PROMETEO

Unidad 1: Programación segura

La **programación segura** no es un añadido opcional, sino una práctica esencial en el desarrollo moderno. Cada línea de código que escribes puede ser una puerta de entrada para un atacante si no aplicas medidas de seguridad desde el principio.

Sesión 1: Principios de programación segura: validación de datos, manejo de información sensible, actualización constante en ciberseguridad.

La **programación segura** no es un añadido opcional, sino una práctica esencial en el desarrollo moderno. Cada línea de código que escribes puede ser una puerta de entrada para un atacante si no aplicas medidas de seguridad desde el principio. Este enfoque forma parte del concepto **"security by design"**, es decir, pensar en la seguridad desde la concepción del software y no como un parche posterior.

Los principios básicos se apoyan en tres ejes:

Validación de datos de entrada

Nunca debes confiar en lo que un usuario introduce en un formulario, una API o un archivo cargado. Los datos pueden contener código malicioso diseñado para romper tu aplicación. La validación estricta —tipo, longitud, formato, rango— previene ataques como la inyección SQL o el Cross-Site Scripting (XSS).

Protección de información sensible

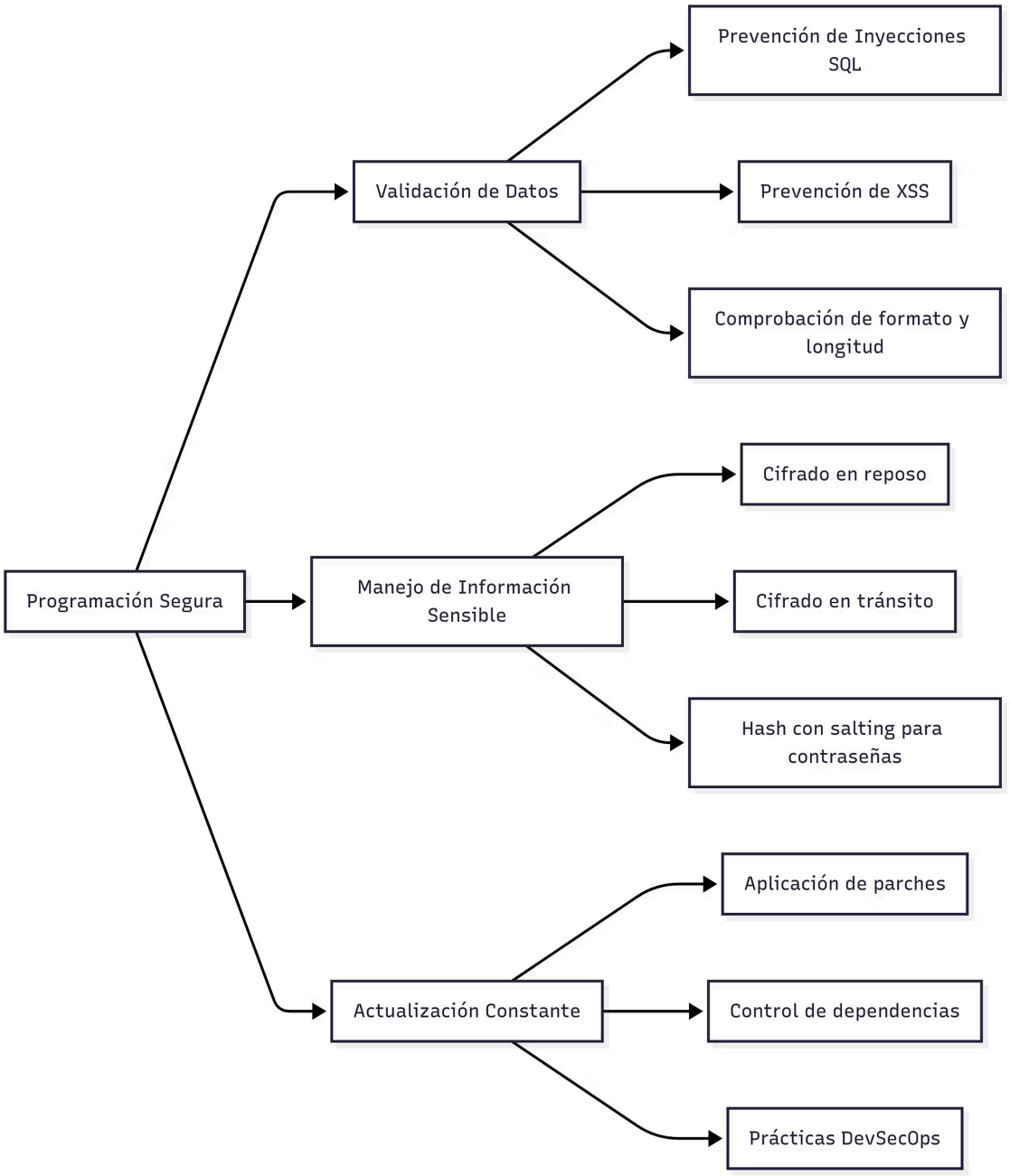
Cualquier dato que manejes —contraseñas, números de tarjeta, historiales médicos— debe almacenarse y transmitirse de forma cifrada. Para contraseñas, lo adecuado es aplicar **hashes con salting** (bcrypt, Argon2). Para comunicaciones, protocolos como **TLS 1.3** son el estándar. Aquí entra en juego la **confidencialidad, integridad y disponibilidad** (CIA triad).

Actualización y gestión de dependencias

Muchas vulnerabilidades no provienen de tu propio código, sino de librerías o frameworks desactualizados. Mantener dependencias al día y aplicar parches de seguridad de inmediato es fundamental. La filosofía **DevSecOps** integra la seguridad en la cadena de despliegue continuo, asegurando que la protección evoluciona junto al software.

En un contexto regulado por normas como **RGPD** en Europa, programar de forma segura no solo evita ataques: garantiza cumplimiento legal y confianza del cliente. Como futuro profesional, debes asumir que la seguridad no es responsabilidad exclusiva del equipo de ciberseguridad: es parte de tu trabajo diario como programador.

Esquema Visual



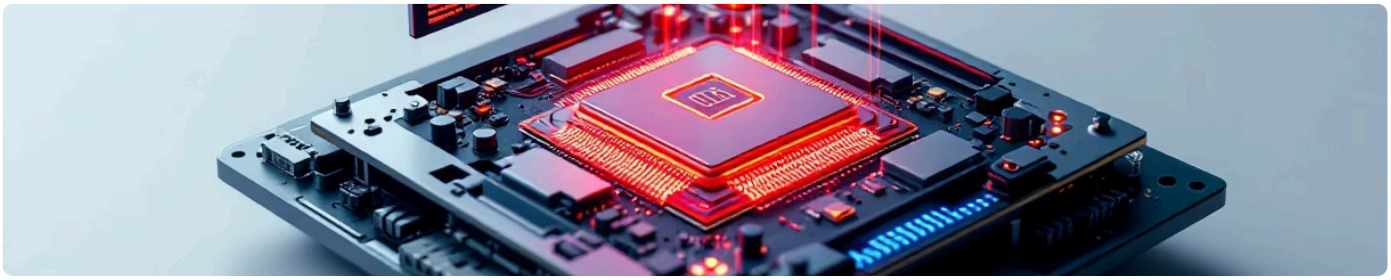
Explicación: El nodo central, *Programación Segura*, se divide en tres ramas que representan los pilares.

- **Validación de datos** asegura que lo que entra en el sistema es fiable, previniendo ataques comunes.
- **Manejo de información sensible** se centra en proteger datos críticos tanto en reposo como en tránsito.
- **Actualización constante** refleja la necesidad de mantener librerías y frameworks al día, con una visión de seguridad continua. La interconexión de las tres ramas forma un sistema de protección integral: si falla una, todo se debilita.

Caso de Estudio: El Ciberataque a Equifax (2017)

Contexto

Equifax, agencia de crédito estadounidense, almacenaba información sensible de más de 800 millones de personas. En 2017 sufrió una brecha de seguridad devastadora.



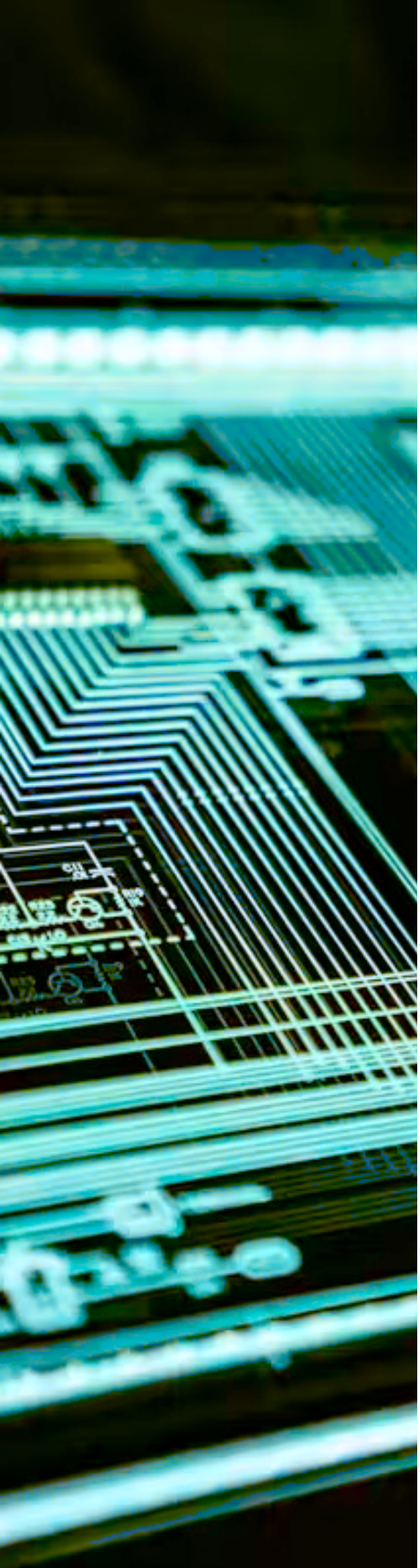
Estrategia (o carencia de ella)

El fallo se produjo por no aplicar un parche de seguridad crítico en **Apache Struts**, un framework web. Aunque la vulnerabilidad y la actualización eran públicas, la empresa no actualizó a tiempo. Los atacantes explotaron esta debilidad para infiltrarse en sus servidores y extraer datos durante meses.

Resultado

147M	$700M	Fatal
Usuarios comprometidos	Pérdidas económicas	Daño reputacional
Datos personales de **millones de usuarios** comprometidos.	En sanciones y compensaciones.	Dimisión de altos directivos y **daño reputacional irreversible**.

> **Conclusión** Slack demuestra que la GUI no es cosmética: **habilita el valor del producto**. Cuando la información se organiza con una jerarquía clara y se suministra feedback contextual, **tareas complejas parecen simples** y la herramienta se vuelve imprescindible.



Herramientas y Consejos

1

Para validar datos

- Usa Joi (Node.js) o Hibernate Validator (Java) para implementar reglas robustas.
- Aplica validación tanto en frontend (para usabilidad) como en backend (para seguridad).

2

Para proteger datos sensibles

- Implementa bcrypt o Argon2 para contraseñas.
- Gestiona secretos con HashiCorp Vault o AWS Secrets Manager.
- Activa siempre protocolos seguros: HTTPS con TLS 1.3.

3

Para actualizar dependencias

- Herramientas como Dependabot (GitHub) o Snyk automatizan alertas de vulnerabilidades.
- Integra estas revisiones en pipelines de CI/CD.

4

Referencias clave

Consulta el documento OWASP Top 10, guía internacional sobre vulnerabilidades más comunes en aplicaciones web.

Mitos y Realidades

❌ **MITO**: "La seguridad es tarea exclusiva del equipo de ciberseguridad."

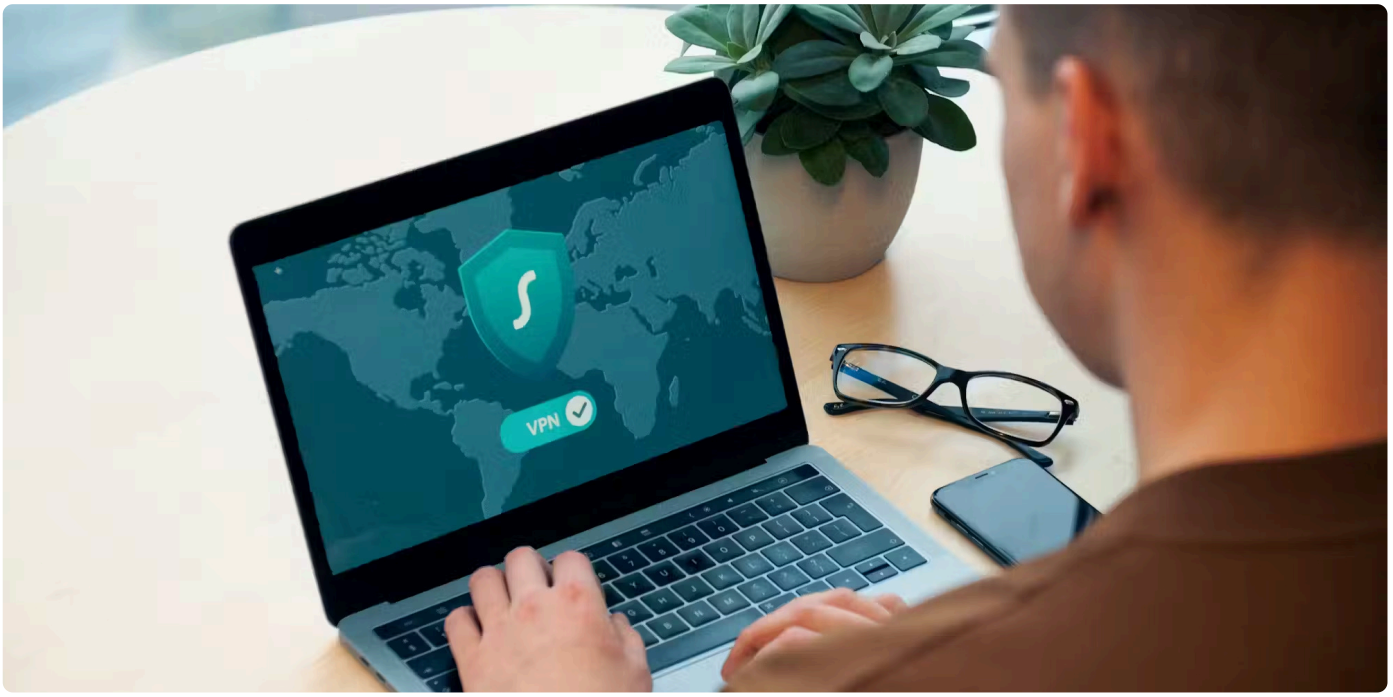
→ **FALSO**: El equipo de seguridad puede poner normas, pero si el código está mal escrito, el riesgo persiste. La seguridad debe ser responsabilidad compartida, desde el programador hasta el administrador de sistemas (**DevSecOps**).

❌**MITO:** "Aprendo una tecnología de GUI y me sirve para siempre."

→ **FALSO:** El sector evoluciona. Dominar fundamentos (eventos, data binding, layouts, estados, accesibilidad) te habilita para migrar entre JavaFX, .NET MAUI, Flutter o React Native. La adaptabilidad es la ventaja profesional real.

Resumen Final

- Programación segura = validación, cifrado y actualización.
- Validar entradas previene inyecciones y XSS.
- Contraseñas siempre con hash + salting.
- Actualizar dependencias y aplicar parches salva de desastres.
- Seguridad = responsabilidad compartida.



Sesión 2: Criptografía: simétrica, asimétrica, funciones hash. Protocolos HTTPS, SSL/TLS, SSH.

La **criptografía** es el cimiento de la seguridad digital. Sin ella, cada comunicación en internet podría ser interceptada y cada dato almacenado, manipulado. Su finalidad es proteger la **confidencialidad, integridad y autenticidad** de la información en un entorno donde el riesgo de ataques es permanente. Existen tres grandes técnicas que debes dominar:

Criptografía simétrica

Utiliza una única clave compartida para cifrar y descifrar. Es rápida y eficiente, ideal para grandes volúmenes de datos. El estándar más extendido es **AES (Advanced Encryption Standard)**, usado en discos duros, VPNs o cifrado de bases de datos. Su debilidad es el intercambio seguro de la clave.

Criptografía asimétrica

Emplea un par de claves: una pública para cifrar y una privada para descifrar. Ejemplos: **RSA** o **Elliptic Curve Cryptography (ECC)**. Es más lenta que la simétrica, pero resuelve el problema de compartir claves de forma segura. Es esencial en certificados digitales y protocolos de autenticación.

Funciones hash

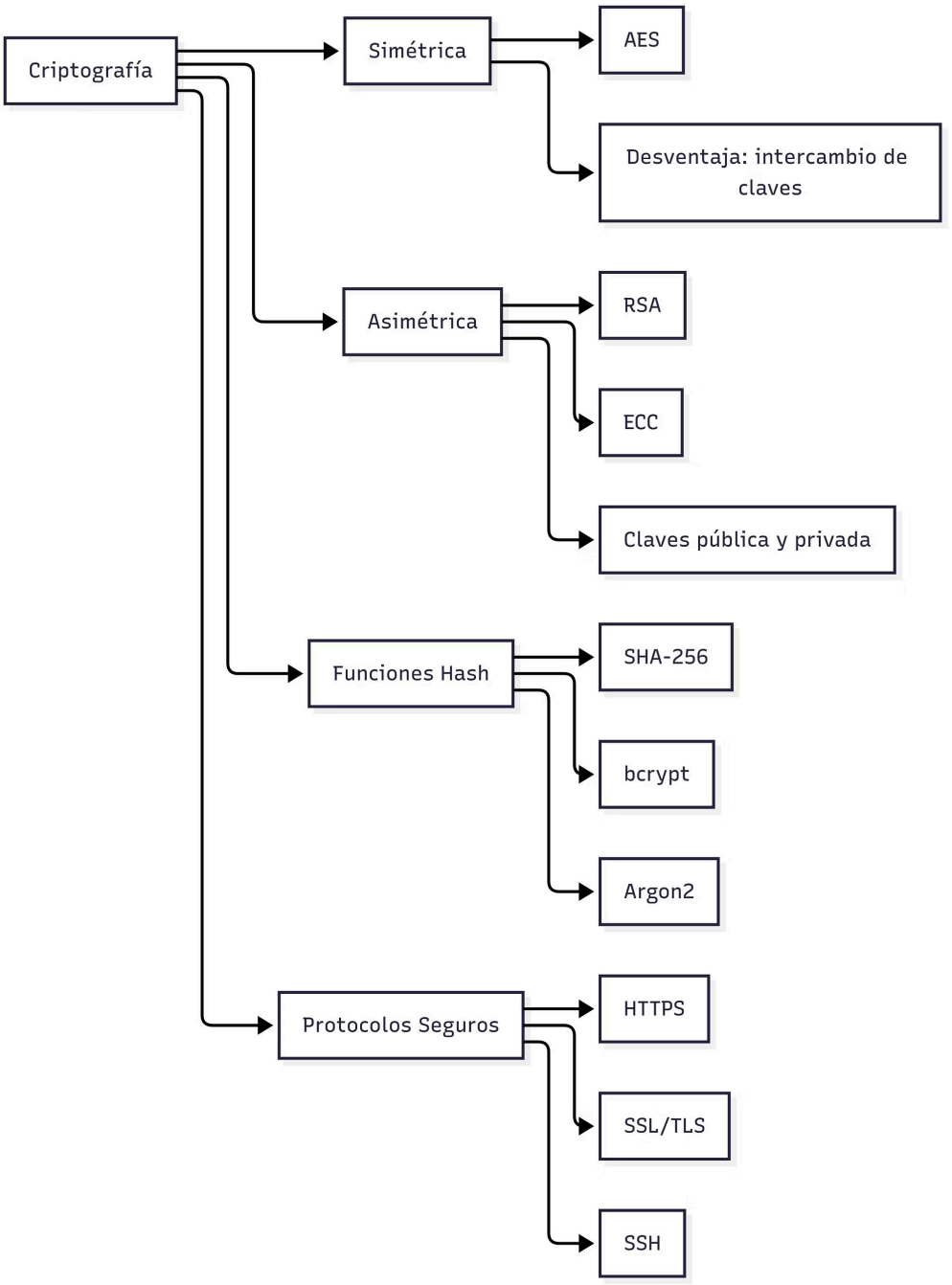
Transforman datos en una cadena de longitud fija e irreversible. Se utilizan para verificar integridad de archivos o almacenar contraseñas. Algoritmos modernos como **SHA-256, bcrypt o Argon2** son la norma actual. A diferencia del cifrado, el hash no se descifra: se compara.

Además de las técnicas, la criptografía se aplica a través de **protocolos de comunicación seguros**, que hacen posible la navegación web, la conexión remota o las transacciones online:

- **HTTPS**: versión segura del HTTP, que cifra la comunicación entre cliente y servidor mediante **SSL/TLS**. Es obligatorio para cualquier sitio web profesional y es un requisito del **RGPD** para proteger datos personales.
- **SSL/TLS**: protocolos que establecen canales seguros mediante el uso de certificados digitales y criptografía híbrida (asimétrica para el inicio de la conexión y simétrica para el intercambio de datos).
- **SSH (Secure Shell)**: protocolo que permite conectarse de forma remota a servidores, sustituyendo a protocolos inseguros como Telnet. Utiliza criptografía asimétrica para autenticar y cifrar la sesión.

Como futuro profesional, comprender cómo se combinan estas técnicas y protocolos es esencial para desarrollar software seguro y cumplir con normativas de protección de datos.

Esquema Visual



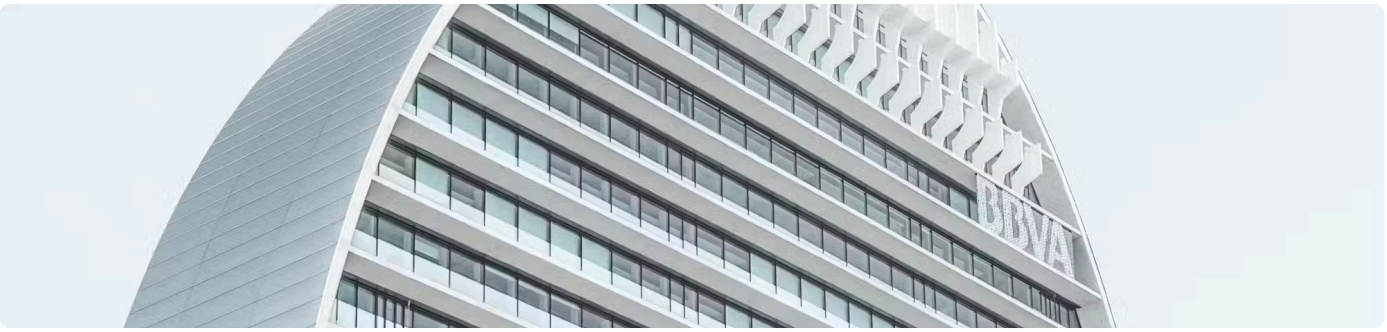
Explicación del esquema:

- La **criptografía** se desglosa en tres técnicas (simétrica, asimétrica y hash) y en protocolos que las aplican en la práctica.
- La simétrica (AES) es rápida pero necesita una forma segura de compartir la clave.
- La asimétrica (RSA, ECC) permite autenticación y distribución segura de claves.
- El hashing protege contraseñas y asegura integridad.
- Los protocolos HTTPS, SSL/TLS y SSH utilizan estas técnicas combinadas para garantizar comunicaciones seguras en web, servidores y conexiones remotas.

Caso de Estudio – HTTPS en la Banca Online de BBVA

Contexto

BBVA, uno de los bancos líderes en España y Latinoamérica, gestiona operaciones financieras de millones de clientes cada día. Garantizar la seguridad en la banca online es crítico para evitar fraudes y cumplir con la normativa PSD2 y el RGPD.



Estrategia

- Implementación obligatoria de **HTTPS** en todos sus servicios digitales, basado en **TLS 1.3**.
- Uso de **cifrado asimétrico** (RSA/ECC) para establecer la conexión inicial y simétrico (AES-256) para las transacciones.
- **Firmas digitales** para validar certificados de sus servidores y evitar ataques de suplantación.
- Autenticación reforzada mediante **hashing de contraseñas** y validaciones multifactor.

Resultado

Reducción drástica de intentos exitosos de *phishing* y *man-in-the-middle*.	Reconocimiento como uno de los bancos más avanzados en ciberseguridad en Europa.	Generación de confianza en sus clientes gracias a un canal de comunicación seguro.

> **Lección clave**: la aplicación correcta de protocolos criptográficos no solo protege al usuario, sino que fortalece la reputación de la empresa.

Herramientas y Consejos

1

Para cifrado

- Emplea OpenSSL para implementar cifrado simétrico y asimétrico en proyectos.
- En entornos de desarrollo, utiliza bibliotecas seguras como Crypto (Node.js) o javax.crypto (Java).

2

Para protocolos

- Usa Wireshark para verificar que la comunicación realmente viaja cifrada bajo HTTPS o SSH.
- Configura servidores con Let's Encrypt, que ofrece certificados SSL/TLS gratuitos y renovables.

3

Para gestión de claves y hashes

- Implementa bcrypt o Argon2 en el almacenamiento de contraseñas.
- Gestiona claves y certificados en servicios en la nube como AWS KMS o Azure Key Vault.

4

Buenas prácticas

- Obliga a usar TLS 1.2 o superior en tus proyectos. TLS 1.0 y 1.1 ya no son seguros.
- Evita protocolos obsoletos (como FTP o Telnet); sustitúyelos siempre por sus equivalentes seguros (SFTP, SSH).

Mitos y Realidades

❌ **Mito**: "Con poner HTTPS en mi web ya estoy completamente protegido."

→ **FALSO**. HTTPS protege la transmisión de datos, pero no evita vulnerabilidades en tu aplicación. Si tu web es vulnerable a inyecciones SQL o XSS, el ataque seguirá siendo posible aunque uses HTTPS.

❌ **Mito**: "El hashing de contraseñas con MD5 es suficiente."

→ **FALSO**. MD5 y SHA-1 están rotos y permiten colisiones. Hoy en día, los atacantes pueden descifrar contraseñas hash con estas funciones en segundos. La realidad es que debes usar bcrypt o Argon2 con salting para garantizar seguridad.

Resumen Final

- Criptografía = simétrica (AES), asimétrica (RSA, ECC) y hash (SHA-256, bcrypt).
- Protocolos clave: HTTPS, SSL/TLS, SSH.
- HTTPS = obligatorio para toda web moderna.
- SSL/TLS = base de las comunicaciones seguras.
- SSH = acceso remoto seguro a servidores.
- Caso BBVA = aplicación de TLS 1.3 en banca online.
- Mitos: HTTPS no cubre todas las vulnerabilidades, y MD5 está obsoleto.

Sesión 3: Control de acceso (autenticación, autorización), documentación de aspectos de seguridad en aplicaciones.

El **control de accesos** es un principio esencial de la seguridad informática que responde a una pregunta crítica: **¿quién puede acceder a qué, cuándo y cómo?**. No basta con proteger la comunicación o cifrar los datos; si no defines reglas claras de acceso, cualquier usuario podría entrar en recursos que no le corresponden. Existen tres elementos clave en este proceso:

Autenticación

Es el mecanismo mediante el cual un sistema verifica la identidad de un usuario o entidad. Los métodos más comunes incluyen:

- **Contraseñas**: todavía el más usado, aunque con debilidades si no se complementa.
- **Factores biométricos**: huella dactilar, reconocimiento facial.
- **Tokens**: físicos (tarjetas, llaves USB) o virtuales (apps móviles de autenticación).

Hoy en día, se recomienda usar **autenticación multifactor (MFA)** para reforzar la seguridad.

Autorización

Una vez autenticado, el sistema determina qué acciones puede realizar el usuario. Aquí entran en juego los modelos de control de accesos:

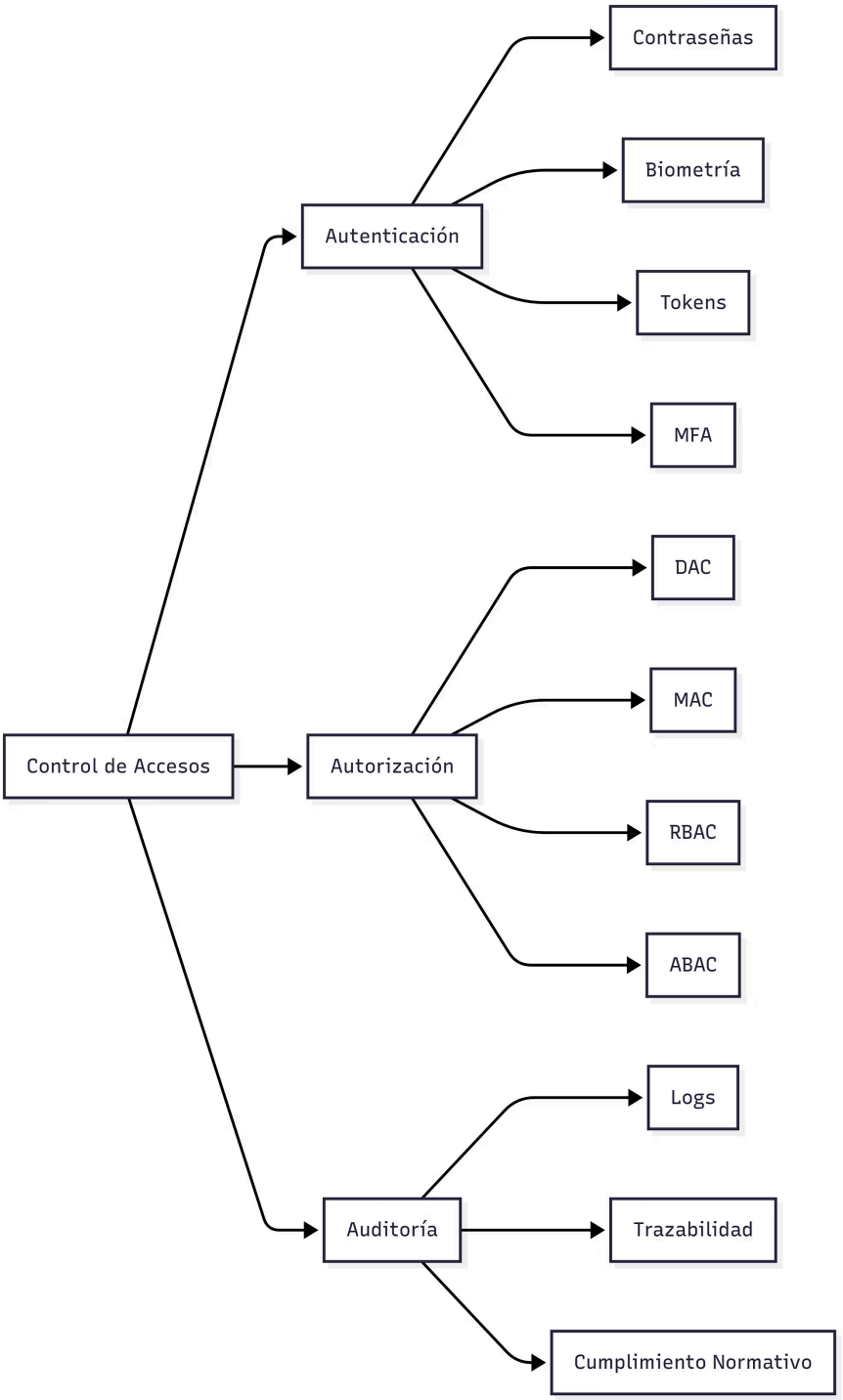
- **DAC (Discretionary Access Control)**: el propietario decide quién accede a los recursos.
- **MAC (Mandatory Access Control)**: las reglas de acceso son centralizadas y no modificables por los usuarios.
- **RBAC (Role-Based Access Control)**: los permisos se asignan según roles predefinidos (ej. administrador, editor, lector).
- **ABAC (Attribute-Based Access Control)**: las reglas dependen de atributos dinámicos (ej. ubicación, hora del día).

Trazabilidad y auditoría

El control de accesos no termina con autenticar y autorizar; también es vital **registrar quién accedió a qué y cuándo**. Los logs permiten detectar intentos sospechosos, cumplir con normativas como el RGPD y realizar investigaciones en caso de incidentes.

En el mundo laboral, dominar el control de accesos significa diseñar sistemas resilientes frente a accesos indebidos, minimizar riesgos internos (empleados con permisos excesivos) y garantizar cumplimiento normativo. Es un equilibrio entre **seguridad y usabilidad**: demasiado laxo genera riesgos, demasiado restrictivo frustra a los usuarios.

Esquema Visual



Explicación del esquema:

- El nodo central **Control de Accesos** se ramifica en tres fases: **Autenticación, Autorización y Auditoría**.
- La autenticación asegura la identidad del usuario (con contraseñas, biometría, tokens o MFA).
- La autorización regula lo que puede hacer, usando modelos como DAC, MAC, RBAC o ABAC.
- La auditoría registra todas las acciones para garantizar trazabilidad y cumplir regulaciones.

Caso de Estudio – Google Workspace y el Control Basado en Roles

Contexto

Google Workspace es una suite utilizada por millones de empresas en todo el mundo. Al gestionar información sensible en la nube (documentos, correos, datos de clientes), el control de accesos es crítico.

Estrategia

- Google implementa un modelo **RBAC (Role-Based Access Control)**, donde los permisos se asignan a roles predefinidos: administrador, editor, lector, etc.
- Incorpora autenticación multifactor (MFA) obligatoria en entornos corporativos para reducir el riesgo de accesos indebidos.
- Añade características de **ABAC**, como limitar accesos según la ubicación geográfica o la dirección IP.
- Incluye auditoría avanzada, donde los administradores pueden ver quién accedió a qué archivo y cuándo.

Resultado

Mayor protección

Contra ataques de phishing y robo de credenciales.

Reducción de riesgos

Internos gracias a una gestión granular de permisos.

Cumplimiento normativo

De normativas internacionales de seguridad y protección de datos.

> **Lección clave**: un control de accesos bien diseñado combina autenticación robusta, asignación de permisos eficiente y registro continuo de actividades.

Herramientas y Consejos

1

Gestión de autenticación

- Implementa OAuth 2.0 y OpenID Connect para la autenticación en aplicaciones web.
- Usa plataformas de identidad como Okta o Auth0 para integrar MFA y control centralizado.

2

Gestión de permisos

- Diseña roles claros con RBAC en sistemas empresariales (ej. SAP, Salesforce).
- Para entornos cloud, usa los sistemas de control nativos: IAM en AWS, IAM en Google Cloud o Azure AD.

3

Auditoría y trazabilidad

- Configura logs centralizados con herramientas como Splunk o Elastic Stack.
- Activa alertas automáticas para accesos sospechosos (ej. inicio de sesión desde países inusuales).

4

Buenas prácticas

- Aplica el principio de mínimo privilegio: cada usuario debe tener solo los permisos necesarios.
- Revisa periódicamente permisos obsoletos o cuentas inactivas.

Mitos y Realidades

❌ Mito: "Si una contraseña es fuerte, ya no hace falta nada más."

→ FALSO. Una contraseña robusta es útil, pero los ataques de phishing o robo de credenciales siguen siendo efectivos. La realidad es que necesitas MFA para proteger identidades.

❌ Mito: "El control de accesos es estático: configuras los permisos una vez y ya está."

→ FALSO. Los permisos deben revisarse y actualizarse continuamente, adaptándose a cambios en los roles de los empleados o en el entorno de amenazas.

Resumen Final

- Control de accesos = autenticación, autorización y auditoría.
- Autenticación reforzada con MFA.
- Autorización mediante DAC, MAC, RBAC y ABAC.
- Auditoría = trazabilidad y cumplimiento normativo.
- Caso Google Workspace: RBAC + MFA + auditoría.
- Mitos: contraseñas no bastan y permisos no son estáticos.