refuerzo conceptual: jerarquía de CAs vs web of trust.

- En **S/MIME**, una CA (como DigiCert o Sectigo) certifica la identidad del usuario. Si confías en la CA, confías en el certificado.
- En **PGP**, cualquier usuario puede firmar la clave de otro. La confianza es transitiva y social.

Algoritmos Criptográficos S/MIME

Función	Requisito
Resumen del mensaje (firma)	MUST SHA-1; SHOULD MD5 (retrocompat.)
Cifrado del resumen (firma digital)	MUST DSS; SHOULD RSA
Cifrado de clave de sesión	SHOULD DH; MUST RSA (512–1024 bits)
Cifrado del mensaje	MUST Triple-DES; SHOULD AES; SHOULD RC2/40
Código de autenticación (MAC)	MUST HMAC-SHA-1

Importante: consolidación de estándares.

Aunque tanto PGP como S/MIME están en vías de estandarización, S/MIME tiende a consolidarse como el estándar para **uso comercial**, mientras que PGP permanece predominante para **uso personal y técnico**.

4. Conexión Remota Segura

4.1 Telnet y FTP — Protocolos Inseguros

⚠ Importante: comunicación en claro.

Tanto Telnet como FTP transmiten **toda la información en texto claro**, incluyendo credenciales (usuario y contraseña). No se recomienda su uso en redes no confiables.

Protocolo	Puerto	Función
Telnet	23 (TCP)	Acceso remoto a sistemas; el terminal local emula el terminal remoto
FTP	20/21 (TCP)	Transferencia de ficheros entre recursos remotos

Problema fundamental: La información transferida y las acciones remotas se realizan completamente en claro, exponiendo contraseñas, comandos y datos.

4.2 SSH: Secure Shell v2

💡 Definición.

SSH (Secure Shell) es una herramienta similar a Telnet que opera en el **puerto 22 (TCP)**. Permite acceso remoto seguro mediante el modelo cliente/servidor, **cifrando todas las transacciones** usando criptografía de clave pública.

Arquitectura de Capas de SSH

```
Cliente SSH

      Capa de Aplicación
           Autenticación del cliente:
           - Modo básico: usuario + contraseña
           - Modo avanzado: criptografía de clave pública (par de claves SSH)

      Capa de Transporte
           - Intercambio inicial de claves
           - Establecimiento de modos de cifrado y compresión
```

```
      Capa de Red
           - Conexión directa cliente-servidor
           - Modo túnel (cifrado simétrico negociado en la capa de transporte)

Servidor SSH
```

Protección contra Ataques

SSH mitiga o evita ataques específicos:

- **IP Spoofing:** suplantación de identidad por nodo remoto.
- **DNS Spoofing:** el atacante suplanta el nombre del servidor.

Clientes SSH

Dos clientes de referencia multiplataforma:

- **PuTTY** (Windows/Linux): cliente SSH gráfico muy extendido.
- **OpenSSH** (Linux/macOS/Windows): implementación open-source de SSH.

4.3 SFTP y FTPS — Transferencia Segura de Ficheros

⚠ Confusión habitual: SFTP FTPS.

SFTP y **FTPS** son dos protocolos completamente distintos que se confunden frecuentemente por sus nombres similares.

- **SFTP** = SSH File Transfer Protocol → funciona **dentro del túnel SSH** (puerto 22).
- **FTPS** = FTP Secure → FTP con soporte para **TLS/SSL** (puerto 990 o 21).

Son protocolos diferentes, con arquitecturas y mecanismos de cifrado distintos.

SFTP (SSH File Transfer Protocol)

SSH puede usarse también para transferencia de ficheros, conocido como **SFTP** (y **SCP** — Secure Copy).

Modos de autenticación en SFTP:

- **Modo básico:** usuario y contraseña convencionales.
- **Modo avanzado:** uso de claves públicas SSH previamente generadas. El cliente transmite su clave pública al servidor para autenticación; no se envía ninguna contraseña.

Todas las conexiones SFTP están **cifradas a nivel de red** mediante el túnel SSH.

FTPS (FTP Secure)

FTPS proporciona soporte adecuado para **TLS (Transport Layer Security)**.

Funcionamiento:

1. El cliente verifica que el certificado del servidor es correcto y de confianza.
2. Se negocia una clave de sesión (proceso similar al handshake TLS).
3. Las credenciales (usuario y contraseña) se cifran a lo largo de la conexión FTPS.

```
Cliente FTPS

      1. Verificación del certificado del servidor (X.509)
      2. Negociación de clave de sesión (TLS handshake)
      3. Transferencia cifrada de datos y credenciales

Servidor FTPS
```

Comparativa SSH / SFTP / FTPS vs Telnet / FTP

Característica	Telnet/FTP	SSH	SFTP	FTPS
Cifrado	No (texto claro)	Sí (simétrico + asimétrico)	Sí (tunel SSH)	Sí (TLS)
Autenticación	Usuario/contraseña (en claro)	Contraseña o clave pública	Contraseña o clave pública	Certificado X.509
Puerto	23 / 20-21	22	22	990 o 21
Uso recomendado	Obsoleto	Acceso remoto	Transferencia de ficheros	FTP con TLS

---

5. Herramientas de Cifrado

5.1 Herramientas Open-Source

TrueCrypt

Definición.

TrueCrypt es una herramienta de cifrado de discos duros disponible para Windows (XP/2000/2003) y Linux. Soporta los algoritmos: **AES-256, Blowfish, CAST5, Serpent, Triple DES y Twofish**. Permite también la ocultación de particiones mediante cifrado y aleatoriedad de la información.

DiskCryptor

Similar a TrueCrypt, pero con la capacidad adicional de **cifrar dispositivos de almacenamiento externo USB**. Incluye algoritmos: **AES, Twofish y Serpent**.

VeraCrypt

Definición.

VeraCrypt es el sucesor espiritual de TrueCrypt. La diferencia clave es que incluye un número específico de iteraciones para el proceso de cifrado, lo que aumenta la resistencia a ataques de fuerza bruta a costa de una mayor lentitud en operaciones de lectura/escritura en disco. Aplica: **AES, Twofish y Serpent**.

Herramientas de Esteganografía

Definición: Esteganografía.

La **Esteganografía** (del griego "cubierto" u "oculto" + "escritura") es la técnica de ocultar información dentro de otro medio (imagen, audio, vídeo) de forma que su existencia sea imperceptible. A diferencia del cifrado (que oculta el *contenido*), la esteganografía oculta la *existencia* del mensaje.

- **OpenStego:** aplica técnicas de esteganografía para ocultar información usando imágenes o archivos multimedia.
- **OpenPuff:** similar a OpenStego para Windows, con soporte para BMP, JPG, MP3, WAV y MP4.

5.2 Herramientas Propietarias

BitLocker

Definición.

BitLocker es una herramienta de cifrado de disco completo desarrollada por **Microsoft** para proteger los datos almacenados en discos duros o unidades flash USB. Disponible en varias ediciones de Windows, compatible con los sistemas de archivos NTFS y exFAT.

Característica destacada: hace uso de **hardware criptográfico** para el cifrado del disco mediante el **TPM (Trusted Platform Module)**, un chip de seguridad integrado en la placa base que almacena las claves de cifrado y valida la integridad del sistema en el arranque.

Herramienta	Plataforma	Algoritmos	Soporte USB	TPM
TrueCrypt	Windows / Linux	AES-256, Blowfish, Serpent, Twofish, 3DES, CAST5	No	No
DiskCryptor	Windows	AES, Twofish, Serpent	Sí	No
VeraCrypt	Windows / Linux / Mac	AES, Twofish, Serpent	Sí	No
BitLocker	Windows	AES-128/256	Sí	Sí

Segunda Parte: Seguridad en Pagos Electrónicos

1. Conceptos Generales

1.1 Introducción

En el ámbito del e-commerce se han desarrollado esquemas de pago electrónico que proporcionan en el mundo digital la misma heterogeneidad y funcionalidad que los sistemas de pago tradicionales. Los pagos se realizan a través de redes abiertas como Internet, pero la correspondencia entre los pagos electrónicos y la transferencia del valor real es realizada y garantizada por los bancos, a través de los **sistemas financieros de compensación**, que operan sobre redes cerradas de las instituciones bancarias.

Participantes en un Pago Electrónico

```
[Comprador] ↔ [Internet] ↔ [Vendedor]

                              [Pasarela de pago]

                           [Red bancaria privada]

                      [Banco emisor] ↔ [Banco adquirente]

                      [TTP / Árbitro de disputas] (opcional)
```

En general, los pagos electrónicos involucran a un comprador y a un vendedor, aunque pueden existir:

- **TTPs** (Trusted Third Parties): entidades de confianza que participan en la transacción.
- **Mediadores** para la resolución de disputas (en muchos protocolos, las disputas se resuelven fuera del sistema de pago).

1.2 Clasificación de los Sistemas de Pago

Por el momento en que el vendedor contacta con el banco

Tipo	Descripción
On-line	Antes de enviar el producto, el vendedor verifica con la entidad financiera la validez del pago del comprador
Off-line	El vendedor acepta el pago y envía el producto sin contactar con el banco durante la compra-venta; deposita el dinero para verificación posteriormente

Por el momento en que se retira el dinero de la cuenta del comprador

Tipo	Analogía	Descripción
Pre-pago	Monedero electrónico, tarjeta telefónica, papel moneda	La cuenta del comprador se decrementa **antes** de la compra
Pago instantáneo	Tarjeta de débito	El cargo se realiza **en el momento exacto** de la compra

Tipo	Analogía	Descripción
Post-pago	Tarjeta de crédito	El banco garantiza el pago al vendedor; el comprador ve decrementada su cuenta **tiempo después**

⚠ Confusión habitual: pre-pago vs pago instantáneo.

El **pre-pago** implica cargar fondos en un monedero o tarjeta **con antelación**, antes de saber qué se va a comprar. El **pago instantáneo** (débito) se produce en el momento de la compra, pero sin reserva previa de fondos.

Por la cantidad implicada en la transacción

Categoría	Umbral
Micropagos	< 1 €
Pagos	Entre 1 € y 10 €
Macropagos	> 10 €

ℹ Refuerzo conceptual: micropagos.

Los pagos inferiores a 10 € presentan el problema del **coste de implementación**: no tendría sentido utilizar un sistema de pago cuyo coste económico sea del orden de magnitud del importe de la transacción. Esto motiva el uso de protocolos criptográficos más ligeros (hash, criptografía simétrica) para micropagos.

1.3 Problemas de Seguridad y Soluciones

Los sistemas de pago electrónico están expuestos a:

- Ataques de escucha (eavesdropping).
- Suplantación de identidad (del cliente o del comerciante).
- Generación de datos falsos.
- Modificación de datos en tránsito.

Para mitigarlos, los protocolos suelen implementar: primitivas criptográficas, mecanismos de autenticación y autorización de usuarios, firmas digitales y certificados digitales.

1.4 Breve Historia: El Papel de SSL

Inicialmente se aplicaba **SSL (Secure Sockets Layer)** como medida de protección de los canales de comunicación en pagos electrónicos.

SSL fue creado por Netscape en 1994. Proporcionaba criptografía híbrida:

- **Asimétrica:** autenticación mediante certificados X.509 v3 y negociación de claves de sesión (RSA o Diffie-Hellman).
- **Simétrica:** cifrado de punto a punto (DES, 3DES, RC2, RC4, IDEA).
- **Integridad:** mediante MAC (MD5 o SHA-1).

> ! Importante: SSL está obsoleto.
>
> La última versión SSLv3 está obsoleta. Fue reemplazado por TLS. **No debe usarse SSL en ningún sistema moderno.**

Sin embargo, SSL presentaba problemas para pagos electrónicos:

- Solo protege transacciones entre **dos puntos**, mientras que una transacción con tarjeta involucra al menos un banco.
- **No protege al comprador frente al vendedor**: el vendedor obtiene datos de la tarjeta que puede usar ilícitamente.
- No hay mecanismos de autenticación de tarjetas ni de facturación/recibos.

Esto motivó el desarrollo de protocolos de pago más específicos.

---

2. Protocolo SET (Secure Electronic Transaction)

> Definición.
>
> **SET (Secure Electronic Transaction)** es un protocolo desarrollado en 1996 por VISA y Mastercard (con American Express, IBM, Microsoft, Verisign, RSA, Netscape y GTE) para proporcionar seguridad a las transacciones electrónicas basadas en tarjetas de crédito, reducir el fraude mercantil y garantizar el pago a través de redes abiertas.
> **No es un sistema de pago en sí mismo**, sino un conjunto de protocolos de seguridad y formatos estándar que permiten usar de forma segura la infraestructura ya existente de tarjetas de crédito.

2.1 Servicios de Seguridad que Proporciona

- **Confidencialidad** en las comunicaciones entre todas las entidades de la transacción.

- **Autenticación** mediante certificados digitales X.509 (obligatorio para cliente, vendedor y pasarela de pago).
- **Privacidad:** la información solo está disponible para cada entidad cuando y donde es necesario.
- **Integridad** mediante firmas digitales.
- **No repudio**, reduciendo disputas.
- **Autorización de pago** y confirmación de la transacción.

2.2 Participantes en SET

```
[Comprador / Cardholder]

                    → [Vendedor / Merchant]

                              [Pasarela de Pago / Payment Gateway]

               → [Banco Emisor] ↔ [Banco Adquirente]

                         [CA / PKI SET]
```

2.3 Pasos del Protocolo SET

Paso	Acción
1. Petición del producto	El comprador selecciona el producto y solicita el pedido al vendedor
2. Inicialización	El vendedor envía sus certificados digitales al comprador
3. Información del pedido e instrucciones de pago	El comprador envía la descripción de la compra (OI) y las instrucciones de pago (PI) usando la firma dual
4. Petición de autorización	El vendedor contacta con la pasarela de pago; la pasarela contacta con el banco emisor
5. Aprobación de autorización	El banco emisor autoriza (o deniega) el pago; la pasarela responde al vendedor
6. Finalización	El vendedor reclama la cantidad a la pasarela; petición de compensación hacia el banco del vendedor

2.4 Firma Dual (Innovación Técnica de SET)

Definición: Firma Dual.

La **firma dual** es una innovación técnica de SET cuyo propósito es **enlazar dos mensajes que han de ir a receptores distintos** sin que cada receptor conozca el contenido del mensaje del otro:

- La **información de pago (PI)** va al banco.
- La **información del pedido (OI)** va al comerciante.

Objetivo de privacidad:

- El comerciante recibe los detalles del pedido, pero **NO** el número de tarjeta del cliente.
- El banco recibe las instrucciones de pago, pero **NO** los detalles del pedido.
- Sin embargo, ambos documentos quedan **criptográficamente enlazados** para una posible resolución de disputas posterior.

Notación:

Símbolo	Significado
PI	Payment Information (información de pago)
OI	Order Information (información del pedido)
H	Función hash SHA-1
||	Concatenación
PIMD	PI message digest = H(PI)
OIMD	OI message digest = H(OI)
POMD	Payment Order message digest = H(PIMD || OIMD)
E	Cifrado RSA
PRc	Clave privada del cliente

Construcción de la Firma Dual:

```
H(PI) → PIMD
                  H(PIMD || OIMD) → POMD → E(PRc, POMD) → Firma Dual
H(OI) → OIMD
```

- **Verificación por el vendedor:** recibe OI, PIMD y la Firma Dual. Puede verificar que el PIMD es coherente sin conocer PI.
- **Verificación por el banco:** recibe PI, OIMD y la Firma Dual. Puede verificar que el OIMD es coherente sin conocer OI.

! ¡Hay NO-REPUDIO!

Gracias a la firma dual, el cliente no puede negar haber enviado tanto las instrucciones de pago como la información del pedido.

2.5 Ventajas y Desventajas de SET

Ventajas	Desventajas
Seguro y bien diseñado	Dependiente de algoritmos específicos (RSA, DES, SHA-1)
Garantiza autenticación, confidencialidad, integridad y no-repudio	Gestión compleja de certificados digitales
Garantiza privacidad (vendedor no accede a datos de tarjeta; banco no accede a detalles del pedido)	Fuerte esfuerzo de implantación (especialmente para el vendedor)
Reduce disputas gracias al no-repudio	No adaptado a micropagos

---

3. Protocolo CyberCash

Definición.

CyberCash es un protocolo de pago electrónico basado en el uso de una **pasarela propia** que gestiona los pagos electrónicos. Permite el uso de cualquier tipo de tarjeta e integra el software de cliente (*CyberWallet*) con la red financiera del banco del comprador.

3.1 Pasos del Protocolo CyberCash

Paso	Acción
1	Orden de compra (descripción del producto)
2	Petición de pago (precio)
3	Orden de pago (firmada por el CyberWallet)
4	Redirección de la orden de pago
5	Verificación del pedido y petición de autorización
6	Petición de autorización al banco emisor
7	Respuesta de autorización (banco adquirente y pasarela)
8	Envío de facturas cifradas (cliente y comerciante)

Paso	Acción
9	Redirección de la factura del cliente

3.2 Problemas de CyberCash

⚠ Importante: problema de privacidad en CyberCash.

Parte de la información del cliente es conocida por la pasarela "privada", lo que permite **analizar los hábitos del cliente** — problema de privacidad que no existía en SET.

- Hace uso de **DES y RSA (1024 bits)**.
- Tras caer en bancarrota, **Verisign** adquirió los derechos. Posteriormente, **PayPal** compró la solución a Verisign.

---

4. Protocolo iKP (i-Key Protocol)

💡 Definición.

iKP (donde i = 1, 2 o 3) es una **familia de protocolos de pago** basados en criptografía de clave pública, desarrollada por IBM. La implantación es gradual para conseguir un pago seguro multiparte completo, requiriendo una infraestructura de certificación avanzada.

Los tres protocolos (1KP, 2KP, 3KP) se diferencian en **el número de entidades que poseen certificado digital**:

Protocolo	Entidades con certificado	Nivel de seguridad
1KP	Solo el banco (adquirente)	Básico
2KP	Banco + Vendedor	Intermedio
3KP	Banco + Vendedor + Cliente	Completo

4.1 Elementos Intercambiados en iKP

Item	Descripción
CAN	Número de cuenta del cliente (ej. número de tarjeta)
ID_M	Identificador del comerciante
TID_M	ID de transacción (identificador único)

Item	Descripción
DESC	Descripción de los bienes; incluye nombre del titular y BIN bancario
$SALT_C$	Número aleatorio del cliente para aleatorizar DESC (privacidad del enlace M→A)
$NONCE_M$	Número aleatorio del vendedor para proteger frente a replay
DATE	Fecha/hora actual del vendedor
PIN	PIN del cliente (opcional en 1KP)
CID	Pseudo-ID del cliente: $CID = H(R_C, CAN)$
Common	Información en común: PRICE, ID_M, TID_M, DATE, $NONCE_M$, CID, $H(DESC, SALT_C)$
SLIP	Instrucciones de pago: PRICE, $H(Common)$, CAN, R_C
EncSlip	SLIP cifrado con la clave pública del banco: $PK_A(SLIP)$

No es necesario memorizar todos los campos; sirven de referencia para comprender la estructura del protocolo.

4.2 Protocolo 1KP

Solo el banco necesita poseer (y distribuir) su certificado digital $CERT_A$.

Actor	Información inicial
Cliente	DESC, CAN, PK_{CA}, [PIN], $CERT_A$
Vendedor	DESC, PK_{CA}, $CERT_A$
Banco (adquirente)	SK_A, $CERT_A$

Desventajas de 1KP:

- El cliente se autentica solo con número de tarjeta y PIN opcional, no con firma digital.
- El vendedor **no se autentica** ante el cliente ni ante el banco.
- Ni el vendedor ni el cliente proporcionan evidencias de intervención en la transacción.

4.3 Protocolo 2KP

Además del banco, cada vendedor necesita un par `<clave pública, clave privada>` y distribuir su certificado $CERT_M$ al cliente y al banco.

Actor	Información inicial
Cliente	DESC, CAN, PK_{CA}, $CERT_A$
Vendedor	DESC, PK_{CA}, $CERT_A$, SK_M, $CERT_M$
Banco (adquirente)	PK_{CA}, SK_A, $CERT_A$

Con 2KP, el vendedor puede firmar mensajes y queda autenticado ante el banco.

---

5. Micropagos y Protocolo Millicent

5.1 El Problema de los Micropagos

En algunos escenarios hay que transferir cantidades muy pequeñas. El objetivo es minimizar el tráfico y los recursos utilizados para que los costes de realizar el pago sean mínimos en comparación con el pago en sí mismo.

Soluciones aplicadas:

- Servicios de prepago.
- Autorizaciones off-line.
- Agrupación de facturas para micropago por lotes.
- Soluciones online usando funciones hash (la criptografía de clave pública es demasiado costosa para micropagos).

⚠ Importante: hash y no-repudio en micropagos.

El uso de funciones hash en micropagos conlleva la **imposibilidad de garantizar el no-repudio**, ya que no se generan firmas digitales verificables.

5.2 Protocolo Millicent

Definición.

Millicent es un protocolo para gestionar micropagos basado en **criptografía simétrica** que **NO** utiliza procesamiento online. Introduce la figura del *agente de negocios (broker)* y usa una forma de moneda electrónica denominada **scrip**.

Concepto de Scrip:

Los **scrips** son "cupones electrónicos" que representan dinero, con los que el comprador obtiene la mercancía del vendedor. No sería eficiente comprar scrips

para cada vendedor por separado; el broker vende lotes mixtos de scrips de distintos vendedores.

Participantes:

- **Cliente / Comprador**
- **Vendedor / Comerciante (A)**
- **Agente de negocios / Broker** (institución financiera)

Flujo del protocolo Millicent:

```
1. Compra-Venta de scrips de A (broker  vendedor A)
2. Cliente compra scrips de agente (MACROPAGO)
3. Agente envía scrips al cliente
4. Cliente compra scrips de A (MICROPAGO mediante scrips de agente)
5. Agente envía scrips de A al cliente
6. Cliente envía petición + MICROPAGO (scrips de A) al vendedor
7. Vendedor envía el producto al cliente
```

Modelo de anonimato en Millicent:

- El **agente** conoce la identidad del comprador y su número de tarjeta, pero **nunca sabe qué producto compra**.
- El **vendedor** sabe lo que el cliente compra, pero **desconoce su identidad**.

Este modelo multiparte proporciona un cierto grado de anonimato por parte del comprador.

Otros sistemas de micropago: Subscrip, Kleline, Flattr, M-coin.

Tercera Parte: Privacidad de los Usuarios en Aplicaciones

---

1. Conceptos Generales de Privacidad

1.1 Definición de Privacidad

Definición: Privacidad.

La **privacidad** puede definirse como el derecho de los individuos y entidades de proteger, salvaguardar y controlar el acceso, almacenamiento, distribución y uso de información sobre su "propia persona". La información a proteger depende de lo que el usuario considere privado: identidad, localización, preferencias, rutinas, etc.

1.2 Privacidad Confidencialidad

> ! Importante: distinción fundamental.
>
> **Confidencialidad** y **privacidad** son conceptos distintos que se confunden frecuentemente:
> - **Confidencialidad** → relativa a los **datos**: mantener datos en secreto.
> - **Privacidad** → relativa a las **personas**: mantener la integridad de la persona.
>
> El cifrado de un mensaje garantiza confidencialidad pero no necesariamente privacidad: las cabeceras IP (origen, destino) siguen siendo visibles.

1.3 Formas de Violar la Privacidad

Rastreo de Actividad en la Red

Todo lo que se sube a la red es público y se pierde el control sobre esa información. Las redes sociales facilitan la adquisición de información personal que puede usarse para: despidos, robos, discriminación, incriminación, etc.

Análisis de Tráfico

El análisis de tráfico permite a observadores externos obtener información a través de las comunicaciones de un usuario:

- Si la comunicación está en claro: el atacante accede directamente a la carga útil.
- Si la comunicación está cifrada: el cifrado solo protege los datos, **no las cabeceras** (dirección IP origen y destino). Además, se puede estimar: tamaño de los paquetes, número de paquetes, frecuencia y tiempos de envío.

```
[Comunicación cifrada]
    → Dirección IP origen: VISIBLE
    → Dirección IP destino: VISIBLE
    → Tamaño de paquetes: ESTIMABLE
    → Frecuencia y tiempos: ESTIMABLE
    → Contenido del mensaje: CIFRADO
```

1.4 Enfoques para Proteger la Privacidad

Enfoque	Descripción
Legislativo	Creación de leyes que impongan sanciones y limiten prácticas abusivas (ej. GDPR en Europa)
Tecnológico	Mecanismos de preservación de la privacidad basados en algoritmos criptográficos, probabilísticos y de aleatorización

---

1.5 Propiedades de la Privacidad

- **No-vinculación (unlinkability):** incapacidad de un atacante para relacionar dos mensajes o entidades observadas.
- **No-observabilidad (unobservability):** imposibilidad de distinguir la presencia de mensajes o la identidad de las entidades.

1.6 Tipos de Anonimato

El **anonimato** se define como "el estado de no ser identificable entre un grupo de sujetos". Las técnicas de anonimato buscan hacer indistinguible a un individuo dentro de un conjunto de identidades con características similares.

Tipo	Descripción
Pseudónimos	Oculta la identidad mediante alias; el uso continuado puede permitir derivar la identidad real
Anonimato rastreable	Ofrece anonimato, pero en caso de necesidad se puede revelar la identidad del usuario (ej. banco/vendedor honesto)
Anonimato no rastreable	Garantiza que la identidad de los usuarios no puede revelarse bajo ninguna circunstancia
Anonimato no rastreable y no vinculante	Además de no poder revelar la identidad, el conjunto de operaciones del usuario no puede vincularse entre sí

1.7 Técnicas para Conseguir Privacidad

- **Esquemas avanzados de firma digital:** permiten que entidades autorizadas firmen sin revelar su identidad.
- **Protocolos criptográficos y de enrutado:** evitan que la dirección de red o el camino de los paquetes identifiquen a las partes comunicantes.
- **Técnicas de ofuscación:** mecanismos basados en generalización o supresión de información para limitar la precisión de lo que se revela.

---

2. Privacidad Basada en Esquemas Avanzados de Firma Digital

A partir del concepto básico de firma digital surgen esquemas más avanzados con objetivos más ambiciosos:

Esquema	Anonimato	Descripción breve
Firma ciega	Rastreable	El firmante firma sin conocer el contenido
Firma de grupo	Rastreable	Cualquier miembro firma en nombre del grupo; el administrador puede revelar al firmante
Firma de anillo	**No rastreable ni vinculante**	Similar a grupo pero sin administrador ni posibilidad de revelar al firmante

---

2.1 Firma Ciega (Blind Signature)

Definición.

Con la **firma ciega**, el mensaje M generado por Bob es **firmado por Alice sin que esta conozca su contenido**. La firma resultante puede verificarse públicamente con posterioridad como una firma digital normal. Proporciona **anonimato rastreable**.

Caso de uso típico: voto electrónico, donde la autoridad electoral firma el voto para validar que el votante tiene derecho a votar, pero sin conocer el contenido del voto.

Implementación con RSA:

Sean e y d las claves pública y privada de Alice, n el módulo RSA, y r un número aleatorio mod n (**blinding factor**).

1. Bob genera r aleatorio.
2. Bob calcula: $M^* = M \cdot r^e \mod n$ (mensaje "cegado").
3. Bob envía M^* a Alice.
4. Alice firma: $(M^*)^d \mod n = [M^d \cdot (r^e)^d \mod n] = [M^d \cdot r \mod n]$.
5. Bob elimina el factor cegador: $M^d \cdot r \cdot r^{-1} \mod n = M^d \mod n$.

El resultado $M^d \mod n$ es la firma legítima de Alice sobre M, pero Alice nunca vio M.

```
BOB                                    ALICE

   r (random)  → M* = M·r  mod n

          M*          →
                                          (M*)  = M ·r mod n  (firma ciega)
   ←        (M*)

   Elimina r: M ·r·r ¹ = M  mod n
     → Firma válida de Alice sobre M
```

2.2 Firma de Grupo

> Definición.
>
> Los **esquemas de firma de grupo** permiten que cualquier miembro de un grupo firme en nombre del grupo de forma anónima. Un **administrador del grupo** puede, en caso de disputa, revelar la identidad del firmante. Proporciona **anonimato rastreable**.

Ejemplo: un empleado de un banco firma documentos en nombre del banco sin revelar cuál empleado firmó.

Participantes:

- **Administrador del grupo:** controla la membresía, emite la clave de firma del grupo y puede "abrir" cualquier firma.
- **Miembro del grupo:** firma mensajes de forma anónima en nombre del grupo.
- **Verificador:** receptor del documento firmado; puede verificar que la firma pertenece al grupo.

Propiedades requeridas (anonimato rastreable):

- Solo los miembros del grupo pueden firmar correctamente (infalsificable).

- Nadie excepto el administrador puede descubrir qué miembro firmó (anonimato).
- Nadie excepto el administrador puede saber si dos firmas provienen del mismo miembro (no-vinculación).
- Los miembros no pueden evitar que el administrador abra la firma.

Procedimientos de un esquema de firma de grupo:

Procedimiento	Descripción
Establecimiento	Protocolo entre administrador y miembros. Genera: clave pública del grupo Y, claves privadas individuales X_i, clave secreta de administración Z
Firma	$S = E_{X_i}(M)$: el miembro i firma el mensaje M con su clave privada
Verificación	$E_Y(S) == M$: cualquier usuario verifica con la clave pública del grupo
Apertura	El administrador identifica al firmante usando la clave secreta Z; devuelve la identidad y una prueba de este hecho

Ejemplo básico de diseño (Ejemplo 1):

- El administrador proporciona a cada miembro i una lista disjunta de claves privadas: $L_i = \{X_{i,1}, \dots, X_{i,n}\}$.
- Se publican en un directorio público, en orden aleatorio, todas las claves públicas correspondientes (clave pública del grupo Y).
- Cada miembro firma un mensaje con **una sola clave privada de su lista** (para evitar vinculación).
- El receptor verifica con la clave pública del directorio.
- El administrador, conociendo todas las claves privadas, puede identificar al firmante.

Ejemplo 2 — basado en RSA:

Sea e el exponente público del grupo (clave pública Y), $C_p = (p_1, p_2, \dots, p_L)$ primos relativos a e, y $n = p_1 \cdot p_2 \cdots p_L$ el módulo compuesto.

Se generan módulos individuales para cada miembro: $n_i = p_{i1} \cdot p_{i2}$

Se calculan exponentes privados: $d_i = e^{-1} \mod \theta(n_i)$

- **Firma del miembro k:** $S = M^{d_k} \bmod n_k$
- **Verificación:** $M' = S^e \bmod n$
- **Apertura (administrador):** prueba firma con las claves privadas de cada miembro hasta encontrar la coincidencia.

2.3 Firma de Anillo

> Definición.
>
> Las **firmas de anillo** son similares a las firmas de grupo, pero con **anonimato total e incondicional**: no existe administrador, ni procedimiento de establecimiento, ni mecanismo de revocación del anonimato. Proporciona **anonimato no rastreable ni vinculante**.

Características diferenciadoras:

- **No hay administrador de grupo** ni clave secreta de administración.
- **No hay procedimiento de establecimiento** formal entre los posibles firmantes.
- **No hay revocación del anonimato** bajo ninguna circunstancia.
- Cualquier usuario puede elegir un conjunto de posibles firmantes (incluyéndose él mismo) **sin la aprobación ni ayuda de esos otros miembros**.
- Computa la firma usando solo su propia clave privada y las claves públicas de los demás.
- Los otros miembros pueden **desconocer totalmente** que sus claves públicas están siendo utilizadas.

---

3. Privacidad Basada en Protocolos Criptográficos y de Enrutado

Los protocolos criptográficos y de enrutado tratan de proteger el tráfico frente a entidades que observan las comunicaciones. Las soluciones se clasifican en cuatro categorías:

Categoría	Arquitectura	Latencia
a) Proxy	Centralizada	Baja
b) Mezcladores (Mix Networks)	Centralizada	Alta
c) Enrutado de Cebolla (Onion Routing)	Centralizada	Muy alta
Tor (2ª generación de Onion Routing)	Centralizada	Media-baja
d) Crowds / Hordes	Distribuida	Media-baja

3.1 Proxy

Definición.

Un **servidor proxy** actúa como intermediario en la comunicación, aceptando conexiones de los clientes y reenviándolas, ocultando así la dirección IP del verdadero origen. El destino cree que el origen de la comunicación es el proxy.

Limitaciones:

- No protegen frente a atacantes externos que controlen parte de la red.
- El proxy puede ser "honesto pero curioso": conoce la identidad real del usuario y su destino.

3.2 Mix Networks (Mezcladores)

Definición.

Los **mezcladores (mixers)** son un tipo de proxy capaz de ocultar la relación entre los mensajes que recibe y los que envía. Son dispositivos de almacenamiento y reenvío.

Funcionamiento:

1. El usuario envía el mensaje cifrado al mixer.
2. El mixer lo almacena durante cierto tiempo para mezclarlo con otros mensajes recibidos.
3. Los mensajes salen del mixer en un **orden diferente** al de llegada.

El esquema funciona **solo si el número de mensajes almacenados temporalmente es suficientemente grande** (para que la mezcla sea efectiva).

Limitaciones:

- Los retrasos introducidos pueden ser grandes.
- El mezclador puede ser honesto o deshonesto, pero siempre curioso.

3.3 Onion Routing y Tor

Onion Routing

Definición.

El **Onion Routing (enrutado de cebolla)** es un mecanismo en el que el *onion proxy* crea un paquete cifrado en capas, que se irán "pelando" a medida que el paquete atraviese una cadena de *onion routers*. Cada nodo solo descifra su capa y solo conoce el nodo anterior y el siguiente.

```
[Alice] → Cifra con K_nodo3(Cifra con K_nodo2(Cifra con K_nodo1(Mensaje)))
                ↓
          [Nodo 1] → descifra con K_nodo1 → ve solo "enviar a Nodo 2"
                ↓
          [Nodo 2] → descifra con K_nodo2 → ve solo "enviar a Nodo 3"
                ↓
          [Nodo 3] → descifra con K_nodo3 → ve solo "enviar a Destino"
                ↓
          [Destino]
```

Analogía: como una cebolla con capas. Cada nodo "pela" una capa (descifra su nivel) pero no puede ver las capas internas.

Limitación de Onion Routing: vulnerable a ataques en un extremo (el atacante controla el nodo de entrada o de salida).

Tor — Segunda Generación de Onion Routing

Definición.

Tor (The Onion Router) es la segunda generación de Onion Routing. Evita algunas deficiencias del diseño original añadiendo control de congestión, comprobación de integridad y *forward secrecy*.

Características principales:

- Permite navegar a través de una ruta privada creada aleatoriamente entre distintos **repetidores (onion routers)** de la red Tor.
- Se negocian nuevos caminos frecuentemente y, tras su uso, **las claves se borran** (forward secrecy).
- Se aplican **guardianes de entrada** (el onion proxy) para reducir posibles ataques de correlación.
- Incluye **servidores ocultos** que solo son accesibles desde dentro de la red Tor.

Funcionamiento detallado:

1. **Elección de ruta:** los paquetes se envían a través de varios onion routers elegidos de forma aleatoria por un servidor central. Todos los nodos Tor

se eligen al azar; ningún nodo se usa dos veces en la misma ruta.

2. **Establecimiento del túnel:** el cliente se conecta a un nodo aleatorio a través de una conexión cifrada. Una vez conocido el camino, cada salto de la red es cifrado, **excepto la conexión del nodo de salida al destino final** (no cifrada).
3. **Cifrado asimétrico por capas:** Alice cifra el mensaje primero con la clave pública del último router onion, luego con la del penúltimo, y así sucesivamente. Cada nodo solo puede descifrar su capa.
4. **Rotación periódica:** cada 10 minutos se cambian los nodos de la conexión, escogiendo nuevos nodos, para evitar el análisis de tráfico pasivo.

⚠ Importante: Tor no garantiza privacidad total.

- Tor es **lento** por su arquitectura: múltiples saltos y uso de criptografía de clave pública.
- **Garantiza anonimato**, pero **no garantiza la privacidad de los datos** (el nodo de salida puede ver el contenido si el tráfico no está cifrado end-to-end).
- El anonimato en Tor no es incondicional: ataques de correlación entre nodo de entrada y salida pueden desanonimizar al usuario.

Formas de acceder a la red Tor:

- Usar el **Tor Browser** (navegador compatible, también disponible para móviles).
- Usar una extensión del navegador habitual.

3.4 Crowds y Hordes

Crowds

Los **Crowds** agrupan a los emisores creando una "multitud" en la que todos enrutan de manera aleatoria los paquetes recibidos de sus compañeros. Cada nodo solo conoce el destino y el nodo del que recibe el paquete.

- Los nodos comparten claves con sus vecinos para cifrar los mensajes.
- El camino seguido por el mensaje se almacena temporalmente en cada nodo para enrutar las respuestas.

Hordes

Hordes es un esquema muy similar a Crowds con una diferencia principal: la respuesta se envía mediante **broadcast** desde el router a todos los miembros del grupo donde se encuentra el originador del mensaje.

Aspecto	Crowds	Hordes
Enrutado de respuesta	Punto a punto (inverso del camino)	Broadcast al grupo
Rendimiento (tiempo)	Menor	**Mayor** (broadcast es más rápido)
Robustez	**Mayor**	Menor (el broadcast da indicios sobre la localización del usuario y sobrecarga el sistema)

3.5 Comparativa de Protocolos de Privacidad

Protocolo	Arquitectura	Latencia	Nivel de anonimato
Proxy	Centralizada	Baja	Bajo (el proxy conoce origen y destino)
Mezcladores	Centralizada	Alta	Medio (depende del volumen de mensajes)
Enrutado de cebolla	Centralizada	Muy alta	Alto
Tor	Centralizada	Media-baja	Alto (con limitaciones en nodo de salida)
Crowds / Hordes	Distribuida	Media-baja	Medio-alto

Referencias Bibliográficas

Bibliografía básica

- *Cryptography and Network Security: Principles and Practice* — William Stallings. Prentice Hall, 2010 (5ª edición).
- *Electronic Payment Systems for E-commerce* — Donal O'Mahony. Artech House, 2001 (2ª edición).
- *Blind Signatures for Untraceable Payments* — David Chaum.
- *Group Signatures* — David Chaum, Eugène van Heyst.

Referencias

- "The International PGP Home Page" — http://www.pgpi.org
- RFC 3156: MIME Security with OpenPGP
- RFC 5652: Cryptographic Message Syntax (CMS)
- RFC 5750: Secure/Multipurpose Internet Mail — Version 3.2 — Certificate Handling
- RFC 5751: Secure/Multipurpose Internet Mail Extensions — Version 3.2 — Message Specification
- SET Secure Electronic Transaction Specification (V1.0), Books 1, 2 and 3
- MITRE ATT&CK Framework — https://attack.mitre.org

5. Seguridad de la Información — Tema 5: Seguridad en Redes TCP/IP.

1. Introducción y motivación.

La expansión de la *World Wide Web* durante los años 90 y su adopción para realizar transacciones electrónicas planteó nuevos riesgos de seguridad: en el lado del cliente, en el lado del servidor, y sobre la propia información intercambiada entre ambos. Las primeras versiones de HTTP no contemplaban ningún mecanismo de protección, por lo que la información viajaba en claro y sin autenticar.

Para abordar el problema, el IETF formó a mediados de los 90 el grupo de trabajo **Web Transaction Security (WTS)**, con el objetivo de desarrollar los requisitos y especificaciones de los servicios de seguridad necesarios para las transacciones web.

1.1. Amenazas y consecuencias.

Las amenazas que afectan a las transacciones web pueden clasificarse según el servicio de seguridad comprometido:

Servicio	Amenaza típica	Consecuencia
Integridad	Modificación de datos de usuario, troyano en el navegador, modificación en tránsito	Pérdida de información, afectación del dispositivo, otros ataques

Servicio	Amenaza típica	Consecuencia
Confidencialidad	Escuchas en el canal (*eavesdropping*), robo de información, robo de configuración	Pérdida de información y privacidad
Disponibilidad	Interrupción de procesos en cliente o servidor (DoS)	Interrupción del servicio, inaccesibilidad
Autenticación	Suplantación de identidad, falsificación de datos	Confianza en datos falsos o entidades ilegítimas

! Importante.

La correspondencia entre tipo de ataque y servicio de seguridad necesario es directa: los ataques de *eavesdropping* exigen **confidencialidad**, los de *tampering* exigen **integridad** y los de *forgery* exigen **autenticación**. Toda la arquitectura de SSL/TLS se construye en torno a esta tríada.

1.2. Dos enfoques iniciales: SHTTP frente a SSL.

Frente al problema de la seguridad web, surgieron dos aproximaciones contemporáneas pero distintas:

- **SHTTP (Secure HyperText Transfer Protocol):** Solución del grupo WTS del IETF, situada en la capa de aplicación. Quedó especificada en los RFC 2084, 2659 y 2660 (categoría experimental).
- **SSL (Secure Sockets Layer):** Solución de Netscape, situada como una **subcapa entre aplicación y transporte**. Aprovecha que TCP es orientado a conexión y fiable, y proporciona seguridad de forma transparente al protocolo de aplicación.

El enfoque de Netscape se impuso: SSL acabó sustituyendo por completo a SHTTP como mecanismo estándar para proteger las comunicaciones en Internet.

Modelo mental.

SSL/TLS no es ni un protocolo de aplicación ni un protocolo de transporte: es una **subcapa intermedia** que se "intercala" sobre TCP y por debajo del protocolo de aplicación, lo que permite proteger cualquier aplicación basada en TCP sin modificarla.

1.3. Evolución histórica de SSL/TLS.

Año	Organización	Versión
Mediados de 1994	Netscape	SSL v1.0 (nunca pública)
Finales de 1994	Netscape	SSL v2.0
1995	Netscape	SSL v2.0 (actualizada)
1996	Netscape	SSL v3.0 (obsoleta)
1999	IETF	TLS v1.0 — SSL v3.1 — RFC 2246
2006	IETF	TLS v1.1 — SSL v3.2 — RFC 4346
2008	IETF	TLS v1.2 — SSL v3.3 — RFC 5246
2018	IETF	TLS v1.3 — SSL v3.4 — RFC 8446

Refuerzo conceptual: relación SSL TLS.

TLS es la estandarización por parte del IETF del protocolo SSL de Netscape. Como recoge el propio RFC 2246, las diferencias entre TLS 1.0 y SSL 3.0 no son drásticas, pero sí lo suficiente como para impedir interoperabilidad entre ambos. A efectos didácticos, **SSL y TLS son equivalentes** en su arquitectura general, y entender SSL es la base para entender TLS.

---

2. Servicios de seguridad y arquitectura de SSL.

2.1. Servicios proporcionados.

El protocolo SSL es un protocolo **cliente/servidor** que proporciona prevención contra:

- **Ataques de escucha (*eavesdropping*):** combatidos mediante **confidencialidad**.
- **Ataques de manipulación (*tampering*):** combatidos mediante **integridad**.
- **Ataques de falsificación (*forgery*):** combatidos mediante **autenticación**.

Concretamente, SSL garantiza:

- **Autenticación** de las entidades origen y destino, opcionalmente mediante certificados X.509 y MAC.
- **Confidencialidad** de la conexión.
- **Integridad** de la conexión.

> ❗ Importante.
>
> A pesar de emplear criptografía de clave pública, SSL **no proporciona el servicio de no repudio**, ni de origen ni de entrega. Si la aplicación requiere no repudio (p. ej. firma electrónica de documentos), debe gestionarse a nivel de aplicación.

2.2. Independencia del protocolo de aplicación.

Una ventaja fundamental de SSL es que es independiente del protocolo de aplicación: cualquier protocolo basado en TCP puede beneficiarse de los servicios de seguridad. Esto da lugar a múltiples variantes "sobre SSL/TLS":

Protocolo	Descripción	Puerto
https	HTTP sobre SSL/TLS	443
ldaps	LDAP sobre SSL/TLS	636
ftps-data	FTP Data sobre SSL/TLS	989
ftps	FTP Control sobre SSL/TLS	990
telnets	Telnet sobre SSL/TLS	992
imaps	IMAP4 sobre SSL/TLS	993
pop3s	POP3 sobre SSL/TLS	995

2.3. Criptografía híbrida.

SSL combina dos familias de algoritmos:

- **Criptografía de clave pública:** Para la **autenticación** de las entidades y el **establecimiento de la clave** de sesión. SSL v3.0 admitía RSA, Diffie-Hellman y Fortezza KEA (este último sólo hasta SSL v3.0).
- **Criptografía simétrica:** Para la **autenticación de los datos** (mediante MAC) y el **cifrado** del contenido aplicativo.

> ⚠ Idea clave.
>
> La criptografía híbrida permite combinar las ventajas de ambas familias: la clave pública resuelve el problema de establecer una clave secreta entre desconocidos, y la criptografía simétrica proporciona la velocidad necesaria para cifrar volúmenes grandes de datos.

2.4. Sesión y conexión SSL.

SSL distingue dos conceptos relacionados pero diferentes:

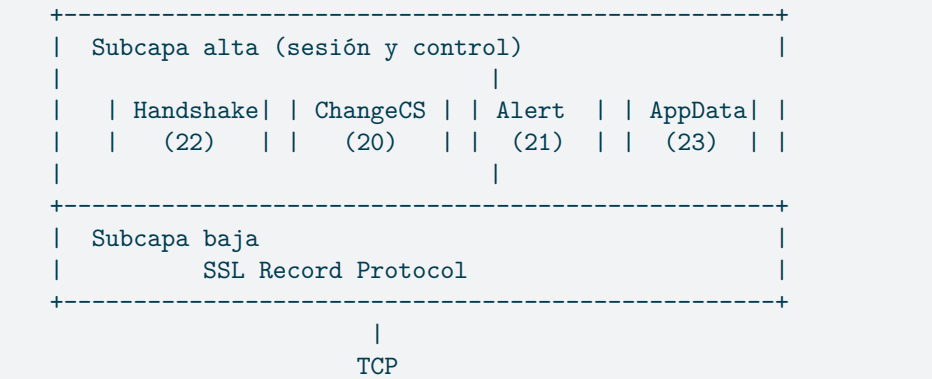
- **Sesión SSL:** Asociación entre cliente y servidor en la que se **negocian los parámetros de seguridad** (versión, *cipher suite*, claves maestras, etc.) que se aplicarán en las conexiones asociadas a esa sesión.
- **Conexión SSL:** Transmisión efectiva de datos entre cliente y servidor, protegida criptográficamente según lo negociado en la sesión.

Una misma sesión puede dar lugar a múltiples conexiones, lo que evita renegociar parámetros costosos cada vez que se abre un canal nuevo.

> Refuerzo conceptual: las dos fases de SSL.
>
> - **Fase I — Sesión SSL:** Negociación de parámetros de seguridad mediante el *handshake*.
> - **Fase II — Conexión SSL:** Comunicación segura (cifrada y autenticada) sobre los parámetros negociados.

2.5. Arquitectura en dos subcapas.

Para cubrir su doble misión (establecer la conexión segura y transmitir datos sobre ella), SSL se organiza en **dos subcapas** y varios subprotocolos:



- **Subcapa alta:** Contiene los subprotocolos que gestionan la sesión y el control:
  - **(22) SSL Handshake Protocol:** Permite a las partes autenticarse mutuamente, negociar un *cipher suite* y, opcionalmente, un método de compresión.
  - **(20) SSL Change Cipher Spec Protocol:** Activa el *cipher suite* negociado.
  - **(21) SSL Alert Protocol:** Intercambia mensajes de alerta sobre incidencias.
  - **(23) SSL Application Data Protocol:** Es el propio protocolo de aplicación (p. ej. HTTP) cuyos datos alimentan al SSL Record Protocol.

- **Subcapa baja:** Contiene el **SSL Record Protocol**, que fragmenta los datos procedentes de la subcapa alta y los procesa de forma individual.

---

3. SSL Record Protocol.

El **SSL Record Protocol** es la pieza de la subcapa baja responsable de transformar los datos de la subcapa alta en unidades transmisibles sobre TCP, denominadas **SSL records**.

3.1. Formato del SSL record.

Cada SSL record tiene la siguiente estructura de cabecera:

Tipo (1 byte)	Versión (2 bytes)	Longitud (2 bytes)	Fragmento (0 bytes)

- **Tipo:** Indica el protocolo de la subcapa alta del que procede el fragmento. Valores principales:
  - `20` → SSL Change Cipher Spec Protocol
  - `21` → SSL Alert Protocol
  - `22` → SSL Handshake Protocol
  - `23` → SSL Application Data Protocol
- **Versión:** Dos bytes que codifican versión mayor y menor (en SSLv3, valores `3` y `0` respectivamente).
- **Longitud:** Tamaño del fragmento en bytes.
- **Fragmento:** Carga útil ya procesada (comprimida, autenticada y cifrada).

3.2. Procesamiento del fragmento.

El Record Protocol aplica una secuencia ordenada de transformaciones sobre los datos:

```
[datos de subcapa alta]
        |
        v
   1. Fragmentar   → bloques de longitud  2^14 bytes (16 384)
        |
   2. Comprimir    → opcional
        |
   3. Añadir MAC   → usa client_write_MAC_secret / server_write_MAC_secret
        |
   4. Cifrar       → usa client_write_key / server_write_key
        |                 y client_write_IV / server_write_IV
```

```
5. Añadir cabecera SSL record
     |
     v
[segmento TCP]
```

En el destino, el proceso se ejecuta a la inversa: los datos recibidos son descifrados, verificados (MAC), descomprimidos y reensamblados antes de entregarlos a la capa de aplicación.

> **i** Refuerzo conceptual: dónde se usa cada clave.
>
> - El **MAC secret** garantiza la integridad y la autenticidad del fragmento.
> - La **write key** y el **IV** se usan para el cifrado simétrico.
> - Hay claves separadas para cliente y servidor, lo que evita ataques de reflexión y simplifica el análisis del flujo.

---

4. SSL Handshake Protocol.

El **SSL Handshake Protocol** es la parte más compleja de SSL y se ejecuta **antes de transmitir cualquier dato de aplicación**. Permite a cliente y servidor:

- **Autenticarse mutuamente.**
- **Negociar** un algoritmo de cifrado y una función MAC (el *cipher suite*).
- **Negociar las claves** que se usarán para proteger los datos en el SSL Record Protocol.

4.1. Formato de los mensajes de handshake.

Los mensajes de handshake viajan como fragmento de un SSL record con `Tipo = 22`. La estructura interna de cada mensaje es:

Tipo (1 byte)	Longitud (3 bytes)	Contenido (0 bytes)

- **Tipo:** Identifica uno de los diez posibles tipos de mensaje (ver tabla en §4.3).
- **Longitud:** Longitud del mensaje en bytes.
- **Contenido:** Parámetros específicos del mensaje.

4.2. Las cuatro fases del handshake.

El intercambio completo de mensajes se organiza en cuatro fases:

- **Fase 1 — Negociación de capacidades.** Cliente y servidor intercambian `ClientHello` y `ServerHello` para acordar versión del protocolo, ID de sesión, *cipher suite*, número aleatorio (*random*) y, opcionalmente, método de compresión.
- **Fase 2 — Autenticación del servidor y envío de parámetros.** El servidor puede enviar su certificado (opcional), los parámetros necesarios para la gestión de la clave secreta (`ServerKeyExchange`) y, si lo desea, solicitar un certificado al cliente (`CertificateRequest`). Termina con `ServerHelloDone`.
- **Fase 3 — Autenticación del cliente y aportación de la clave.** Si se le ha solicitado, el cliente envía su certificado. A continuación envía `ClientKeyExchange` con los parámetros de seguridad necesarios para derivar la clave de sesión. Si ha enviado certificado, añade `CertificateVerify` firmado para verificación de origen.
- **Fase 4 — Activación y cierre.** Ambas partes envían `ChangeCipherSpec` y `Finished`, activando el *cipher suite* negociado y comprobando que todo el handshake se ha realizado correctamente.

4.3. Tabla de mensajes del Handshake.

Mensaje	Tipo	Contenido principal
`Hello_request`	0	Vacío. Lo envía el servidor para solicitar renegociación.
`Client_hello`	1	Versión, `client_random`, ID de sesión, *cipher suites* soportados, métodos de compresión.
`Server_hello`	2	Versión, `server_random`, ID de sesión, *cipher suite* elegido, método de compresión.
`Certificate`	11	Cadena de certificados X.509.
`Server_key_exchange`	12	Parámetros para computar la clave de sesión (p. ej. parámetros DH) y firma del hash de `client_random + server_random + parámetros`.
`Certificate_request`	13	Tipos de certificado y autoridades aceptadas.
`Server_hello_done`	14	Vacío. Indica fin de Fase 2.

Mensaje	Tipo	Contenido principal
`Client_key_exchange`	16	`pre_master_secret` cifrado con RSA, o parámetros DH/Fortezza para computar el `master_key` en ambas partes.
`Certificate_verify`	15	Firma sobre `hash(master_key + hash(todos los mensajes intercambiados hasta el momento))`. Sólo si se envió certificado de cliente.
`Finished`	20	Cifrado del `hash(master_key + hash(hash(todos los mensajes hasta el momento) + ID_sender))`.

```
SSL record                 |  Mensaje de Handshake
                           |
Tipo Versión Long. Frag.   |  Tipo Long. Versión  Contenido
 22    3,0    Y    ...     |   1    Z    3,0      client_random,
                           |                       ID session,
                           |                       cipher suite1,
                           |                       cipher suite2, ...
```

```
SSL record                 |  Mensaje de Handshake
                           |
Tipo Versión Long. Frag.   |  Tipo Long.  Contenido
 22    3,0    Y    ...     |  16    Z     pre-shared master key
```

Esquema rápido del handshake (SSL/TLS 1.2).

```
Cliente                                        Servidor

       ClientHello

        ServerHello
           [Certificate]
           [ServerKeyExchange]
           [CertificateRequest]
            ServerHelloDone

       [Certificate]
          ClientKeyExchange
         [CertificateVerify]
          ChangeCipherSpec
          Finished

        ChangeCipherSpec
            Finished

         datos de aplicación
```

Los corchetes `[ ]` indican mensajes **opcionales**.

---

5. Establecimiento de la clave de sesión.

Una vez completada la negociación inicial, cliente y servidor deben derivar el material criptográfico necesario para proteger la conexión. Concretamente, hay que obtener:

1. La **clave de sesión** (para cifrado simétrico de los datos).
2. Una **clave para el MAC** (para integridad y autenticidad).
3. Un **IV** (vector de inicialización) para el modo de operación del cifrado en bloque.

5.1. Parámetros que intervienen en la derivación.

El material criptográfico se calcula a partir de:

- Una **semilla** (`pre_master_secret`, también llamada *pre-shared master key*).
- Dos **valores aleatorios** (*nonces*): `client_random` y `server_random`.

- Una **función pseudoaleatoria (PRF — *pseudorandom function*)** que combina los anteriores.

El flujo de derivación, simplificado, es:

```
pre_master_secret    client_random   server_random

                    PRF (función
                   pseudoaleatoria)

                     master_key

                       PRF + más
                       aleatoriedad

                   key_block →  [ client_write_MAC_secret,
                                  server_write_MAC_secret,
                                  client_write_key,
                                  server_write_key,
                                  client_write_IV,
                                  server_write_IV ]
```

> **!** Importante.
>
> La **PRF de SSL es distinta a la de TLS**. SSL usa una combinación de MD5 y SHA-1; TLS 1.0 y 1.1 también, pero TLS 1.2 y 1.3 emplean SHA-256. Es uno de los puntos clave en los que SSL 3.0 y TLS no son interoperables.

5.2. Intercambio de claves con RSA.

El esquema más sencillo de intercambio de claves usa RSA y se desarrolla en tres pasos:

- **Paso 1 — `ClientHello`.** El cliente genera `client_random` y propone *cipher suites*. Envía toda la información al servidor.

```
ClientHello = ( { TLS_RSA_WITH_AES_256_CBC_SHA, ... }, client_random, ... )
```

- **Paso 2 — `ServerHello` y certificado.** El servidor genera `server_random`, elige el *cipher suite* y entrega su certificado X.509

que contiene su clave pública RSA.

- **Paso 3 — `ClientKeyExchange`.** El cliente genera la semilla `pre_master_secret`, la cifra con la **clave pública RSA del servidor** y la envía:

```
ClientKeyExchange = E_pub_server( pre_master_secret )
```

A partir de ahí, cliente y servidor disponen de `client_random`, `server_random` y `pre_master_secret`, lo que les permite derivar la `master_key` y, de ella, las claves de sesión.

5.3. Intercambio de claves con Diffie-Hellman.

Antes de introducir DHE, conviene recordar el esquema básico de **Diffie-Hellman**:

```
Valores públicos: q (primo grande),  (raíz primitiva de q)

   Alice                                    Bob

   Xa < q  (clave privada)                  Xb < q  (clave privada)
   Ya =  ^Xa mod q  (clave pública)         Yb =  ^Xb mod q  (clave pública)
                  intercambian Ya, Yb
   KAB = (Yb)^Xa mod q                      KAB = (Ya)^Xb mod q

             KAB =  ^(Xa·Xb) mod q =  ^(Xb·Xa) mod q
```

Ambos llegan a la misma clave compartida sin haberla transmitido en ningún momento.

5.4. DHE — Diffie-Hellman efímero y Perfect Forward Secrecy.

Tanto SSL como TLS usan **DHE (Diffie-Hellman Efímero)** como mejora del esquema anterior:

- Cliente y servidor **generan valores secretos nuevos en cada negociación**, en lugar de reutilizar valores estáticos.
- Esta renegociación constante proporciona **Perfect Forward Secrecy (PFS)** en la creación del secreto compartido.
- PFS evita el riesgo de **MitM (Man-in-the-Middle)** retrospectivo.

> ⚠ Idea clave.
>
> Sin PFS, si la **clave privada RSA del servidor se filtra** (o un MitM compromete la negociación), un atacante puede derivar **todas las claves**

de sesión pasadas entre las entidades legítimas, descifrando comunicaciones antiguas. Con PFS, comprometer una clave privada de largo plazo **no compromete las sesiones anteriores**.

5.5. DHE-RSA: combinando autenticación e intercambio efímero.

El esquema más típico combina autenticación con RSA e intercambio de clave con DHE:

```
Cliente                                                    Servidor
      ClientHello (TLS_DHE_RSA_WITH_AES_256_CBC_SHA,
       client_random)

                                                          1. Genera server_random
                                                          2. Genera G, N (G raíz primitiva
                                                          3. Genera privada Xb < N
                                                          4. Calcula Yb = G^Xb mod N

       ServerHello (G, N, Yb,
         RSAsign(G, N, Yb, server_random) )
        [Certificate]
        ServerHelloDone

      1. Verifica certificado, extrae Kpub_serv
      2. Verifica firma RSAsign sobre (G, N, Yb, server_random)
      3. Genera privada Xa < N
      4. Calcula Ya = G^Xa mod N

      ClientKeyExchange (Ya = G^Xa mod N)

      5. pre_master_secret = G^(Xa·Xb) mod N
                                                 5. pre_master_secret = G^(Xa·Xb) mod N
```

Refuerzo conceptual: por qué este esquema cubre todo.

- **RSA** se usa **solo para firmar**, garantizando que los parámetros DH provienen efectivamente del servidor (autenticación).
- **DHE** se usa para **generar la semilla compartida** (`pre_master_secret`) de forma efímera (confidencialidad con PFS).
- A partir de la semilla y los *randoms*, la PRF deriva el resto de material criptográfico.

6. Subprotocolos auxiliares.

6.1. SSL Change Cipher Spec Protocol.

Es un protocolo extremadamente simple. Consta de un **único mensaje de un solo byte con valor** `1`, cuya función es **activar el *cipher suite*** negociado en el handshake. A partir de ese momento, todos los SSL records subsiguientes se procesan con los nuevos algoritmos y claves.

```
SSL record
 Tipo  Versión  Longitud   Fragmento
  20    3,0       1            1          ← un único byte
```

6.2. SSL Alert Protocol.

Se usa para **comunicar al otro extremo incidencias relacionadas con SSL**. Cada mensaje consta de dos bytes y se comprime (opcional) y cifra según lo establecido en la sesión.

```
SSL record
 Tipo  Versión  Longitud   Fragmento (2 bytes)
  21    3,0       2
                              Nivel     Descripción
                             (1 / 2)      (1 byte)
```

- **Primer byte — Nivel de severidad:**
  - `1` → *warning*.
  - `2` → *fatal*.
- **Segundo byte — Código de alerta** específico. Ejemplos: `unexpected_message`, `bad_record_mac`, `decompression_failure`, `illegal_parameter`, etc.

> **!** Importante.
>
> Si el nivel es **fatal**, SSL termina la conexión de forma **inmediata**. Otras conexiones de la misma sesión pueden continuar abiertas, pero **no se permiten nuevas conexiones** dentro de esa sesión.

6.3. SSL Record Protocol.

Ya descrito en §3. Es el responsable de fragmentar, comprimir, autenticar (MAC), cifrar y añadir cabecera a los datos procedentes de la subcapa alta antes de entregarlos a TCP.

7. TLS 1.2 y TLS 1.3.

TLS hereda toda la arquitectura SSL pero introduce mejoras significativas en seguridad y rendimiento. Las dos versiones modernas relevantes son **TLS 1.2 (2008, RFC 5246)** y **TLS 1.3 (2018, RFC 8446)**.

7.1. TLS 1.2.

Los cambios respecto a SSL/TLS anteriores son sustanciales:

- **Cálculo del *master key*:** La PRF se basa en **SHA-256**, en lugar de la combinación MD5 + SHA-1 de TLS 1.0/1.1. Concretamente:

```
master_key (48 bytes) = PRF-SHA-256( pre_master_secret,
                                     "master_key",
                                     client_random + server_random )

key_block             = PRF( master_key,
                             "key_expansion",
                             client_random + server_random )
```

del que se derivan:

- `client_write_MAC_secret`, `server_write_MAC_secret`
- `client_write_key`, `server_write_key`
- `client_write_IV`, `server_write_IV`
- **Extensiones en `ClientHello`/`ServerHello`:** Permite añadir información sobre certificados, parámetros de seguridad, autorizaciones, tipos de certificados, etc.
- **Actualización del *cipher suite*:** Se eliminan DES e IDEA, se añade **AES** y se introduce criptografía de clave pública basada en **curvas elípticas** con **ECDHE** para la negociación de claves.
- **AEAD (Authenticated Encryption with Associated Data):** Se introducen modos que combinan cifrado y autenticación en una sola operación:
  - **AES-CBC-MAC / AES-CCM**.
  - **AES-GCM (Galois Counter Mode)**, que funciona de forma similar al modo CTR pero usa Carter-Wegman MAC en un campo de Galois; es rápido y eficiente.

7.2. TLS 1.3.

TLS 1.3 introduce cambios todavía más profundos, orientados a **rendimiento** y **seguridad**:

- **Un único RTT (*Round-Trip Time*) en el handshake.** Frente a los 2 RTT de versiones anteriores, TLS 1.3 reduce el coste de establecer la conexión.
- **0-RTT en reconexión.** Si cliente y servidor ya se conocen, pueden reutilizar credenciales (**PSK** — **pre-shared key**) y empezar a enviar datos junto con el primer mensaje.
- **Prevalece Perfect Forward Secrecy.** Solo se permiten *cipher suites* del tipo **DHE-RSA / ECDHE**.
- **Reducción de modos de operación.** Solo se admiten **CBC** y **AEAD** (GCM y CCM). Se eliminan los algoritmos débiles o cuestionables: DES, RC4, MD5, SHA-1, etc.
- **No usa compresión** (la compresión a nivel TLS había abierto la puerta a ataques como CRIME).
- **Cifrado de la práctica totalidad del handshake**, incluyendo certificados, lo que reduce el *fingerprinting* del cliente y del servidor.

Esquema rápido del handshake TLS 1.3 (1-RTT).

```
Cliente                                          Servidor

      ClientHello
       + ClientKeyShare

       ServerHello
         + ServerKeyShare
         {EncryptedExtensions}
         {Certificate}
         {CertificateVerify}
         {Finished}

      {Finished}

         datos de aplicación
```

Las llaves `{ }` indican mensajes **cifrados** con las claves derivadas tras el intercambio de `KeyShare`. El handshake completo se realiza en **un solo RTT**.

7.3. Comparativa TLS 1.2 vs TLS 1.3.

Aspecto	TLS 1.2	TLS 1.3
RTT del handshake	2 RTT	1 RTT (0-RTT en reconexión)
Cipher suites soportados	Amplio catálogo, muchos legados	Catálogo reducido, solo AEAD

Aspecto	TLS 1.2	TLS 1.3
PFS	Opcional (DHE/ECDHE recomendado)	**Obligatorio**
Algoritmos eliminados	DES, IDEA	Además: RC4, MD5, SHA-1, modos no-AEAD
Compresión TLS	Opcional (deshabilitada en práctica)	Eliminada
Cifrado del handshake	Parcial	Casi completo (incluidos certificados)

⚠ Idea clave.

TLS 1.3 representa una **simplificación radical** del protocolo: menos opciones, menos algoritmos, menos rondas. Esta reducción no es casual; cada elemento eliminado había sido fuente de vulnerabilidades reales (BEAST, CRIME, POODLE, FREAK, Logjam, etc.) en versiones anteriores.

---

8. DTLS — Datagram Transport Layer Security.

8.1. Motivación.

SSL/TLS está diseñado sobre **TCP**, que es orientado a la conexión y fiable. Sin embargo, hay aplicaciones que se ejecutan sobre **UDP** (VoIP, vídeo en tiempo real, IoT, DNS sobre UDP, etc.) y que también requieren seguridad. Para cubrir este caso se definió **DTLS (Datagram Transport Layer Security)** en el **RFC 6347** (creado en 2006, última versión de enero de 2012).

DTLS está adquiriendo un papel relevante en escenarios de comunicación en **tiempo real** o **entornos restringidos** como IoT (RFC 7925 — TLS/DTLS Profiles for the Internet of Things).

8.2. Adaptaciones respecto a TLS.

La estructura general es similar a la de TLS, pero la transmisión se realiza sobre UDP, **no fiable** (puede haber pérdidas o entrega desordenada de datagramas). Para compensar esto, DTLS añade:

- **Mecanismos de control de flujo** de la información a nivel del propio protocolo.
- Un **número de secuencia explícito** en cada DTLS record, en lugar del implícito de TLS.

- Mecanismos de **retransmisión** y **fragmentación** propios del handshake para tolerar pérdidas.

```
+-----------------+
|   Aplicación    |
+-----------------+
|  DTLS Record    |
+-----------------+
|  UDP   |  TCP   |   ← DTLS sobre UDP; TLS sobre TCP
+--------+--------+
|     Internet    |
+-----------------+
|     Enlace      |
+-----------------+
|     Física      |
+-----------------+
```

Cuándo usarlo.

DTLS es la opción adecuada cuando la aplicación **no puede o no quiere usar TCP**: comunicación en tiempo real con tolerancia a pérdidas (voz, vídeo), entornos con dispositivos limitados (IoT, sensores) o protocolos diseñados sobre UDP (DTLS-SRTP, CoAP sobre DTLS, etc.).

---

9. Síntesis y modelo mental.

Modelo mental.

SSL/TLS es una **subcapa** entre aplicación y transporte que cumple tres funciones, en orden:

1. **Acordar quién es quién y con qué algoritmos** se va a hablar (handshake, certificados, *cipher suite*).
2. **Acordar una clave compartida** que ninguna escucha pueda recuperar (intercambio de claves, idealmente con PFS vía DHE/ECDHE).
3. **Proteger cada paquete de datos** con esa clave (Record Protocol: fragmentar, comprimir, autenticar, cifrar).

El resto de subprotocolos —Change Cipher Spec y Alert— son simples mecanismos de control: uno para "pasar de modo claro a modo cifrado" y otro para "avisar de errores".

Refuerzo conceptual: línea de tiempo conceptual.

- **SSL 2.0 / 3.0 (1994-1996)** — Solución pragmática de Netscape; SSL 3.0 ya está obsoleta.
- **TLS 1.0 / 1.1 (1999-2006)** — Estandarización por IETF; vulnerabilidades acumuladas en CBC y hashes.
- **TLS 1.2 (2008)** — SHA-256, AES, ECDHE, AEAD; el "estándar de trabajo" durante una década.
- **TLS 1.3 (2018)** — Simplificación, 1-RTT, PFS obligatorio, eliminación de criptografía débil.
- **DTLS** — Misma arquitectura adaptada a UDP, con retransmisión y secuencia explícita.