Blockchain Learning Path

Manual del usuario · Programa educativo en español para aprender blockchain de cero a producción

v0.8.1

19 módulos · 50 prácticas · https://github.com/vladimiracunadev-create/blockchain-learning-path

Índice

Introducción

⛓️ Blockchain Learning Path

19 módulos · 50 prácticas · de cero a producción

Programa educativo en español para aprender blockchain desde los fundamentos hasta llevarla a producción en una empresa — criptografía, Bitcoin, Ethereum/EVM, Solidity, dApps, seguridad, L2, DAO, infraestructura, casos de negocio e implementación, con laboratorios ejecutables y un proyecto integrador.

CI passing · Security passing · Deploy Pages passing

VERSIÓN 0.8.1 · MÓDULOS 19 · PRÁCTICAS 50 · NIVEL INICIAL → PRODUCCIÓN

IDIOMA ESPAÑOL · CODE MIT · CONTENIDO CC BY 4.0

Node.js LTS · pnpm workspace · Solidity contratos · Foundry pruebas · viem dApps · TypeScript dApp & indexer

🌱 Empieza aquí · 📖 Glosario · 🌐 Sitio · 📕 Manual (PDF) · 📥 Apps · 📚 Currículo · 🏭 Industria · 🧪 Laboratorios · 🗺️ Roadmap · 🧭 Rutas por perfil · 🤝 Contribuir · 🔐 Seguridad

---

🌱 ¿Es tu primer contacto con blockchain?

👉 EMPIEZA AQUÍ

No hace falta que sepas nada todavía. Esa página te dice qué necesitas antes de arrancar, qué instalar y en qué momento, cómo se lee un módulo, qué hacer cuando te atasques — y te deja en la puerta del módulo 00, que es por donde se empieza.

Ten a mano el **📖 glosario** (120 términos, enlazado desde todos los módulos) para cuando una palabra te frene.

> **Blockchain no es sinónimo de criptomoneda.** En este recorrido aprenderás cuándo una cadena de bloques aporta valor, cuándo una base de datos tradicional es mejor y cómo construir —y llevar a producción— sistemas descentralizados de forma responsable.

🎯 Qué es esto

Un currículo modular y **secuencial** que cubre el espectro completo de blockchain, paso a paso, en **19 módulos numerados (00→18)** agrupados en seis etapas, más un proyecto final. Cada módulo es una carpeta con un `README.md` que incluye:

- 🎯 **Objetivos** medibles y **resultados de aprendizaje** verificables.
- 🗺️ **Temas** con el porqué de cada uno y **conceptos** con definiciones.
- 🧩 **Esquema visual** (diagramas Mermaid que GitHub renderiza nativamente).
- 🧠 **Modelo mental** con su analogía y los **límites** de la analogía.
- 🔬 **Profundización** con ejemplos numéricos y casos reales verificables.
- 🧪 **Laboratorio guiado** ejecutable y un **reto verificable** con criterio de aceptación.
- ⚠️ **Errores frecuentes** (síntoma → causa) y 🛡️ **seguridad y ética**.
- 🔗 **Referencias** a los libros y fuentes primarias del área.

No enseña a especular: enseña a decidir **cuándo** usar la tecnología, a **construirla** con pruebas y a **llevarla a una empresa** con infraestructura, costos y casos reales.

📚 Pauta derivada de los mejores libros

Cada etapa se apoya explícitamente en la literatura de referencia del sector; el contenido es **original en su redacción** y **no reproduce** las obras.

Área	Libros y fuentes de referencia
Fundamentos y panorama	Bashir — *Mastering Blockchain* · Narayanan et al. — *Bitcoin and Cryptocurrency Technologies* · Werbach — *The Blockchain and the New Architecture of Trust*
Criptografía	Aumasson — *Serious Cryptography* · Katz, Lindell — *Introduction to Modern Cryptography* · Ferguson/Schneier/Kohno — *Cryptography Engineering*
Sistemas distribuidos y consenso	Cachin, Guerraoui, Rodrigues — *Reliable and Secure Distributed Programming* · Nakamoto (whitepaper) · Castro, Liskov — *PBFT*
Bitcoin y Ethereum	Antonopoulos — *Mastering Bitcoin* · Antonopoulos, Wood — *Mastering Ethereum* · Wood — *Yellow Paper*
Contratos y seguridad	Docs de Solidity · The Foundry Book · Trail of Bits — *Building Secure Contracts* · ConsenSys — *Smart Contract Best Practices*
Escalabilidad, ZK y arquitectura	Buterin — *An Incomplete Guide to Rollups* · Thaler — *Proofs, Arguments, and Zero-Knowledge* · ERC-4337 · Flashbots
Empresa e infraestructura	BIS · World Economic Forum · documentación de clientes de nodo y de nube

> **¿Dónde se usa cada libro?** La bibliografía incluye la tabla qué obra sustenta cada módulo: enlaza cada obra con su fuente oficial y con el módulo concreto que la usa, así que puedes ir del libro al módulo o del módulo al libro. Varias de las obras clave —*Mastering Bitcoin*, *Mastering Ethereum*, *Proofs, Arguments, and Zero-Knowledge*— tienen **edición legalmente gratuita**: el programa completo se puede seguir sin comprar un solo libro.
>
> La bibliografía recoge además los **hitos recientes** del ecosistema (Merge, Dencun/EIP-4844, Pectra/EIP-7702).

📥 Llévate el curso: apps y manual

Todo el contenido —los 19 módulos, la industria, los laboratorios, los ADR y el proyecto final— viaja contigo. **Las apps funcionan sin conexión**: sirven para estudiar en el metro o en un aula sin wifi.

Formato	Descarga	Notas
🖥️ **Windows**	Instalador o portable	Curso completo dentro del ejecutable · cómo está hecha
📱 **Android**	APK (~8,5 MB)	Curso completo dentro del APK · cómo está hecha
📕 **Manual PDF**	MANUAL.pdf (~340 páginas)	También adjunto en cada release
🌐 **Web**	Sitio del programa	Con buscador, progreso y autoevaluación

Las tres versiones salen del **mismo build**, así que ninguna se queda atrás. Cada binario se verifica en CI abriéndolo y contando los módulos, las páginas y las preguntas que lleva dentro: un build en verde no prueba que el artefacto contenga el curso.

> Los binarios **no están firmados** con certificado de código (cuesta cientos de dólares al año), así que Windows SmartScreen y Android avisarán del origen desconocido. Compara el `SHA256` publicado en la release antes de ejecutarlos.

Para generarlo todo desde el repositorio: `pnpm build:manual`, `pnpm app:windows`, `pnpm app:android`.

🗂️ Los 19 módulos en seis etapas

Cada etapa tiene su índice de currículo con mapa visual. Estúdialos **en orden**: cada módulo asume el anterior.

Etapa	Módulos	Foco	Resultado
Orientación	00	¿Necesito blockchain?	Distinguir blockchain de una base de datos
Fundamentos	01–03	Criptografía, redes P2P, consenso	Entender qué hace verificable a una cadena
Desarrollo	04–07	Bitcoin, EVM, Solidity, dApps	Crear y probar contratos y una dApp
Profesional	08–11	Tokens, seguridad, oráculos, DAO	Diseñar protocolos seguros y gobernados
Avanzado	12–15	L2, interoperabilidad, ZK, arquitectura	Auditar, investigar y decidir arquitectura
Producción	16–18	Infraestructura, empresa, implementación	Llevar la tecnología a una empresa real
Proyecto final	capstone	Integración	Protocolo documentado, probado y desplegable

🧪 Laboratorios y proyectos ejecutables

- **50 prácticas** guiadas con actividad, evidencia y criterio de aceptación (catálogo).
- **Contratos con Foundry**: vault, protocolos, token, oráculo y gobernador con timelock, con pruebas, fuzzing e invariantes.
- **Retos de seguridad**: contratos vulnerables y sus correcciones (security-challenges).
- **dApp** de financiamiento comunitario (viem/TypeScript), **indexador** de eventos y **panel** de progreso.
- **Bitcoin Core en `regtest`** y un **nodo Geth real en Docker** para operar infraestructura.

```
git clone https://github.com/vladimiracunadev-create/blockchain-learning-path.git
cd blockchain-learning-path
corepack enable && pnpm install
pnpm check      # valida estructura, enlaces y conteos
pnpm test       # pruebas de Node y de scripts
pnpm lab:hash   # tu primer laboratorio
pnpm serve      # panel de seguimiento
```

> 💻 **Sin instalar nada:** abre el repo en GitHub Codespaces — el devcontainer deja listos Node, pnpm, Foundry y Docker.

Luego empieza por curriculum/00-orientacion. Cada una de las 50 prácticas trae su **resolución explicada** —cómo se implementa, el comando, la salida esperada y su

interpretación— en el cuaderno de laboratorios.

🏭 La industria por dentro

Además del currículo, la sección Industria es la lectura profesional extendida: cómo se **construye** una red, el **stack** real del ecosistema, cómo trabajan y se **comunican** los equipos, **casos empresariales** (éxitos y fracasos) y **modelos de negocio**. Y para llevarlo a la práctica, los módulos 16–18 lo convierten en laboratorios: infraestructura real, caso de negocio con costos e implementación end-to-end. Incluye una guía dedicada de **cómo explicar blockchain a clientes y personas no técnicas**: discurso de 30 segundos, traducción de jerga y manejo de objeciones.

👩‍🏫 Para instructores

El programa está listo para el aula: guía del instructor, syllabus de 26 semanas, checklist de laboratorios y rúbricas de evaluación. Los archivos de `solutions/` son **criterios de revisión, no respuestas para copiar**. El estudiante lleva su avance en una copia de `student/progress.example.json`.

🚀 Cómo usar el programa

1. **Sigue el orden.** La numeración 00→18 es secuencial por diseño: cada módulo asume el anterior.
2. **Aplica el ciclo** de cada módulo: comprender → experimentar → explicar → construir → verificar.
3. **Ejecuta los laboratorios** en local (Anvil) o testnet; registra la evidencia en tu bitácora de progreso.
4. **Haz el reto verificable** de cada módulo: ahí se fija el aprendizaje con un criterio de aceptación explícito.
5. **Comprueba tu trabajo sin depender de nadie:** 31 de las 50 prácticas traen verificación ejecutable (`pnpm test` , `forge test`), así que estudiando solo tienes señal inmediata de si tu solución funciona. El resto produce una evidencia revisable con rúbrica.
6. **Cierra con la autoevaluación** del módulo (4 preguntas al final de cada uno, también en el sitio y en las apps). Cada opción incorrecta es un error frecuente documentado en ese mismo módulo: si fallas, la explicación te dice exactamente qué releer. Al terminar el programa, el quiz global repasa las 19 etapas.
7. **Usa los libros de referencia** de cada área para profundizar.

Plan completo en el ROADMAP de 26 semanas · ruta intensiva en docs/ruta-rapida.md.

🧭 Rutas por perfil profesional

Recorridos ordenados para **desarrollo, arquitectura, auditoría, producto, investigación y empresa**: elige el tuyo en learning-paths.

✅ Calidad y CI

- **CI**: lint de Markdown, JavaScript (ESLint) y Solidity (`forge fmt`), validación de estructura, enlaces, autoevaluación y cadena de módulos (`pnpm check`), pruebas de Node y de contratos con Foundry, y análisis estático con Slither.
- **Security**: escaneo de secretos con `gitleaks` en cada push y semanalmente, más CodeQL sobre el JavaScript.
- **Enlaces**: revisión semanal de los ~240 enlaces externos del material; si alguno muere, se abre un issue.
- **Apps**: cada binario se construye y se **abre** en CI para contar los módulos, las páginas y las preguntas que lleva dentro. Compilar no es evidencia de que el artefacto contenga el curso.
- **Deploy Pages**: el sitio y el manual en PDF se generan desde el repo y se publican automáticamente.
- **Dependabot** mantiene al día las dependencias y las GitHub Actions.

Consulta la guía de contribución, el código de conducta y la política de seguridad.

🎯 Qué es y qué no es este programa

✅ Lo que sí es

- Un currículo para **entender, construir y llevar a producción** blockchain de forma responsable.
- Material **original**, con fuentes citadas y actualizado a los hitos recientes del ecosistema.
- Laboratorios **ejecutables** en local/testnet, con criterios de aceptación verificables.

❌ Lo que no es

- **No es asesoría** financiera, tributaria ni legal, ni promete rentabilidad alguna.
- **No es una guía para especular** con criptomonedas ni para operar con fondos reales.
- **No sustituye** una auditoría profesional ni la operación real de una infraestructura.

💡 Idea fuerza

La madurez técnica no se demuestra usando blockchain en todo, sino sabiendo **cuándo aporta valor, cuándo una base de datos es mejor, y cómo llevar a producción con seguridad** aquello que sí lo justifica.

📄 Licencia

🌱 Empieza aquí (si no sabes nada de blockchain)

⬅️ Volver al programa · 📖 Glosario · 📚 Currículo

Esta página es para quien abre el repositorio y piensa "*esto parece enorme y no entiendo la mitad de las palabras*". Es una reacción razonable. Aquí está el orden en que conviene mirarlo.

Regla número uno: no tienes que entenderlo todo hoy. El programa está pensado para 26 semanas. Si intentas leerlo de corrido te vas a atascar, y el atasco no significa que no sirvas para esto: significa que te saltaste el andamio.

¿Esto es para mí?

Tu situación	Respuesta honesta
Sé programar (cualquier lenguaje)	Sí. Es el perfil para el que está escrito
Nunca he programado	Sí para los módulos 00–03 y 17, que son conceptuales. Para los demás necesitarás aprender programación en paralelo: sin eso, los laboratorios serán copiar y pegar
Quiero invertir en criptomonedas	**No.** Este programa enseña la tecnología, no a especular. No encontrarás recomendaciones de inversión y es deliberado
Soy de negocio, no técnico	Sí, por otra puerta: módulo 00, módulo 17, la sección de industria y cómo explicarlo a no técnicos

Lo primero: cuatro palabras que se confunden siempre

Antes de empezar, distingue esto. Es el 80 % de la confusión inicial:

- **Blockchain** — la *tecnología*: un registro compartido que muchas máquinas mantienen a la vez y que nadie puede reescribir a solas.
- **Criptomoneda** — un *uso* de esa tecnología: dinero digital nativo de una red concreta (bitcoin en Bitcoin, ether en Ethereum).
- **Token** — algo que alguien crea *encima* de una red existente, con un programa. No es la moneda nativa de la red.
- **Contrato inteligente** — un *programa* que se ejecuta en la red y aplica reglas sin que nadie pueda pararlo a mitad.

Blockchain **no** es sinónimo de criptomoneda, igual que "base de datos" no es sinónimo de "banco".

El resto del vocabulario está en el glosario, con 120 términos. No lo leas entero: consúltalo cuando una palabra te frene.

Qué necesitas instalar (y cuándo)

No instales nada todavía. Cada cosa se necesita en un momento distinto:

Cuándo	Qué	Para qué
Módulo 00	Nada	Es lectura y una decisión razonada
Módulo 01	Node.js 22+	Ejecutar los primeros laboratorios
Módulo 06	Foundry	Escribir y probar contratos
Módulo 07	Una wallet de navegador, **en red de prueba**	Interactuar con una dApp

¿No quieres instalar nada? Abre el repositorio en **GitHub Codespaces**: viene con todo listo. O usa la app de escritorio o la app Android para leer el curso sin conexión.

> ⚠️ **Nunca uses una wallet con dinero real** para los ejercicios. Todo el programa funciona en redes de prueba, donde el dinero no vale nada. Es lo que permite equivocarse sin consecuencias, que es como se aprende esto.

Cómo se lee un módulo

Cada uno tiene la misma estructura. No se lee de arriba abajo de una sentada:

1. 🎯 **Objetivos** — mira qué vas a poder hacer al terminar.
2. 🧠 **Modelo mental** — la analogía. Empieza aquí si el tema es nuevo.
3. 📖 **Conceptos** — el vocabulario del módulo. Vuelve aquí cuando algo no encaje.
4. 🔬 **Profundización** — el detalle. Viene en capas: la idea en llano, el cálculo trabajado, y un bloque plegable 🎓 **Si ya dominas esto** que puedes saltarte sin perder nada.
5. 🧪 **Laboratorio** — hazlo. Leer sobre criptografía no enseña criptografía.
6. ⚠️ **Errores frecuentes** — léelo *antes* de atascarte, no después.
7. 🧠 **Autoevaluación** — cuatro preguntas al final. Cada opción incorrecta es un error frecuente real: si fallas, la explicación te dice qué releer.

Al pie de cada módulo hay enlaces al **anterior** y al **siguiente**. El orden importa: cada módulo asume el anterior.

Cuando te atasques

Te vas a atascar. Es parte del proceso, y hay un orden para salir:

1. **Relee los Conceptos del módulo.** La mayoría de los atascos son una palabra que creías entender.
2. **Busca el término en el glosario.**
3. **Mira los Errores frecuentes** del módulo: la tabla está ordenada por síntoma, así que busca lo que te pasa a ti.
4. **Ejecuta el laboratorio y cambia un número.** Ver qué se rompe enseña más que leer la explicación otra vez.
5. **Vuelve al módulo anterior.** Un atasco persistente casi siempre es un hueco en lo previo, no en lo actual.
6. **Abre un issue.** Si algo está mal explicado, es un fallo del material y arreglarlo ayuda a quien venga detrás.

Cuánto tiempo lleva

Ritmo	Dedicación	Duración
Recomendado	6–10 h/semana	26 semanas
Intensivo (solo si ya programas bien)	15–20 h/semana	8 semanas
Solo entender de qué va	—	Módulos 00 a 03, unas 6 horas

Cinco cosas que conviene saber desde el principio

1. **La mayoría de las veces, la respuesta correcta es "no uses blockchain".** El módulo 00 te enseña a llegar a esa conclusión con argumentos. No es cinismo: es criterio profesional.
2. **Lo irreversible es irreversible.** No hay servicio de atención al cliente ni botón de deshacer. Por eso el programa insiste tanto en probar antes.
3. **Nadie sabe todo esto.** El campo mezcla criptografía, sistemas distribuidos, economía y derecho. Quien afirme dominarlo entero, desconfía.
4. **El material puede quedar obsoleto.** El ecosistema cambia rápido; por eso cada afirmación lleva su fuente enlazada y hay que contrastarla. Cómo se valida está explicado en la bibliografía.
5. **Si alguien te promete rentabilidad, no es tecnología: es una venta.** Este programa no te va a hacer rico; te va a enseñar a construir y a decidir.

Siguiente paso

👉 **Módulo 00 · Orientación** — empieza por aquí.

Si prefieres ver primero el mapa completo: currículo · roadmap de 26 semanas · rutas por perfil.

> ⬅️ Volver al programa · 📖 Glosario · 📚 Currículo

Currículo

📚 Currículo

⬅️ Volver al programa · 📖 Bibliografía y fuentes · 🧪 Laboratorios · 🗺️ Roadmap

19 módulos progresivos (00–18), de los fundamentos criptográficos a llevar la tecnología a producción en una empresa. Cada módulo enlaza al siguiente y trae su **fuente de referencia**, un **esquema visual**, un laboratorio y un reto verificable. Estúdialos **en orden**: cada uno asume el anterior.

Mapa del programa

Índice

Cómo está construido cada módulo

Todos siguen la misma estructura (ver `MODULE_TEMPLATE.md`): objetivos medibles, resultados de aprendizaje, tabla de temas, modelo mental, **esquema visual** (diagramas Mermaid), conceptos con definiciones, **profundización** con casos reales y ejemplos numéricos,

laboratorio guiado, reto verificable con criterio de aceptación, errores frecuentes, seguridad y ética, **referencias a libros y fuentes primarias**, y navegación al módulo anterior y siguiente.

Para la dimensión profesional del ecosistema —cómo se construye una red, el stack, los equipos, las empresas y los modelos de negocio— consulta la sección Industria.

Las fuentes se detallan en la bibliografía central, que también recoge los **hitos recientes del ecosistema** (Merge, Dencun/EIP-4844, Pectra/EIP-7702) para mantener el material al día.

Ruta recomendada

Nivel	Módulos	Resultado
Orientación	00	Distinguir blockchain de una base de datos
Fundamentos	01–03	Criptografía, redes y consenso
Desarrollo	04–07	Bitcoin, EVM, contratos y una dApp
Profesional	08–11	Tokens, seguridad, oráculos y DAO
Avanzado	12–15	L2, interoperabilidad, ZK y arquitectura
Producción	16–18	Infraestructura real, caso de negocio e implementación en la empresa

Empieza por el Módulo 00 · Orientación.

🧭 Navegación

🏠 Programa · 📖 Bibliografía · 🧪 Laboratorios · ➡️ Módulo 00 · Orientación

00 · Orientación

> **Nivel:** Inicial · ⏱️ **Duración estimada:** 90 min · **Fuente:** *Mastering Blockchain* (Bashir) y *The Blockchain and the New Architecture of Trust* (Werbach) ⬅️ Currículo · 📚 Bibliografía 🧭 ⬅️ **Anterior:** 🏠 Programa · 📚 Índice · ➡️ **Siguiente:** 01 · Criptografía aplicada 📖 Glosario de términos · 🌱 ¿Nuevo en esto? Empieza aquí

🎯 Objetivos

- Distinguir con precisión los conceptos de blockchain, DLT, criptomoneda y contrato inteligente.
- Situar la descentralización como un espectro (técnico, político y arquitectónico), no como un valor binario.
- Aplicar un conjunto de preguntas de diseño para decidir si un problema justifica una blockchain.
- Reconocer cuándo una base de datos tradicional es la opción más adecuada.
- Documentar una decisión técnica con criterios explícitos y trazables.

📚 Resultados de aprendizaje

Al finalizar, el estudiante podrá:

1. **Diferenciar** blockchain, DLT, criptomoneda y contrato inteligente con ejemplos propios.
2. **Clasificar** un sistema según su grado de descentralización en varias dimensiones.
3. **Evaluar** un caso de uso con seis preguntas de decisión estructuradas.
4. **Justificar** por qué una base de datos centralizada suele bastar cuando una sola organización controla la escritura.
5. **Producir** una matriz de decisión defendible para tres casos reales.

🗺️ Temas

#	Tema	Por qué importa
1	Blockchain vs. DLT	Toda blockchain es un DLT, pero no todo DLT encadena bloques con hash.
2	Criptomoneda vs. token vs. contrato	Separar el activo del programa evita confusiones frecuentes.
3	Descentralización como espectro	Permite medir en vez de etiquetar "descentralizado".
4	Permisionadas vs. públicas	Define quién puede leer, escribir y validar.
5	Modelo de confianza	La blockchain reubica la confianza, no la elimina.
6	Cuándo NO usar blockchain	Evita sobreingeniería y costos innecesarios.
7	Costos y compensaciones	La redundancia y el consenso tienen un precio real.

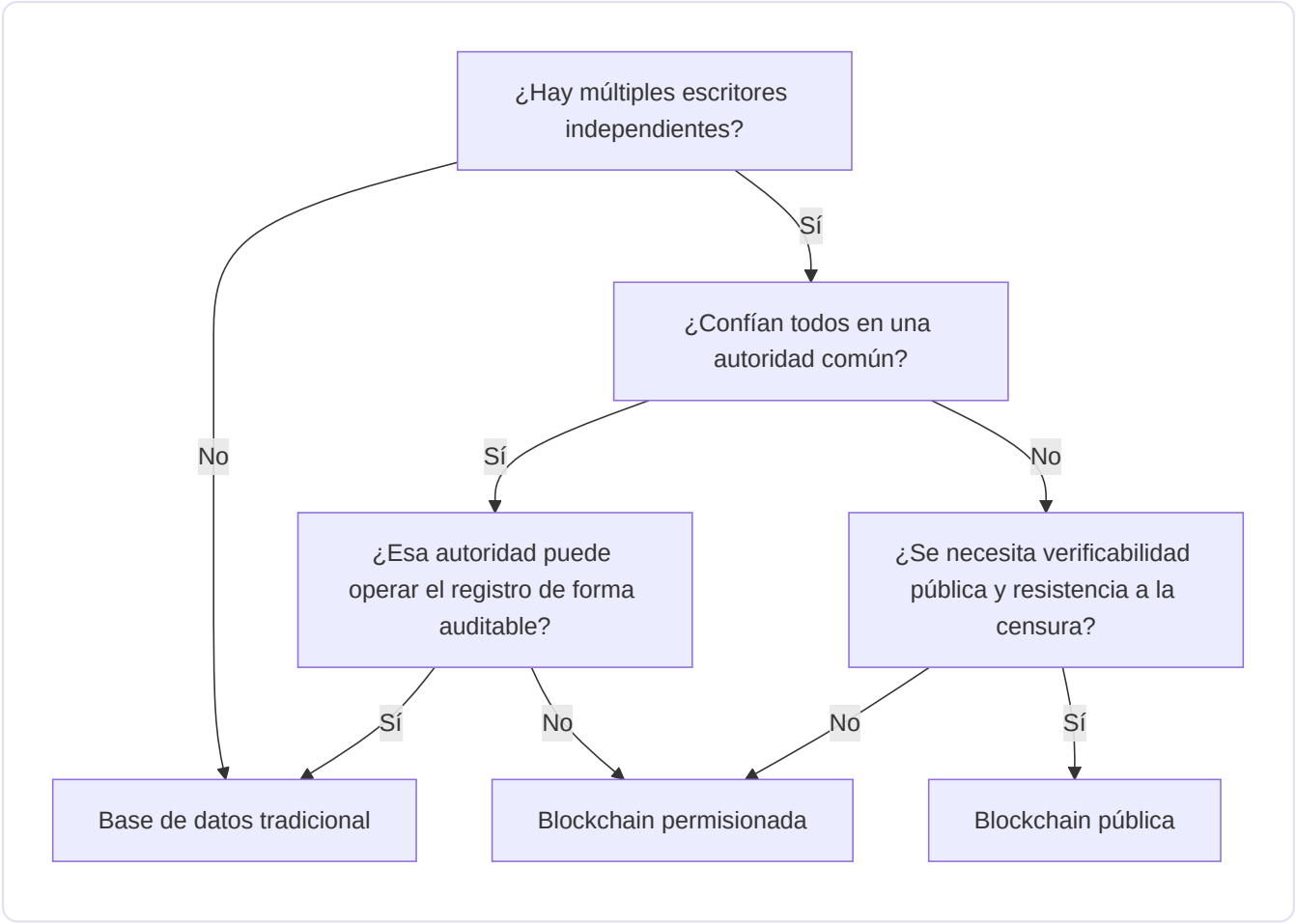
🧠 Modelo mental

Piensa en una blockchain como un libro contable compartido que muchas partes que no se conocen mantienen simultáneamente, donde cada página nueva referencia criptográficamente la anterior. Nadie es dueño del cuaderno y cambiar una página pasada obligaría a reescribir todas las siguientes ante la vista de todos. Esta analogía explica bien la inmutabilidad y la ausencia de un administrador único.

El límite de la analogía es importante: un cuaderno compartido no dice por sí mismo qué versión es la verdadera cuando dos personas escriben a la vez, ni impide que alguien registre un dato falso pero bien formado. Resolver "cuál historia es la válida" es trabajo del consenso (módulo 03), y garantizar que el dato de entrada sea cierto es un problema externo que la cadena no resuelve.

🧩 Esquema visual

El siguiente árbol de decisión resume las preguntas clave para determinar si un problema justifica una blockchain o si basta con una base de datos tradicional.



📖 Conceptos y definiciones

- **Blockchain**: estructura de datos de bloques enlazados por hash y replicada entre nodos; ejemplo: Bitcoin, Ethereum.
- **DLT (Distributed Ledger Technology)**: categoría amplia de registros distribuidos; no todos usan cadena de bloques (por ejemplo, los DAG).
- **Criptomoneda**: activo digital nativo de una red usado para transferir valor y pagar comisiones; ejemplo: ether (ETH).
- **Token**: unidad de valor definida por un contrato sobre una red existente, no nativa del protocolo base.
- **Contrato inteligente**: programa determinista que se ejecuta en la red y automatiza reglas sin intermediario.
- **Descentralización**: distribución del control entre múltiples partes independientes; se mide, no se afirma.
- **Red permisionada**: solo participantes autorizados validan o escriben; útil en consorcios.
- **Inmutabilidad**: dificultad práctica y económica de alterar el historial ya confirmado.
- **Resistencia a la censura**: propiedad de que ninguna parte pueda impedir transacciones válidas.
- **Oráculo (adelanto)**: mecanismo que introduce datos externos; la cadena no verifica su veracidad.

🔬 Profundización

El espectro de descentralización y el coeficiente de Nakamoto

Decir "esta red es descentralizada" no es medir nada. El **coeficiente de Nakamoto**, propuesto por Balaji Srinivasan y Leland Lee en 2017, ofrece una métrica concreta: es el número mínimo de entidades independientes que tendrían que coludirse para comprometer un subsistema crítico de la red (producción de bloques, stake, clientes de software, hosting, gobernanza).

Ejemplo numérico: imagina una red PoS con 10 validadores cuyos pesos de stake son 30, 20, 15, 10, 8, 7, 4, 3, 2 y 1 (total = 100). Si comprometer el consenso requiere controlar más del 33 % del stake, basta con que coludan los dos mayores validadores (30 + 20 = 50 > 33). El coeficiente de Nakamoto de ese subsistema es **2**, por muchos que sean los nodos totales. La lección: el número de nodos no mide descentralización; la distribución del poder sí. Además, el coeficiente debe calcularse por subsistema — una red puede tener miles de validadores y depender de 2 o 3 proveedores de nube o de un único equipo de desarrollo del cliente mayoritario.

Taxonomía de DLT más allá de la blockchain

No todo registro distribuido encadena bloques. La siguiente tabla resume las familias principales:

Familia	Estructura de datos	Ejemplo	Rasgo distintivo
Blockchain	Cadena lineal de bloques enlazados por hash	Bitcoin, Ethereum	Un solo historial canónico; bifurcaciones se resuelven por consenso
DAG	Grafo acíclico dirigido de transacciones	IOTA, Kaspa	Varias transacciones pueden confirmarse en paralelo sin bloques estrictos
Hashgraph	Grafo de eventos con "gossip sobre gossip"	Hedera	Consenso por timestamp virtual; patentado y con consejo de gobierno permisionado
Ledger permisionado sin cadena global	Canales o subledgers entre pares	Corda	Solo las partes de una transacción la ven; no hay difusión global

Todas comparten replicación y verificación criptográfica, pero difieren en el modelo de consenso, la privacidad y quién puede participar. "Es un DLT" no implica "es una blockchain pública".

Mini-caso real: TradeLens y el fracaso por gobernanza

TradeLens, la plataforma de blockchain permisionada para logística marítima creada por IBM y Maersk (lanzada en 2018), anunció su cierre en noviembre de 2022 y cesó

operaciones a inicios de 2023. La tecnología funcionaba: procesaba documentos de embarque y eventos logísticos reales. El fallo fue de **gobernanza y de incentivos**: las navieras competidoras de Maersk tenían pocos motivos para volcar sus datos operativos en una plataforma cofundada y percibida como controlada por su mayor rival, por mucho que la infraestructura fuera "neutral" sobre el papel. Sin la masa crítica de escritores independientes — justamente la primera pregunta del árbol de decisión — el registro compartido no aportaba más valor que una base de datos bien administrada. Moraleja verificable: antes de evaluar la tecnología, evalúa si los participantes que deben escribir tienen incentivos reales para hacerlo bajo esa gobernanza (véase el anuncio oficial de Maersk: https://www.maersk.com/news/articles/2022/11/29/maersk-and-ibm-to-discontinue-tradelens).

El árbol de decisión, aplicado a tres casos reales

Las seis preguntas del módulo se vuelven útiles cuando se aplican a casos concretos y **la respuesta sale "no" la mayoría de las veces**. Eso no es un fallo del ejercicio: es el resultado honesto.

La cadena de decisión, en orden. Basta un "no" para detenerse:

1. ¿Hay **varias organizaciones** que escriben, no solo que leen?
2. ¿**Desconfían** entre sí lo suficiente como para no aceptar la base de datos de una de ellas?
3. ¿Es **inaceptable** poner a un tercero neutral (un notario, una cámara de compensación) en medio?
4. ¿El dato es **nativo digital**, o depende de que alguien afirme algo del mundo físico?
5. ¿Se puede vivir con **latencia y coste** mayores que los de una base de datos?
6. ¿Existe **presupuesto continuo** para operar, auditar y cumplir normativa?

Caso	Dónde se detiene	Veredicto
Historial médico de un hospital	Pregunta 1: escribe una sola organización	Base de datos con auditoría y firma. Blockchain no aporta nada y añade riesgo de privacidad con datos irreversibles
Trazabilidad de café de origen	Pregunta 4: el dato entra cuando alguien escanea un saco	La cadena garantiza que el registro no se alteró, no que el saco contenga lo que dice. El problema real es el oráculo humano, y eso no lo resuelve la tecnología
Liquidación entre bancos que no se fían	Llega al final	Candidato legítimo: varios escritores, desconfianza mutua, dato nativo digital y presupuesto

El error de razonamiento más común es responder que sí a la 1 confundiendo *leer* con *escribir*. Que diez empresas consulten un sistema no las convierte en escritoras; si una sola decide qué se guarda, hay una autoridad central y la pregunta ya está respondida.

Y el segundo: dar por hecha la 6. Un piloto lo paga un presupuesto de innovación; la operación continua —nodos, auditorías, cumplimiento— necesita una línea permanente. La mayoría de los proyectos que se abandonan no fracasan técnicamente: se quedan sin quien pague el mantenimiento.

> 💡 **En una frase:** la pregunta correcta no es "¿puedo usar blockchain?" sino "¿quién escribe, y por qué no aceptan la base de datos de otro?". Casi siempre la respuesta cierra el caso en el primer paso.

▶ 🎓 **Si ya dominas esto** — el matiz que separa el análisis serio del entusiasmo

🧪 Laboratorio guiado

> 🧪 Estas prácticas están catalogadas y **resueltas paso a paso** en el catálogo de laboratorios.

Este módulo es de análisis: no ejecuta código. Construirás una matriz de decisión para tres casos.

1. Para cada caso responde las seis preguntas de decisión: (a) ¿hay múltiples escritores independientes?, (b) ¿existe una autoridad confiable disponible?, (c) ¿se necesita resistencia a la censura?, (d) ¿quién corrige errores?, (e) ¿qué datos jamás deberían ser públicos?, (f) ¿el beneficio supera el costo de operar una red distribuida?
2. Aplica las preguntas al **registro académico** de una universidad.
3. Aplica las preguntas a los **pagos internacionales** entre entidades sin confianza mutua.
4. Aplica las preguntas a un programa de **puntos de fidelidad** de una sola empresa.
5. Registra tu veredicto por caso en una tabla como la siguiente:

```
Caso | Escritores | Autoridad | Censura | Corrige | Datos privados | Veredicto
-----|------------|-----------|---------|---------|----------------|----------
...  | ...        | ...       | ...     | ...     | ...            | BD / Blockchain
```

6. Concluye para cada caso si una base de datos tradicional es preferible o si se justifica una blockchain.

📝 Reto verificable

Redacta una recomendación de una página por cada uno de los tres casos, respondiendo las seis preguntas y cerrando con una decisión.

Criterio de aceptación: para el caso de puntos de fidelidad la recomendación debe ser una base de datos tradicional, con la justificación explícita de que una sola organización controla la escritura y no requiere resistencia a la censura ni múltiples escritores independientes.

⚠️ Errores frecuentes

Síntoma	Causa y cómo comprobarlo
"Todo problema mejora con blockchain"	Falta el análisis de escritores/autoridad; revísalo con las seis preguntas.
Confundir blockchain con base de datos	No distingues replicación sin confianza vs. control único; contrasta quién valida.
Igualar criptomoneda con blockchain	La moneda es un uso; la cadena es la infraestructura. Sepáralos con ejemplos.
Asumir que "descentralizado" es binario	No mides dimensiones; clasifícalo en técnico, político y arquitectónico.
Creer que la cadena garantiza datos verdaderos	Ignoras el problema del oráculo; comprueba de dónde viene el dato.

🛡️ Seguridad y ética

- Trabaja siempre en entorno local o testnet; en este módulo no se usan claves ni fondos reales.
- No introduzcas datos personales reales en los ejercicios de análisis.
- Reconoce que registrar datos inmutables puede entrar en conflicto con el derecho al olvido y la privacidad.
- Evalúa el costo energético y operativo como parte de la ética de la decisión técnica.
- Documenta tus supuestos: una recomendación honesta expone sus límites.

🔗 Referencias

- Imran Bashir, *Mastering Blockchain* — https://www.packtpub.com/
- Kevin Werbach, *The Blockchain and the New Architecture of Trust*, MIT Press — https://mitpress.mit.edu/
- Arvind Narayanan et al., *Bitcoin and Cryptocurrency Technologies* — https://bitcoinbook.cs.princeton.edu/
- Fuente primaria: Satoshi Nakamoto, *Bitcoin: A Peer-to-Peer Electronic Cash System* — https://bitcoin.org/bitcoin.pdf

✅ Criterio de dominio

- Entregas la matriz de decisión de los tres casos con veredicto justificado.
- Explicas sin ambigüedad la diferencia entre blockchain, DLT, criptomoneda y contrato inteligente.
- Argumentas al menos un escenario en el que una base de datos tradicional es superior.

🧭 Navegación

⬅️ 🏠 Inicio del programa · 📚 Índice del currículo · ➡️ Módulo 01 · Criptografía aplicada

01 · Criptografía aplicada

> **Nivel:** Inicial · ⏱️ **Duración estimada:** 120 min · **Fuente:** *Serious Cryptography* (Aumasson) y *Introduction to Modern Cryptography* (Katz, Lindell) ⬅️ Currículo · 📚 Bibliografía 🧭 ⬅️ **Anterior:** 00 · Orientación · 📚 Índice · ➡️ **Siguiente:** 02 · Sistemas distribuidos y redes P2P 📖 Glosario de términos · 🌱 ¿Nuevo en esto? Empieza aquí

🎯 Objetivos

- Explicar las propiedades de una función hash: resistencia a preimagen, a colisiones y efecto avalancha.
- Diferenciar cifrado (confidencialidad) de firma digital (autenticación e integridad).
- Construir y verificar una prueba de inclusión sobre un árbol de Merkle.
- Identificar los riesgos operativos de la gestión de claves privadas.
- Justificar por qué un hash rápido es inadecuado para almacenar contraseñas.

📚 Resultados de aprendizaje

Al finalizar, el estudiante podrá:

1. **Describir** qué garantiza y qué no garantiza una función hash criptográfica.
2. **Distinguir** una firma digital de un cifrado, y clave pública de clave privada.
3. **Generar** una raíz de Merkle y **verificar** una prueba de inclusión.
4. **Diagnosticar** vulnerabilidades por reutilización de claves o aleatoriedad débil.
5. **Seleccionar** la primitiva adecuada (SHA-256, Keccak-256, ECDSA, Argon2) según el objetivo.

🗺️ Temas

#	Tema	Por qué importa
1	Funciones hash	Detectan cualquier cambio en los datos y anclan la integridad.
2	Preimagen y colisiones	Definen la seguridad real de un hash frente a ataques.
3	Criptografía asimétrica	Permite firmar y verificar sin compartir secretos.
4	Firmas ECDSA (secp256k1)	Es el esquema de firma usado por Bitcoin y Ethereum.
5	Árboles de Merkle	Resumen muchos elementos con pruebas pequeñas y verificables.
6	Derivación de contraseñas	Un hash rápido es inseguro; se necesitan funciones lentas.
7	Gestión de claves	La mayoría de los incidentes reales nacen de claves mal manejadas.

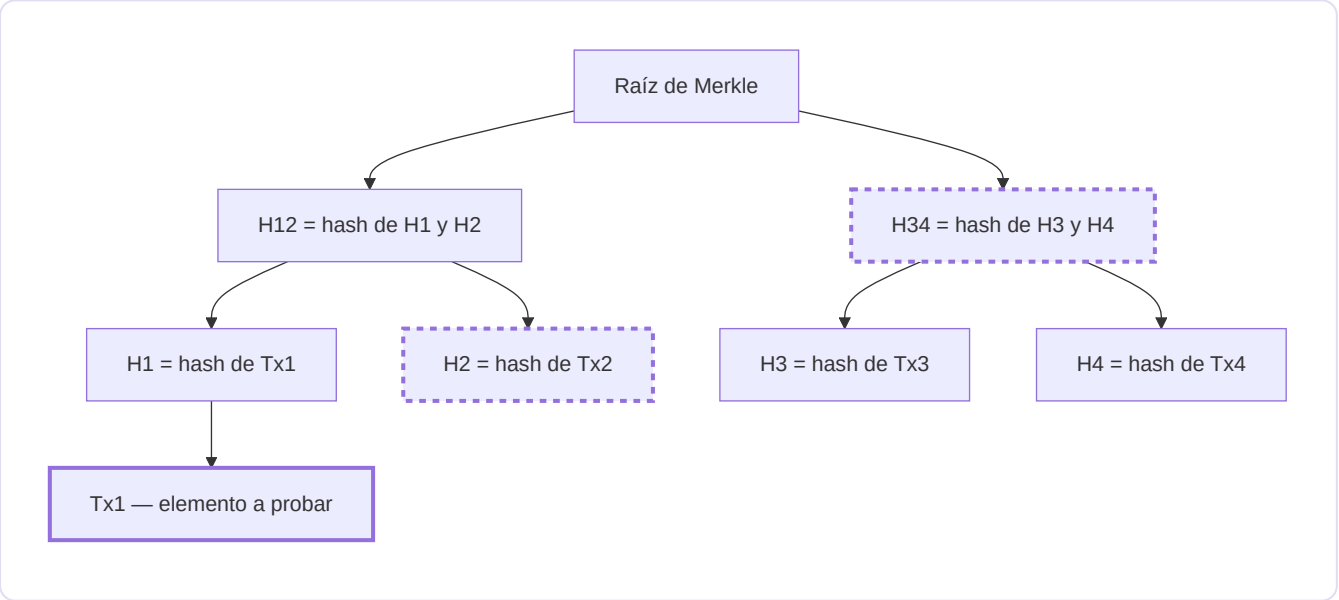
🧠 Modelo mental

Imagina un sello de lacre sobre un sobre. El hash es como una huella del contenido: si alguien altera una sola letra, la huella cambia por completo (efecto avalancha) y se nota la manipulación. La firma digital es como un sello personal imposible de falsificar sin tu anillo (clave privada), que cualquiera puede reconocer con la impronta pública. El árbol de Merkle es como un índice que permite probar que una carta está dentro de un archivo enorme mostrando solo unos pocos sellos, sin abrir todas las cajas.

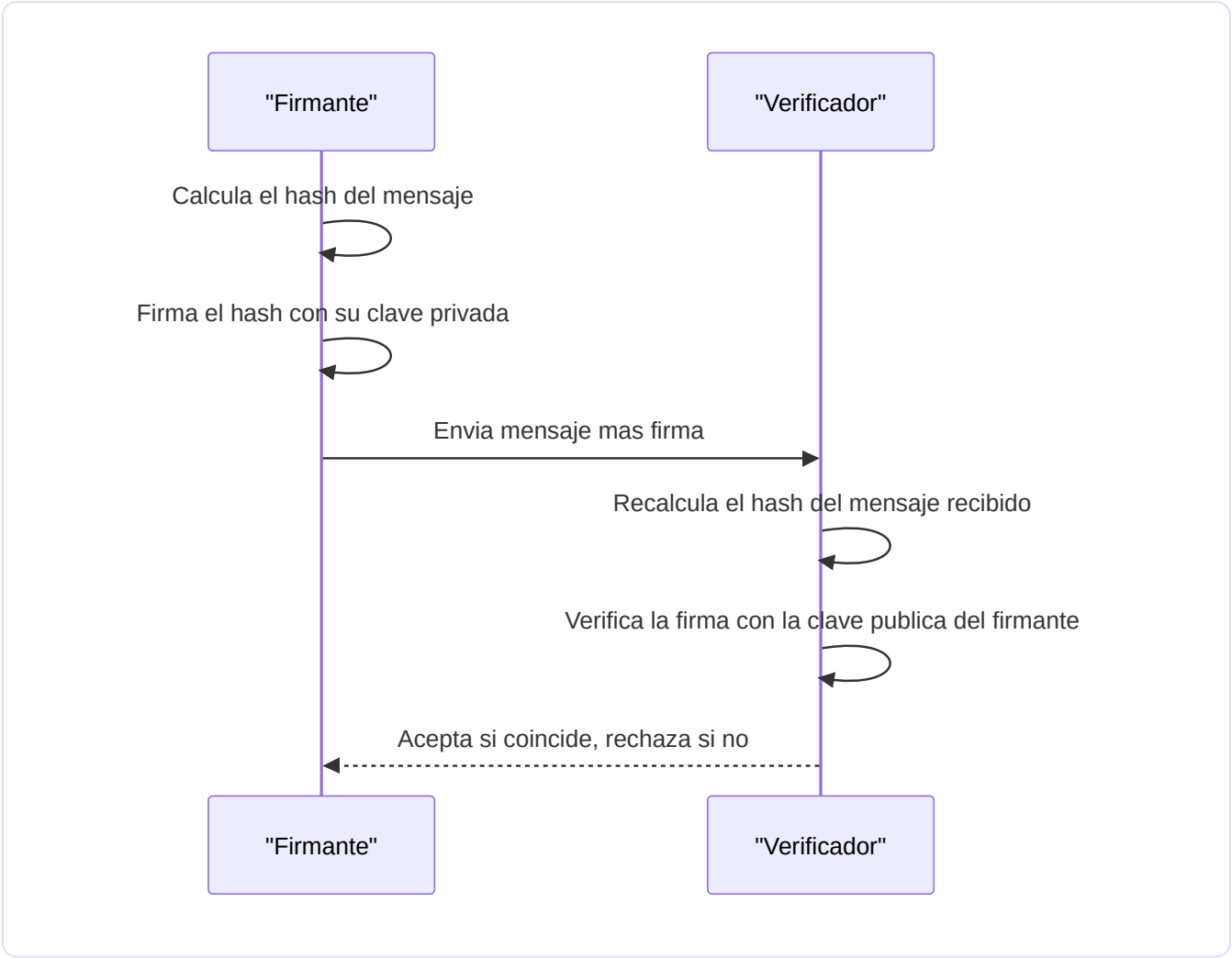
El límite de la analogía es que ninguna de estas primitivas decide qué historia es la verdadera. El hash detecta cambios pero no dice cuál versión debe prevalecer; la firma prueba quién autorizó un mensaje pero no si ese mensaje es la transacción correcta del sistema. Ordenar y validar el historial es tarea del consenso (módulo 03), no de la criptografía por sí sola.

🧩 Esquema visual

Árbol de Merkle de cuatro hojas: para probar que Tx1 está incluida basta con revelar H2 y H34 (la ruta de prueba, resaltada) y recomputar hacia la raíz.



Flujo de firma y verificación: la clave privada nunca viaja; el verificador solo necesita el mensaje, la firma y la clave pública.



📖 Conceptos y definiciones

- **Función hash criptográfica**: transforma datos de longitud arbitraria en un valor fijo; ejemplo: SHA-256, Keccak-256.
- **Resistencia a preimagen**: dado un hash, es inviable hallar una entrada que lo produzca.
- **Resistencia a colisiones**: es inviable hallar dos entradas distintas con el mismo hash.
- **Efecto avalancha**: un cambio mínimo en la entrada altera radicalmente la salida.
- **Clave pública / privada**: par asimétrico; la privada firma y descifra, la pública verifica y cifra.
- **Firma digital**: prueba de que el poseedor de una clave privada autorizó un mensaje concreto; ejemplo: ECDSA sobre secp256k1.
- **Cifrado**: protege la confidencialidad; no garantiza por sí solo autenticidad.
- **Árbol de Merkle**: árbol binario de hashes cuya raíz resume todo el conjunto de hojas.
- **Prueba de inclusión**: conjunto mínimo de hashes que demuestra que un elemento pertenece al árbol.

- **KDF (Argon2, bcrypt)**: función lenta y con sal para derivar claves de contraseñas y frenar la fuerza bruta.

🔬 Profundización

De clave pública a dirección Ethereum: Keccak-256 y checksum EIP-55

Una dirección Ethereum no es la clave pública, sino un resumen de ella. El proceso exacto:

1. De la clave privada (32 bytes) se deriva por multiplicación en la curva secp256k1 la clave pública sin comprimir: 64 bytes (coordenadas X e Y, sin el prefijo `0x04`).
2. Se calcula **Keccak-256** de esos 64 bytes (ojo: Keccak-256 original, no el SHA-3 estandarizado por NIST en FIPS 202, que difiere en el padding).
3. Se toman los **últimos 20 bytes** del hash: esa es la dirección.
4. Para el formato con checksum **EIP-55** se calcula Keccak-256 de la dirección en hexadecimal minúscula y se pone en mayúscula cada letra cuyo nibble correspondiente del hash sea ≥ 8.

Ejemplo verificable (vector de prueba del propio EIP-55): la dirección en minúsculas `0x5aaeb6053f3e94c9b9a09f33669435e7ef1beaed` se convierte con checksum en `0x5aAeb6053F3E94C9b9A09f33669435E7Ef1BeAed`. Un error de tipeo en una sola letra hace que el patrón de mayúsculas no cuadre, y cualquier wallet moderna rechaza la dirección: el checksum detecta erratas sin necesitar ningún registro central. Especificación: https://eips.ethereum.org/EIPS/eip-55.

Aleatoriedad débil en ECDSA: el caso Sony PlayStation 3

Cada firma ECDSA requiere un número efímero secreto, el **nonce k**, que debe ser único e impredecible por firma. Si k se repite en dos firmas con la misma clave, un observador puede plantear dos ecuaciones con dos incógnitas (k y la clave privada) y **despejar la clave privada con álgebra elemental**.

Caso real: en diciembre de 2010, el grupo fail0verflow mostró en el congreso 27C3 que Sony firmaba el software de la PlayStation 3 usando un k **constante** en todas las firmas. Con dos firmas cualesquiera bastó para recuperar la clave privada maestra de la consola, lo que permitió firmar software arbitrario como si fuera oficial. El mismo fallo ha drenado fondos reales en Bitcoin y Ethereum cuando wallets generaron nonces con mala entropía.

La defensa estándar es **RFC 6979**: derivar k de forma determinista a partir de la clave privada y del hash del mensaje mediante HMAC. Así, k es único por mensaje, reproducible y no depende de la calidad del generador aleatorio del dispositivo. Especificación: https://www.rfc-editor.org/rfc/rfc6979.

Hashes rápidos vs. KDF: cada primitiva tiene su propósito

Que SHA-256 sea velocísimo es una virtud para integridad y una catástrofe para contraseñas: un atacante con GPU prueba miles de millones de candidatos por segundo. Las KDF modernas se diseñan deliberadamente lentas y con consumo de memoria configurable.

Propiedad	Hash rápido (SHA-256, Keccak-256)	KDF (Argon2id, scrypt, bcrypt)
Objetivo	Integridad, punteros, Merkle, PoW	Derivar claves desde contraseñas
Velocidad deseada	Máxima	Deliberadamente lenta y ajustable
Uso de memoria	Mínimo	Alto y configurable (Argon2id, scrypt) para frenar GPU y ASIC
Sal	No aplica de serie	Obligatoria y única por usuario
Uso en blockchain	Bloques, transacciones, direcciones	Cifrado de keystores de wallets (scrypt en los keystore de Ethereum)

Regla práctica: si la entrada es de baja entropía (una contraseña humana), nunca un hash rápido a secas; siempre una KDF con sal y parámetros de costo actualizados (OWASP publica recomendaciones vigentes para Argon2id).

Por qué "inviable" no significa "imposible"

Cuando este módulo dice que encontrar una colisión de SHA-256 es *inviable*, no está diciendo que sea imposible: está diciendo que **cuesta más energía de la que hay disponible**. Conviene ver el número, porque es lo que convierte un acto de fe en un argumento.

Por la paradoja del cumpleaños, encontrar una colisión en un hash de n bits no cuesta 2^n intentos sino aproximadamente **2^(n/2)**. Para SHA-256:

```
2^128 ≈ 3,4 × 10^38 intentos
```

Pongamos ese número en perspectiva con el hardware más rápido que existe para hashear: toda la red de Bitcoin junta, que ronda los 10^21 hashes por segundo.

```
3,4 × 10^38 ÷ 10^21 hashes/s ≈ 3,4 × 10^17 segundos
                             ≈ 10 800 millones de años
```

Casi **cien veces la edad del universo**, usando todo el hardware de minería del planeta y sin parar. Por eso "inviable" es una afirmación económica y física, no una promesa matemática de imposibilidad.

Y por eso mismo el tamaño importa tanto. Cada bit que se le quita al hash divide el trabajo por la raíz cuadrada de dos:

Hash	Bits	Colisión (2^(n/2))	Estado
MD5	128	2^64	**Roto**: colisiones en segundos en un portátil
SHA-1	160	2^80	**Roto**: colisión real demostrada en 2017 (SHAttered)
SHA-256	256	2^128	Sin ataques prácticos conocidos

SHA-1 no cayó porque apareciera un fallo repentino: cayó porque 2^80 dejó de ser inalcanzable cuando el hardware avanzó y alguien decidió pagar el cómputo. **La criptografía no se rompe de golpe; se erosiona.** Esa es la razón de que los protocolos serios prevean cómo migrar de algoritmo antes de necesitarlo.

> 💡 **En una frase:** "seguro" en criptografía significa "demasiado caro de romper hoy". Es una afirmación con fecha, no una garantía permanente.

▶ 🎓 **Si ya dominas esto** — precisiones que importan al implementar

🧪 Laboratorio guiado

> 🧪 Estas prácticas están catalogadas y **resueltas paso a paso** en el catálogo de laboratorios.

1. Ejecuta el laboratorio de hashing y observa el efecto avalancha al cambiar un byte:

```
pnpm lab:hash
```

2. Ejecuta el laboratorio de árboles de Merkle para construir una raíz y una prueba de inclusión:

```
pnpm lab:merkle
```

3. Modifica una transacción de una hoja y vuelve a calcular la raíz; anota cómo cambia respecto de la original.
4. Verifica que una prueba de inclusión válida deja de serlo tras la modificación.

5. Ejecuta la batería de pruebas del repositorio para confirmar tu implementación:

```
pnpm test
```

📝 Reto verificable

Toma un conjunto de transacciones, construye su raíz de Merkle, altera una transacción y documenta el cambio de raíz junto con una prueba de inclusión antes y después.

Criterio de aceptación: demuestras que al modificar una transacción cambia la raíz de Merkle y que la prueba de inclusión original ya no valida contra la nueva raíz; `pnpm test` pasa sin errores.

⚠️ Errores frecuentes

Síntoma	Causa y cómo comprobarlo
Confundir cifrar con firmar	Usas la primitiva equivocada; verifica si buscas confidencialidad o autenticación.
Contraseñas con SHA-256 "a secas"	Hash rápido sin sal; comprueba tiempo de cómputo y adopta Argon2/bcrypt.
Firmar bytes distintos a lo mostrado	La interfaz muestra algo y firmas otra cosa; compara el mensaje real byte a byte.
Reutilizar el nonce en ECDSA	Aleatoriedad débil filtra la clave privada; revisa la fuente de entropía.
Tratar una dirección como identidad legal	Una dirección es un pseudónimo; no prueba quién es la persona.

🛡️ Seguridad y ética

- Trabaja solo en local o testnet; nunca uses claves privadas ni fondos reales en el laboratorio.
- Genera claves de práctica desechables y no las reutilices fuera del ejercicio.
- Nunca subas claves privadas, semillas ni archivos `.env` al control de versiones.
- Usa siempre aleatoriedad criptográficamente segura para generar claves y nonces.
- Recuerda que la seudonimia no es anonimato: las direcciones son trazables en cadena.

🔗 Referencias

- Jean-Philippe Aumasson, *Serious Cryptography*, 2.ª ed. — https://nostarch.com/serious-cryptography-2nd-edition
- Jonathan Katz y Yehuda Lindell, *Introduction to Modern Cryptography* — https://www.cs.umd.edu/~jkatz/imc.html
- Ferguson, Schneier y Kohno, *Cryptography Engineering* — https://www.schneier.com/books/cryptography-engineering/
- Fuente primaria: NIST, FIPS 180-4 *Secure Hash Standard (SHA)* — https://csrc.nist.gov/

✅ Criterio de dominio

- Construyes y verificas una prueba de Merkle y explicas cómo cambia la raíz ante una alteración.
- Distingues correctamente cifrado de firma y justificas la primitiva elegida.
- Argumentas por qué las contraseñas requieren una KDF lenta en lugar de un hash rápido.

🧭 Navegación

⬅️ Módulo 00 · Orientación · 📚 Índice del currículo · ➡️ Módulo 02 · Sistemas distribuidos y redes P2P

02 · Sistemas distribuidos y redes P2P

> **Nivel:** Inicial-Intermedio · ⏱️ **Duración estimada:** 120 min · **Fuente:** *Introduction to Reliable and Secure Distributed Programming* (Cachin, Guerraoui, Rodrigues) y *Distributed Systems* (Tanenbaum, van Steen) ⬅️ Currículo · 📚 Bibliografía 🧭 ⬅️ **Anterior:** 01 · Criptografía aplicada · 📚 Índice · ➡️ **Siguiente:** 03 · Consenso 📖 Glosario de términos · 🌱 ¿Nuevo en esto? Empieza aquí

🎯 Objetivos

- Explicar los efectos de la latencia, las particiones de red y la replicación en un sistema distribuido.
- Diferenciar tolerancia a fallos por caída de la tolerancia a fallos bizantinos.
- Relacionar los teoremas CAP y FLP con las decisiones de diseño de una red P2P.
- Distinguir finalidad probabilística de finalidad económica.
- Analizar la propagación por gossip, el mempool y la resistencia Sybil.

📚 Resultados de aprendizaje

Al finalizar, el estudiante podrá:

1. **Definir** seguridad (safety) y vivacidad (liveness) y dar un ejemplo de cada una.
2. **Explicar** la compensación del teorema CAP ante una partición de red.
3. **Justificar** por qué el resultado FLP limita el consenso determinista en redes asíncronas.
4. **Simular** el comportamiento de una red de cinco nodos ante fallos y desórdenes.
5. **Comparar** finalidad probabilística y económica con casos concretos.

🗺️ Temas

#	Tema	Por qué importa
1	Latencia y asincronía	Los mensajes tardan y no llegan en orden; el diseño debe tolerarlo.
2	Particiones de red	La red puede dividirse; CAP obliga a elegir consistencia o disponibilidad.
3	Replicación y relojes	Sin reloj global, ordenar eventos requiere relojes lógicos.
4	Tolerancia bizantina	Los nodos pueden mentir, no solo caerse.
5	Mempool y gossip	Las transacciones se difunden por propagación entre pares.
6	Resistencia Sybil	Sin costo por identidad, un atacante crea nodos falsos ilimitados.
7	Finalidad	Determina cuándo una transacción se considera irreversible.

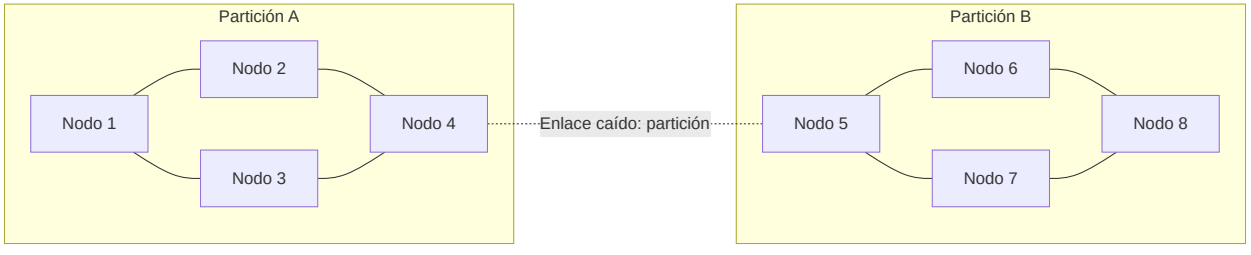
🧠 Modelo mental

Imagina una red de corresponsales que se envían cartas por correo postal sin un reloj compartido: las cartas se cruzan, algunas se pierden, otras llegan en desorden y alguno podría enviar versiones contradictorias a distintos destinatarios. Para ponerse de acuerdo sobre "qué pasó y en qué orden" no basta con esperar; hacen falta reglas que funcionen aunque haya cartas perdidas (particiones) y aunque algún corresponsal mienta (fallo bizantino). Esta imagen explica por qué el acuerdo distribuido es difícil incluso sin adversarios.

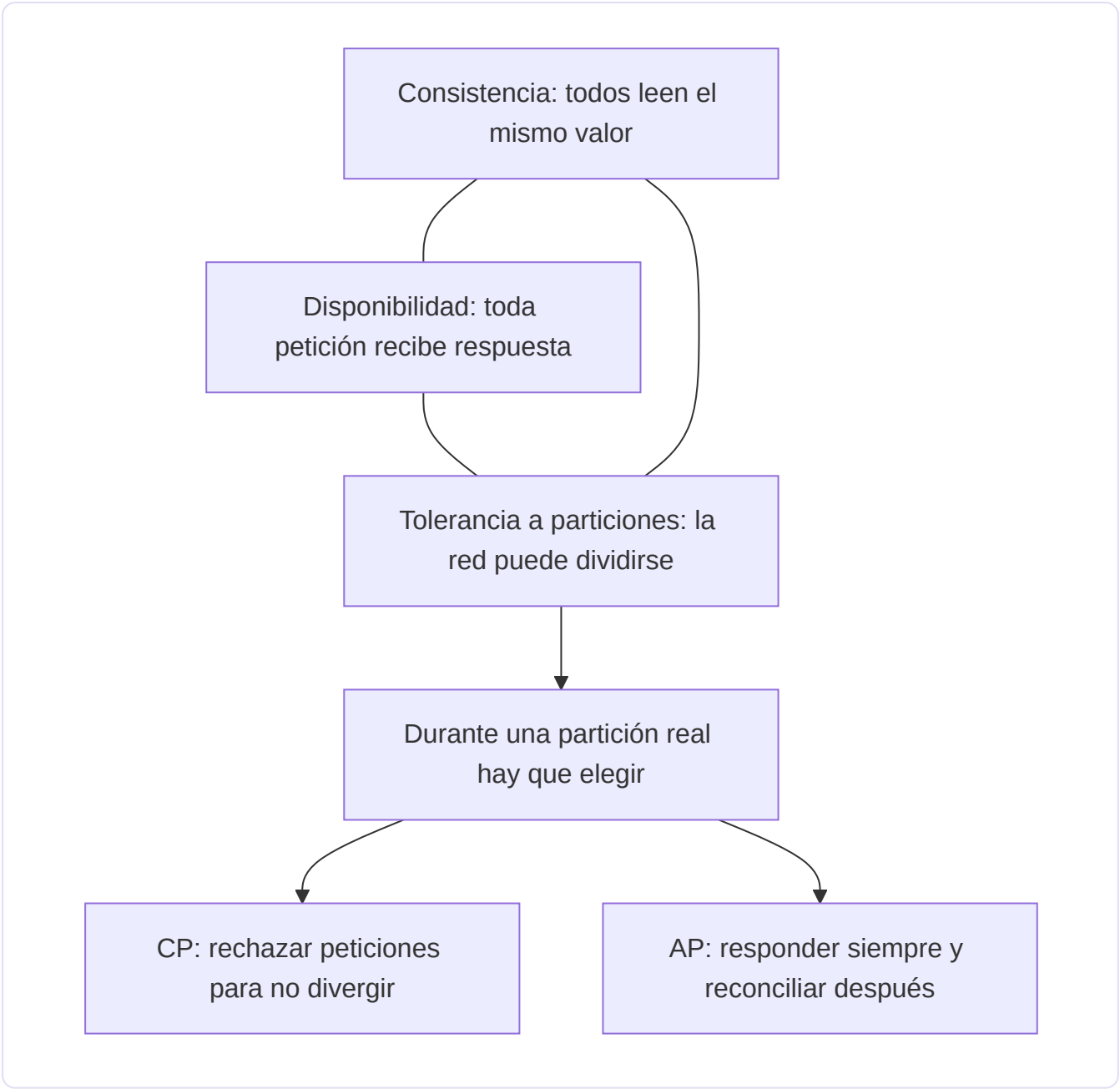
El límite de la analogía es que los corresponsales humanos pueden llamarse por teléfono para desempatar; en un sistema abierto no existe ese canal privilegiado ni una autoridad central. Además, "descentralizado" no significa ausencia de gobierno: siempre hay reglas de protocolo, incentivos y procesos sociales que gobiernan el sistema, aunque no haya un administrador único.

🧩 Esquema visual

Propagación por gossip en una red de ocho nodos: cada nodo reenvía a sus pares, pero el enlace caído entre los nodos 4 y 5 parte la red en dos mitades que dejan de verse.



El triángulo CAP: las tres propiedades deseables y la elección forzada cuando la partición ocurre de verdad.



📖 Conceptos y definiciones

- **Seguridad (safety)**: nada malo ocurre; nunca se acepta un estado inválido o contradictorio.
- **Vivacidad (liveness)**: algo bueno acaba ocurriendo; el sistema sigue progresando.
- **Teorema CAP**: ante una partición hay que elegir entre consistencia y disponibilidad.
- **Resultado FLP**: en un sistema asíncrono no hay consenso determinista garantizado si puede fallar un proceso.
- **Fallo bizantino**: un nodo se comporta de forma arbitraria o maliciosa, no solo cae.
- **Mempool**: conjunto de transacciones pendientes que cada nodo mantiene antes de incluirlas.

- **Gossip**: protocolo de difusión en el que cada nodo reenvía a sus pares.
- **Ataque Sybil**: un atacante crea muchas identidades falsas para ganar influencia.
- **Finalidad probabilística**: la reversión es cada vez menos probable con el tiempo (Bitcoin).
- **Finalidad económica**: revertir implica una pérdida económica prohibitiva (Ethereum PoS).

🔬 Profundización

¿Qué elige una blockchain en términos CAP?

Una blockchain pública de tipo Nakamoto elige, en la práctica, **disponibilidad con consistencia eventual**: durante una partición, cada mitad de la red sigue produciendo bloques sobre su propia vista, y al reunificarse la regla de elección de cadena descarta una de las ramas — eso es un **reorg**. Los nodos que consideraban confirmadas las transacciones de la rama perdedora ven cómo vuelven al mempool. Por eso la finalidad de Bitcoin es probabilística: más profundidad, menos probabilidad de reversión, pero nunca cero.

Caso real verificable: el 25 de mayo de 2022, la Beacon Chain de Ethereum sufrió un **reorg de 7 bloques** — siete bloques ya propuestos fueron descartados de la cadena canónica. No hubo ataque: fue una consecuencia de la propagación desigual entre clientes actualizados y no actualizados en la implementación del boost del fork choice. La lección de sistemas distribuidos es doble: (1) incluso sin adversarios, la latencia y la heterogeneidad de clientes bastan para producir divergencias temporales; (2) el protocolo se diseña para que esas divergencias se resuelvan solas — la capa de finalidad (checkpoints de Casper FFG, módulo 03) marca el punto tras el cual un reorg ya no es una molestia sino una catástrofe económica. Análisis técnico: https://barnabe.substack.com/p/pos-ethereum-reorg.

Modelos de sincronía y por qué FLP no condena el consenso

El resultado FLP (Fischer, Lynch y Paterson, 1985) prueba que en un sistema **asíncrono puro** — sin ninguna cota en los retrasos de mensajes — no existe un algoritmo determinista que garantice consenso si un solo proceso puede fallar. Suena letal, pero se aplica a un modelo extremo. Los tres modelos habituales:

Modelo	Supuesto sobre los retrasos	Consecuencia práctica
Síncrono	Existe una cota conocida para todo retraso	Protocolos simples, pero el supuesto es irreal en Internet
Parcialmente síncrono	La cota existe pero se desconoce, o rige solo tras un instante GST	El estándar de diseño real: PBFT, Tendermint y Gasper operan aquí
Asíncrono	Ningún límite en los retrasos	Aplica FLP: imposibilidad de consenso determinista

Las salidas de la trampa FLP son tres, y todas se usan: **sincronía parcial** (esperar timeouts y reintentar rondas, como PBFT), **aleatorización** (el sorteo del líder en PoW y PoS rompe la simetría que FLP explota) y **relajar la garantía** (aceptar finalidad probabilística en vez de acuerdo instantáneo). FLP dice que no puedes tener siempre terminación garantizada en el peor caso adversarial; no dice que el consenso falle en las redes reales, donde los periodos de buen comportamiento abundan.

Resistencia Sybil: qué recurso encarece las identidades

Crear una identidad en una red P2P abierta es gratis; por eso el voto "un nodo, un voto" es inviable. Cada mecanismo anti-Sybil ancla el peso del voto a un recurso costoso:

Mecanismo	Recurso escaso	Costo de ataque	Límite práctico
Proof of Work	Cómputo y energía	Adquirir u alquilar más hash que la red honesta, de forma sostenida	Hardware y electricidad tienen mercados observables; el costo es externo y recurrente
Proof of Stake	Capital bloqueado en el protocolo	Comprar y arriesgar una fracción grande del stake, expuesta a slashing	El propio ataque destruye el valor del capital atacante; costo interno
Identidad (PoA, consorcios)	Autorización verificada fuera de cadena	Corromper o suplantar a los miembros autorizados	No sirve para redes abiertas; reintroduce una autoridad de admisión

La conclusión conecta con el módulo 03: el mecanismo de consenso no "elige al mejor", solo hace que fingir ser muchos resulte más caro que el beneficio esperado del ataque.

CAP con un ejemplo que se puede seguir a mano

CAP suena abstracto hasta que se ve el instante de la decisión. Dos centros de datos, uno en Santiago y otro en Madrid, replican el mismo saldo: **100 €**. Se corta el enlace entre ambos y siguen recibiendo peticiones.

Llega una retirada de 80 € a cada lado, casi a la vez. Los dos caminos posibles:

Opción CP (priorizar consistencia): cada centro se pregunta si puede confirmar con la mitad de la red incomunicada. La respuesta es no.

```
Santiago → "no puedo confirmar" → el cliente no puede sacar dinero
Madrid   → "no puedo confirmar" → el cliente no puede sacar dinero
Saldo al reconectar: 100 €. Correcto, pero el servicio estuvo caído.
```

Opción AP (priorizar disponibilidad): cada centro responde con lo que sabe.

```
Santiago → entrega 80 € → apunta saldo 20 €
Madrid   → entrega 80 € → apunta saldo 20 €
Al reconectar: se entregaron 160 € de una cuenta que tenía 100.
```

Ese descubierto de 60 € **no es un bug del código**: es el precio explícito de haber elegido responder durante la partición. Alguien lo asumirá —el banco, el comercio o el cliente—, y esa decisión se toma en el diseño, no en el incidente.

Aquí es donde encaja una blockchain pública: **elige CP**. Durante una partición no confirma nada de forma definitiva, y de ahí sale la recomendación de esperar confirmaciones. Cuando alguien dice "la transacción tarda", en realidad está describiendo el precio de no permitir jamás un doble gasto.

> 💡 **En una frase:** en una partición no eliges entre bueno y malo, sino entre negarte a responder o arriesgarte a contradecirte. No hay tercera opción, y elegir por omisión es elegir igual.

▶ 🎓 **Si ya dominas esto** — las precisiones que suelen faltar

🧪 Laboratorio guiado

> 🧪 Estas prácticas están catalogadas y **resueltas paso a paso** en el catálogo de laboratorios.

Este módulo es un ejercicio de diseño y simulación en papel; no requiere código específico. Puedes usar `pnpm test` para correr las pruebas del repositorio si tu diseño incluye un componente verificable.

1. Dibuja una red de cinco nodos conectados como pares (P2P) e indica sus canales.
2. Simula un **nodo desconectado**: describe cómo el resto continúa y cómo se reincorpora luego.
3. Simula **dos mensajes válidos que llegan en distinto orden** en diferentes nodos y anota el conflicto.
4. Simula un **nodo malicioso** que envía información contradictoria a distintos pares.
5. Simula una **partición temporal** que divide la red en dos mitades y su posterior reunificación.
6. Para cada escenario registra qué propiedades priorizas en una tabla:

```
Escenario           | Safety | Liveness | Finalidad | Disponibilidad
--------------------|--------|----------|-----------|---------------
Nodo desconectado   | ...    | ...      | ...       | ...
Desorden de mensajes| ...    | ...      | ...       | ...
Nodo malicioso      | ...    | ...      | ...       | ...
Partición temporal  | ...    | ...      | ...       | ...
```

📝 Reto verificable

Entrega el diseño de la red de cinco nodos con los cuatro escenarios simulados y la tabla de propiedades priorizadas por escenario.

Criterio de aceptación: en el escenario de partición de red identificas explícitamente la compensación del teorema CAP (elegir consistencia o disponibilidad) y justificas cuál propiedad sacrificas y por qué.

⚠️ Errores frecuentes

Síntoma	Causa y cómo comprobarlo
Suponer entrega ordenada y confiable	Ignoras la asincronía; revisa qué pasa si un mensaje llega tarde o nunca.
Creer que se puede tener CAP completo	Confundes el enunciado; comprueba qué cedes durante una partición.
Igualar fallo por caída con fallo bizantino	Un nodo caído no miente; un bizantino sí. Distingue el modelo de fallo.
Confiar en relojes del sistema para ordenar	Sin reloj global hay deriva; usa relojes lógicos o el orden del protocolo.
Pensar que descentralizado es sin gobierno	Siempre hay reglas e incentivos; identifica quién decide los cambios.

🛡️ Seguridad y ética

- Realiza las simulaciones en local; no despliegues nodos con fondos ni claves reales.
- No uses datos personales reales en los mensajes simulados.
- Considera que la resistencia Sybil tiene un costo: entiende quién lo paga y cómo.
- Reflexiona sobre el gobierno del protocolo: la descentralización técnica no elimina el poder, lo distribuye.
- Documenta los supuestos de red (síncrona, parcialmente síncrona o asíncrona) de cada escenario.

🔗 Referencias

- Cachin, Guerraoui y Rodrigues, *Introduction to Reliable and Secure Distributed Programming* — https://link.springer.com/book/10.1007/978-3-642-15260-3
- Tanenbaum y van Steen, *Distributed Systems* — https://www.distributed-systems.net/
- Martin Kleppmann, *Designing Data-Intensive Applications* — https://dataintensive.net/
- Fuente primaria: Leslie Lamport, *The Part-Time Parliament* (Paxos) — https://lamport.azurewebsites.net/pubs/lamport-paxos.pdf

✅ Criterio de dominio

- Entregas el diseño de la red y los cuatro escenarios con su tabla de propiedades.
- Explicas la compensación CAP y la limitación FLP con tus propias palabras.
- Diferencias correctamente finalidad probabilística de finalidad económica.

🧭 Navegación

⬅️ Módulo 01 · Criptografía aplicada · 📚 Índice del currículo · ➡️ Módulo 03 · Consenso

03 · Consenso

> **Nivel:** Intermedio · ⏱️ **Duración estimada:** 120 min · **Fuente:** whitepaper de Bitcoin (Nakamoto) y *Practical Byzantine Fault Tolerance* (Castro, Liskov) ⬅️ Currículo · 📚 Bibliografía 🧭 ⬅️ **Anterior:** 02 · Sistemas distribuidos y redes P2P · 📚 Índice · ➡️ **Siguiente:** 04 · Bitcoin 📖 Glosario de términos · 🌱 ¿Nuevo en esto? Empieza aquí

🎯 Objetivos

- Comparar Proof of Work, Proof of Stake, BFT y Proof of Authority más allá de energía y velocidad.
- Explicar la regla de la cadena más larga de Nakamoto y su finalidad probabilística.
- Describir el consenso de Ethereum tras The Merge (Casper FFG + LMD-GHOST, Gasper) y el slashing.
- Relacionar la cota 3f+1 de PBFT con la tolerancia a fallos bizantinos.
- Distinguir el recurso anti-Sybil de cada mecanismo y su tipo de finalidad.

📚 Resultados de aprendizaje

Al finalizar, el estudiante podrá:

1. **Comparar** cuatro mecanismos de consenso según recurso anti-Sybil, finalidad y uso típico.
2. **Explicar** por qué la regla de la cadena más larga ofrece finalidad probabilística.
3. **Describir** cómo el slashing genera finalidad económica en Ethereum PoS.
4. **Justificar** la cota 3f+1 de PBFT para tolerar f nodos bizantinos.
5. **Interpretar** un laboratorio de PoW sin confundirlo con la seguridad de una red real.

🗺️ Temas

#	Tema	Por qué importa
1	Problema del consenso	Acordar un único historial entre partes que no confían entre sí.
2	Proof of Work	Ancla la seguridad en cómputo y energía; base de Bitcoin.
3	Cadena más larga (Nakamoto)	Define cómo se resuelven las bifurcaciones sin autoridad.
4	Proof of Stake (Ethereum)	Ancla la seguridad en capital bloqueado y penalizaciones.
5	Casper FFG + LMD-GHOST (Gasper)	Combina elección de cadena con finalidad periódica.
6	PBFT y cota 3f+1	Da finalidad rápida en conjuntos conocidos de validadores.
7	Proof of Authority	Consenso por identidades autorizadas en redes permisionadas.

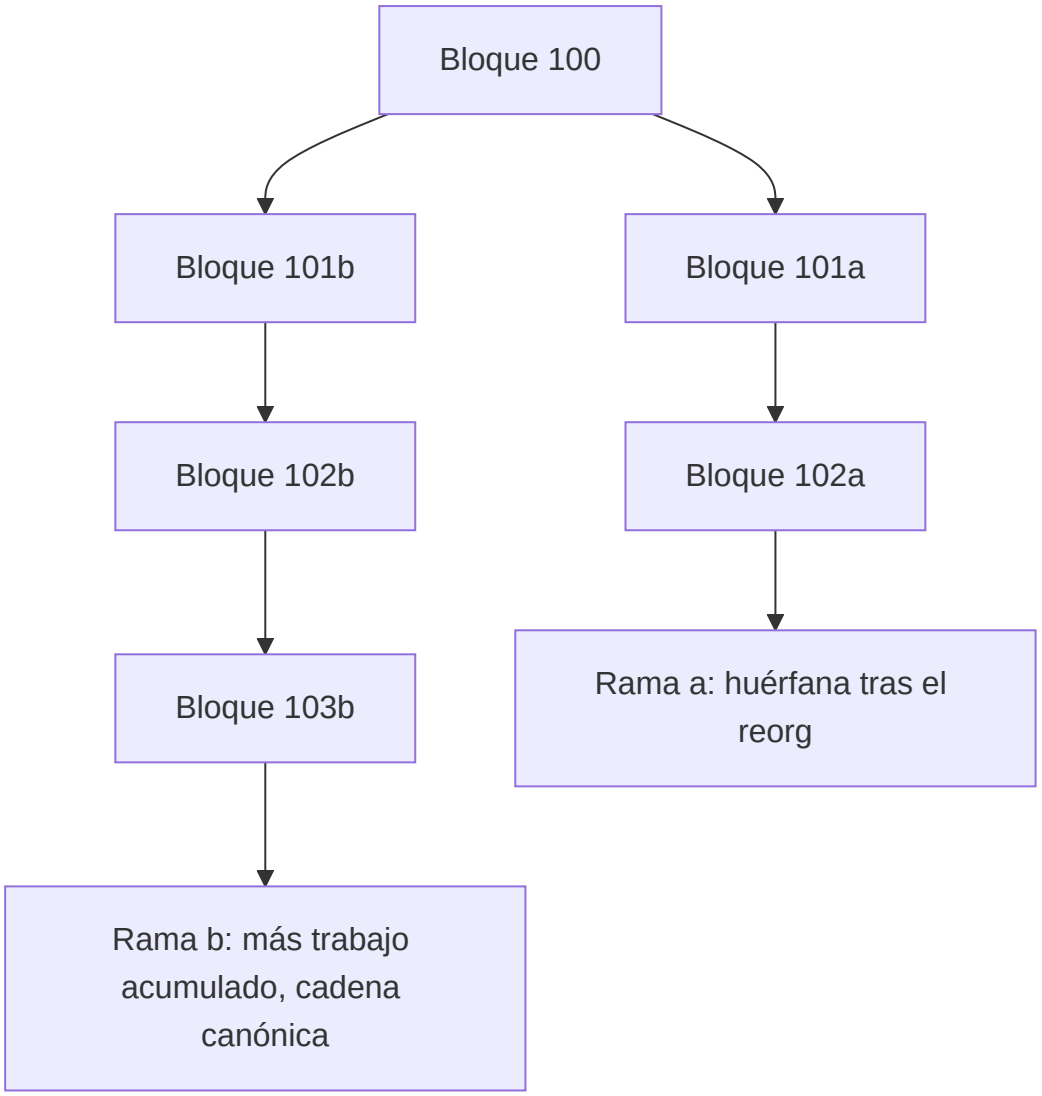
🧠 Modelo mental

Piensa en el consenso como el modo en que una multitud sin líder decide cuál de dos relatos contradictorios es el oficial. En Proof of Work la multitud "vota" gastando trabajo real: la versión respaldada por más esfuerzo acumulado gana, y rehacerla costaría repetir todo ese esfuerzo. En Proof of Stake los validadores ponen un depósito y firman la historia; si mienten pierden su depósito (slashing), de modo que la honestidad es la opción económicamente racional. En PBFT un grupo conocido vota en rondas y basta que más de dos tercios sean honestos para fijar el resultado de inmediato.

El límite de la analogía del "voto" es que no se trata de personas iguales: el peso proviene del recurso escaso (cómputo, capital o pertenencia autorizada), que es justamente el mecanismo anti-Sybil. Además, no todos los mecanismos dan el mismo tipo de garantía: la cadena más larga ofrece finalidad probabilística (cada bloque adicional reduce la probabilidad de reversión), mientras que Gasper y PBFT ofrecen finalidad explícita una vez alcanzado el quórum.

🧩 Esquema visual

Elección de cadena por regla longest-chain: dos mineros hallan bloque a la vez y la red se bifurca; cuando la rama b acumula más trabajo, la rama a se descarta (reorg) y sus transacciones vuelven al mempool.



Pipeline del Proof of Stake de Ethereum: de la propuesta en un slot a la finalización del checkpoint, dos épocas después.

📖 Conceptos y definiciones

- **Mecanismo de consenso**: reglas para que nodos independientes acuerden un mismo historial.
- **Recurso anti-Sybil**: bien escaso que da peso a un participante (cómputo, capital o identidad autorizada).
- **Proof of Work (PoW)**: se compite resolviendo un puzle de hash; el ganador propone el bloque.

- **Regla de la cadena más larga**: se sigue la cadena con más trabajo acumulado; base de la finalidad probabilística.
- **Proof of Stake (PoS)**: los validadores depositan capital y son elegidos para proponer y atestiguar bloques.
- **Slashing**: penalización que confisca parte del depósito ante conductas maliciosas; sostiene la finalidad económica.
- **Gasper**: combinación de Casper FFG (finalidad) y LMD-GHOST (elección de cadena) usada por Ethereum.
- **PBFT**: protocolo que tolera f nodos bizantinos con al menos 3f+1 participantes y da finalidad inmediata.
- **Proof of Authority (PoA)**: validadores identificados y autorizados firman bloques; típico en redes permisionadas.
- **The Merge (2022)**: transición de Ethereum de PoW a PoS; desde entonces Ethereum es Proof of Stake.

🔬 Profundización

Economía de la seguridad: qué cuesta atacar cada mecanismo

En PoW, un ataque del 51 % exige controlar la mayoría del hashrate de forma sostenida. Contra Bitcoin es hoy inviable en la práctica: su hashrate se mide en cientos de exahashes por segundo (el valor exacto es volátil — consúltalo en vivo), y no existe mercado de alquiler capaz de suministrar esa capacidad; habría que fabricar y alimentar millones de ASIC. El riesgo real lo sufren las cadenas PoW pequeñas cuyo hashrate sí cabe en los mercados de alquiler: **Ethereum Classic sufrió ataques del 51 % verificados en enero de 2019 y tres veces en agosto de 2020**, con dobles gastos que en un solo incidente superaron los 5 millones de dólares. La seguridad PoW no es una propiedad del algoritmo sino del tamaño económico de la red concreta.

En PoS los umbrales son distintos: con **1/3 del stake** un atacante puede impedir la finalización (ataque a la vivacidad), y revertir un checkpoint ya finalizado exige que al menos **2/3 del stake** firme historias contradictorias — lo que implica que como mínimo 1/3 queda probadamente equivocado y es **slasheable**. En Ethereum hay decenas de millones de ETH en stake (la cifra exacta y su valor en dólares son volátiles — consúltalos en vivo, por ejemplo en https://beaconcha.in/); el costo de romper la finalidad no es alquilar un recurso externo, sino comprar y luego **destruir** una fracción enorme de ese capital. La diferencia clave de orden de magnitud no está solo en el precio de entrada, sino en que en PoW el hardware sobrevive al ataque y en PoS el capital atacante se quema.

Ataques clásicos a PoS y sus mitigaciones

- **Nothing-at-stake**: en un PoS ingenuo, votar por todas las ramas de una bifurcación no cuesta nada (no hay energía que dividir), así que la estrategia racional sería

apoyar todo a la vez y cobrar en la rama ganadora. Mitigación: el **slashing** convierte la firma de bloques contradictorios en una conducta cara — en Ethereum, un validador que firma dos cabeceras en conflicto pierde parte de su depósito y es expulsado.
- **Long-range**: un atacante que controló claves de validadores antiguos (o las compró baratas una vez retirado su stake) puede reescribir la historia desde un punto lejano del pasado, porque firmar bloques antiguos no cuesta nada hoy. Mitigación: **checkpoints de subjetividad débil** — los nodos nuevos o largamente desconectados no aceptan reorganizaciones que crucen un checkpoint reciente obtenido de una fuente confiable, y los clientes incorporan estos puntos de anclaje al sincronizar.
- **Ataques de corto alcance al fork choice** (balancing, bouncing): intentan mantener a la red dividida entre dos ramas manipulando el momento de publicación de atestaciones. Mitigación: el **proposer boost** y los refinamientos sucesivos de LMD-GHOST tras incidentes como el reorg de 7 bloques de mayo de 2022.

Gasper: dos protocolos complementarios

Componente	Pregunta que responde	Mecanismo	Garantía que aporta
LMD-GHOST	¿Sobre qué cabeza de cadena construyo y atestiguo ahora?	Sigue la rama con más peso de últimas atestaciones válidas	Vivacidad: la cadena avanza cada slot aunque no haya finalidad
Casper FFG	¿Qué historia es ya irreversible?	Votos de checkpoint por épocas; justificación y finalización con 2/3 del stake	Seguridad económica: revertir lo finalizado cuesta al menos 1/3 del stake slasheado

La separación importa: si más de 1/3 del stake se desconecta, LMD-GHOST mantiene la cadena viva pero Casper FFG deja de finalizar; el protocolo activa entonces la **fuga de inactividad** (inactivity leak), que drena el depósito de los validadores ausentes hasta que los activos vuelven a superar los 2/3 y la finalidad se recupera. Especificación y análisis: Buterin et al., *Combining GHOST and Casper* — https://arxiv.org/abs/2003.03052.

Cuánto cuesta de verdad un ataque del 51 %

"Con la mayoría del poder se puede reescribir la cadena" es cierto y poco útil sin el número. Pongámoslo, porque el resultado explica por qué unas redes se atacan y otras no.

Lo que un atacante puede y no puede hacer. Es más limitado de lo que sugiere el titular:

Puede	No puede
Excluir transacciones (censurar)	Robar monedas de otras direcciones: no tiene las claves
Revertir sus **propias** transacciones recientes (doble gasto)	Crear monedas de la nada: las reglas las validan todos los nodos
Reordenar transacciones dentro de su ventana	Alterar bloques antiguos y profundos

El ataque real, entonces, es concreto: **depositar en un exchange, cambiar por otra moneda, retirar, y luego reescribir la cadena para recuperar el depósito**.

Qué lo hace rentable. Su viabilidad depende de una relación:

```
beneficio  =  lo que consigues retirar antes de que lo detecten
coste      =  alquiler de hashrate × horas  +  hardware  +  reputación
```

Y aquí está la clave que decide todo: **si existe un mercado donde alquilar hashrate, el coste del hardware desaparece de la ecuación**. Por eso las cadenas pequeñas que comparten algoritmo con una grande son las que se atacan: hay potencia de sobra apuntando a otra cadena, y desviarla un rato es barato. Ethereum Classic sufrió varias reorganizaciones profundas en 2019 y 2020 exactamente así, mientras Bitcoin —cuyo hashrate no se puede alquilar en volumen suficiente— nunca lo ha sufrido.

La defensa real no es solo técnica. Un exchange que exige 100 confirmaciones para una cadena barata está subiendo el coste del ataque de forma lineal: revertir 100 bloques cuesta cien veces más que revertir uno. Por eso los requisitos de confirmación varían tanto entre monedas: no es arbitrario, es una función del coste de atacarlas.

> 💡 **En una frase:** el consenso no hace imposible el ataque; lo hace más caro que el beneficio. Cuando esa desigualdad se invierte —cadena pequeña, hashrate alquilable— el ataque ocurre.

▶ 🎓 **Si ya dominas esto** — donde el modelo simple se queda corto

🧪 Laboratorio guiado

> 🧪 Estas prácticas están catalogadas y **resueltas paso a paso** en el catálogo de laboratorios.

1. Ejecuta el laboratorio pedagógico de Proof of Work:

```
pnpm lab:pow
```

2. Observa cuántos intentos (nonces) se requieren para hallar un hash bajo el objetivo con la dificultad inicial.
3. Aumenta la dificultad un nivel y vuelve a ejecutar; registra intentos y tiempo transcurrido.
4. Repite con varios niveles y tabula la relación entre dificultad, intentos y tiempo:

```
Dificultad | Intentos promedio | Tiempo (s)
-----------|-------------------|-----------
1          | ...               | ...
2          | ...               | ...
3          | ...               | ...
```

5. Concluye cómo escala el costo con la dificultad, recordando que este laboratorio es pedagógico y no representa la seguridad real de una red en producción.

📝 Reto verificable

Completa la tabla de dificultad frente a intentos y tiempo con al menos tres niveles y redacta una breve interpretación del crecimiento observado.

Criterio de aceptación: muestras que al aumentar la dificultad crece de forma marcada el número de intentos y el tiempo esperado, y declaras explícitamente que el laboratorio no modela la seguridad de una red real (número de mineros, hashrate global ni incentivos).

⚠️ Errores frecuentes

Síntoma	Causa y cómo comprobarlo
Reducir el consenso a "energía vs. velocidad"	Ignoras recurso anti-Sybil y finalidad; compara las tres dimensiones.
Creer que Ethereum sigue en PoW	Desde The Merge (2022) es PoS; verifica en la documentación oficial.
Confundir finalidad probabilística con inmediata	En PoW la reversión disminuye con el tiempo; en Gasper/PBFT hay finalidad explícita.
Pensar que el laboratorio mide seguridad real	Es pedagógico; no incluye hashrate global ni incentivos económicos.
Suponer que PoS elimina toda centralización	El capital puede concentrarse; analiza la distribución de validadores.

🛡️ Seguridad y ética

- Ejecuta el laboratorio solo en local; no uses claves ni fondos reales ni conectes a redes de producción.
- No presentes resultados del laboratorio como evidencia de la seguridad de una red pública.
- Reconoce las compensaciones de cada mecanismo (costo energético, concentración de capital, permisos).
- Evita el fraude de simplificación: comunica los supuestos y límites de cualquier comparación.
- Considera el impacto ambiental y de gobernanza al recomendar un mecanismo.

🔗 Referencias

- Fuente primaria: Satoshi Nakamoto, *Bitcoin: A Peer-to-Peer Electronic Cash System* — https://bitcoin.org/bitcoin.pdf
- Miguel Castro y Barbara Liskov, *Practical Byzantine Fault Tolerance*, OSDI 1999 — https://pmg.csail.mit.edu/papers/osdi99.pdf
- Buterin y Griffith, *Casper the Friendly Finality Gadget* — https://arxiv.org/abs/1710.09437
- ethereum.org, documentación sobre Proof of Stake — https://ethereum.org/developers/docs/consensus-mechanisms/pos/

✅ Criterio de dominio

- Entregas la tabla de dificultad e interpretas el crecimiento del costo.
- Comparas los cuatro mecanismos por recurso anti-Sybil, finalidad y uso típico.
- Explicas la diferencia entre finalidad probabilística y económica con ejemplos actuales.

🧭 Navegación

⬅️ Módulo 02 · Sistemas distribuidos y redes P2P · 📚 Índice del currículo · ➡️ Módulo 04 · Bitcoin

04 · Bitcoin

> **Nivel:** Intermedio · ⏱️ **Duración estimada:** 150 min · **Fuente:** *Mastering Bitcoin* (Antonopoulos) y *Mastering the Lightning Network* (Antonopoulos, Osuntokun, Pickhardt) ⬅️ Currículo · 📚 Bibliografía 🧭 ⬅️ **Anterior:** 03 · Consenso · 📚 Índice · ➡️ **Siguiente:** 05 · Ethereum y EVM 📖 Glosario de términos · 🌱 ¿Nuevo en esto? Empieza aquí

🎯 Objetivos

- Explicar el modelo UTXO identificando entradas, salidas, scripts y firmas en una transacción real.
- Calcular la comisión de una transacción y su tasa en sat/vB a partir de datos de un explorador público.
- Distinguir el papel de un full node frente a un cliente SPV en la verificación.
- Describir la política de emisión (halving) y su efecto sobre la oferta a lo largo del tiempo.
- Diferenciar custodia propia y custodia delegada, y el rol de la seed phrase (BIP-39).

📚 Resultados de aprendizaje

Al finalizar, el estudiante podrá:

1. **Descomponer** una transacción de Bitcoin en los UTXO que consume y las salidas que crea.
2. **Estimar** comisión y tasa efectiva marcando explícitamente cualquier inferencia sobre el cambio.
3. **Comparar** confirmaciones, profundidad de bloque y el riesgo de reorganización.
4. **Justificar** por qué una dirección no equivale a una persona ni a una identidad.
5. **Seleccionar** UTXO de una cartera aplicando criterios de coste y privacidad.
6. **Relacionar** Lightning con la capa base como red de pagos fuera de cadena.

🗺️ Temas

#	Tema	Por qué importa
1	Modelo UTXO	Es la unidad contable de Bitcoin; sin él no se entiende ninguna transacción.
2	Scripts y firmas	Definen las condiciones de gasto y quién puede reclamar una salida.
3	Mempool y comisiones	Determinan cuándo y a qué coste se confirma una transacción.
4	Minería y confirmaciones	Explican la finalidad probabilística y el riesgo de reorganización.
5	Full nodes y SPV	Marcan la diferencia entre verificar y confiar.
6	Emisión y halving	Fijan la política monetaria y la oferta futura.
7	Lightning Network	Habilita pagos rápidos y baratos fuera de la cadena base.
8	Custodia y BIP-39	La gestión de claves decide quién controla realmente los fondos.

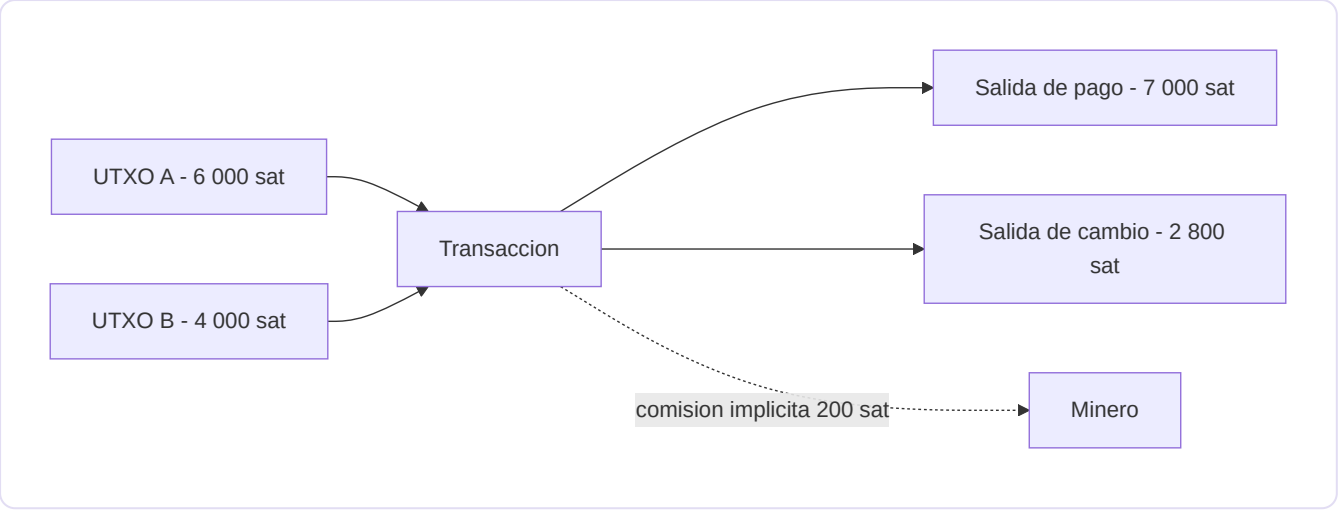
🧠 Modelo mental

Piensa en los UTXO como billetes físicos en una billetera: cada uno tiene un valor fijo y solo puedes gastarlo entero. Para pagar 7 usando dos billetes de 5, entregas ambos (entradas por 10), pagas los 7 y recibes 3 de vuelta como cambio, menos una pequeña propina para el minero (la comisión). No existe un saldo único: tu balance es la suma de todos los billetes que puedes gastar.

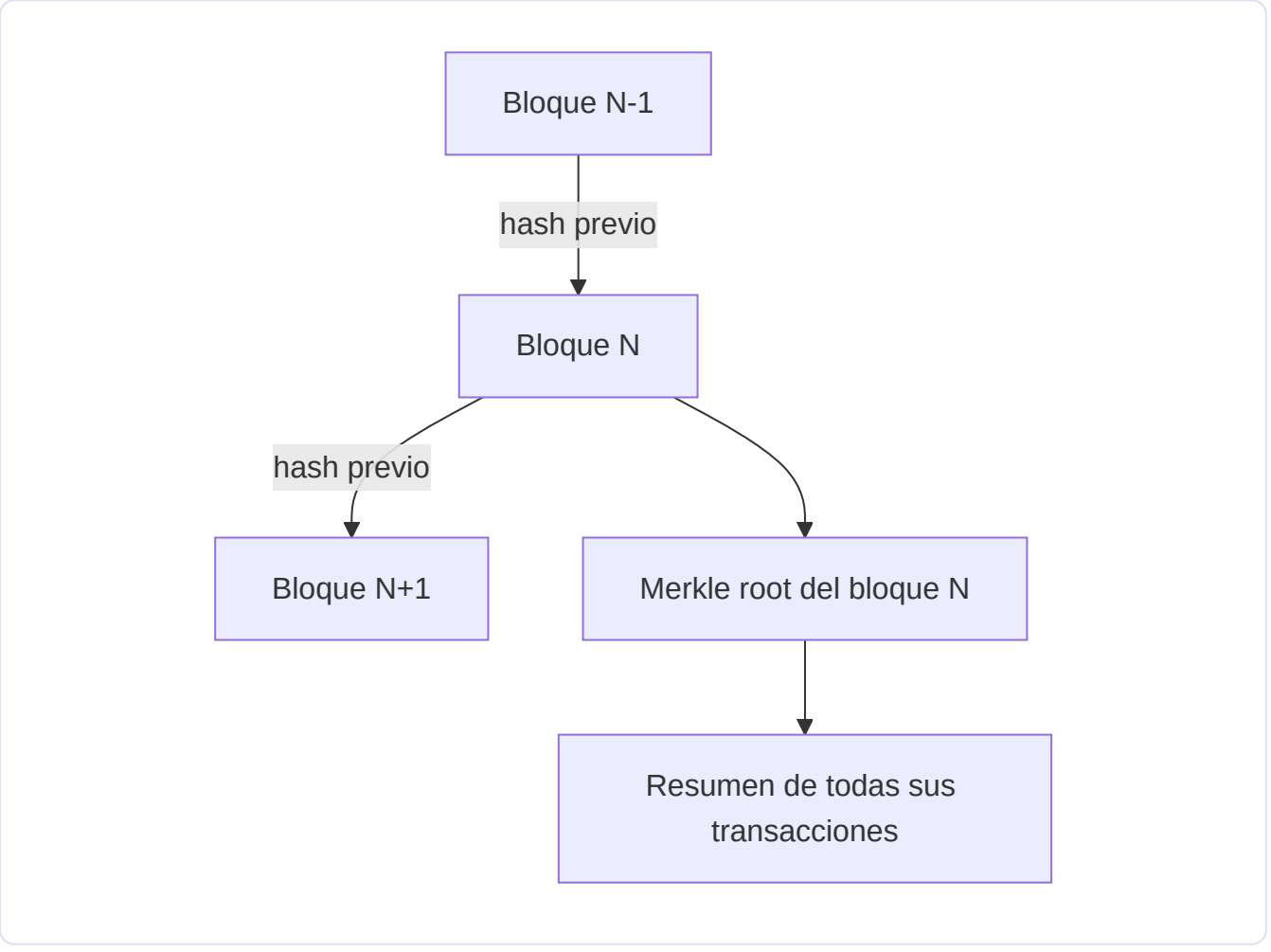
La analogía tiene límites. Los billetes no llevan condiciones de gasto programables, mientras que un UTXO se bloquea con un script que puede exigir una firma, varias firmas o una condición temporal. Además, el cambio no vuelve mágicamente a "tu bolsillo": va a una nueva salida que tú controlas, y confundirla con un pago real es una fuente clásica de errores de análisis.

🧩 Esquema visual

El flujo típico de una transacción UTXO: dos entradas se consumen por completo, se crea una salida de pago y otra de cambio, y la diferencia no asignada es la comisión que cobra el minero.



Cada bloque referencia el hash del anterior y resume sus transacciones en una raíz de Merkle, lo que encadena la historia y permite pruebas de inclusión eficientes.



📖 Conceptos y definiciones

- **UTXO**: salida de transacción no gastada; representa fondos disponibles con una condición de gasto asociada.
- **Entrada (input)**: referencia a un UTXO previo que se consume, acompañada de los datos que satisfacen su script.

- **Salida (output)**: nuevo UTXO creado, con un valor y un script que fija quién podrá gastarlo.
- **Comisión (fee)**: diferencia entre el valor total de entradas y salidas; retribuye al minero. Ejemplo: entradas 10 000 sat, salidas 9 800 sat → comisión 200 sat.
- **Tasa de comisión**: comisión dividida por el tamaño virtual, expresada en sat/vB; guía la prioridad en el mempool.
- **Confirmación**: inclusión de la transacción en un bloque; cada bloque adicional aumenta la profundidad y reduce el riesgo de reorganización.
- **Reorganización (reorg)**: sustitución de bloques recientes por una cadena competidora más trabajada; puede revertir confirmaciones poco profundas.
- **Full node**: nodo que valida por sí mismo todas las reglas de consenso; no delega confianza.
- **SPV**: verificación de pago simplificada; confía en cabeceras y pruebas de inclusión sin validar toda la cadena.
- **Seed phrase (BIP-39)**: lista de palabras que codifica la semilla de la que derivan todas las claves; quien la posee controla los fondos.
- **Taproot/Schnorr (2021)**: actualización que introdujo firmas Schnorr y scripts más privados y eficientes.

🔬 Profundización

Evolución de los tipos de script

Los formatos de salida de Bitcoin han evolucionado para reducir tamaño, mejorar la privacidad y habilitar nuevas criptografías, siempre mediante soft forks compatibles hacia atrás.

Tipo	Año	Qué aportó
P2PKH	2009	Pago a hash de clave pública; el formato "clásico" con direcciones que empiezan por 1.
P2SH	2012 (BIP-16)	Pago a hash de script; permite multisig y condiciones complejas sin revelarlas hasta el gasto.
P2WPKH	2017 (SegWit, BIP-141)	Mueve las firmas al *witness*, corrige la maleabilidad y abarata el peso de la transacción.
P2TR	2021 (Taproot, BIP-341)	Firmas Schnorr agregables; un gasto cooperativo multisig se ve idéntico a uno simple, ganando privacidad.

Con Schnorr, una multifirma agregada ocupa lo mismo que una firma individual (64 bytes), mientras que un multisig ECDSA tradicional publica todas las firmas por separado.

El mercado de comisiones: vbytes, RBF y CPFP

Desde SegWit el tamaño relevante es el **tamaño virtual** (vbytes): los datos de witness pesan una cuarta parte que el resto. La prioridad en el mempool se mide en **sat/vB**.

Ejemplo numérico orientativo: una transacción con 1 entrada P2WPKH y 2 salidas P2WPKH ocupa ≈ 141 vB (≈ 10,5 vB de estructura + ≈ 68 vB la entrada + ≈ 31 vB cada salida). Si el mercado pide 20 sat/vB:

```
comisión ≈ 141 vB × 20 sat/vB = 2 820 sat
```

Si la transacción queda atascada hay dos salidas estándar:

- **RBF (BIP-125)**: el emisor la reemplaza por otra con el mismo UTXO de entrada y mayor tasa.
- **CPFP**: el receptor gasta la salida sin confirmar con una transacción hija de tasa alta; el minero debe incluir ambas y evalúa la tasa del paquete completo.

Las tasas de mercado cambian por hora: consúltalo en vivo en un estimador de comisiones antes de emitir.

Emisión: halvings y el límite de 21 millones

El subsidio por bloque se reduce a la mitad cada 210 000 bloques (≈ 4 años): 50 BTC en 2009, 25 tras el halving de noviembre de 2012, 12,5 en julio de 2016, 6,25 en mayo de 2020 y **3,125 BTC desde abril de 2024**, el subsidio vigente. El siguiente halving se espera hacia 2028.

Como cada término de la serie es la mitad del anterior, la suma converge: 210 000 × 50 × (1 + 1/2 + 1/4 + …) ≈ 21 millones de BTC, que nunca se alcanzan exactamente por el redondeo a satoshis. Más del 94 % del suministro ya fue emitido; la emisión restante se extiende hasta aproximadamente el año 2140, cuando la seguridad dependerá solo de las comisiones.

Una comisión calculada de principio a fin

La mayoría de las dudas con Bitcoin se resuelven haciendo el cálculo una vez completo. Supongamos que quieres pagar **17 000 sat** y tu cartera tiene tres UTXOs P2WPKH: 8 000, 12 000 y 30 000 sat.

Paso 1 — elegir entradas. Con 8 000 no llega. Con 8 000 + 12 000 = 20 000 sí. La transacción queda con **2 entradas** y **2 salidas** (el pago y el cambio).

Paso 2 — estimar el tamaño virtual. Aquí hay que parar un momento, porque es donde se pierde la gente.

En un bloque de Bitcoin no cabe valor: cabe **espacio**. Por eso lo que pagas depende de cuánto ocupa tu transacción, no de cuánto mueves. Y ese "cuánto ocupa" no se cuenta en bytes normales sino en **vbytes** (bytes virtuales), una unidad que da menos peso a la parte de la firma. La razón es una decisión de diseño de SegWit: separó las firmas del resto de la transacción y les puso descuento, para abaratar el uso de bloques.

> Regla práctica: **más entradas y más salidas = más vbytes = más caro**, independientemente del monto.

Cifras estándar por componente para direcciones P2WPKH (las que empiezan por `bc1q`):

Componente	vB cada uno	Cantidad	Total
Sobrecarga (versión, contadores, locktime)	10,5	1	10,5
Entrada P2WPKH	68	2	136
Salida P2WPKH	31	2	62
Total			**≈ 209 vB**

Paso 3 — aplicar la tasa. Si el mempool pide 12 sat/vB para entrar en los próximos bloques:

```
comisión = 209 vB × 12 sat/vB = 2 508 sat
```

Paso 4 — repartir el valor. Aquí está el punto que casi nadie ve la primera vez: **la comisión no se declara en ningún campo**. Es lo que sobra entre entradas y salidas, así que el cambio se despeja, no se elige.

```
cambio = entradas − pago − comisión
       = 20 000 − 17 000 − 2 508
       = 492 sat

entradas           20 000 sat
salida de pago    −17 000 sat
salida de cambio     −492 sat
                  ──────────
sobra               2 508 sat   ← la comisión; se la queda el minero
```

Paso 5 — el polvo. Una salida de 492 sat es *dust*: gastarla en el futuro costaría más de lo que vale (una entrada P2WPKH son 68 vB, que a 12 sat/vB ya son 816 sat). Las carteras, ante esto, hacen una de dos cosas: **omitir la salida de cambio** y regalar esos 492 sat al minero (comisión efectiva 3 000 sat), o **bajar la tasa** para que el cambio supere el umbral de polvo.

El error que cuesta dinero. Si construyes la transacción a mano y olvidas la salida de cambio con los tres UTXOs seleccionados (50 000 sat de entradas, 17 000 de pago), la comisión no es de 2 500 sat: es de **33 000 sat**. El protocolo no te avisa, no hay error, y el minero se lo queda. No existe forma de recuperarlo. Es la razón por la que las prácticas de este módulo exigen comprobar que entradas = salidas + comisión antes de firmar nada.

> 💡 **En una frase:** la comisión no se escribe, se despeja — es lo que sobra entre lo que entra y lo que sale. Suma siempre ambos lados antes de firmar.

Por qué el tamaño manda más que el monto

Consecuencia contraintuitiva del modelo UTXO: **mover 1 BTC puede costar más que mover 100**. Lo que se paga es el espacio en el bloque, no el valor transferido.

Escenario	Entradas	vB aprox.	A 12 sat/vB
Una entrada grande → 100 BTC	1	141	1 692 sat
Cien entradas pequeñas → 1 BTC	100	6 873	82 476 sat

(Ambas con dos salidas: pago y cambio.)

Por eso las carteras hacen *consolidación*: juntar muchos UTXOs pequeños en uno grande cuando las tasas están bajas, para no pagarlo caro cuando haya prisa. Y por eso recibir muchos pagos diminutos tiene un coste futuro que no se ve en el momento de recibirlos.

> 💡 **En una frase:** en Bitcoin pagas por ocupar espacio, no por mover valor. Recibir muchos pagos pequeños te deja una factura futura que no ves al recibirlos.

▶ 🎓 **Si ya dominas esto** — el detalle fino que cambia decisiones reales

🧪 Laboratorio guiado

> 🧪 Estas prácticas están catalogadas y **resueltas paso a paso** en el catálogo de laboratorios.

1. Ejecuta la selección de UTXO del repositorio para observar cómo se eligen entradas ante distintos objetivos de pago.

```
pnpm lab:utxo
```

2. Elige una transacción pública en un explorador de bloques de tu preferencia y anota su identificador.
3. Lista los UTXO que consume (entradas) y las salidas que crea, con sus valores.
4. Identifica cuál salida parece cambio y **marca esa afirmación como inferencia**, no como hecho.
5. Calcula la comisión (suma de entradas menos suma de salidas) y la tasa en sat/vB usando el tamaño virtual.
6. Escribe dos limitaciones del análisis, incluyendo por qué no puedes vincular la dirección a una persona.

📝 Reto verificable

Documenta el análisis de una transacción pública en un texto breve: identificador, entradas consumidas, salidas creadas, comisión, tasa en sat/vB y la salida que infieres como cambio.

Criterio de aceptación: el documento marca explícitamente la inferencia de cambio, muestra el cálculo de la comisión con sus cifras y no afirma en ningún punto que una dirección corresponda a una identidad concreta.

⚠️ Errores frecuentes

Síntoma	Causa y cómo comprobarlo
La comisión "no cuadra"	Olvidaste que la comisión es entradas menos salidas; súmalas por separado y resta.
Confundes el cambio con un pago	Asumiste una salida sin marcar la inferencia; verifica qué dirección controla la cartera emisora.
Crees que una confirmación es definitiva	Ignoras el riesgo de reorg; consulta la profundidad y espera más bloques para montos altos.
Atribuyes una dirección a una persona	Confundes seudónimo con identidad; recuerda que el análisis solo observa flujos on-chain.
Comparas tasas sin normalizar	Usaste el tamaño en bytes en vez del virtual; calcula sat/vB con el tamaño virtual.

🛡️ Seguridad y ética

- Trabaja siempre en local o testnet; nunca uses fondos ni claves reales en los ejercicios.
- No introduzcas seed phrases reales en ninguna herramienta del laboratorio.
- El análisis de cadena observa seudónimos: no deanonymices ni atribuyas transacciones a personas.

- Trata la custodia propia con responsabilidad; una seed phrase perdida es dinero perdido y una filtrada es robo.
- Las cifras de comisiones y dificultad cambian constantemente: consúltalo en vivo antes de concluir.

🔗 Referencias

- Antonopoulos, *Mastering Bitcoin*, 3.ª ed., cap. sobre transacciones y UTXO — https://github.com/bitcoinbook/bitcoinbook
- Antonopoulos, Osuntokun y Pickhardt, *Mastering the Lightning Network*, cap. introductorio — https://github.com/lnbook/lnbook
- Narayanan et al., *Bitcoin and Cryptocurrency Technologies*, caps. sobre mecánica de Bitcoin.
- Fuente primaria: whitepaper de Bitcoin — https://bitcoin.org/bitcoin.pdf
- Fuente primaria: repositorio de BIPs — https://github.com/bitcoin/bips

✅ Criterio de dominio

- Explicas de memoria el ciclo entrada → salida → cambio de una transacción real.
- Calculas comisión y tasa en sat/vB y justificas la incertidumbre de la inferencia de cambio.
- Argumentas la diferencia entre verificar con un full node y confiar en SPV.

🧭 Navegación

⬅️ Módulo 03 · Consenso · 📚 Índice del currículo · ➡️ Módulo 05 · Ethereum y EVM

05 · Ethereum y EVM

> **Nivel:** Intermedio · ⏱️ **Duración estimada:** 150 min · **Fuente:** *Mastering Ethereum* (Antonopoulos, Wood) y *Ethereum Yellow Paper* (Wood) ⬅️ Currículo · 📚 Bibliografía
> 🧭 ⬅️ **Anterior:** 04 · Bitcoin · 📚 Índice · ➡️ **Siguiente:** 06 · Solidity y Foundry 📖 Glosario de términos · 🌱 ¿Nuevo en esto? Empieza aquí

🎯 Objetivos

- Distinguir cuentas EOA y de contrato por su nonce, código y forma de iniciar transacciones.
- Explicar el modelo de gas y la estructura de comisiones EIP-1559 (base fee y priority fee).
- Codificar y decodificar una llamada mediante ABI y su selector de función.
- Diferenciar storage, memory y stack, y relacionar una variable de storage con un evento.
- Seguir una transacción desde la firma en la wallet hasta el cambio de estado, identificando dónde puede fallar.

📚 Resultados de aprendizaje

Al finalizar, el estudiante podrá:

1. **Clasificar** una cuenta como EOA o contrato a partir de sus atributos observables.
2. **Descomponer** la calldata de una llamada en selector y argumentos codificados.
3. **Calcular** el coste aproximado de una transacción con base fee y priority fee.
4. **Explicar** por qué la estimación de gas no es una promesa exacta del coste final.
5. **Relacionar** un cambio de estado en storage con el evento emitido que lo registra.
6. **Rastrear** el recorrido de una transacción y ubicar los puntos de fallo (revert, gas insuficiente, nonce).

🗺️ Temas

#	Tema	Por qué importa
1	EOA vs. contrato	Define quién puede firmar y quién solo ejecuta código.
2	Nonce	Ordena las transacciones de una cuenta y evita repeticiones.
3	Gas y EIP-1559	Determina el coste y la inclusión de una transacción.
4	Calldata y ABI	Es el contrato de interfaz entre quien llama y el código.
5	Eventos y logs	Permiten observar cambios de estado de forma barata e indexable.
6	Storage, memory y stack	Explican dónde vive cada dato y cuánto cuesta usarlo.
7	Llamadas y revert	Modelan la composición entre contratos y el manejo de errores.
8	Estado como Merkle-Patricia	Garantiza integridad y pruebas eficientes del estado global.

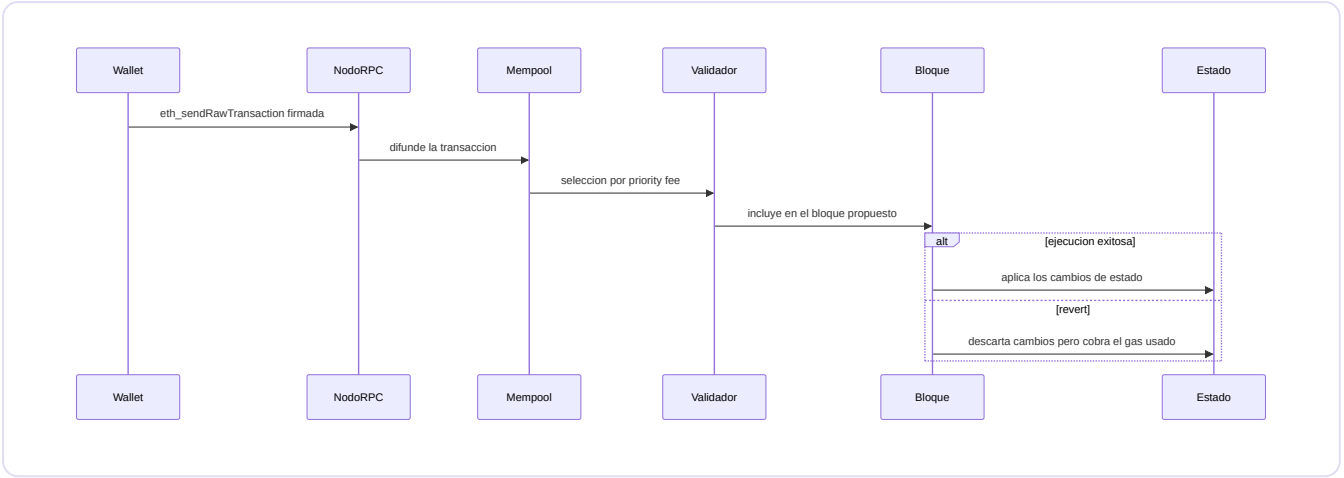
🧠 Modelo mental

Imagina la EVM como una calculadora mundial compartida en la que cada operación tiene un precio en fichas (gas). Antes de ejecutar, cargas suficientes fichas; si se agotan a mitad de camino, la máquina deshace todo el trabajo pero se queda con las fichas gastadas. El estado global es un gran libro contable cuyo resumen (la raíz de Merkle-Patricia) cabe en una sola huella verificable.

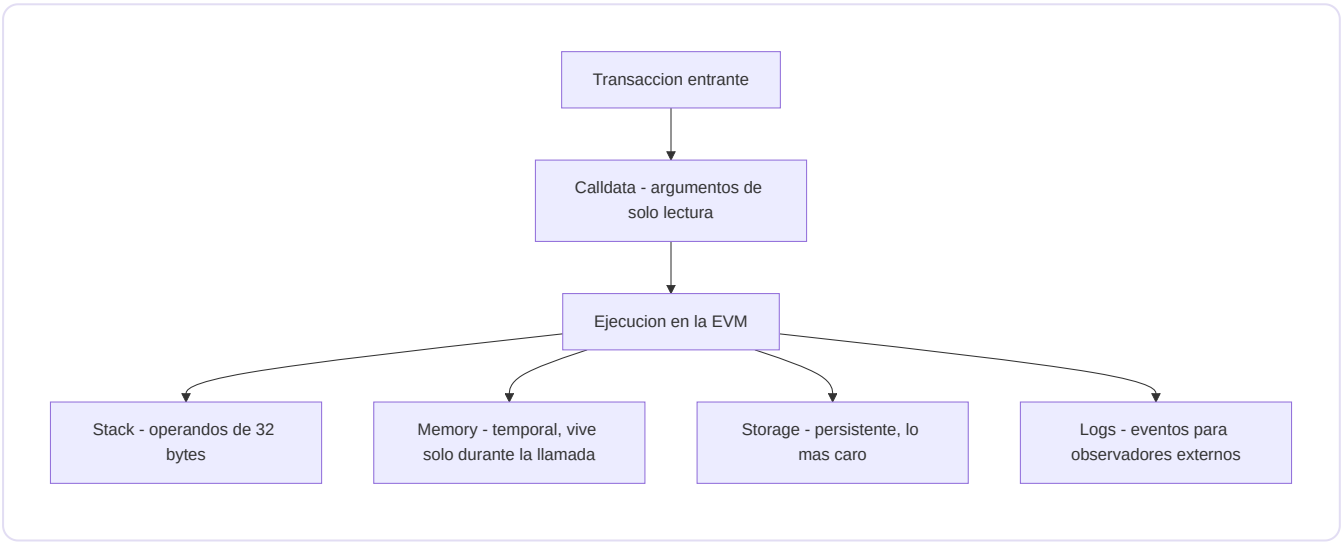
La analogía se rompe en un punto clave: a diferencia de una calculadora aislada, aquí miles de nodos reejecutan la misma operación para acordar el resultado, y el precio por ficha varía en cada bloque según la demanda (la base fee). Por eso la estimación de gas es una previsión, no una garantía: el consumo real depende del estado en el momento exacto de la ejecución.

🧩 Esquema visual

Ciclo de vida de una transacción, desde la firma en la wallet hasta el cambio de estado, con el revert como camino alternativo.



Áreas de datos de la EVM durante una ejecución: cada una tiene un ciclo de vida y un coste distintos.



📖 Conceptos y definiciones

- **EOA**: cuenta controlada por una clave privada; es la única que puede iniciar y firmar transacciones.
- **Cuenta de contrato**: cuenta con código asociado que se ejecuta cuando la invocan; no firma por sí misma.
- **Nonce**: contador por cuenta que ordena las transacciones y previene su repetición.
- **Gas**: unidad que mide el trabajo computacional; cada operación de la EVM tiene un coste fijo en gas.
- **Base fee**: comisión por gas que se quema en cada bloque; se ajusta según la ocupación (EIP-1559).
- **Priority fee**: propina por gas para el validador que incentiva la inclusión rápida.
- **Calldata**: datos de entrada de una llamada; comienza con el selector de 4 bytes y sigue con los argumentos.
- **ABI**: especificación que describe funciones y tipos para codificar y decodificar llamadas y respuestas.

- **Evento (log)**: registro barato e indexable que un contrato emite para señalar un cambio de estado.
- **Storage / memory / stack**: almacenamiento persistente por contrato, memoria temporal por llamada y pila de trabajo de la EVM.
- **Revert**: aborto de una ejecución que deshace los cambios de estado pero consume el gas gastado.

🔬 Profundización

Costos de gas reales: frío vs. caliente

Desde EIP-2929 (hard fork Berlín, 2021) el primer acceso a una cuenta o a un slot de storage dentro de una transacción es "frío" y paga un recargo; los accesos siguientes son "calientes" y resultan mucho más baratos. Cifras orientativas post-Berlín:

Operación	Coste en gas
SLOAD frío (primer acceso al slot en la tx)	2 100
SLOAD caliente (accesos posteriores)	100
SSTORE de cero a distinto de cero, slot frío	22 100
SSTORE actualizando un valor no nulo, slot frío	5 000
SSTORE actualizando un valor no nulo, slot caliente	2 900

De ahí la asimetría clásica: escribir por primera vez un contador cuesta ≈ 22 100 gas (20 000 del SSTORE inicial + 2 100 del acceso frío), mientras que incrementarlo en una transacción posterior cuesta ≈ 5 000 (2 900 + 2 100). Poner un slot de vuelta a cero genera además un reembolso parcial, limitado desde EIP-3529 (Londres, 2021).

Layout de storage: slots, packing y mappings

El storage de un contrato es un arreglo direccionable de slots de 32 bytes. El compilador asigna variables de estado en orden de declaración y **empaqueta** en un mismo slot las que quepan juntas:

```
uint128 a; // slot 0, bytes bajos
uint128 b; // slot 0, bytes altos: comparte slot con a
uint256 c; // slot 1: necesita los 32 bytes completos
```

Leer `a` y `b` en la misma transacción toca un solo slot, así que declarar variables pequeñas contiguas ahorra gas real. Los mappings no ocupan su slot de forma secuencial: el valor de `m[k]`, con el mapping declarado en el slot `p`, vive en `keccak256(abi.encode(k, p))`, lo que dispersa las claves por todo el espacio de storage sin colisiones prácticas.

EIP-1559 en números

Cada bloque tiene un **objetivo de 15 millones de gas** y un **límite de 30 millones**. La base fee se ajusta según la ocupación del bloque anterior: hasta **+12,5 %** si vino completamente lleno y hasta **−12,5 %** si vino vacío. Con seis bloques llenos consecutivos la base fee aproximadamente se duplica ($1{,}125^6 \approx 2{,}03$), lo que hace muy caro sostener congestión artificial.

La base fee se **quema** (sale de circulación) y solo la priority fee llega al validador. El coste total por unidad de gas es:

```
precio efectivo = min(max fee, base fee + priority fee)
```

Los valores vigentes de base fee cambian bloque a bloque: consúltalo en vivo antes de estimar costes.

El coste de una transacción, desglosado

"¿Por qué me costó eso?" es la pregunta más repetida del módulo, y tiene respuesta exacta: se suma.

Antes de los números, la idea. En Ethereum hay **dos cosas separadas** que se suelen confundir en una:

- **El gas** mide *cuánto trabajo* pide tu transacción. Es una cantidad fija para una operación dada: escribir en el estado siempre cuesta lo mismo, esté la red vacía o saturada.
- **El precio del gas** es *cuánto vale ese trabajo ahora mismo*, y sí cambia con la demanda.

> Analogía: el gas son los kilómetros del viaje; el precio del gas, lo que cuesta el litro hoy. La factura es el producto de ambos. Si la red está cara, no es que tu transacción haga más trabajo: es que el litro subió.

El trabajo (el gas) se reparte en tres partidas: **base**, **datos** y **estado**. Vamos con una transferencia ERC-20 corriente.

1. Coste base. Toda transacción paga **21 000 gas** por existir, aunque no haga nada. Es el precio de verificar la firma, actualizar el nonce y mover el saldo.

2. Calldata. Los datos de entrada se cobran por byte, y **no todos los bytes cuestan igual**: un byte cero vale 4 gas y uno distinto de cero vale 16 (EIP-2028). El calldata de un `transfer(address,uint256)` son 68 bytes: 4 de selector + 32 de dirección + 32 de monto.

```
selector  a9059cbb                          →  4 bytes, ninguno cero      = 4 × 16 =
64
dirección 000…0d8dA6BF26964aF9D7eEd9e03E53415D37aA96045 → 32 bytes, 12 en cero
                                                → 12×4 + 20×16 = 48 + 320 = 368
monto     000…00000000000000000000000f4240  → 32 bytes, 29 en cero
                                                → 29×4 + 3×16 = 116 + 48   = 164
                                                                          ──────────
                                                                  calldata ≈ 596 gas
```

Esto explica una rareza que se ve en la práctica: **las direcciones con muchos ceros al principio son literalmente más baratas de usar**, y por eso existen contratos con direcciones "vanity" llenas de ceros en protocolos de alto volumen.

3. Estado. Es la parte cara, y depende de algo que no controlas: si el destinatario **ya tenía** saldo de ese token.

¿Por qué importa tanto? Porque escribir un dato **nuevo** obliga a cada uno de los miles de nodos de la red a guardarlo para siempre; sobrescribir uno que ya existía solo cambia un valor que ya ocupaba sitio. La EVM cobra esa diferencia de forma explícita:

Operación	Cuándo	Gas
`SSTORE` de slot que pasa de 0 a distinto de 0	el destinatario recibe el token por primera vez	20 000
`SSTORE` de slot que ya era distinto de 0	el destinatario ya tenía saldo	2 900
Acceso "frío" a un slot (primera vez en la transacción)	siempre, la primera lectura	2 100
Acceso "templado" (ya tocado en esta transacción)	lecturas siguientes	100

De ahí sale la horquilla que verás en cualquier explorador: el mismo `transfer` consume **≈ 65 000 gas** cuando el destinatario estrena saldo y **≈ 51 000** cuando ya lo tenía. No es que la red esté más cara: es que escribir un cero donde no había nada obliga a la red a guardar una entrada nueva para siempre.

4. Y ahora el precio. El gas es *trabajo*; el precio de ese trabajo se fija aparte, y desde EIP-1559 son dos números:

```
coste total = gas usado × (base fee + priority fee)

  65 000 gas × (12 gwei base + 1 gwei propina)
= 65 000 × 13 gwei
= 845 000 gwei
= 0,000845 ETH
```

De esos, `65 000 × 12 = 780 000 gwei` **se queman** (desaparecen de la circulación) y solo `65 000 × 1 = 65 000 gwei` van al validador. Mezclar ambas es el error tabulado en este módulo: si sumas la base fee al ingreso del validador, te sale un número que no corresponde a nadie.

El `maxFeePerGas` que fija tu cartera es un **techo**, no un precio: si la base fee sube por encima, la transacción espera; si baja, pagas menos de lo autorizado y te devuelven la diferencia.

Por qué la estimación no es una promesa

`eth_estimateGas` ejecuta la transacción contra el estado **actual** y te dice cuánto consumió. Pero se incluirá en un bloque futuro, con otro estado.

El caso canónico: estimas un `transfer` a una dirección que ya tiene saldo → 51 000. Antes de que te incluyan, esa dirección retira todo y su saldo queda en cero. Tu transacción ahora hace un `SSTORE` de 0 a distinto de 0 → necesita 65 000, y con un límite de 51 000 revierte por *out of gas*… **consumiendo igualmente todo el gas del límite**.

Por eso las carteras añaden un margen sobre la estimación, y por eso un revert cuesta dinero: el trabajo de cómputo se hizo y hay que pagarlo; lo que se deshace son los efectos, no el gasto.

> 💡 **En una frase:** el gas mide trabajo y no cambia; el precio del gas mide la demanda y sí cambia. Escribir un dato nuevo cuesta ~7 veces más que sobrescribir uno existente, y esa diferencia explica casi toda la variación que verás.

▶ 🎓 **Si ya dominas esto** — lo que suele fallar en producción

🧪 Laboratorio guiado

> 🧪 Estas prácticas están catalogadas y **resueltas paso a paso** en el catálogo de laboratorios.

1. Inicia un nodo local de desarrollo con Foundry en una terminal aparte.

```
anvil
```

2. Consulta bloque actual, chain id y el balance de una cuenta con `cast`.

```
cast block-number
cast chain-id
cast balance 0xf39Fd6e51aad88F6F4ce6aB8827279cffFb92266
```

3. Ejecuta el laboratorio de selector ABI para codificar una llamada y decodificar su respuesta.

```
pnpm lab:abi
```

4. Compara el valor de una variable en storage con el evento que se emitió al modificarla.
5. Estima el gas de una llamada y explica por qué el coste final puede diferir.
6. Ejecuta la batería de pruebas del repositorio con `pnpm test` para validar tu entorno.

📝 Reto verificable

Sigue una transacción de principio a fin: firma en la wallet, propagación, inclusión en bloque y cambio de estado. Redacta el recorrido señalando dónde podría fallar.

Criterio de aceptación: el recorrido nombra el selector y los argumentos de la calldata, muestra el desglose base fee más priority fee y enumera al menos tres puntos de fallo posibles (revert, gas insuficiente y nonce incorrecto).

⚠️ Errores frecuentes

Síntoma	Causa y cómo comprobarlo
"La transacción se quedó pendiente"	Nonce con hueco o gas price bajo; revisa el nonce esperado y la base fee actual.
El coste no coincide con la estimación	Confundiste estimación con promesa; el consumo depende del estado en ejecución.
No encuentras el cambio de estado	Buscaste en logs en vez de storage; el evento describe, el storage guarda.
La llamada revierte sin motivo claro	Argumentos mal codificados; verifica selector y tipos contra la ABI.
Mezclas base fee y priority fee	Sumaste mal la comisión; sepáralas y recuerda que la base fee se quema.

🛡️ Seguridad y ética

- Trabaja en local con Anvil o en testnet; nunca uses fondos ni claves reales.

- No pegues claves privadas de mainnet en scripts ni variables de entorno del laboratorio.
- Trata la estimación de gas como orientación, no como compromiso, al diseñar interfaces.
- Los valores de base fee y de red varían minuto a minuto: consúltalo en vivo antes de concluir.
- Recuerda que Ethereum opera bajo Proof of Stake desde The Merge; describe el consenso con precisión.

🔗 Referencias

- Antonopoulos y Wood, *Mastering Ethereum*, caps. sobre la EVM y transacciones — https://github.com/ethereumbook/ethereumbook
- Wood, *Ethereum Yellow Paper*, secciones de ejecución y estado — https://ethereum.github.io/yellowpaper/paper.pdf
- Documentación para desarrolladores de ethereum.org — https://ethereum.org/developers/docs/
- Fuente primaria: EIP-1559 — https://eips.ethereum.org/EIPS/eip-1559

✅ Criterio de dominio

- Explicas el recorrido completo de una transacción y ubicas sus puntos de fallo.
- Codificas y decodificas una llamada por su selector y argumentos ABI.
- Justificas la diferencia entre estimación y coste final de gas con base fee y priority fee.

🧭 Navegación

⬅️ Módulo 04 · Bitcoin · 📚 Índice del currículo · ➡️ Módulo 06 · Solidity y Foundry

06 · Solidity y Foundry

> **Nivel:** Intermedio-Avanzado · ⏱️ **Duración estimada:** 180 min · **Fuente:** documentación de Solidity y *The Foundry Book* ⬅️ Currículo · 📚 Bibliografía 🧭 ⬅️ **Anterior:** 05 · Ethereum y EVM · 📚 Índice · ➡️ **Siguiente:** 07 · Aplicaciones descentralizadas 📖 Glosario de términos · 🌱 ¿Nuevo en esto? Empieza aquí

🎯 Objetivos

- Escribir un contrato en Solidity con tipos, visibilidad, modifiers, custom errors y eventos correctos.
- Diseñar invariantes de una bóveda antes de implementar su lógica.
- Ejecutar el flujo de Foundry: compilar, probar con detalle y aplicar fuzzing e invariantes.
- Aplicar el patrón checks-effects-interactions y control de acceso para prevenir reentrancy.
- Razonar sobre transferencias forzadas (selfdestruct/force-send) y su efecto en la contabilidad interna.

📚 Resultados de aprendizaje

Al finalizar, el estudiante podrá:

1. **Estructurar** un contrato con herencia, interfaces y librerías de forma clara.
2. **Formular** invariantes verificables antes de escribir la implementación.
3. **Ejecutar** pruebas unitarias, de fuzzing y de invariantes con Foundry.
4. **Aplicar** checks-effects-interactions para ordenar validaciones, efectos e interacciones externas.
5. **Distinguir** el saldo real del contrato de su contabilidad interna ante force-send.
6. **Justificar** el uso de custom errors y de patrones de control de acceso.

🗺️ Temas

#	Tema	Por qué importa
1	Tipos y visibilidad	Definen qué es accesible y cómo se comporta cada dato.
2	Modifiers y control de acceso	Centralizan las precondiciones y protegen funciones sensibles.
3	Custom errors y eventos	Abaratan el manejo de errores y hacen auditable el estado.
4	Herencia, interfaces y librerías	Permiten componer y reutilizar código de forma segura.
5	receive / fallback	Controlan cómo el contrato recibe ether y llamadas sin datos.
6	Layout de storage	Determina el coste y la seguridad de las variables persistentes.
7	Fuzzing e invariantes	Exploran estados inesperados que las pruebas de ejemplo no cubren.
8	checks-effects-interactions	Es la defensa base contra reentrancy.

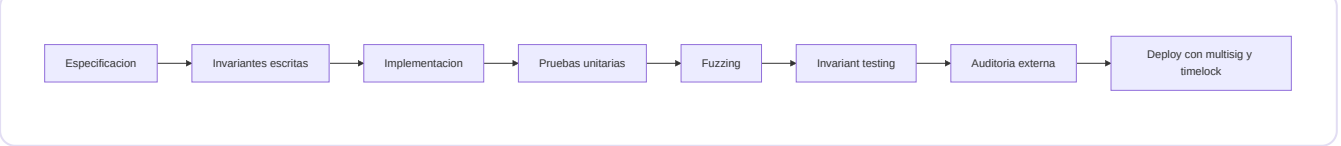
🧠 Modelo mental

Trata el contrato como la caja fuerte de un banco con reglas grabadas en acero: una vez desplegado, nadie cambia las reglas, así que deben ser correctas desde el primer día. Antes de instalar la cerradura defines las promesas que jamás pueden romperse (las invariantes): "nadie retira más de lo que depositó" y "la suma de las cuentas cuadra con el dinero administrado". Luego el fuzzing actúa como un ladrón incansable que prueba millones de combinaciones buscando una promesa rota.

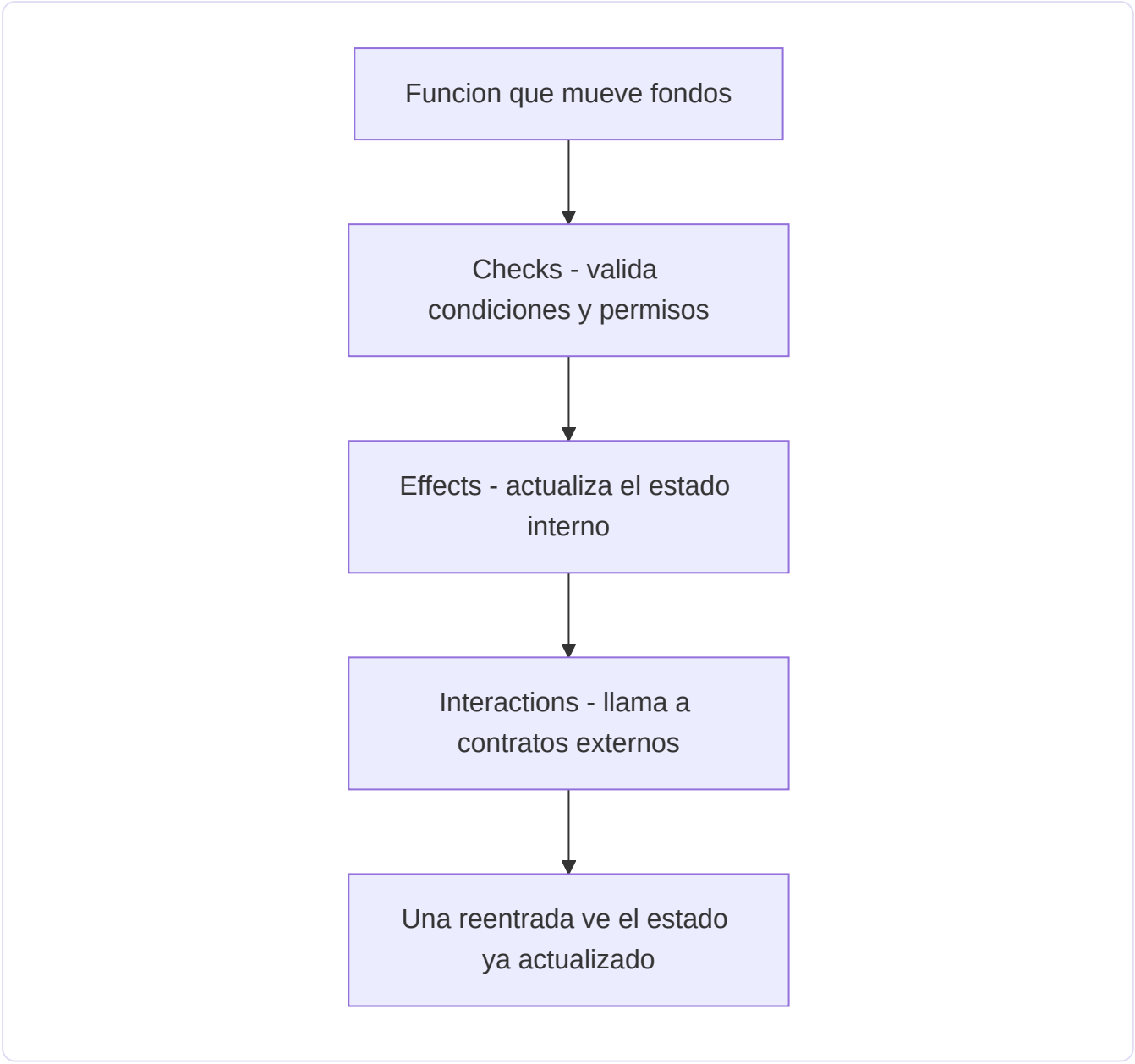
El límite de la analogía está en el force-send: alguien puede empujar ether al contrato sin pasar por la puerta (por ejemplo, mediante selfdestruct), de modo que el saldo real puede superar la contabilidad interna. Por eso una invariante nunca debe confiar en que `address(this).balance` iguale la suma contable; hay que separar lo que el contrato registra de lo que físicamente contiene.

🧩 Esquema visual

Pipeline profesional de un contrato: las invariantes se escriben antes que el código y cada etapa de prueba amplía la cobertura de la anterior.



El patrón checks-effects-interactions ordena cada función que mueve fondos para que una reentrada tardía ya no encuentre estado desactualizado.



📖 Conceptos y definiciones

- **Visibilidad**: `public`, `external`, `internal` o `private`; define quién puede invocar una función o leer una variable.
- **Modifier**: bloque reutilizable de precondiciones que envuelve a una función, por ejemplo un `onlyOwner`.
- **Custom error**: error tipado y barato declarado con `error`; reemplaza cadenas de `require` en revert.
- **Evento**: registro indexable que documenta cambios de estado para observadores externos.
- **Interfaz**: declaración sin implementación que fija cómo interactuar con otro contrato.
- **Librería**: código sin estado propio, reutilizable, que se enlaza o incrusta en contratos.

- **receive / fallback**: funciones especiales que gestionan ether entrante y llamadas sin coincidencia de selector.
- **Layout de storage**: disposición de variables en ranuras de 32 bytes; un orden incorrecto encarece o rompe upgrades.
- **Invariante**: propiedad que debe mantenerse cierta en todo estado alcanzable, verificada por Foundry.
- **checks-effects-interactions**: patrón que valida primero, actualiza el estado después e interactúa con el exterior al final.

🔬 Profundización

Inmutable vs. upgradeable: trade-offs reales

Un contrato inmutable es la promesa más fuerte que puedes dar; un proxy la relaja a cambio de poder corregir errores. Elegir es una decisión de gobernanza, no solo técnica.

Estrategia	Ventaja principal	Riesgo principal
Inmutable	Garantías máximas; sin admin que comprometer	Un bug es permanente; solo queda migrar a un contrato nuevo
Transparent Proxy	Patrón maduro; separa admin de usuarios	Más gas por llamada; el admin es un punto de confianza
UUPS	Lógica de upgrade en la implementación; llamadas más baratas	Si una versión olvida `_authorizeUpgrade`, el contrato queda congelado o secuestrable

El riesgo transversal de cualquier proxy es la **colisión de storage**: la implementación nueva debe respetar exactamente el orden y tipo de las variables previas (solo añadir al final). Cambiar `uint256 total` por `address owner` en el mismo slot reinterpreta bytes existentes como otro tipo, corrompiendo el estado sin ningún error de compilación. Por eso los patrones modernos usan slots deterministas separados (ERC-1967) y herramientas que comparan layouts entre versiones.

Fuzzing e invariantes en Foundry

En una prueba fuzz, Foundry genera cientos de valores para los argumentos de la función de test. Sin acotar, la mayoría de entradas revierten por triviales (montos absurdos, direcciones cero) y el fuzzing pierde potencia; con `bound(monto, 1, saldoMaximo)` concentras las corridas en el rango interesante.

Para invariantes, el **handler** es un contrato intermedio que envuelve a la bóveda y expone solo secuencias de acciones válidas (depositar, retirar, forzar envío de ether), además de llevar contadores fantasma. La invariante contable típica de un vault:

```
function invariant_contabilidadCuadra() public view {
    // el balance real puede EXCEDER lo registrado (force-send),
    // pero nunca ser menor que la suma de depósitos netos
    assertGe(address(vault).balance, handler.sumaDepositos() - handler.sumaRetiros());
}
```

Nota el `>=` en lugar de `==`: es exactamente la lección del force-send aplicada a una invariante ejecutable.

Gas y errores: custom errors y el optimizador

Un `require(cond, "mensaje largo de error")` incrusta la cadena en el bytecode (cada 32 bytes de string son bytecode adicional en el deploy) y al revertir codifica `Error(string)` con su overhead de memoria. Un custom error (`error Unauthorized();` + `revert Unauthorized();`) viaja como un selector de 4 bytes: como cifras orientativas, ahorra cientos de gas por revert y reduce el tamaño de despliegue en decenas de bytes por cada mensaje reemplazado, además de permitir parámetros tipados (`error InsufficientBalance(uint256 pedido, uint256 disponible)`).

El optimizador de Solidity se configura con `runs`: es una estimación de cuántas veces se ejecutará cada función a lo largo de la vida del contrato. `runs = 1` minimiza el tamaño del bytecode (deploy barato, llamadas algo más caras); `runs = 1000000` hace lo contrario. El valor por defecto de 200 es un compromiso; para un contrato que recibirá millones de llamadas, subir `runs` suele pagarse solo.

Leer una traza de Foundry (la habilidad que desatasca)

Cuando un test falla, la diferencia entre media hora y dos minutos es saber leer la traza. Ejecuta con `-vvvv` y Foundry imprime cada llamada, su gas y su resultado:

```
[FAIL. Reason: revert: saldo insuficiente] test_retiro() (gas: 34218)
Traces:
  [34218] VaultTest::test_retiro()
    ├─ [0] VM::prank(alice: [0x1234…])
    │   └─ ← ()
    ├─ [22431] Vault::depositar{value: 1000000000000000000}()
    │   ├─ emit Deposito(quien: alice, monto: 1000000000000000000)
    │   └─ ← ()
    ├─ [2553] Vault::retirar(2000000000000000000)
    │   └─ ← revert: saldo insuficiente
    └─ ← revert: saldo insuficiente
```

Cómo se lee, de arriba abajo:

- `[34218]` al inicio es el gas total del test; los `[…]` de cada línea, el de esa llamada. Un número desproporcionado en una línea señala dónde se va el gas.
- **La sangría es la pila de llamadas.** `Vault::retirar` está dentro del test; si un contrato llamara a otro, aparecería anidado un nivel más. Ahí se ve la reentrancia: la misma función apareciendo dos veces anidada.
- `←` es lo que devolvió. `← ()` es éxito sin retorno; `← revert: …` es el fallo.
- **El primer `revert` de abajo hacia arriba es la causa**; los de encima son la propagación. Buscar el más profundo, no el primero que se lee.
- **Los eventos** (`emit`) aparecen en su sitio exacto: si esperabas un evento y no está, la línea te dice si la función llegó a ejecutarse.

Tres banderas de `forge test` que resuelven casi todo:

Bandera	Para qué
`-vvvv`	Traza completa con llamadas internas y eventos
`--match-test test_retiro`	Ejecuta solo ese test mientras lo depuras
`--gas-report`	Tabla de gas por función: dónde optimizar de verdad

Y dentro del test, `console2.log("saldo", saldo);` imprime en la traza sin romper nada, tras `import {console2} from "forge-std/console2.sol";`.

El storage por dentro: dónde vive cada variable

El *storage layout* deja de ser abstracto cuando cuentas las ranuras. La EVM da a cada contrato ranuras de 32 bytes numeradas desde 0, y Solidity las asigna **en el orden de declaración**, empaquetando variables pequeñas si caben juntas.

```
contract Ejemplo {
    uint256 public total;      // ranura 0  (32 bytes, ocupa una entera)
    uint128 public techo;      // ranura 1, bytes 0–15   ┐ comparten
    uint128 public piso;       // ranura 1, bytes 16–31  ┘ la ranura 1
    address public duenio;     // ranura 2 (20 bytes)  ┐ comparten
    bool    public pausado;    // ranura 2 (1 byte)    ┘ la ranura 2
    mapping(address => uint256) public saldos;  // ranura 3 (reserva)
}
```

Dos consecuencias prácticas:

1. El orden de declaración cambia el gas. Si intercalas `uint256` entre los dos `uint128`, dejan de empaquetarse y el contrato usa una ranura más: un `SSTORE` extra cada vez que se escriben juntos. Ordenar de mayor a menor tamaño no es cosmética.

2. Los mappings no guardan nada en su ranura. La ranura 3 queda reservada pero vacía; el valor de `saldos[alice]` vive en `keccak256(abi.encode(alice, 3))`. Por eso un

mapping no se puede recorrer y por eso no "cabe" en un slot: sus entradas están dispersas por todo el espacio de direcciones.

Ahora se entiende exactamente por qué un upgrade que **inserta** una variable en medio corrompe los datos: si añades un `uint256 nuevo;` entre `total` y `techo`, `techo` y `piso` pasan a la ranura 2 — pero ahí siguen los bytes de `duenio` y `pausado` de la versión anterior. El contrato leerá una dirección como si fueran dos enteros. No hay error de compilación: solo datos que dejaron de significar lo que decían.

Compruébalo tú mismo con `forge inspect Ejemplo storageLayout`, que imprime la tabla de ranuras. Comparar esa salida entre dos versiones es exactamente la revisión que impide el fallo.

> 💡 **En una frase:** el storage son cajones numerados de 32 bytes que Solidity reparte por orden de declaración. Todo lo raro de los upgrades sale de ahí, y `forge inspect … storageLayout` te lo enseña sin adivinar.

▶ 🎓 **Si ya dominas esto** — los bordes que muerden

🧪 Laboratorio guiado

1. Antes de programar, redacta por escrito las dos invariantes de la bóveda: retiros nunca superiores al saldo del usuario y suma contable igual a los fondos administrados, considerando force-send.
2. Sitúate en el laboratorio, compila y ejecuta las pruebas con detalle, luego aumenta el fuzzing.

```
cd labs/06-solidity-vault
forge build
forge test -vv
forge test --fuzz-runs 1000
```

3. Observa la salida del fuzzing y anota qué entradas exploró el motor.
4. Añade o revisa una prueba de invariante que modele transferencias forzadas al contrato.
5. Revisa que cada función que toca fondos siga el orden checks-effects-interactions.
6. Consulta el catálogo de laboratorios en `../../labs/CATALOG.md` y la guía en `../../labs/guides/README.md` para contexto adicional.

📝 Reto verificable

Implementa (o corrige) la bóveda para que respete ambas invariantes y pase el fuzzing con al menos 1000 corridas, incluyendo un escenario de force-send.

Criterio de aceptación: `forge test --fuzz-runs 1000` termina sin fallos, existe una prueba de invariante que introduce fondos por force-send y ninguna función viola el orden checks-effects-interactions.

⚠️ Errores frecuentes

Síntoma	Causa y cómo comprobarlo
El fuzzing rompe la invariante de saldo	La contabilidad confía en `address(this).balance`; sepárala del registro interno.
Reentrancy vacía la bóveda	Interactúas antes de actualizar estado; reordena a checks-effects-interactions.
El upgrade corrompe datos	Cambiaste el orden del layout de storage; compara las ranuras antes y después.
El revert no informa nada útil	Usas cadenas caras o vacías; declara custom errors descriptivos.
Cualquiera ejecuta una función crítica	Falta control de acceso; añade un modifier y verifica el propietario.

🛡️ Seguridad y ética

- Trabaja siempre en local o testnet; nunca despliegues con fondos ni claves reales.
- Reutiliza componentes auditados como los de OpenZeppelin antes de reinventar primitivas de seguridad.
- Escribe las invariantes antes que el código para no ajustar las pruebas a errores existentes.
- Considera siempre las transferencias forzadas: el saldo real puede exceder la contabilidad interna.
- No publiques como seguro un contrato sin fuzzing ni revisión; los costes de gas y las prácticas cambian, consúltalo en vivo.

🔗 Referencias

- Documentación de Solidity, secciones de tipos, contratos y patrones — https://docs.soliditylang.org/
- *The Foundry Book*, capítulos de testing, fuzzing e invariantes — https://book.getfoundry.sh/
- OpenZeppelin Contracts, guía de control de acceso y seguridad — https://docs.openzeppelin.com/contracts/
- Fuente primaria: EIP-4626, estándar de bóvedas tokenizadas — https://eips.ethereum.org/EIPS/eip-4626

✅ Criterio de dominio

- Enuncias las invariantes de la bóveda antes de implementarla y las verificas con Foundry.
- Justificas el orden checks-effects-interactions en cada función que mueve fondos.
- Explicas cómo el force-send desacopla el saldo real de la contabilidad interna.

🧭 Navegación

⬅️ Módulo 05 · Ethereum y EVM · 📚 Índice del currículo · ➡️ Módulo 07 · Aplicaciones descentralizadas

07 · Aplicaciones descentralizadas

> **Nivel:** Intermedio-Avanzado · ⏱️ **Duración estimada:** 150 min · **Fuente:** documentación de ethereum.org y de viem ⬅️ Currículo · 📚 Bibliografía 🧭 ⬅️ **Anterior:** 06 · Solidity y Foundry · 📚 Índice · ➡️ **Siguiente:** 08 · Tokens y estándares
> 📖 Glosario de términos · 🌱 ¿Nuevo en esto? Empieza aquí

🎯 Objetivos

- Describir las partes de una dApp: interfaz, proveedor RPC, wallet, contratos, indexación y servicios externos.
- Distinguir lecturas públicas de escrituras firmadas y cuándo cada una requiere una wallet.
- Simular una transacción antes de enviarla para anticipar su efecto y sus fallos.
- Interpretar los estados de una transacción: pending, confirmed, replaced y reverted.
- Construir una interfaz que muestre contrato, red, valor y efecto esperado antes de pedir la firma.

📚 Resultados de aprendizaje

Al finalizar, el estudiante podrá:

1. **Diagramar** el flujo de datos de una dApp y ubicar dónde vive la fuente de verdad.
2. **Separar** llamadas de solo lectura de transacciones que modifican estado.
3. **Simular** una operación con una llamada previa antes de solicitar la firma.
4. **Mostrar** al usuario el efecto esperado, la red y el contrato antes de firmar.
5. **Gestionar** decimales, unidades y approvals con una UX que evite errores.
6. **Mitigar** la dependencia de un RPC no confiable mediante redundancia y verificación.

🗺️ Temas

#	Tema	Por qué importa
1	Anatomía de una dApp	La interfaz no es fuente de verdad; el estado vive en la cadena.
2	Conexión y cambio de red	Firmar en la red equivocada arruina la operación.
3	Lectura vs. escritura	Diferencia lo gratuito y sin riesgo de lo que gasta y compromete.
4	Simulación previa	Anticipa reverts y efectos antes de gastar gas.
5	Estados de transacción	Permite dar feedback honesto y manejar reemplazos.
6	Decimales y approvals	Un error de unidades o un approval abierto expone fondos.
7	RPC no confiable	Un proveedor único puede mentir o caerse; la redundancia protege.
8	Firmas legibles (EIP-712)	Muestran al usuario qué está firmando en lenguaje comprensible.

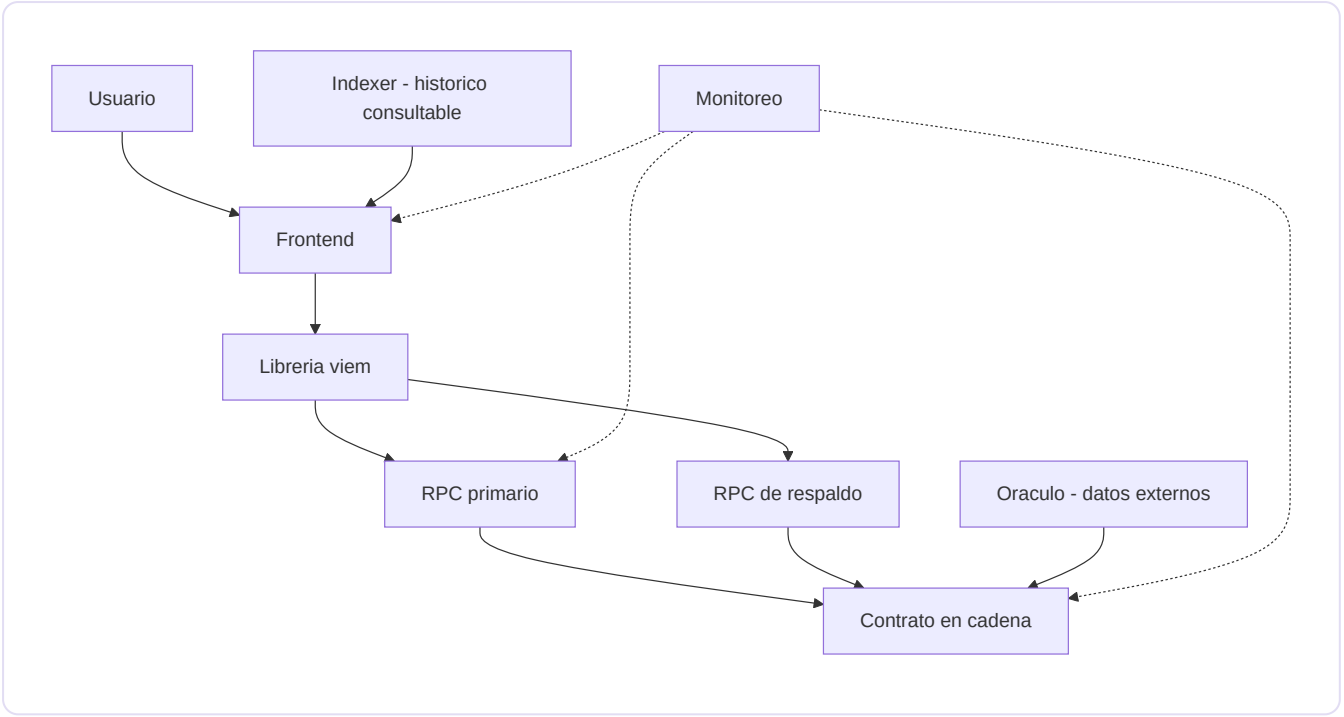
🧠 Modelo mental

Piensa en la dApp como el mostrador de un banco y en la blockchain como la bóveda del fondo. El cajero (la interfaz) te enseña saldos y formularios, pero no guarda el dinero: solo transmite tus órdenes firmadas a la bóveda, donde ocurre lo real. Si el cajero muestra una cifra bonita pero la bóveda dice otra cosa, la bóveda siempre gana. Antes de firmar, un buen mostrador te lee en voz alta a quién pagas, cuánto y en qué red.

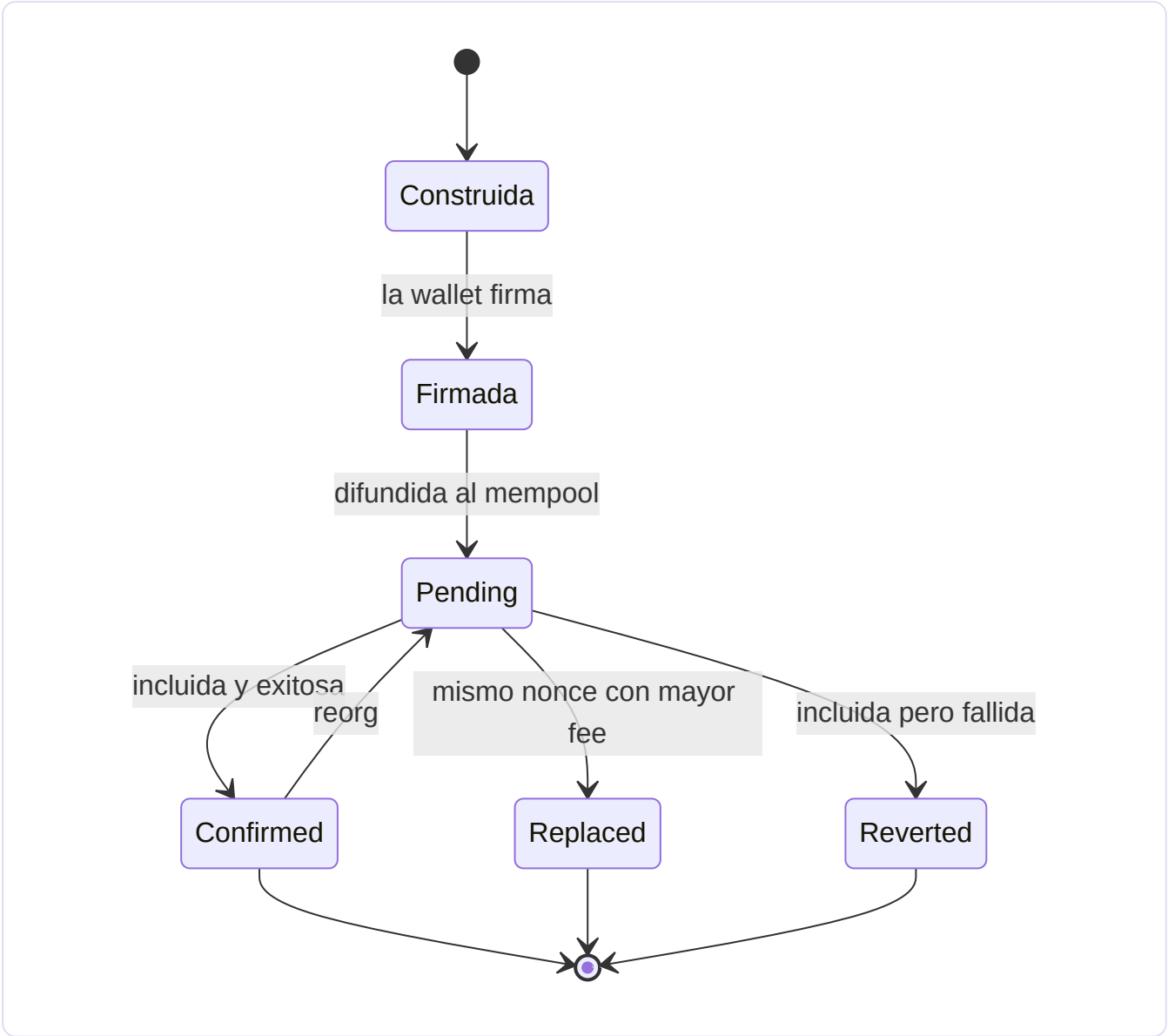
La analogía tiene un límite importante: en la banca tradicional confías en una sola sucursal, mientras que aquí el "cajero" consulta a través de un proveedor RPC que podría estar equivocado o ser malicioso. Por eso la interfaz debe tratar sus propios datos como provisionales, simular antes de enviar y, cuando importa, contrastar con más de una fuente en lugar de creer ciegamente a un único RPC.

🧩 Esquema visual

Arquitectura de una dApp en producción: el frontend nunca habla "directo" con la cadena, sino a través de una librería y RPC redundantes, con indexer y oráculo como fuentes laterales y monitoreo transversal.



Estados por los que pasa una transacción desde que la interfaz la construye; nota que un reorg puede devolver una transacción confirmada a pending.



📖 Conceptos y definiciones

- **Proveedor RPC**: servicio que la dApp consulta para leer estado y difundir transacciones; puede ser no confiable.
- **Lectura pública**: consulta de solo lectura que no cuesta gas ni requiere firma, por ejemplo un `eth_call`.
- **Escritura firmada**: transacción que modifica estado, requiere firma de la wallet y consume gas.
- **Simulación**: ejecución previa (mediante `eth_call` o simulate) que predice el resultado sin enviarlo.
- **Estado pending**: transacción difundida y aún no incluida en un bloque.
- **Estado replaced**: transacción sustituida por otra con el mismo nonce y mayor comisión.
- **Estado reverted**: transacción incluida pero cuya ejecución falló y deshizo sus efectos.

- **Approval**: permiso que autoriza a un contrato a mover tus tokens; un límite abierto es un riesgo.
- **Decimales y unidades**: los tokens usan enteros escalados; confundir unidades altera el monto real.
- **EIP-712**: estándar de firma de datos estructurados que hace legible para el usuario qué está firmando.

🔬 Profundización

Reorgs y finalidad: qué mostrar al usuario

Antes de The Merge la única defensa contra reorganizaciones era esperar N confirmaciones y cruzar los dedos. Desde 2022 el consenso Proof of Stake de Ethereum ofrece garantías explícitas que la interfaz puede consultar por el tag del bloque:

- **latest**: la cabeza de la cadena; puede reorganizarse en los siguientes slots.
- **safe**: bloque justificado por la mayoría de validadores; un reorg es ya muy improbable.
- **finalized**: bloque finalizado tras dos épocas (≈ 12,8 minutos, con épocas de 6,4 minutos); revertirlo exigiría destruir al menos un tercio del stake total.

Una UX honesta refleja esto en capas: "incluida" al ver el recibo sobre `latest`, "confirmada" tras algunos bloques o al alcanzar `safe`, y "definitiva" solo en `finalized` para montos altos. Un mini-caso real: en mayo de 2022 la beacon chain sufrió un reorg de 7 bloques; toda interfaz que hubiera marcado como definitiva una transacción con 5 confirmaciones habría mentido al usuario. Para pagos pequeños, `safe` suele ser el equilibrio razonable entre latencia y riesgo.

EIP-1193 y EIP-6963: cómo la dApp encuentra la wallet

EIP-1193 define la interfaz estándar del provider inyectado: un objeto con `request({ method, params })` y eventos como `accountsChanged` y `chainChanged`. Gracias a ese contrato único, viem o wagmi funcionan con cualquier wallet que lo implemente.

Su punto débil era el descubrimiento: todas las wallets peleaban por el mismo `window.ethereum` y la última en inyectarse "ganaba". **EIP-6963** (2023) lo resuelve con un protocolo de anuncio por eventos del DOM: cada wallet emite `eip6963:announceProvider` con sus metadatos (nombre, icono, identificador) y la dApp las lista todas, dejando elegir al usuario. Toda interfaz moderna debería soportar EIP-6963 con EIP-1193 como respaldo.

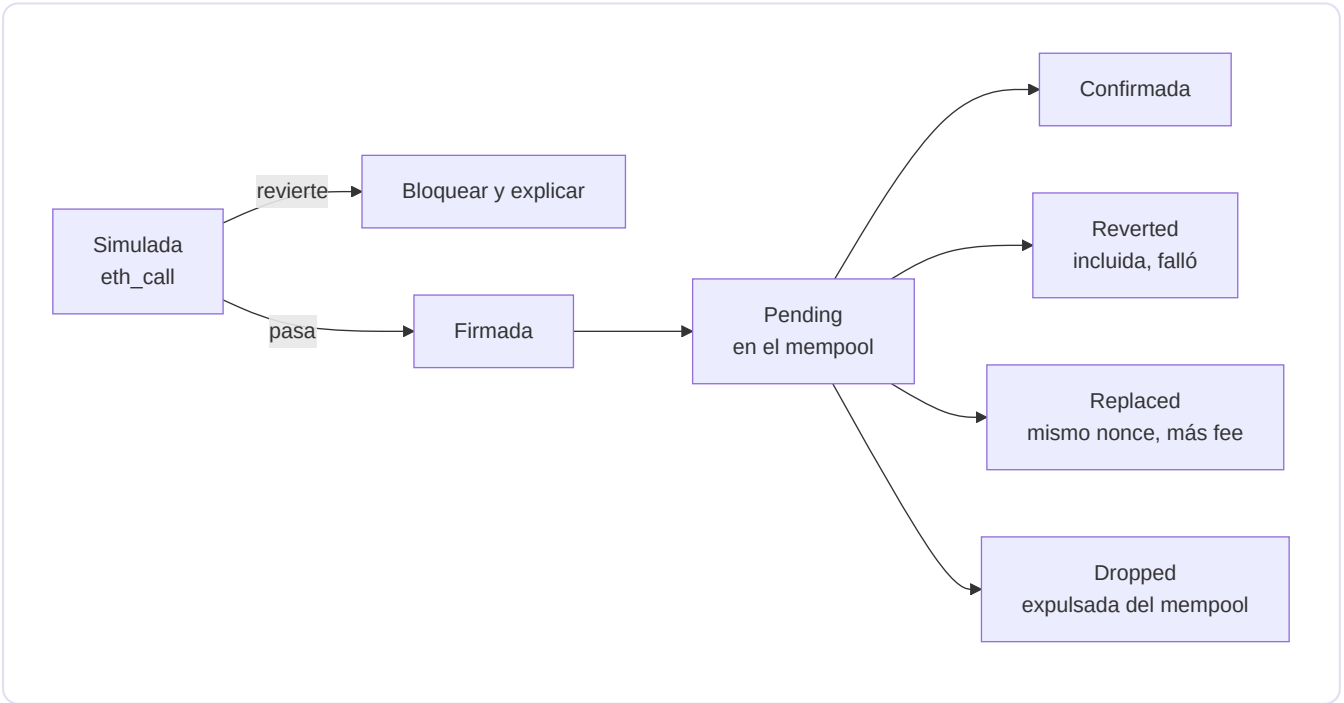
Patrones de robustez RPC

Patrón	Problema que resuelve
Fallback transport de viem	Un RPC caído o lento no tumba la dApp; las peticiones rotan al siguiente proveedor configurado
Multicall (lecturas agregadas)	Cien `eth_call` individuales saturan el rate limit; un solo contrato multicall las resuelve en una petición
Simulación con `eth_call` previa	Enviar sin simular quema gas en reverts previsibles; la simulación anticipa el fallo gratis
Verificación cruzada de datos críticos	Un RPC único puede mentir u ofrecer estado desactualizado; contrastar dos proveedores lo detecta

En viem, el fallback se declara al crear el cliente (`fallback([http(rpcA), http(rpcB)])`) y el multicall está integrado como batching automático de lecturas de contrato: activarlo suele ser un cambio de configuración, no una reescritura. La regla operativa: toda escritura pasa antes por una simulación, y toda lectura que dispare decisiones de dinero se verifica contra más de una fuente.

El ciclo de vida de una transacción, y qué mostrar en cada estado

La mayoría de las dApps que confunden al usuario cometen el mismo error: tratan "enviada" como "hecha". Una transacción atraviesa estados que la interfaz debe distinguir, porque cada uno exige una acción distinta.



Estado	Qué pasó	Qué debe hacer la interfaz
Simulada	`eth_call` con el mismo calldata, sin enviar	Si revierte, **no pedir la firma**: mostrar el motivo. Firmar algo que sabes que falla es cobrarle gas al usuario por nada
Pending	Difundida, sin bloque	Mostrar el hash y que se puede acelerar o cancelar. **No** decir "completado"
Confirmada	En un bloque, `status = 1`	Releer el estado desde el RPC, no asumirlo
Reverted	En un bloque, `status = 0`	Se pagó el gas y no pasó nada. Decirlo así: el usuario ve el cobro y no ve el efecto
Replaced	Otra transacción con el mismo nonce y más comisión ocupó su lugar	Seguir el **nonce**, no el hash. Si sigues el hash, la transacción "desaparece"
Dropped	El mempool la descartó (llevaba demasiado tiempo o subió el mínimo)	Ofrecer reenviar. No se quedará pendiente para siempre

La regla que se deriva: **la fuente de verdad es la cadena, no tu estado local**. Tras confirmar, vuelve a leer el saldo desde el RPC. Si la interfaz suma el monto a su copia en memoria, cualquier divergencia (un revert, otra transacción del mismo usuario en otra pestaña) deja al usuario mirando un número que no existe.

Decimales: el error de tres ceros

Es el fallo más caro que comete un principiante y el más fácil de evitar. **Los tokens no tienen decimales: tienen enteros y un número que dice dónde imaginar la coma.**

```
USDC  → decimals = 6      1 USDC  = 1 000 000 unidades
WETH  → decimals = 18     1 WETH  = 1 000 000 000 000 000 000 unidades
WBTC  → decimals = 8      1 WBTC  = 100 000 000 unidades
```

Si asumes 18 porque "todos usan 18" y el token es USDC, enviar "5" da:

```
lo que pretendías:  5 USDC     =         5 000 000 unidades
lo que enviaste:    5×10¹⁸     = 5 000 000 000 000 000 000 unidades
                                = 5 000 000 000 000 USDC
```

Un factor de 10^{12}. No hay confirmación que te salve: la transacción es válida y el token se movió.

Tres reglas que lo cierran:

1. **Nunca escribas el factor a mano.** `parseUnits("5", decimals)` y `formatUnits(valor, decimals)`; el `parseEther` de viem asume 18 y solo vale para ETH.

2. **Lee `decimals()` del contrato**, no de una lista. Un token puede tener el `decimals` que quiera.
3. **Nada de flotantes.** `0.1 + 0.2 !== 0.3` en JavaScript, y aquí eso es dinero. Los montos van en `BigInt` de punta a punta; el formateo a texto es lo último que se hace, solo para mostrar.

El mismo razonamiento aplica a los feeds de precio: un oráculo con `decimals = 8` que devuelve `312450000000` son 3 124,50, no 312 450 000 000. Normaliza antes de comparar cualquier cosa con cualquier cosa.

> 💡 **En una frase:** los tokens solo manejan enteros, y `decimals` dice dónde imaginar la coma. Lee ese número del contrato, convierte con `parseUnits` / `formatUnits` y no dejes que un flotante toque un monto.

▶ 🎓 **Si ya dominas esto** — el detalle que rompe integraciones

🧪 Laboratorio guiado

> 🧪 Estas prácticas están catalogadas y **resueltas paso a paso** en el catálogo de laboratorios.

1. Levanta el panel del repositorio para navegar los recursos del curso.

```
pnpm serve
```

2. Compila la aplicación web del repositorio, ubicada en `apps/community-funding-web`.

```
pnpm build:web
```

3. En la interfaz del Vault construida con TypeScript y viem, conecta una wallet y verifica la red activa.
4. Realiza una lectura pública del estado del contrato y muéstrala sin pedir firma.
5. Simula una escritura y presenta al usuario contrato, red, valor y efecto esperado antes de firmar.
6. Provoca un caso de revert en la simulación y confirma que la interfaz lo comunica con claridad.

📝 Reto verificable

Entrega una interfaz para el Vault que, antes de solicitar cualquier firma, muestre la dirección del contrato, la red, el valor y el efecto esperado, y que simule la operación previamente.

Criterio de aceptación: la interfaz bloquea la firma si la red no coincide, muestra el efecto simulado con unidades correctas y comunica explícitamente los estados pending, confirmed y reverted.

⚠️ Errores frecuentes

Síntoma	Causa y cómo comprobarlo
La UI muestra un saldo que la cadena desmiente	Tratas la interfaz como fuente de verdad; relee el estado desde el RPC.
El usuario firma en la red equivocada	No validas chain id antes de firmar; bloquea la acción hasta cambiar de red.
El monto enviado es mil veces mayor	Confundiste decimales/unidades; convierte con la función adecuada y verifica.
Approval deja fondos expuestos	Autorizaste un límite abierto; usa montos acotados y revócalos tras usarlos.
La transacción "desaparece"	Fue replaced por mismo nonce y mayor fee; sigue el nonce, no solo el hash.

🛡️ Seguridad y ética

- Trabaja en local o testnet; nunca uses fondos ni claves reales en los laboratorios.
- Simula siempre antes de enviar y muestra el efecto esperado antes de pedir la firma.
- No confíes en un único RPC: contrasta datos críticos y prevé su indisponibilidad.
- Prefiere firmas legibles con EIP-712 para que el usuario entienda qué autoriza.
- Evita approvals ilimitados y explica sus riesgos; las condiciones de red cambian, consúltalo en vivo.

🔗 Referencias

- Documentación de ethereum.org sobre dApps — https://ethereum.org/developers/docs/dapps/
- Documentación de viem, guías de clientes y acciones — https://viem.sh/
- Documentación de wagmi, hooks para interfaces React — https://wagmi.sh/
- Fuente primaria: EIP-712, firma de datos estructurados — https://eips.ethereum.org/EIPS/eip-712

✅ Criterio de dominio

- Explicas por qué la interfaz no es fuente de verdad y dónde vive el estado real.
- Muestras contrato, red, valor y efecto esperado antes de solicitar una firma.
- Manejas los estados de una transacción y justificas la redundancia de RPC.

🧭 Navegación

⬅️ Módulo 06 · Solidity y Foundry · 📚 Índice del currículo · ➡️ Módulo 08 · Tokens y estándares

08 · Tokens y estándares

> **Nivel:** Intermedio-Avanzado · ⏱️ **Duración estimada:** 150 min · **Fuente:** EIPs de Ethereum y OpenZeppelin Contracts ⬅️ Currículo · 📚 Bibliografía 🧭 ⬅️ **Anterior:** 07 · Aplicaciones descentralizadas · 📚 Índice · ➡️ **Siguiente:** 09 · Seguridad y auditoría 📖 Glossario de términos · 🌱 ¿Nuevo en esto? Empieza aquí

🎯 Objetivos

- Distinguir con precisión los estándares ERC-20, ERC-721, ERC-1155 y ERC-4626, y cuándo elegir cada uno.
- Explicar el modelo de allowances, el riesgo del approve infinito y la alternativa de firmas con ERC-2612 (`permit`).
- Implementar un token educativo apoyado en una biblioteca auditada, documentando autoridad de mint/burn, suministro máximo y controles de emergencia.
- Analizar la distribución y concentración inicial de un suministro y justificar por qué el token debe existir.
- Ubicar la abstracción de cuenta (ERC-4337 y EIP-7702, activo desde Pectra en 2025) dentro de la experiencia de usuario de un token.

📚 Resultados de aprendizaje

Al finalizar, el estudiante podrá:

1. **Comparar** los estándares fungibles y no fungibles según interfaz, casos de uso y coste de gas.
2. **Implementar** un token ERC-20 con suministro máximo, pausa y recuperación usando OpenZeppelin.
3. **Justificar** la política de mint/burn y de allowances frente a los riesgos de concentración y approve infinito.
4. **Diseñar** metadata coherente para un ERC-721/1155 y explicar las royalties con ERC-2981.
5. **Explicar** cómo `permit` (ERC-2612) y la abstracción de cuenta mejoran la UX sin sacrificar seguridad.
6. **Modelar** un escenario de emisión con el simulador de tokenomics del repositorio.

🗺️ Temas

#	Tema	Por qué importa
1	ERC-20 y allowances	Es el estándar fungible base; sus allowances concentran el riesgo de aprobaciones.
2	ERC-721 y metadata	Define la unicidad y la referencia a atributos fuera de cadena.
3	ERC-1155 multi-token	Un solo contrato gestiona fungibles y no fungibles con transferencias por lotes.
4	ERC-2612 `permit`	Elimina la transacción de `approve` mediante firma, reduciendo fricción y coste.
5	Royalties ERC-2981	Estandariza cómo se señala una regalía, aunque su cumplimiento sea voluntario.
6	Bóvedas ERC-4626	Unifica la contabilidad de bóvedas tokenizadas y evita errores de conversión.
7	Abstracción de cuenta	ERC-4337 y EIP-7702 permiten cuentas programables y patrocinio de gas.
8	Autoridad y suministro	Quién puede mintar, quemar o pausar determina el poder real sobre el token.

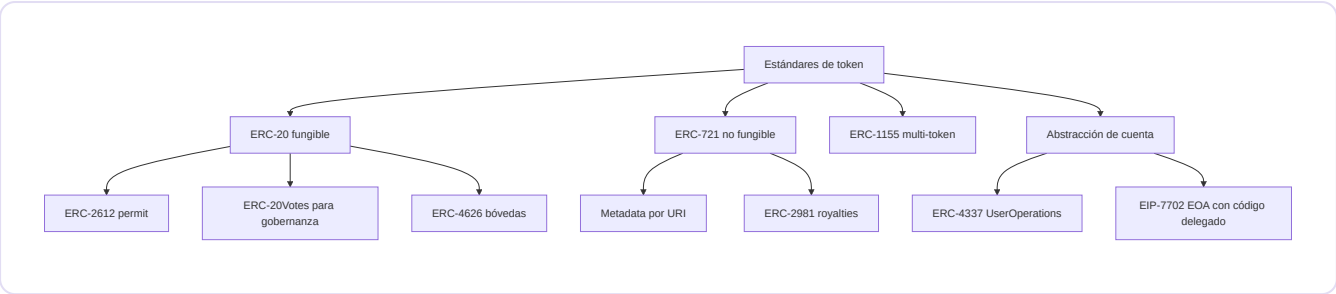
🧠 Modelo mental

Un estándar de token es como el formato de un enchufe eléctrico: no dice qué aparato conectas ni cuánta energía consume, solo garantiza que cualquier billetera o protocolo pueda "enchufarse" y hablar el mismo idioma de funciones y eventos. Gracias a esa interfaz común, un token ERC-20 recién creado ya es visible en billeteras y exploradores sin que nadie lo haya integrado a mano.

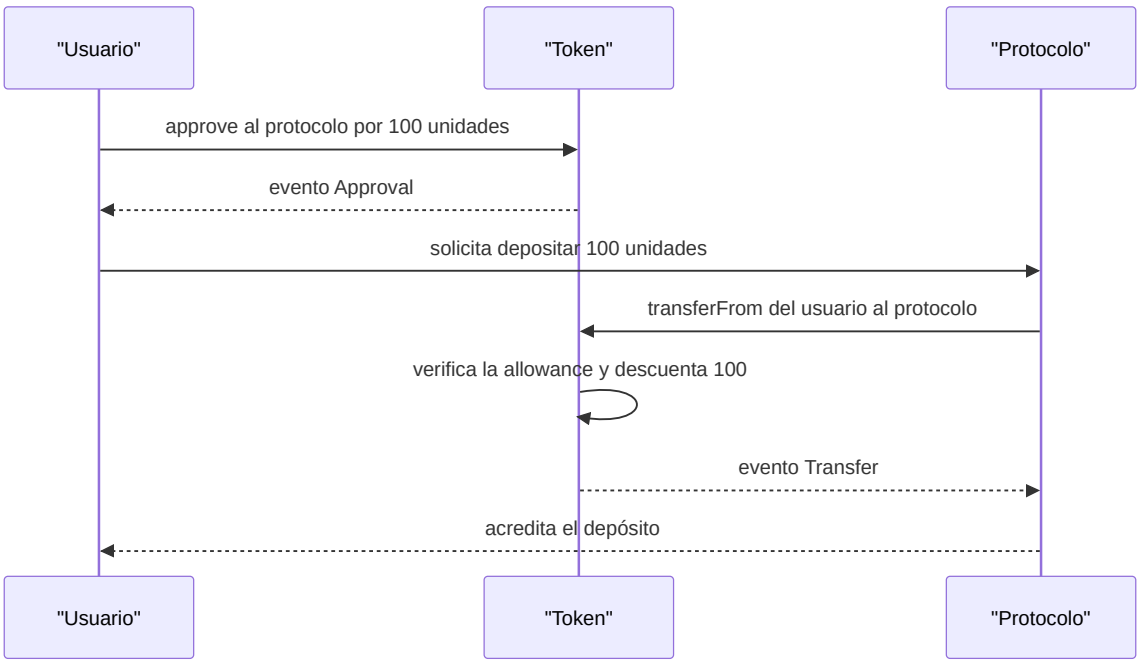
El límite de la analogía es importante: el enchufe no juzga la calidad del aparato. Que un contrato implemente correctamente `transfer` y `balanceOf` no dice nada sobre si su suministro está concentrado en una dirección, si el dueño puede mintar sin tope o si el token tiene alguna utilidad real. Crear un token es trivial; crear utilidad, seguridad y gobernanza responsable es el trabajo difícil.

🧩 Esquema visual

El mapa de estándares parte de dos preguntas: si las unidades son intercambiables entre sí y qué extensiones exige el caso de uso.



El patrón `approve` + `transferFrom` requiere dos transacciones y deja viva una autorización que conviene acotar en monto y en tiempo.



📖 Conceptos y definiciones

- **Allowance**: monto que una dirección autoriza a que otra gaste en su nombre; el approve infinito lo fija en el máximo y expone todo el saldo si el gastador se ve comprometido.
- **Mint / burn**: creación y destrucción de unidades; su autoridad debe estar acotada por roles, tope de suministro y, si es posible, renuncia verificable.
- **Suministro máximo (cap)**: límite superior de unidades emitibles; sin él, la dilución queda a discreción del emisor.
- **`permit` (ERC-2612)**: aprobación mediante firma off-chain con `nonce` y `deadline`, que evita una transacción separada de `approve`.
- **Metadata**: descripción de un token (nombre, imagen, atributos) referenciada por URI; en NFT suele apuntar a IPFS para direccionamiento por contenido.
- **Royalty (ERC-2981)**: interfaz que indica beneficiario y porcentaje de regalía, cuyo pago depende de que el mercado decida respetarlo.

- **Bóveda ERC-4626**: estándar que tokeniza depósitos en un activo subyacente unificando `deposit`, `mint`, `withdraw` y `redeem` y su contabilidad de `shares`.
- **Abstracción de cuenta**: capacidad de que una cuenta ejecute lógica programable; ERC-4337 usa UserOperations y EIP-7702 permite a una EOA delegar temporalmente en código.
- **Hook**: función que se ejecuta antes o después de una transferencia para añadir lógica como pausas o listas de bloqueo.

🔬 Profundización

Contabilidad de una bóveda ERC-4626: shares vs. assets

Una bóveda ERC-4626 no guarda "saldos en USDC" por usuario: emite *shares* que representan una fracción proporcional del total de activos. El tipo de cambio es `assets / shares` y crece cuando la bóveda gana rendimiento. Ejemplo numérico: una bóveda con 12 500 USDC y 10 000 shares tiene un tipo de cambio de 1.25; si depositas 500 USDC recibes `500 / 1.25 = 400` shares, y si más tarde el total sube a 15 000 USDC, tus 400 shares valen `400 × 1.5 = 600` USDC.

Esa aritmética habilita el **ataque de inflación del primer depósito**: en una bóveda vacía, el atacante deposita 1 unidad mínima y recibe 1 share; luego "dona" 10 000 USDC transfiriéndolos directamente al contrato, sin pasar por `deposit`. Cuando la víctima deposita 19 999 USDC, la fórmula `shares = 19 999 × 1 / 10 000` redondea hacia abajo a 1 share: víctima y atacante quedan con la mitad de una bóveda de casi 30 000 USDC, y el atacante retira ~15 000 USDC habiendo aportado ~10 000. Las mitigaciones estándar son los *virtual shares/assets* (un offset interno que usa OpenZeppelin desde la versión 4.9 y encarece el ataque hasta hacerlo antieconómico) o un depósito inicial del propio protocolo cuyas shares se queman o quedan bloqueadas.

El riesgo real de las approvals: del approve infinito a Permit2

El `approve` por el máximo (`type(uint256).max`) es cómodo, pero convierte cada allowance en una llave permanente: si el contrato aprobado se ve comprometido, o si el usuario firma ante un *drainer* de phishing, todo el saldo queda expuesto sin necesidad de robar la clave privada. Los kits de drenado que operan desde 2022 explotan justamente firmas de `approve`, `permit` y `Permit2` obtenidas con interfaces engañosas, y han causado pérdidas acumuladas de cientos de millones de dólares; la cifra exacta varía por informe, consúltala en vivo.

Mecanismo	Cómo autoriza	Ventaja	Riesgo característico
`approve` clásico	Transacción on-chain por gastador	Universal, soportado por todo ERC-20	Allowances infinitas olvidadas durante años
`permit` (ERC-2612)	Firma off-chain EIP-712 con `nonce` y `deadline`	Sin transacción previa; caducidad explícita	Solo lo implementan tokens que adoptaron el estándar; una firma robada vale hasta su `deadline`
Permit2 (Uniswap)	Un `approve` único a Permit2 + firmas por protocolo con monto y expiración	Lleva `permit` a cualquier ERC-20; permisos granulares y revocables	Permit2 se vuelve punto de concentración: una firma engañosa autoriza a un tercero

La higiene mínima: aprobar montos acotados, revisar y revocar allowances periódicamente y desconfiar de cualquier firma cuyo contenido la interfaz no muestre con claridad.

Decimales y aritmética: por qué USDC usa 6 y casi todo lo demás 18

`decimals` es solo metadato de presentación: el contrato opera siempre con enteros. USDC y USDT usan 6 decimales por herencia de sus sistemas contables originales, mientras que ETH (18) fijó la convención que la mayoría de tokens copia. Mezclarlos sin normalizar produce errores de factor 10^{12}: 1 USDC son `1 000 000` unidades base, pero 1 DAI son `1 000 000 000 000 000 000`. Un contrato que compara ambos crudos concluiría que 1 DAI "vale" un billón de veces más que 1 USDC.

Además, la división entera siempre trunca: convertir 1 unidad base de USDC a un token de 18 decimales y de vuelta puede perder el resto del redondeo. La regla profesional es redondear siempre en contra del usuario que retira y a favor del protocolo (como exige ERC-4626 en `previewWithdraw` y `previewRedeem`), porque los restos acumulados a favor del usuario son exactamente la grieta que explota el ataque de inflación descrito arriba.

Cómo se vacía una cartera con una firma

El titular "firmó algo y perdió todo" suena a descuido. Casi nunca lo es: es una cadena de decisiones razonables que termina mal. Verla completa es lo que enseña a cortarla.

El montaje. Un sitio ofrece reclamar un airdrop. Para "verificar que eres titular" pide una firma. No pide dinero, no pide la clave privada, no envía ninguna transacción — por eso parece inofensivo.

Los tres pasos:

1. **La víctima firma un `permit` (ERC-2612).** Es una firma off-chain: no cuesta gas, no aparece en el explorador y la wallet la muestra como un texto que casi nadie lee. Lo que autoriza es una allowance por el máximo posible.

2. **El atacante lleva esa firma a la cadena.** Llama a `permit` en el contrato del token con la firma de la víctima. La allowance queda registrada, y **la paga él**: para la víctima no hay ni una transacción sospechosa en su historial.
3. **Llama a `transferFrom`** y se lleva el saldo entero. Legítimo desde el punto de vista del contrato: hay una autorización válida firmada por la titular.

Dónde se corta la cadena. Fíjate en que cada eslabón tiene una defensa distinta:

Eslabón	Defensa	Quién la implementa
La firma parece inofensiva	EIP-712 muestra **qué** se autoriza en texto legible	La wallet
La allowance es infinita	Autorizar solo lo necesario	La dApp, al construir la petición
El plazo es eterno	`deadline` corto: minutos, no años	La dApp
La autorización sobrevive al uso	Revocar tras operar	La persona usuaria

El detalle que lo hace peligroso: una firma no aparece en el historial de transacciones. Alguien puede revisar su cartera, no ver nada raro, y tener una autorización activa firmada hace semanas esperando el momento. Por eso la revisión periódica de allowances no es paranoia: es la única forma de ver lo que el historial no muestra.

> 💡 **En una frase:** una firma sin gas no es una firma inofensiva. Lo que autoriza puede ejecutarlo otro, cuando quiera, y pagándolo él.

▶ 🎓 **Si ya dominas esto** — el detalle que separa un token correcto de uno que rompe integraciones

🧪 Laboratorio guiado

> 🧪 Estas prácticas están catalogadas y **resueltas paso a paso** en el catálogo de laboratorios.

1. Explora el simulador de suministro para ver el efecto de distintas políticas de emisión.

```
pnpm lab:tokenomics
```

2. Modela dos escenarios: uno con suministro máximo fijo y otro con emisión continua, y compara la concentración resultante en las primeras direcciones.

3. Abre los contratos del módulo en `labs/08-protocols` y ejecuta la suite de pruebas con trazas para observar transferencias, allowances y eventos.

```
forge test -vv
```

4. Localiza en el código de prueba una llamada a `permit` y verifica cómo la firma sustituye a una transacción de `approve`.

📝 Reto verificable

Implementa un token educativo ERC-20 apoyado en OpenZeppelin con: suministro máximo fijo, roles separados para mint y pausa, función de recuperación de tokens enviados por error y documentación de la autoridad de cada rol.

Criterio de aceptación: `forge test -vv` pasa en verde; el suministro no puede superar el cap ni siquiera por el rol de mint; existe una prueba que demuestra que una cuenta sin rol no puede mintar ni pausar; y el README del token justifica en un párrafo por qué el token debe existir y cómo se distribuye.

⚠️ Errores frecuentes

Síntoma	Causa y cómo comprobarlo
El saldo de un usuario se vacía tras firmar en un sitio	Approve infinito a un contrato malicioso; revisa las allowances activas y revócalas.
El `permit` falla siempre	`deadline` vencido o `nonce` reutilizado; imprime ambos valores en la prueba y compáralos.
La imagen del NFT desaparece	Metadata en un servidor no persistente; usa un CID de IPFS con pinning verificado.
Las regalías no se pagan	ERC-2981 solo señala la regalía; comprueba si el mercado la respeta antes de asumir ingresos.
El suministro crece sin control	Falta de `cap` o mint sin restricción; añade una prueba que intente superar el tope y espere revert.
Confundir `shares` con activos en la bóveda	Conversión ERC-4626 mal interpretada; valida con `convertToAssets` y `convertToShares`.

🛡️ Seguridad y ética

- Trabaja siempre en local o testnet; nunca uses fondos ni claves privadas reales en los laboratorios.
- Prefiere bibliotecas auditadas (OpenZeppelin) antes que reimplementar estándares desde cero.

- Documenta y minimiza los privilegios: quién puede mintar, quemar, pausar o recuperar, y cómo se renuncia a ellos.
- Advierte sobre el approve infinito en cualquier interfaz y ofrece aprobaciones acotadas o mediante `permit`.
- Evalúa la concentración del suministro: un token con utilidad real y distribución transparente es más defendible que uno especulativo.

🔗 Referencias

- Ethereum Foundation, *EIPs — Ethereum Improvement Proposals* — https://eips.ethereum.org/
- ERC-20, *Token Standard* — https://eips.ethereum.org/EIPS/eip-20
- ERC-721, *Non-Fungible Token Standard* — https://eips.ethereum.org/EIPS/eip-721
- ERC-1155, *Multi Token Standard* — https://eips.ethereum.org/EIPS/eip-1155
- ERC-4626, *Tokenized Vaults* — https://eips.ethereum.org/EIPS/eip-4626
- OpenZeppelin, *Contracts — documentación* — https://docs.openzeppelin.com/contracts/
- Antonopoulos & Wood, *Mastering Ethereum*, cap. sobre tokens — https://github.com/ethereumbook/ethereumbook
- Fuente primaria: ERC-2612, *`permit` — aprobaciones firmadas (712)* — https://eips.ethereum.org/EIPS/eip-2612

✅ Criterio de dominio

- Entregas un token ERC-20 con cap, roles y pruebas en verde, más un README que justifica su existencia y distribución.
- Explicas sin apoyo el riesgo del approve infinito y cómo `permit` lo mitiga.
- Eliges de forma razonada entre ERC-20, ERC-721, ERC-1155 y ERC-4626 para un caso dado.

🧭 Navegación

⬅️ Módulo 07 · Aplicaciones descentralizadas · 📚 Índice del currículo · ➡️ Módulo 09 · Seguridad y auditoría

09 · Seguridad y auditoría

> **Nivel:** Avanzado · ⏱️ **Duración estimada:** 180 min · **Fuente:** Trail of Bits *Building Secure Contracts* y ConsenSys *Smart Contract Best Practices* ⬅️ Currículo · 📚 Bibliografía 🧭 ⬅️ **Anterior:** 08 · Tokens y estándares · 📚 Índice · ➡️ **Siguiente:** 10 · Oráculos, almacenamiento e indexación 📖 Glosario de términos · 🌱 ¿Nuevo en esto? Empieza aquí

🎯 Objetivos

- Reconocer las clases de vulnerabilidad más frecuentes en contratos y sus señales típicas en el código.
- Aplicar un proceso de auditoría reproducible de siete pasos, desde el alcance hasta la verificación de la corrección.
- Combinar revisión manual con pruebas unitarias, fuzzing, pruebas de invariantes y análisis estático.
- Reproducir un hallazgo con una prueba mínima y clasificarlo por impacto y probabilidad.
- Practicar la seguridad respetando límites éticos y legales, sin atacar sistemas ajenos.

📚 Resultados de aprendizaje

Al finalizar, el estudiante podrá:

1. **Identificar** reentrancia, fallos de control de acceso, lógica económica y manipulación de oráculo en código real.
2. **Ejecutar** un ciclo de auditoría con alcance, actores, activos e invariantes documentados.
3. **Construir** pruebas de fuzzing e invariantes que expongan una vulnerabilidad concreta.
4. **Reproducir** un exploit con una prueba mínima y explicar la corrección que lo neutraliza.
5. **Clasificar** hallazgos por impacto y probabilidad para priorizar su remediación.
6. **Aplicar** herramientas de análisis estático como Slither dentro del flujo de revisión.

🗺️ Temas

#	Tema	Por qué importa
1	Reentrancia	Permite reentrar antes de actualizar estado y drenar fondos; patrón clásico y aún vigente.
2	Control de acceso	Un `onlyOwner` faltante o mal aplicado entrega el contrato a cualquiera.
3	Lógica económica	Errores de precisión, redondeo y flujos de valor generan pérdidas silenciosas.
4	Front-running y MEV	El orden de las transacciones es manipulable y afecta a subastas y liquidaciones.
5	Manipulación de oráculo	Un precio spot barato de mover habilita ataques, a veces vía flash loan.
6	`delegatecall` y upgrades	Ejecutar código externo sobre el propio almacenamiento puede corromper el estado.
7	DoS y agotamiento de gas	Bucles no acotados o dependencias externas pueden bloquear funciones críticas.
8	Firmas y replay	Firmas sin `nonce` o sin `chainId` se reutilizan en otra cadena o transacción.

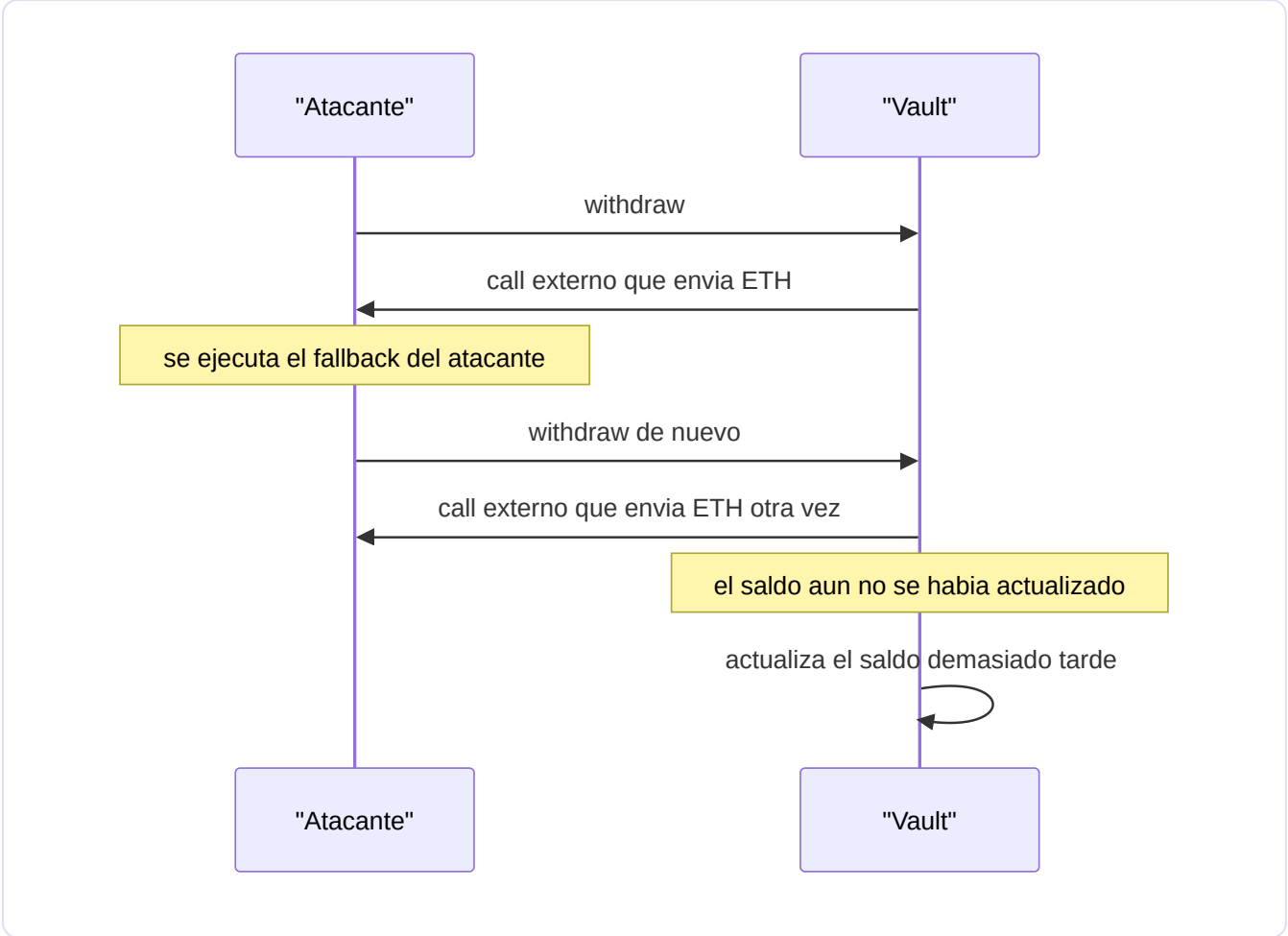
🧠 Modelo mental

Auditar un contrato se parece a inspeccionar un edificio antes de abrirlo al público: no basta con que las luces enciendan. Recorres cada puerta preguntando quién puede abrirla, qué pasa si alguien fuerza el orden de entrada y salida, y qué ocurre si un proveedor externo, como el ascensor, deja de responder. El código "funciona" en la ruta feliz; el auditor busca las rutas que nadie quiso recorrer.

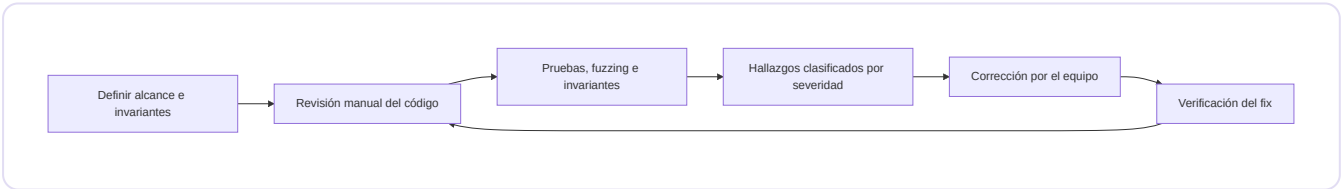
El límite de la analogía: un edificio es estático y su inspector confía en normas maduras, mientras que un contrato opera en un entorno adversarial y componible donde otros contratos pueden llamarlo en cualquier orden y un atacante puede pedir prestado capital enorme por unos segundos. Por eso la seguridad no termina en una lista de comprobación: exige razonar sobre invariantes que deben cumplirse pase lo que pase.

🧩 Esquema visual

La reentrancia clásica explota que el contrato envía valor antes de actualizar su estado: el fallback del atacante reingresa mientras el saldo sigue intacto.



Una auditoría profesional es un ciclo con verificación final, no una lectura única del código.



📖 Conceptos y definiciones

- **Invariante**: propiedad que siempre debe ser cierta (por ejemplo, la suma de saldos igual al suministro); es la base de las pruebas de invariantes.
- **Reentrancia**: reingreso a una función antes de que actualice su estado; se mitiga con el patrón checks-effects-interactions y guardas.
- **Front-running / MEV**: extracción de valor reordenando, insertando o censurando transacciones en el mempool.
- **Manipulación de oráculo**: distorsión temporal de un precio de referencia para engañar la lógica del contrato.
- `delegatecall`: llamada que ejecuta código ajeno usando el almacenamiento del llamante; base de los proxies actualizables y de riesgos graves.

- **Fuzzing**: generación de entradas aleatorias para violar una propiedad; Echidna y el fuzzer de Foundry son herramientas habituales.
- **Análisis estático**: inspección del código sin ejecutarlo para detectar patrones peligrosos; Slither y Mythril son ejemplos.
- **Impacto y probabilidad**: dimensiones para clasificar un hallazgo y priorizar su corrección.

🔬 Profundización

Severidad: la matriz impacto × probabilidad

Las firmas de auditoría no reportan "bugs" sueltos: clasifican cada hallazgo cruzando cuánto daño causaría (fondos perdidos, protocolo congelado, dato incorrecto) con qué tan plausible es que ocurra (¿lo dispara cualquiera, o exige condiciones improbables y capital enorme?). Una matriz típica:

Impacto \ Probabilidad	**Alta**	**Media**	**Baja**
Alto (pérdida de fondos)	Crítica	Alta	Media
Medio (funcionalidad degradada)	Alta	Media	Baja
Bajo (molestia o gas extra)	Media	Baja	Informativa

Trail of Bits, OpenZeppelin y las plataformas de concursos como Code4rena o Sherlock usan variantes de este esquema; lo importante no es la etiqueta exacta sino que la clasificación sea argumentada y reproducible. Un hallazgo "crítico pero teórico" mal justificado destruye la credibilidad de un informe tanto como uno real que se pasó por alto.

Mini-casos históricos: la misma lección, distinta década

Caso	Año	Pérdida histórica	Causa raíz	Lección
The DAO	2016	~3.6 M ETH	Reentrancia: enviaba ETH antes de actualizar el saldo	Checks-effects-interactions no es opcional; el incidente motivó el fork que separó Ethereum de Ethereum Classic
Ronin Bridge	2022	~624 M USD	Claves de 5 de 9 validadores comprometidas vía ingeniería social	La criptografía perfecta no protege un quorum de confianza demasiado pequeño y mal custodiado
Wormhole	2022	~326 M USD	Verificación de firma defectuosa: aceptaba una función de validación obsoleta	Todo lo que "verifica" debe probarse con entradas hostiles, no solo con las válidas
Euler Finance	2023	~197 M USD	Una función de donación rompía el invariante de solvencia usado por la liquidación	Cada función nueva debe evaluarse contra los invariantes de todo el sistema; los fondos fueron devueltos tras negociación

Nótese el patrón: solo uno de los cuatro es "un bug de Solidity" clásico. Los otros tres son fallos de diseño, de custodia de claves o de interacción entre módulos correctos por separado.

Qué detecta cada técnica (y qué no)

Ninguna herramienta cubre todo el espectro; una auditoría seria las apila porque sus puntos ciegos son complementarios:

Clase de bug	Análisis estático (Slither)	Fuzzing (Foundry/Echidna)	Invariantes	Revisión manual
Reentrancia por patrón de código	✅ detecta el patrón	⚠️ solo si la prueba lo ejercita	✅ si el invariante de saldos está definido	✅
Control de acceso faltante	✅ parcial	✅ con llamadas desde cuentas aleatorias	✅	✅
Redondeo y precisión	❌	✅ excelente con entradas extremas	✅	⚠️ fácil de pasar por alto
Lógica económica multi-contrato	❌	⚠️ requiere entorno completo	✅ la técnica más fuerte	✅
Error de diseño del protocolo	❌	❌	❌	✅ única técnica que lo ve

La consecuencia práctica: un `slither .` limpio y un fuzzing en verde acotan clases enteras de errores, pero el caso Euler demuestra que el fallo puede vivir en la interacción entre funciones individualmente correctas, territorio exclusivo de los invariantes bien elegidos y de la revisión humana.

Anatomía de un ataque de préstamo relámpago

Los retos de este módulo se entienden mejor viendo cómo se combinan. Un *flash loan* no es una vulnerabilidad: es una herramienta que **elimina el capital como barrera de entrada**. Quien no tiene un millón puede operar como si lo tuviera, siempre que lo devuelva en la misma transacción.

Eso convierte ataques que eran teóricos en ataques que cualquiera puede ejecutar.

El ataque completo, en una sola transacción atómica:

```
1. Pedir prestados 10 000 000 USDC          ← sin colateral: se devuelve al final o revierte todo
2. Comprar TOKEN en un pool pequeño         ← el precio spot de ese pool se dispara
3. Depositar TOKEN como garantía en el      ← el protocolo lee el precio del pool manipulado
   protocolo víctima, que lo valora caro       y concede un préstamo desproporcionado
4. Retirar el préstamo en USDC
5. Deshacer la compra: el precio vuelve
6. Devolver los 10 000 000 + comisión
7. Quedarse la diferencia
```

Lo que hay que ver: el atacante no rompió ninguna criptografía ni encontró un desbordamiento. **Usó cada contrato exactamente como estaba escrito.** El fallo está en un supuesto: que el precio de un pool refleja el valor de mercado. Es cierto en condiciones normales y falso durante un bloque con liquidez prestada.

Por qué la atomicidad lo hace posible: si algo sale mal en cualquier paso, toda la transacción revierte y el préstamo nunca existió. El atacante no arriesga capital, solo gas. Un intento fallido cuesta unos céntimos.

Las tres defensas, por orden de eficacia:

Defensa	Qué logra	Límite
TWAP en lugar de spot	Manipular la media exige sostener el precio muchos bloques, lo que multiplica el coste	Reacciona con retraso: en un movimiento real de mercado, va por detrás
Agregar varias fuentes	Un solo pool manipulado no mueve la mediana	Añade dependencia de más proveedores, cada uno con su riesgo
Circuit breaker	Detiene la operación si el precio se sale de rango	Alguien tiene que definir "rango razonable", y ese alguien es un punto de confianza

> 💡 **En una frase:** los ataques caros de verdad no rompen el código, rompen un supuesto. Escribe los supuestos de tu contrato antes de escribir el contrato.

▶ 🎓 **Si ya dominas esto** — método de auditoría, más allá del catálogo de bugs

🧪 Laboratorio guiado

1. Abre los retos del repositorio en `security-challenges` y compila los contratos para partir de una base limpia.

```
forge build
```

2. Ejecuta las pruebas con trazas para observar el flujo de un exploit reproducido paso a paso.

```
forge test -vv
```

3. Pasa el análisis estático sobre los contratos y contrasta cada alerta de `slither` con una revisión manual del flujo señalado.

4. Consulta el catálogo de laboratorios para elegir el reto adecuado a la vulnerabilidad que estudias.

📝 Reto verificable

Toma un contrato vulnerable de `security-challenges`, escribe una prueba mínima que reproduzca el exploit, corrige la causa raíz y documenta el hallazgo con su clasificación de impacto y probabilidad.

Criterio de aceptación: existe una prueba que falla contra el contrato vulnerable y pasa contra el corregido; el informe describe alcance, activos, invariante violado y la corrección; y `forge test -vv` queda en verde tras el arreglo sin desactivar la prueba del exploit.

⚠️ Errores frecuentes

Síntoma	Causa y cómo comprobarlo
Fondos drenados en una sola transacción	Reentrancia; busca llamadas externas antes de actualizar estado y añade una prueba de reingreso.
Cualquiera ejecuta una función privilegiada	Falta o mal uso de modificadores de acceso; prueba llamando desde una cuenta sin rol.
El precio usado es absurdo por un instante	Oráculo spot manipulable; valida antigüedad, rango y considera TWAP.
Un proxy queda inutilizable tras actualizar	Colisión de almacenamiento en `delegatecall`; compara los layouts antes y después.
Una firma vale en otra cadena	Falta de `chainId` / `nonce`; revisa el dominio EIP-712 y añade protección de replay.
El análisis estático "no encuentra nada"	Falsa sensación de seguridad; ninguna herramienta sustituye la revisión manual de invariantes.

🛡️ Seguridad y ética

- Nunca pruebes exploits contra sistemas en producción ni contra contratos de terceros sin autorización explícita.
- Trabaja solo en local o testnet, con contratos propios o de laboratorio, y sin fondos ni claves reales.
- Practica la divulgación responsable: si descubres una vulnerabilidad real, repórtala de forma privada al equipo afectado.
- Documenta cada hallazgo de forma reproducible para que la corrección pueda verificarse de manera independiente.
- Recuerda que las herramientas automáticas complementan, pero no reemplazan, el razonamiento sobre invariantes y privilegios.

🔗 Referencias

- Trail of Bits, *Building Secure Contracts* — https://secure-contracts.com/
- ConsenSys, *Smart Contract Best Practices* — https://consensysdiligence.github.io/smart-contract-best-practices/
- SWC Registry, *Smart Contract Weakness Classification* — https://swcregistry.io/
- *Damn Vulnerable DeFi* — https://www.damnvulnerabledefi.xyz/
- Fuente primaria: EIP-155, *Simple replay attack protection* — https://eips.ethereum.org/EIPS/eip-155

✅ Criterio de dominio

- Reproduces y corriges al menos una vulnerabilidad con una prueba que documenta el antes y el después.
- Redactas un informe con alcance, invariantes y clasificación de impacto y probabilidad.
- Justificas por qué la revisión manual sigue siendo imprescindible pese al análisis estático y el fuzzing.

🧭 Navegación

⬅️ Módulo 08 · Tokens y estándares · 📚 Índice del currículo · ➡️ Módulo 10 · Oráculos, almacenamiento e indexación

10 · Oráculos, almacenamiento e indexación

> **Nivel:** Avanzado · ⏱️ **Duración estimada:** 150 min · **Fuente:** documentación de Chainlink y de The Graph ⬅️ Currículo · 📚 Bibliografía 🧭 ⬅️ **Anterior:** 09 · Seguridad y auditoría · 📚 Índice · ➡️ **Siguiente:** 11 · DAO y gobernanza 📖 Glosario de términos · 🌱 ¿Nuevo en esto? Empieza aquí

🎯 Objetivos

- Explicar el "problema del oráculo" y por qué introducir datos externos añade un modelo de confianza.
- Diseñar un consumo de oráculo robusto que valide antigüedad, rango, decimales y ronda completa, con fallback y circuit breaker.
- Distinguir precio spot, TWAP y agregación de múltiples fuentes según su resistencia a manipulación.
- Usar eventos e indexadores para consultar historial sin confundirlos con estado verificable en cadena.
- Comprender el direccionamiento por contenido de IPFS (CID) y las estrategias de persistencia y disponibilidad.

📚 Resultados de aprendizaje

Al finalizar, el estudiante podrá:

1. **Explicar** por qué la blockchain no conoce datos del mundo real y qué garantiza un oráculo.
2. **Implementar** validaciones de antigüedad, rango y ronda al leer un feed de precios.
3. **Comparar** precio spot, TWAP y agregación frente a un ataque de manipulación por flash loan.
4. **Diseñar** un subgraph o indexador que exponga eventos históricos de un contrato.
5. **Justificar** cuándo un dato debe vivir como estado en cadena y cuándo basta con un evento indexado.
6. **Evaluar** una estrategia de pinning para asegurar la disponibilidad de contenido en IPFS.

🗺️ Temas

#	Tema	Por qué importa
1	El problema del oráculo	Todo dato externo importa su propio modelo de confianza a la cadena.
2	Validación de feeds	Antigüedad, rango, decimales y ronda completa evitan usar precios inválidos.
3	Spot vs. TWAP	El precio instantáneo es fácil de mover; el promedio temporal encarece el ataque.
4	Agregación de fuentes	Combinar proveedores reduce el impacto de una fuente comprometida.
5	Eventos e indexación	Permiten reconstruir historial off-chain con The Graph y subgraphs.
6	Estado vs. evento	Un evento no es verificable por otro contrato; el estado sí.
7	IPFS y CID	El direccionamiento por contenido garantiza integridad, no disponibilidad.
8	Aleatoriedad (VRF)	El azar verificable evita que un validador manipule sorteos y loterías.

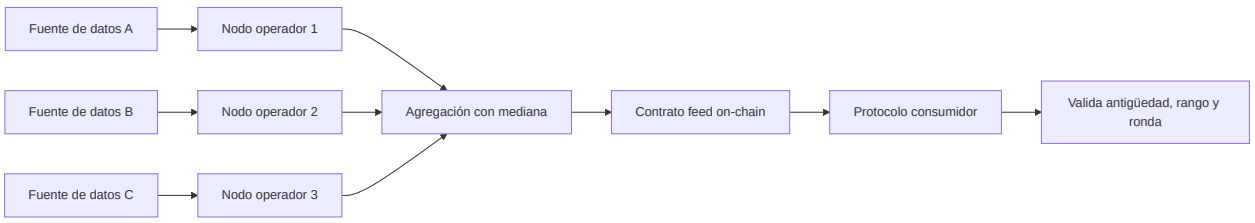
🧠 Modelo mental

Un contrato inteligente es como una persona encerrada en una habitación sin ventanas: puede razonar con impecable lógica, pero solo conoce lo que alguien desliza por debajo de la puerta. El oráculo es ese mensajero. Por muy correcta que sea la lógica interna, si el papel que entra dice un precio falso, la decisión será igual de falsa. Por eso la pregunta clave nunca es "¿el contrato calcula bien?", sino "¿puedo confiar en quién y cómo entró el dato?".

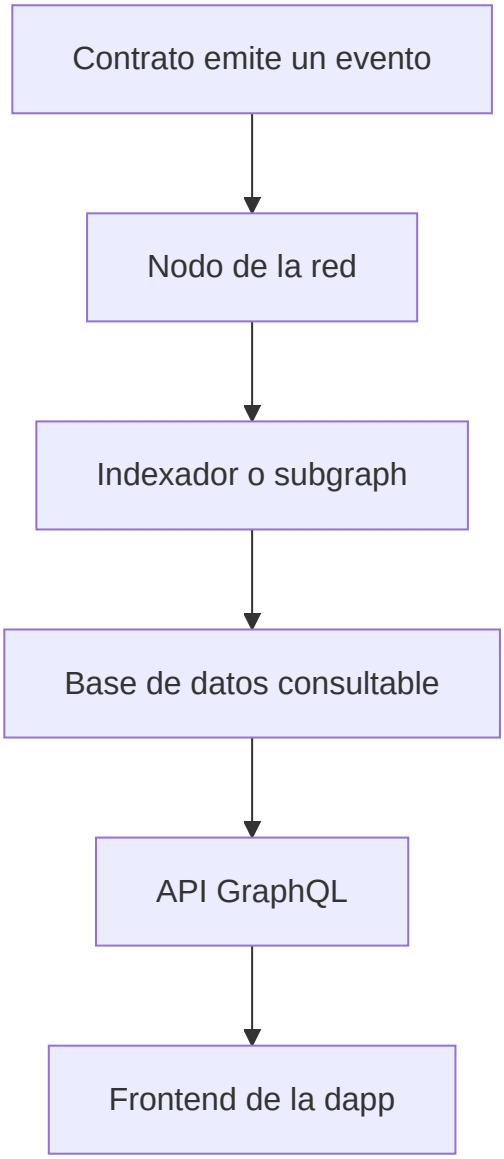
El límite de la analogía: un solo mensajero es un punto único de fallo, así que en la práctica se usan varios mensajeros y se descartan los que llegan tarde o con cifras fuera de rango. Además, distinguir estado de evento es clave: los eventos son como las notas que el mensajero deja apiladas fuera de la habitación (útiles para reconstruir la historia desde afuera), pero el ocupante no puede leerlas ni actuar sobre ellas; solo el estado en cadena está dentro de la habitación.

🧩 Esquema visual

En un oráculo *push*, varios nodos independientes leen múltiples fuentes, agregan el valor y lo publican en un contrato feed que el protocolo consume con sus propias validaciones.



La indexación recorre el camino inverso: convierte eventos emitidos en cadena en datos consultables fuera de ella.



📖 Conceptos y definiciones

- **Oráculo**: mecanismo que introduce datos externos en la cadena; su seguridad depende del modelo de confianza que impone.

- **Staleness (antigüedad)**: tiempo desde la última actualización de un feed; un dato viejo puede ser tan peligroso como uno falso.
- **TWAP**: precio medio ponderado en el tiempo, como el de Uniswap v3, mucho más costoso de manipular que el spot.
- **Agregación**: combinación de varias fuentes en un solo valor para reducir la dependencia de un proveedor único.
- **Circuit breaker**: mecanismo que detiene operaciones cuando un dato sale de límites razonables.
- **Evento**: registro emitido por un contrato para consumo off-chain; no puede leerse desde otro contrato.
- **Subgraph**: definición de indexación de The Graph que transforma eventos en datos consultables por GraphQL.
- **CID**: identificador de contenido de IPFS derivado del hash; garantiza integridad pero no que alguien conserve el archivo.
- **Pinning**: acción de mantener un contenido disponible en IPFS; alternativas de persistencia incluyen Arweave y Filecoin.
- **VRF**: función aleatoria verificable que aporta azar con prueba criptográfica de imparcialidad.

🔬 Profundización

Parámetros reales de un feed: heartbeat y deviation threshold

Un feed push de Chainlink no publica un precio nuevo en cada bloque: se actualiza cuando se cumple cualquiera de dos condiciones. El *deviation threshold* dispara una actualización si el precio observado off-chain se desvía del último publicado más de un porcentaje dado; el *heartbeat* fuerza una actualización si pasó demasiado tiempo desde la anterior, aunque el precio no se haya movido. Un feed mayor como ETH/USD en Ethereum mainnet opera típicamente con una desviación de ±0.5 % y un heartbeat de 3600 s, mientras que feeds de activos menos líquidos usan umbrales más laxos — verifica siempre los parámetros del feed concreto en vivo, porque cambian por activo y por red.

La consecuencia para el consumidor es directa: tu validación de antigüedad debe tolerar al menos el heartbeat (un dato de 50 minutos puede ser normal en un feed de 3600 s), y tu lógica debe asumir que el precio on-chain puede diferir del de mercado hasta el umbral de desviación. Un contrato que trate esa banda como error se detendrá en operación normal; uno que la ignore por completo subestima su margen de error económico.

Manipulación de precio spot con flash loan: los números

Supón un pool AMM de producto constante con 100 ETH y 200 000 USDC (`k = 20 000 000`), es decir, un precio spot de 2000 USDC/ETH. Un atacante pide un flash loan de 200 000 USDC y los mete al pool: las reservas pasan a 400 000 USDC y 50 ETH, y el precio spot instantáneo salta a 8000 USDC/ETH — se cuadruplicó dentro de una sola transacción.

Si un protocolo de préstamos lee ese spot como oráculo, el atacante deposita ETH "valorado" a 8000, pide prestado contra ese colateral inflado, deshace el swap, devuelve el flash loan y se queda con la diferencia.

Un TWAP encarece esto radicalmente: si la ventana es de 1800 s y un bloque dura ~12 s, un pico de un solo bloque pesa apenas `12 / 1800 ≈ 0.7 %` del promedio, así que sostener un precio falso exige mantener el capital en riesgo durante muchos bloques, expuesto al arbitraje. El caso ilustrativo es Mango Markets (octubre de 2022, ~114 M USD): el atacante infló el precio del token MNGO —de baja liquidez— en los mercados que alimentaban el oráculo y usó su posición revalorizada como colateral para vaciar la plataforma. La lección no es "los oráculos fallan", sino que el coste de manipular la fuente debe superar siempre al botín alcanzable con ella.

Oráculos push vs. pull

El modelo push (Chainlink Data Feeds) publica proactivamente en cadena y todos los consumidores leen el mismo valor; el modelo pull u on-demand (Pyth) mantiene los precios firmados off-chain y es el usuario quien los sube en la misma transacción que los consume.

Dimensión	Push (Chainlink feeds)	Pull (Pyth on-demand)
Quién paga el gas de actualizar	Los operadores del feed, de forma continua	El consumidor, solo cuando necesita el dato
Frescura	Limitada por heartbeat y deviation	Precio de hace segundos, firmado off-chain
Coste para el protocolo	Lectura barata de un valor ya publicado	Verificación de la firma y publicación en cada uso
Riesgo característico	Dato añejo dentro de la banda permitida	El consumidor puede elegir qué actualización sube; hay que validar el timestamp
Encaja mejor en	Préstamos y colateral en L1	Perps y trading de alta frecuencia en L2

Ninguno domina: el push amortiza el coste entre todos los usuarios y simplifica el consumo; el pull ofrece latencia mínima a cambio de trasladar validaciones al integrador.

Los cuatro modos de fallo de un oráculo, con su defensa

"Usa un oráculo fiable" no es un diseño. Un oráculo puede fallar de cuatro maneras distintas y **cada una necesita una defensa diferente**; confundirlas es cómo se construye un sistema que parece protegido y no lo está.

Modo de fallo	Qué ocurre	Lo que NO lo detecta	Lo que sí
Se detiene	El feed deja de actualizarse y devuelve el último precio, correcto en su día	Validar el valor: el número es plausible	Comprobar el **timestamp**: rechazar si supera `maxAge`
Se manipula	Alguien mueve el precio spot durante un bloque	Validar la antigüedad: el dato es de hace un segundo	**TWAP** o **agregación** de varias fuentes
Se equivoca	La fuente publica un valor real pero absurdo (un flash crash, un error del proveedor)	Ni antigüedad ni agregación si todas leen lo mismo	**Circuit breaker**: rango de cordura y pausa
Desaparece	El proveedor deja de operar el feed	Nada de lo anterior	**Fallback** a otra fuente y plan de migración

La comprobación mínima de una lectura, con las dos primeras defensas juntas:

```
(, int256 respuesta, , uint256 actualizadoEn, ) = feed.latestRoundData();

if (respuesta <= 0) revert PrecioInvalido();
if (block.timestamp - actualizadoEn > MAX_AGE) revert PrecioObsoleto();
// El feed tiene sus propios decimales: normalizar antes de comparar con nada.
uint256 precio = uint256(respuesta) * 1e18 / (10 ** feed.decimals());
```

Tres líneas que faltan en la mayoría de las integraciones improvisadas, y que cubren dos de los cuatro modos.

El matiz que decide el diseño: `MAX_AGE` no es un número universal. Depende del *heartbeat* del feed —cada cuánto se actualiza por contrato— y de la tolerancia de tu operación. Un feed que actualiza cada hora con un `MAX_AGE` de 10 minutos revierte constantemente; uno que actualiza cada minuto con un `MAX_AGE` de un día acepta datos inútiles. Ese número se saca de la documentación del feed, no de la intuición.

> 💡 **En una frase:** un oráculo puede darte un dato viejo, manipulado, erróneo o ninguno. Si tu contrato solo se defiende de uno, está protegido contra una cuarta parte del problema.

▶ 🎓 **Si ya dominas esto** — lo que decide la arquitectura de datos

🧪 Laboratorio guiado

> 🧪 Estas prácticas están catalogadas y **resueltas paso a paso** en el [catálogo de

> laboratorios.

1. Revisa el indexador de eventos del repositorio, ubicado en `apps/event-indexer`, y observa cómo transforma eventos en datos consultables.
2. Ejecuta su suite de pruebas para validar que la indexación reproduce el historial esperado.

```
pnpm test:indexer
```

3. En los datos indexados, identifica un evento y razona por qué otro contrato no podría depender de él como fuente de verdad.
4. Diseña sobre papel el consumo de un feed de precios con validación de antigüedad, rango, ronda y un fallback, y compáralo con un TWAP.

📝 Reto verificable

Especifica el consumo seguro de un oráculo de precios para un contrato hipotético: define validaciones de antigüedad, rango y ronda completa, una fuente de respaldo y un circuit breaker, y explica cómo un TWAP mitigaría un ataque de manipulación por flash loan.

Criterio de aceptación: el diseño enumera cada validación con su umbral y su comportamiento ante fallo; distingue explícitamente cuándo usar spot, TWAP o agregación; y justifica por qué el dato crítico se maneja como estado verificable y no como simple evento.

⚠️ Errores frecuentes

Síntoma	Causa y cómo comprobarlo
El contrato usa un precio congelado	No se valida la antigüedad; comprueba el timestamp de la última ronda antes de usarla.
Un ataque mueve el precio por un bloque	Uso de precio spot manipulable; sustitúyelo por un TWAP o por agregación de fuentes.
Los decimales dan resultados absurdos	Se ignoran los `decimals` del feed; normaliza siempre antes de operar.
Otro contrato "lee" un evento	Confusión entre evento y estado; los eventos solo se consumen off-chain.
El NFT pierde su imagen	El CID apunta a contenido sin pinning; asegura persistencia con un servicio o con Arweave/Filecoin.
El sorteo parece amañado	Aleatoriedad predecible en cadena; usa un VRF con prueba verificable.

🛡️ Seguridad y ética

- Trabaja en local o testnet; nunca uses fondos ni claves reales al integrar oráculos o indexadores.
- Trata todo dato externo como no confiable hasta validarlo: antigüedad, rango, decimales y ronda completa.
- Prefiere TWAP o agregación frente al spot en cualquier decisión con valor económico relevante.
- No expongas datos personales en metadata pública ni en contenido subido a IPFS, que es difícil de retirar.
- Documenta el modelo de confianza de cada fuente para que un tercero pueda auditarlo.

🔗 Referencias

- Chainlink, *Documentación* — https://docs.chain.link/
- Chainlink, *Whitepaper 2.0* — https://chain.link/whitepaper
- The Graph, *Documentación* — https://thegraph.com/docs/
- IPFS, *Documentación* — https://docs.ipfs.tech/
- Fuente primaria: Uniswap v3, *Oracle (TWAP)* — https://docs.uniswap.org/concepts/protocol/oracle

✅ Criterio de dominio

- Diseñas el consumo de un feed con todas sus validaciones y su comportamiento ante fallo.
- Explicas sin apoyo la diferencia entre spot, TWAP y agregación y cuándo usar cada uno.
- Justificas cuándo un dato debe ser estado en cadena y cuándo basta con un evento indexado.

🧭 Navegación

⬅️ Módulo 09 · Seguridad y auditoría · 📚 Índice del currículo · ➡️ Módulo 11 · DAO y gobernanza

11 · DAO y gobernanza

> **Nivel:** Avanzado · ⏱️ **Duración estimada:** 150 min · **Fuente:** OpenZeppelin Governor y Compound Governance ⬅️ Currículo · 📚 Bibliografía 🧭 ⬅️ **Anterior:** 10 · Oráculos, almacenamiento e indexación · 📚 Índice · ➡️ **Siguiente:** 12 · Escalabilidad y capas 2 📖 Glosario de términos · 🌱 ¿Nuevo en esto? Empieza aquí

🎯 Objetivos

- Describir el ciclo de vida de una propuesta: creación, votación, quorum, cola en timelock y ejecución.
- Explicar la delegación y los snapshots de poder de voto y por qué se mide en un bloque pasado.
- Comparar el voto on-chain con el off-chain (Snapshot) y sus implicaciones de coste y confianza.
- Diseñar una DAO donde una propuesta crítica exija votación, demora y ejecución verificable.
- Identificar amenazas de gobernanza, como el flash-loan governance o la captura por delegados, y sus mitigaciones.

📚 Resultados de aprendizaje

Al finalizar, el estudiante podrá:

1. **Explicar** cómo un snapshot de poder de voto impide comprar votos justo antes de la votación.
2. **Configurar** un Governor de OpenZeppelin con quorum, periodo de votación y `TimelockController`.
3. **Justificar** el uso de un timelock como demora entre la aprobación y la ejecución.
4. **Analizar** un ataque de flash-loan governance y las defensas que lo neutralizan.
5. **Diseñar** un mecanismo de emergencia acotado, transparente y con límites explícitos.
6. **Distinguir** el voto on-chain del off-chain según coste, participación y verificabilidad.

🗺️ Temas

#	Tema	Por qué importa
1	Propuestas y ciclo de vida	Estructuran cómo la comunidad decide y ejecuta cambios.
2	Delegación de voto	Permite a titulares delegar su poder sin transferir tokens.
3	Snapshots de poder	Fijan el poder en un bloque pasado para evitar compra de última hora.
4	Quorum y participación	Sin participación mínima, una minoría decide por todos.
5	Timelock y ejecución	La demora da tiempo a reaccionar ante una propuesta maliciosa.
6	On-chain vs. off-chain	Snapshot abarata el voto pero traslada la confianza fuera de la cadena.
7	Tesorería multisig	Un Safe con múltiples firmantes protege los fondos de la DAO.
8	Amenazas de gobernanza	Flash loans, captura y baja participación erosionan la legitimidad.

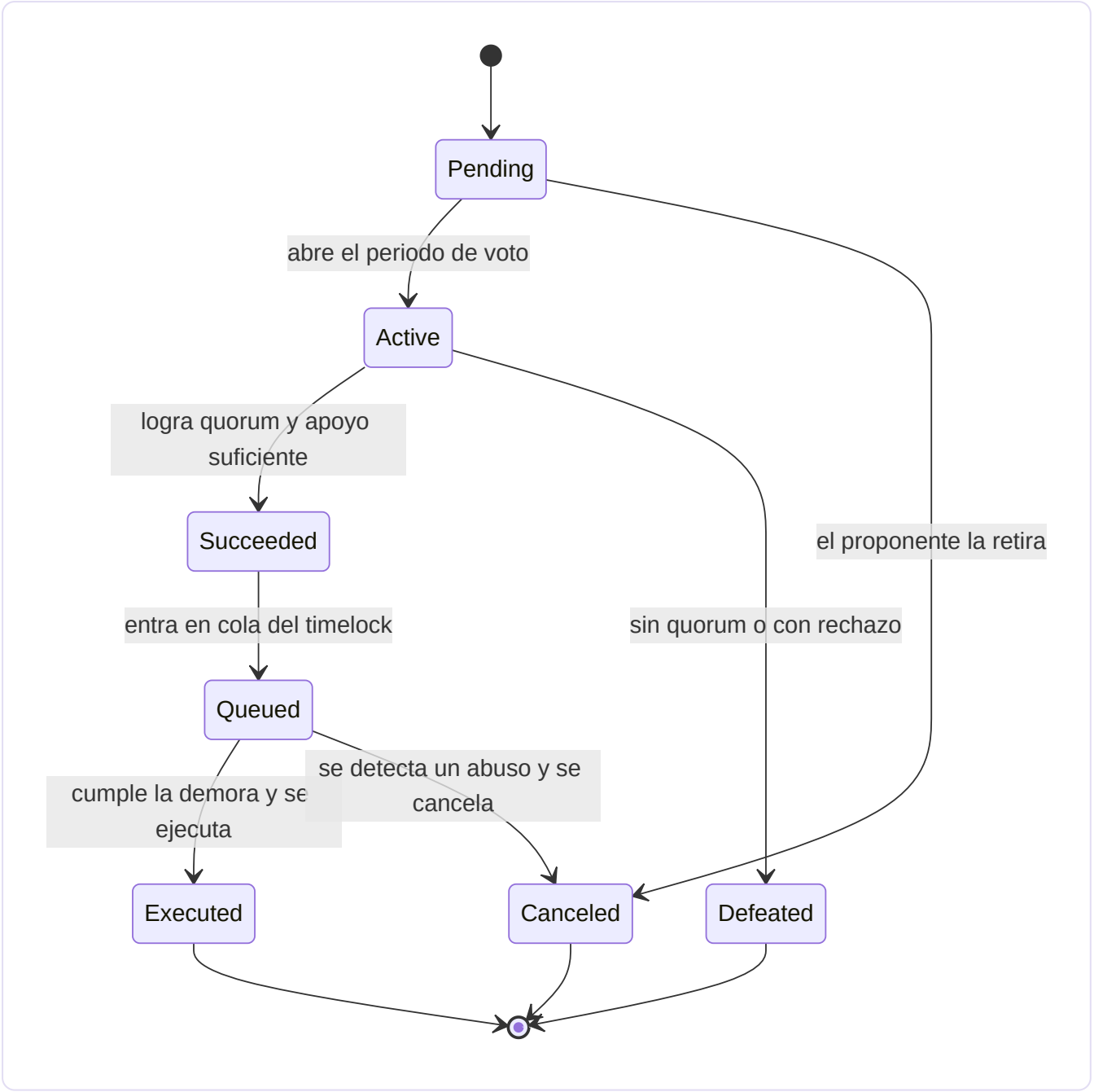
🧠 Modelo mental

Una DAO bien diseñada se parece a un parlamento con reglas de procedimiento: no basta con que una mayoría levante la mano una vez. Hay un registro de quién tiene derecho a voto en cierto momento (el snapshot), un debate con quorum, y un periodo obligatorio entre la aprobación y la entrada en vigor de la ley. Ese retraso no es burocracia inútil: es la ventana en la que la comunidad puede leer la letra pequeña y, si detecta un abuso, retirar sus fondos o reaccionar antes de que el cambio se aplique.

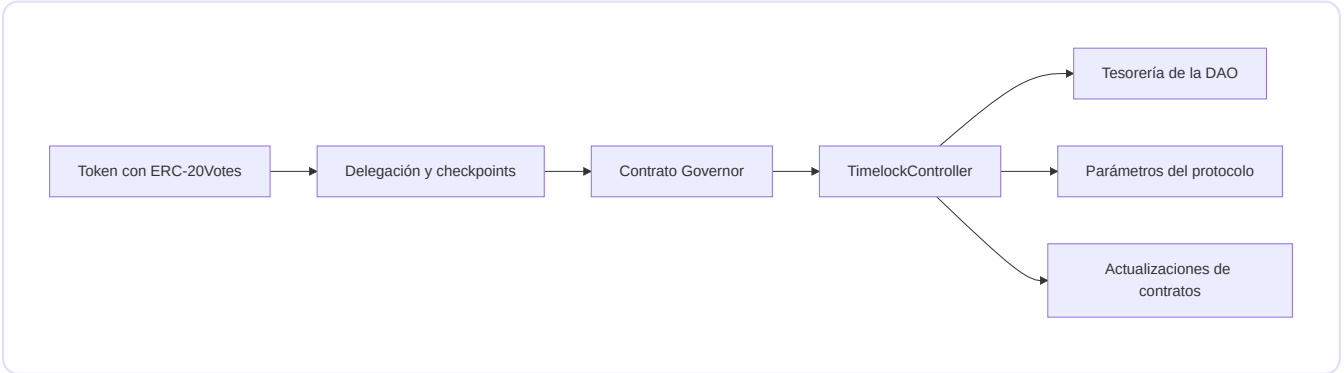
El límite de la analogía: en un parlamento tradicional el poder de voto no se puede alquilar por diez segundos, pero en cadena sí. Un atacante puede pedir prestado un capital enorme vía flash loan, votar y devolverlo en la misma transacción. Por eso el snapshot de poder en un bloque anterior y el timelock no son adornos, sino defensas centrales: separan el momento en que se mide el poder del momento en que se vota, y el momento en que se decide del momento en que se ejecuta.

🧩 Esquema visual

El ciclo de vida de una propuesta en un Governor de OpenZeppelin impone etapas obligatorias entre la creación y la ejecución.



La arquitectura separa quién mide el poder de voto, quién decide y quién ejecuta: solo el timelock es dueño de los contratos administrados.



📖 Conceptos y definiciones

- **Propuesta**: conjunto de acciones que la DAO somete a votación y, si se aprueba, ejecuta de forma verificable.
- **Delegación**: acto de asignar el poder de voto de los propios tokens a una dirección, sin transferir la propiedad.
- **Snapshot de poder**: registro del poder de voto en un bloque pasado, base de ERC-20Votes y los checkpoints de ERC-6372.
- **Quorum**: participación mínima requerida para que una votación sea válida.
- **Timelock**: contrato que impone una demora obligatoria entre la aprobación y la ejecución; en OpenZeppelin es `TimelockController`.
- **Voto off-chain**: señalización de voto fuera de la cadena, como en Snapshot, que ahorra gas pero no ejecuta por sí misma.
- **Tesorería multisig**: fondo gobernado por varias firmas, típicamente con un Safe, para evitar un único punto de control.
- **Flash-loan governance**: ataque que toma poder de voto prestado por instantes para forzar una decisión.
- **ERC-6372**: estándar de reloj que permite a un Governor usar bloques o marcas de tiempo para sus checkpoints.

🔬 Profundización

Anatomía de un ataque de gobernanza: Beanstalk (2022)

En abril de 2022, el protocolo Beanstalk perdió unos 182 M USD en el mayor ataque de gobernanza con préstamo relámpago registrado. El atacante había creado una propuesta maliciosa días antes (disfrazada, con ironía, de donación benéfica) y luego, en una sola transacción, pidió prestados cientos de millones de dólares en stablecoins vía flash loan, los depositó para obtener la supermayoría de poder de voto, ejecutó la propuesta mediante un mecanismo de *emergency commit* que no exigía demora, transfirió la tesorería a su propia dirección y devolvió el préstamo. Beneficio neto para el atacante: alrededor de 76 M USD.

El ataque funcionó porque fallaron a la vez las tres defensas canónicas:

- **Snapshot de voto en bloque pasado**: si el poder se mide con `getPastVotes` en un bloque anterior a la propuesta, los tokens prestados dentro de la misma transacción valen cero votos.
- **Timelock obligatorio**: una demora entre aprobación y ejecución impide que voto y ejecución convivan en una transacción, y da a la comunidad tiempo de auditar la propuesta y reaccionar.
- **Quorum y umbral de propuesta**: elevan el capital que hay que reunir y hacen que el ataque sea visible antes de consumarse.

Ninguna defensa aislada basta: el snapshot sin timelock deja pasar propuestas maliciosas votadas con poder legítimo comprado barato, y el timelock sin snapshot solo retrasa el saqueo.

Parámetros de diseño de un Governor

Parámetro	Qué protege	Trade-off al subirlo
Voting delay (entre propuesta e inicio de votación)	Da tiempo a delegar, informarse y detectar propuestas hostiles	Retrasa decisiones urgentes legítimas
Voting period (duración de la votación)	Participación real en distintas zonas horarias y contextos	Alarga todo el ciclo; más exposición a campañas de compra de votos
Proposal threshold (poder mínimo para proponer)	Filtra spam y propuestas triviales de atacantes sin capital	Concentra la iniciativa en ballenas y grandes delegados
Quorum (participación mínima)	Impide que una minoría diminuta decida por todos	Con apatía alta, ningún cambio legítimo alcanza el umbral
Timelock delay (demora antes de ejecutar)	Ventana de reacción y salida ante una propuesta aprobada maliciosa	Ralentiza correcciones de emergencia; exige un mecanismo de guardián acotado

No existen valores universales: un protocolo con tesorería enorme y comunidad madura tolera ciclos largos; uno joven que necesita iterar rápido suele empezar con parámetros bajos y endurecerlos a medida que crece el valor en juego.

veTokens y delegación líquida

El modelo *vote-escrowed* (veToken), popularizado por Curve con veCRV, ataca dos males crónicos: el capital mercenario que vota hoy y se va mañana, y la apatía del votante pequeño. El titular bloquea sus tokens por un plazo elegido (hasta 4 años en Curve) y recibe poder de voto proporcional al monto y al tiempo restante de bloqueo, que decae linealmente: quien más se compromete a largo plazo, más pesa. La delegación líquida (el patrón de ERC-20Votes) resuelve el otro flanco: permite ceder el poder de voto a delegados activos sin transferir la propiedad, elevando la participación efectiva.

Las críticas también son serias: el bloqueo prolongado ilíquido concentra el poder en quienes pueden permitirse inmovilizar capital años; alrededor de los veTokens surgieron mercados de sobornos de voto (*bribes*) y capas como Convex que re-concentran el poder que el diseño quería dispersar; y la delegación líquida tiende a oligarquías de delegados profesionales con baja rendición de cuentas. La lección de diseño: ningún mecanismo de tokenomics sustituye a una comunidad que vigila la concentración de poder — solo cambia dónde hay que mirar.

Un ataque de gobernanza, y por qué cada defensa existe

Las piezas de un Governor (snapshot, quorum, timelock) parecen burocracia hasta que se ve el ataque que cada una impide. Sigamos uno completo.

El objetivo: una DAO con una tesorería de 50 millones y un token de gobernanza que cotiza en mercado.

Sin ninguna defensa, el ataque cuesta una transacción:

```
1. Flash loan de 20 millones          ← sin colateral, se devuelve al final
2. Comprar tokens de gobernanza
3. Crear la propuesta "transferir la tesorería a esta dirección"
4. Votar a favor con los tokens recién comprados
5. Ejecutar
6. Vender los tokens, devolver el préstamo
```

Todo en un bloque. Y ahora, dónde se rompe la cadena según qué defensa esté puesta:

Defensa	En qué paso muerde	Qué impide exactamente
Snapshot en bloque pasado (`getPastVotes`)	4	Los tokens comprados en el paso 2 tienen **cero** poder de voto: el peso se leyó de un bloque anterior a la compra. Mata el ataque entero
Retraso de votación	3→4	Entre crear la propuesta y poder votar pasan bloques, así que ninguna operación atómica los abarca
Periodo de votación (días)	4	Sostener un préstamo relámpago durante días es imposible por definición
Quorum	4	Obliga a movilizar una fracción real del suministro, no solo mayoría de los que votaron
Timelock	5	Aunque todo lo anterior fallara, la ejecución se retrasa: da tiempo a ver la propuesta y salir

Lo importante: el snapshot solo por sí mismo ya cierra este ataque. Las demás no son redundancia inútil: cada una cubre un camino distinto (compra progresiva, colusión de delegados, propuesta maliciosa disfrazada). Un diseño con snapshot pero sin timelock es vulnerable a un ataque más lento y más difícil de detectar.

La consecuencia práctica que sorprende: con `ERC20Votes`, tener tokens **no da poder de voto**. Hay que delegar explícitamente, aunque sea a uno mismo. Es la causa número uno de participación baja en DAOs recién lanzadas, y no es un bug: es lo que permite que el snapshot funcione.

> 💡 **En una frase:** cada pieza de un Governor existe porque alguien ya ejecutó el ataque que impide. El snapshot es la que hace inviable comprar la votación en el último segundo.

▶ 🎓 **Si ya dominas esto** — donde la gobernanza se rompe de verdad

🧪 Laboratorio guiado

> 🧪 Estas prácticas están catalogadas y **resueltas paso a paso** en el catálogo de laboratorios.

1. Abre los contratos de gobernanza y timelock del módulo, ubicados en `labs/08-protocols`, y revisa cómo se enlaza el Governor con el `TimelockController`.
2. Ejecuta la suite de pruebas con trazas para seguir una propuesta desde su creación hasta su ejecución.

```
forge test -vv
```

3. En las trazas, localiza el snapshot de poder de voto y comprueba que corresponde a un bloque anterior al inicio de la votación.
4. Sigue el paso por el timelock e identifica la demora obligatoria entre la aprobación y la ejecución.

📝 Reto verificable

Diseña e implementa una DAO en la que una propuesta crítica requiera votación con quorum, una demora en `TimelockController` y una ejecución verificable, más un mecanismo de emergencia acotado y transparente.

Criterio de aceptación: `forge test -vv` pasa en verde; una prueba demuestra que una propuesta no puede ejecutarse antes de cumplir la demora del timelock; otra prueba demuestra que el poder de voto se mide por snapshot en un bloque pasado; y el mecanismo de emergencia tiene límites explícitos y deja rastro on-chain.

⚠️ Errores frecuentes

Síntoma	Causa y cómo comprobarlo
Alguien aprueba una propuesta con tokens prestados	Falta de snapshot en bloque pasado; verifica que el poder se lea con `getPastVotes`.
Una propuesta se ejecuta al instante	No hay timelock entre aprobación y ejecución; comprueba la demora en `TimelockController`.
Casi nadie vota y una minoría decide	Quorum demasiado bajo o apatía; revisa el umbral y los incentivos de participación.
El poder de voto no aparece	Titular no delegó, ni siquiera a sí mismo; recuerda que ERC-20Votes exige delegación explícita.
Un delegado concentra el control	Captura por delegación; monitoriza la distribución del poder delegado.
El "botón de emergencia" hace demasiado	Mecanismo sin límites; acótalo, hazlo transparente y auditable.

🛡️ Seguridad y ética

- Trabaja en local o testnet; nunca uses fondos ni claves reales al desplegar contratos de gobernanza.
- Mide el poder de voto por snapshot en un bloque pasado para neutralizar la compra o el préstamo temporal de poder.
- Impón un timelock entre aprobación y ejecución para dar a la comunidad tiempo de reaccionar.
- Diseña cualquier mecanismo de emergencia con alcance mínimo, transparencia total y rastro on-chain.
- Vigila la concentración de poder delegado y la baja participación, que erosionan la legitimidad de la DAO.

🔗 Referencias

- OpenZeppelin, *Governance* — https://docs.openzeppelin.com/contracts/governance
- Compound, *Governance* — https://docs.compound.finance/governance/
- Snapshot, *Documentación* — https://docs.snapshot.org/
- Voshmgir, *Token Economy* — https://tokeneconomy.co/
- Fuente primaria: OpenZeppelin Governor y Compound Governor Bravo (documentación enlazada arriba) — https://docs.openzeppelin.com/contracts/governance

✅ Criterio de dominio

- Entregas una DAO cuyas pruebas demuestran snapshot de poder y demora obligatoria en el timelock.
- Explicas sin apoyo cómo un flash loan podría atacar la gobernanza y qué lo impide.
- Justificas la elección entre voto on-chain y off-chain para un caso concreto.

🧭 Navegación

⬅️ Módulo 10 · Oráculos, almacenamiento e indexación · 📚 Índice del currículo · ➡️ Módulo 12 · Escalabilidad y capas 2

12 · Escalabilidad y capas 2

> **Nivel:** Avanzado · ⏱️ **Duración estimada:** 150 min · **Fuente:** *An Incomplete Guide to Rollups* (Buterin) y L2BEAT ⬅️ Currículo · 📚 Bibliografía 🧭 ⬅️ **Anterior:** 11 · DAO y gobernanza · 📚 Índice · ➡️ **Siguiente:** 13 · Interoperabilidad y ecosistemas 📖 Glosario de términos · 🌱 ¿Nuevo en esto? Empieza aquí

🎯 Objetivos

- Distinguir seis familias de escalado (canales, sidechains, optimistic rollups, ZK rollups, validiums y appchains) según dónde ejecutan y dónde publican datos.
- Comparar mecanismos de prueba de estado inválido: fraud proofs frente a validity/ZK proofs, con sus implicaciones de latencia y confianza.
- Evaluar la disponibilidad de datos (DA) de un diseño y explicar cómo EIP-4844 redujo su costo desde 2024.
- Analizar el riesgo de secuenciación, censura y retiro, incluyendo el challenge period y los mecanismos de escape.
- Argumentar por qué el TPS aislado no mide seguridad ni experiencia de usuario.

📚 Resultados de aprendizaje

Al finalizar, el estudiante podrá:

1. **Clasificar** una solución de capa 2 real según ejecución, publicación de datos y modelo de prueba.
2. **Comparar** optimistic y ZK rollups en tiempo de retiro, supuestos criptográficos y costo.
3. **Explicar** el papel de los blobs de EIP-4844 en el costo de DA y por qué el danksharding completo aún es roadmap.
4. **Identificar** los vectores de censura asociados a un secuenciador centralizado y sus mitigaciones.
5. **Evaluar** un puente L1↔L2 distinguiendo mensajes canónicos de puentes de liquidez de terceros.
6. **Justificar** por qué la seguridad de un rollup depende de la capa 1 y de la disponibilidad de datos, no solo del rendimiento.

🗺️ Temas

#	Tema	Por qué importa
1	Ejecución fuera de cadena vs. publicación en L1	Define cuánta seguridad se hereda de Ethereum
2	Optimistic rollups y fraud proofs	El challenge period (~7 días) determina el tiempo de retiro seguro
3	ZK rollups y validity proofs	La prueba de validez habilita finalidad rápida sin periodo de disputa
4	Validiums y DA fuera de cadena	Sacrifican garantías de datos por costo; cambian el modelo de confianza
5	Disponibilidad de datos y EIP-4844	Los blobs abarataron drásticamente publicar datos de rollup en 2024
6	Secuenciador y riesgo de censura	Quién ordena las transacciones condiciona neutralidad y liveness
7	Puentes, forced inclusion y escape hatch	Determinan si el usuario puede salir sin permiso del operador
8	Appchains y soberanía	Ceden componibilidad a cambio de control y espacio de bloque propio

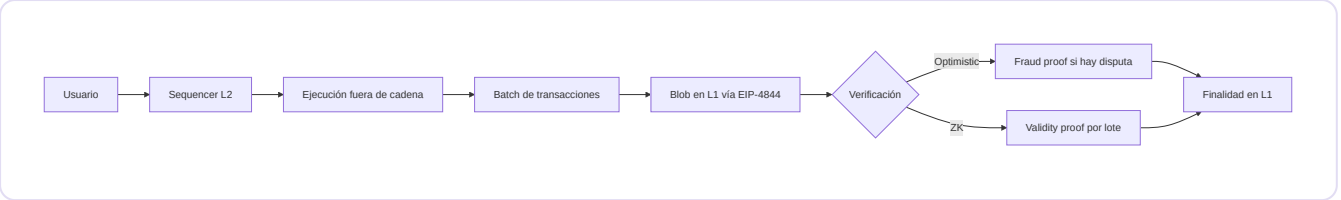
🧠 Modelo mental

Piensa en la capa 1 como un juzgado lento pero incorruptible y en cada capa 2 como una oficina que tramita miles de acuerdos y solo lleva al juzgado un resumen. Un optimistic rollup entrega ese resumen dando por buena la palabra del operador, salvo que alguien presente pruebas de fraude dentro del plazo de impugnación; un ZK rollup adjunta un certificado matemático que el juzgado verifica de inmediato. En ambos casos, los datos que respaldan el resumen deben quedar disponibles para que cualquiera pueda reconstruir el estado y ejercer su defensa.

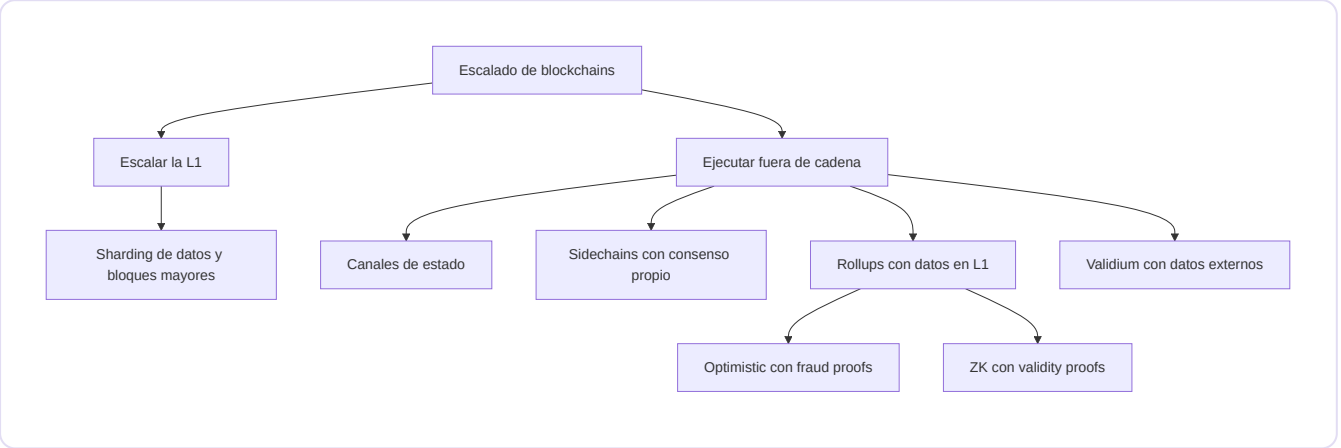
La analogía tiene límites: el "juzgado" no revisa cada caso, solo verifica pruebas o espera impugnaciones, y su garantía se evapora si los datos no están disponibles. Por eso la disponibilidad de datos es el eje del diseño, y por eso comparar cadenas solo por TPS es como juzgar un tribunal por cuántos papeles archiva sin mirar si sus sentencias son ejecutables.

🧩 Esquema visual

El ciclo de vida de una transacción en un rollup, desde que el usuario la envía hasta que alcanza finalidad en la capa 1:



Taxonomía de las estrategias de escalado según dónde ejecutan y dónde publican los datos:



📖 Conceptos y definiciones

- **Rollup**: esquema que ejecuta transacciones fuera de la L1 y publica en ella datos y compromisos de estado; hereda seguridad si los datos están disponibles.
- **Optimistic rollup**: asume que las transiciones son válidas y las revierte solo si un fraud proof lo demuestra dentro del challenge period (~7 días).
- **ZK rollup**: acompaña cada lote con una validity proof (SNARK/STARK) que la L1 verifica, habilitando finalidad sin periodo de disputa.
- **Validium**: como un ZK rollup pero con datos fuera de la L1; reduce costo a cambio de un supuesto adicional de disponibilidad de datos.
- **Disponibilidad de datos (DA)**: garantía de que los datos de cada lote son recuperables por cualquiera para reconstruir y auditar el estado.
- **EIP-4844 (blobs)**: mecanismo de Dencun (2024) que añadió espacio de datos efímero y barato para rollups; el danksharding completo lo amplía y sigue en roadmap.
- **Secuenciador (sequencer)**: componente que ordena y agrupa transacciones; si es único puede censurar o reordenar, salvo mecanismos de inclusión forzada.
- **Challenge period**: ventana durante la cual se pueden impugnar transiciones de un optimistic rollup; retrasa los retiros hacia la L1.
- **Forced inclusion / escape hatch**: rutas que permiten al usuario incluir transacciones o retirar fondos aun si el operador se niega a cooperar.
- **Appchain**: cadena dedicada a una aplicación; gana control y capacidad a cambio de componibilidad y, a veces, de seguridad compartida.

🔬 Profundización

El impacto medible de EIP-4844: blobs frente a calldata

Antes de Dencun (marzo de 2024), los rollups publicaban sus datos como *calldata* en transacciones normales de Ethereum, compitiendo por gas con todas las demás transacciones: cada byte distinto de cero costaba 16 gas y ese coste dominaba la factura de un rollup, llegando a representar más del 90% de sus gastos operativos. EIP-4844 introdujo las *blob-carrying transactions*: cada blob aporta ~128 KB de datos con un mercado de tarifas propio e independiente (fee market separado con su propio precio base), y los blobs se podan de los nodos tras ~18 días, porque solo necesitan estar disponibles durante la ventana de verificación, no para siempre.

El efecto fue inmediato y medible: en las semanas posteriores a Dencun, las comisiones de usuario en los principales L2 cayeron en más de un orden de magnitud (reducciones superiores a 10x fue el patrón general; en varios rollups una transacción pasó de decenas de centavos a fracciones de centavo). Un ejemplo numérico orientativo del mecanismo: si un batch de 100 000 bytes costaba en calldata unos 1 600 000 gas solo en datos, con blobs ese mismo volumen se paga en un mercado que, cuando hay poca demanda de blobs, tiende al precio mínimo (1 wei por gas de blob), es decir, prácticamente gratis en términos relativos. Las cifras actuales de tarifas por L2 son volátiles: consúltalo en vivo en L2BEAT y en Dune. El siguiente paso del roadmap, el danksharding completo, ampliará el número de blobs por bloque con *data availability sampling*; a 2025 sigue siendo trabajo en curso.

Las etapas de madurez de L2BEAT: Stage 0, 1 y 2

L2BEAT clasifica los rollups por cuánto dependen todavía de sus operadores, no por su rendimiento. La pregunta de fondo es: ¿puede el usuario salir con sus fondos aunque el equipo del rollup desaparezca o se vuelva hostil?

Etapa	Exigencia principal	Qué significa para el usuario
Stage 0	Publica datos en L1 y existe software para reconstruir el estado	La seguridad descansa casi por completo en el operador
Stage 1	Sistema de pruebas activo (fraud o validity), salidas sin el operador, y un consejo de seguridad con umbral alto solo para emergencias	El usuario puede salir por sí mismo salvo bug crítico
Stage 2	Pruebas totalmente permissionless, ventana de salida amplia ante upgrades y consejo limitado a errores demostrables en cadena	La confianza en el operador es residual

La mayoría de los rollups en producción aún no alcanza Stage 2 por razones prácticas: mantener un consejo de seguridad con poderes amplios es un seguro frente a bugs en

sistemas de prueba jóvenes, los fraud proofs permissionless son difíciles de blindar contra ataques de espameo y griefing, y renunciar al upgrade rápido implica que un fallo crítico no se puede parchear de inmediato. Es un compromiso deliberado entre inmadurez del software y minimización de confianza; el estado de cada rollup cambia con el tiempo, consúltalo en vivo en L2BEAT.

Fraud proof interactiva frente a validity proof

Los dos modelos de prueba responden a la misma pregunta —¿es válida esta transición de estado?— con filosofías opuestas. La *fraud proof* interactiva (el diseño de bisección usado por los optimistic rollups modernos) no verifica nada por defecto: solo si un retador afirma que el resultado es incorrecto, retador y operador juegan un protocolo de bisección sobre la traza de ejecución. Si la traza tiene, por ejemplo, 2^30 pasos (~mil millones de instrucciones), cada ronda divide el rango en disputa por la mitad: en unas 30 rondas las partes quedan en desacuerdo sobre una única instrucción, y la L1 solo ejecuta esa instrucción para decidir quién miente. El coste en cadena es minúsculo, pero el proceso exige que exista al menos un verificador honesto y vigilante, y justifica el challenge period de ~7 días.

La *validity proof* invierte la carga: el operador demuestra criptográficamente la validez de cada lote antes de que la L1 lo acepte, sin depender de vigilantes ni de plazos de disputa. El contrato verificador comprueba la prueba (SNARK o STARK) en un solo paso, con coste de verificación casi constante aunque el lote contenga miles de transacciones. El precio se paga fuera de la cadena: generar la prueba requiere hardware y tiempo significativos. En resumen: la fraud proof es barata mientras nadie ataque y lenta para salir; la validity proof es cara de producir y rápida para finalizar.

De dónde sale realmente el ahorro de una L2

"Las L2 son más baratas" es cierto, pero la razón que suele darse —"porque procesan fuera de la cadena"— es incompleta. El ahorro tiene dos fuentes de tamaño muy distinto, y saber cuál es cuál explica por qué las comisiones de L2 bajaron de golpe en marzo de 2024.

El coste de una transacción en un rollup son dos partidas:

```
coste total = ejecución en L2  +  parte proporcional de publicar el lote en L1
              (baratísima)        (la que domina la factura)
```

La ejecución en L2 es barata porque la hace un secuenciador con hardware normal, sin miles de nodos replicando. Pero eso es la parte pequeña. **Lo que de verdad se paga es el espacio en L1**, y ahí está la clave: el coste se reparte entre todas las transacciones del lote.

Transacciones en el lote	Coste de publicar por transacción
1	el lote entero
100	1/100
1 000	1/1 000

De ahí sale una propiedad contraintuitiva: **una L2 es más barata cuanto más se usa**. Con poca actividad, cada usuario carga con una porción mayor del coste fijo de publicar.

Y por eso EIP-4844 cambió tanto las cosas. Antes de Dencun (marzo de 2024), los rollups publicaban sus datos como `calldata`, compitiendo por el mismo espacio de bloque que todas las transacciones de L1. Los *blobs* crearon un espacio de datos separado, con su propio mercado de precios y **efímero** (se borra a las pocas semanas, que es tiempo de sobra para que cualquiera descargue y verifique). Resultado: el coste de publicar cayó de forma drástica y las comisiones de L2 se desacoplaron de la congestión de L1.

La pregunta que hay que saber responder: si los blobs se borran, ¿sigue siendo seguro? Sí, y por una razón concreta: la disponibilidad de datos solo necesita ser suficiente para que **cualquiera pueda descargarlos y reconstruir el estado o impugnar un fraude**. Pasada la ventana de disputa, guardar esos datos para siempre en todos los nodos ya no aporta seguridad, solo coste.

> 💡 **En una frase:** en un rollup no pagas por computar, pagas por publicar; por eso se abarata al llenarse y por eso los blobs cambiaron la ecuación entera.

▶ 🎓 **Si ya dominas esto** — lo que separa un rollup de algo que se llama rollup

🧪 Laboratorio guiado

Este módulo es un ejercicio comparativo de análisis, sin código de repositorio. Consulta el índice de prácticas del curso en laboratorios.

1. Elige tres soluciones de capa 2 reales de distinta familia (por ejemplo un optimistic rollup, un ZK rollup y un validium).
2. Para cada una, abre L2BEAT y anota su categoría, su modelo de datos y sus riesgos declarados; recuerda que las cifras cambian, consúltalo en vivo.

```
Dimensión            | L2 A        | L2 B        | L2 C
---------------------+-------------+-------------+-------------
Dónde ejecuta        | ...         | ...         | ...
Dónde publica datos  | ...         | ...         | ...
Prueba de estado     | fraude/ZK   | fraude/ZK   | fraude/ZK
Secuenciador         | ...         | ...         | ...
Tiempo de retiro     | ...         | ...         | ...
Escape hatch         | sí/no       | sí/no       | sí/no
```

3. Registra el tipo de prueba (fraud vs. validity) y el tiempo de retiro estimado de cada opción.
4. Verifica en L2BEAT si la DA es on-chain (blobs/calldata) o externa, e indica el supuesto de confianza que introduce.
5. Redacta una conclusión de un párrafo sobre cuál ofrece mejores garantías de salida sin permiso y por qué.

📝 Reto verificable

Entrega una tabla comparativa de las tres soluciones más un informe breve que responda, para cada una: dónde se ejecuta, dónde se publican los datos, cómo se prueba un estado inválido, quién secuencia y cuánto tarda un retiro seguro.

Criterio de aceptación: la tabla incluye las seis dimensiones del laboratorio para las tres soluciones, cada afirmación de datos en vivo se marca como "consúltalo en vivo" con su fuente (L2BEAT), y el informe justifica por qué el TPS no basta para evaluar seguridad.

⚠️ Errores frecuentes

Síntoma	Causa y cómo comprobarlo
Afirmar que un rollup es seguro "porque usa ZK"	Se ignora la DA; comprueba en L2BEAT si los datos están on-chain o en un comité externo
Comparar cadenas solo por TPS	Se confunde rendimiento con seguridad; contrasta modelo de prueba y de datos, no solo throughput
Suponer retiros instantáneos en optimistic rollups	Existe un challenge period (~7 días); revísalo en la documentación del proyecto
Tratar un puente de liquidez como el puente canónico	Son distintos; verifica si el retiro pasa por el contrato oficial de la L1
Dar por hecho el danksharding completo	Hoy solo hay blobs (EIP-4844); el sharding de datos completo sigue en roadmap
Ignorar la centralización del secuenciador	Un secuenciador único puede censurar; busca si existe inclusión forzada

🛡️ Seguridad y ética

- Trabaja siempre en local o testnet; no muevas fondos reales ni uses claves privadas reales durante el análisis.
- No firmes transacciones ni conectes wallets con activos a exploradores o dApps mientras investigas.
- Cita las cifras de rendimiento y coste como observaciones datadas ("consúltalo en vivo"); nunca las presentes como constantes.
- Distingue el marketing del proyecto de sus garantías verificables; apóyate en fuentes independientes como L2BEAT.
- Reconoce el conflicto de interés: quien opera un secuenciador puede extraer valor del orden de las transacciones.

🔗 Referencias

- Buterin, V., *An Incomplete Guide to Rollups* — https://vitalik.eth.limo/general/2021/01/05/rollup.html
- ethereum.org, documentación de escalado — https://ethereum.org/developers/docs/scaling/
- L2BEAT, riesgos y estado de las capas 2 (datos en vivo) — https://l2beat.com/
- Fuente primaria: EIP-4844 (Shard Blob Transactions) — https://eips.ethereum.org/EIPS/eip-4844

✅ Criterio de dominio

- Explicas, sin notas, cómo un optimistic y un ZK rollup prueban su estado y por qué difieren en tiempo de retiro.
- Argumentas el papel de la disponibilidad de datos y de EIP-4844 en la seguridad y el coste de un rollup.
- Detectas cuándo una comparación por TPS oculta diferencias reales de confianza.

🧭 Navegación

⬅️ Módulo 11 · DAO y gobernanza · 📚 Índice del currículo · ➡️ Módulo 13 · Interoperabilidad y ecosistemas

13 · Interoperabilidad y ecosistemas

> **Nivel:** Avanzado · ⏱️ **Duración estimada:** 150 min · **Fuente:** documentación de Cosmos IBC y de Polkadot (XCM) ⬅️ Currículo · 📚 Bibliografía 🧭 ⬅️ **Anterior:** 12 · Escalabilidad y capas 2 · 📚 Índice · ➡️ **Siguiente:** 14 · Privacidad y zero knowledge
> 📖 Glosario de términos · 🌱 ¿Nuevo en esto? Empieza aquí

🎯 Objetivos

- Distinguir los mecanismos de puente (lock-and-mint, burn-and-mint), los light clients, la mensajería y los atomic swaps (HTLC).
- Comparar los modelos de interoperabilidad de EVM, Solana, Cosmos/IBC, Polkadot/XCM y redes empresariales como Hyperledger Fabric.
- Construir el modelo de amenazas de un puente identificando cada confianza añadida.
- Explicar por qué los puentes concentran riesgo y han sido el mayor vector de pérdidas del sector.
- Situar enfoques de mensajería como CCIP y LayerZero dentro del espectro confianza/verificación.

📚 Resultados de aprendizaje

Al finalizar, el estudiante podrá:

1. **Clasificar** un puente según su mecanismo de custodia y su método de verificación de mensajes.
2. **Comparar** IBC (verificación por light client) con puentes basados en un conjunto externo de validadores.
3. **Enumerar** los vectores de ataque de un puente y asociarlos a controles concretos.
4. **Evaluar** cómo la diferencia de finalidad entre cadenas habilita ataques de reorganización o replay.
5. **Justificar** cuándo conviene mensajería genérica frente a puentes de activos dedicados.
6. **Modelar** la confianza añadida de una arquitectura cross-chain y proponer su reducción.

🗺️ Temas

#	Tema	Por qué importa
1	Lock-and-mint vs. burn-and-mint	Definen dónde queda la custodia y cómo se conserva el suministro
2	Light clients y verificación nativa	Sustituyen la confianza en terceros por verificación criptográfica
3	Message passing (IBC, XCM)	Permiten transferir datos, no solo activos, entre cadenas
4	Atomic swaps y HTLC	Intercambian valor sin custodio, con garantías de atomicidad
5	Diferencia de finalidad entre cadenas	Una cadena reorganizable puede invalidar mensajes ya aceptados
6	Validadores/guardianes de puente	Su compromiso es el mayor riesgo histórico del sector
7	Replay y verificación incompleta	Mensajes repetidos o mal validados permiten acuñar de más
8	Interoperabilidad empresarial (Fabric)	Muestra otro modelo: permisos, canales y confianza acotada

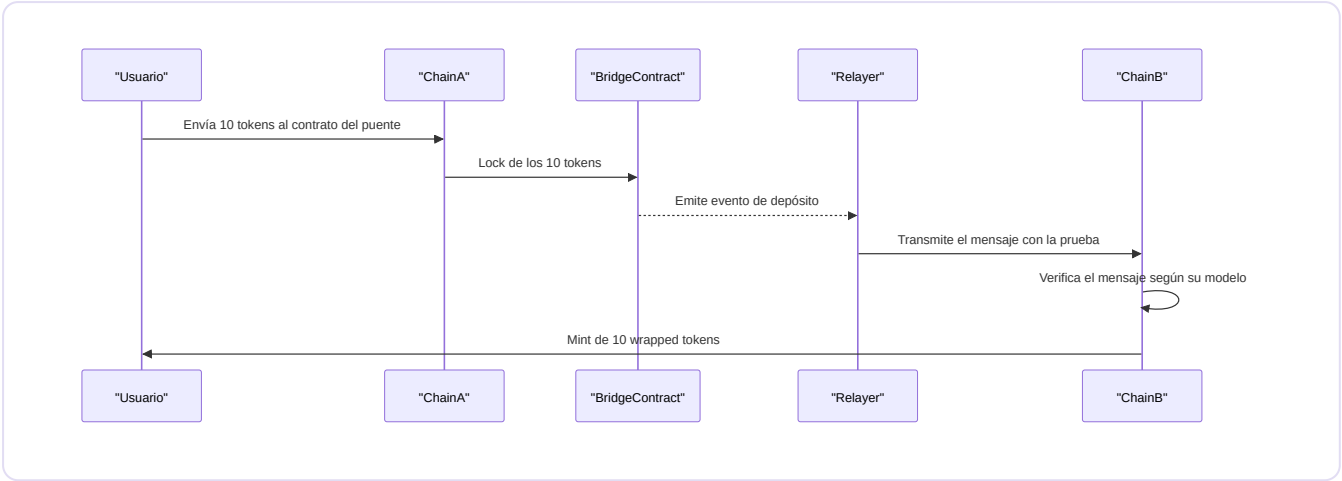
🧠 Modelo mental

Un puente es como una casa de cambio entre dos países con leyes distintas: deposita moneda en un país y recibe un pagaré canjeable en el otro. La seguridad no depende de la moneda, sino de quién custodia el depósito y de cómo se verifica que el pagaré es legítimo. Un light client es un notario que revisa por sí mismo las pruebas del país de origen; un conjunto de validadores externos es un grupo de firmantes en quien hay que confiar. Cuanto más se parezca el puente a "confiar en firmantes" y menos a "verificar pruebas", más superficie de ataque introduce.

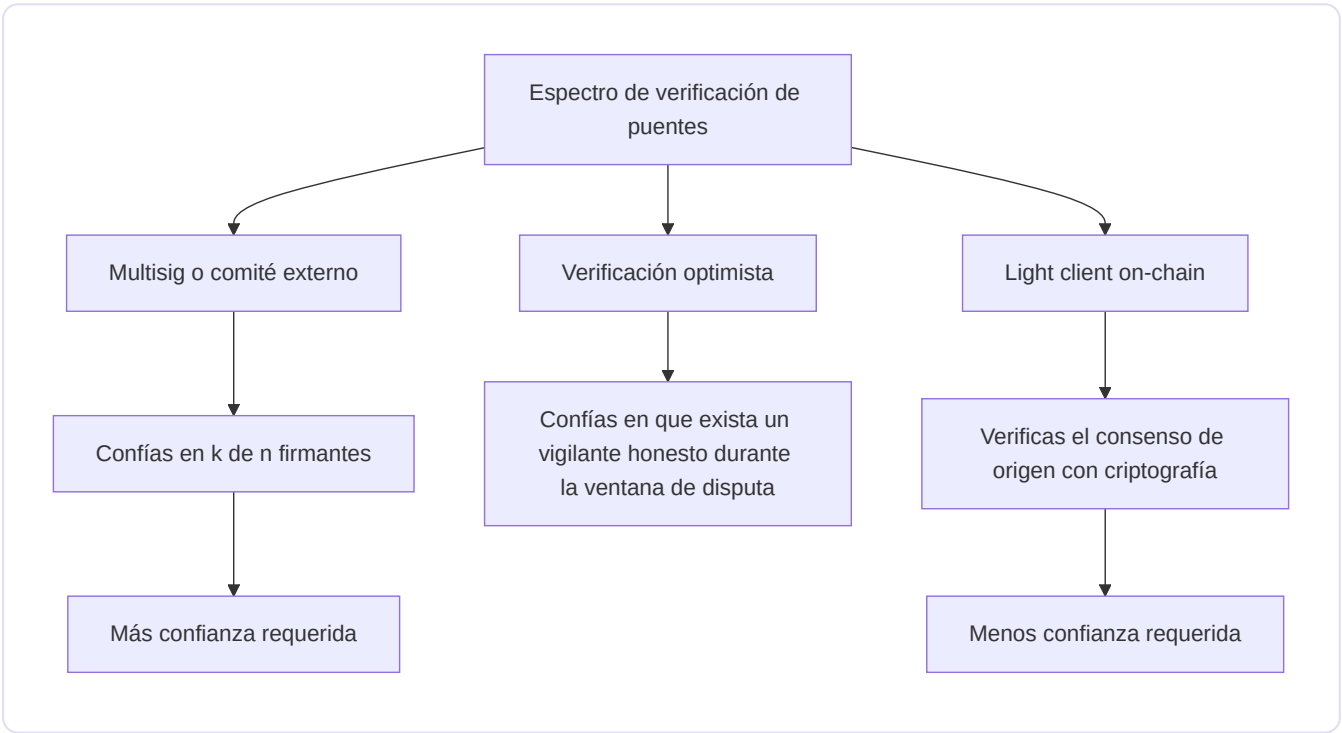
La analogía se rompe en dos puntos: los países no se reorganizan, pero una cadena con finalidad probabilística sí puede revertir bloques, invalidando un pagaré ya emitido; y un pagaré físico no se puede "reproducir", mientras que un mensaje mal protegido puede reejecutarse (replay) para acuñar de más. Por eso modelar interoperabilidad es, ante todo, enumerar cada confianza añadida y cada supuesto de finalidad.

🧩 Esquema visual

Flujo completo de un puente lock-and-mint: el activo queda bloqueado en origen y una representación se acuña en destino.



Espectro de verificación de los puentes, ordenado de mayor a menor confianza requerida en terceros:



📖 Conceptos y definiciones

- **Lock-and-mint**: bloquea el activo en la cadena de origen y acuña una representación en destino; el custodio del bloqueo es el punto crítico.
- **Burn-and-mint**: quema la representación en una cadena y acuña la equivalente en otra, conservando el suministro total sin custodia acumulada.
- **Light client**: verificador ligero que comprueba pruebas de consenso de otra cadena sin confiar en intermediarios; base de IBC.
- **Message passing**: transporte verificable de datos arbitrarios entre cadenas; IBC en Cosmos y XCM en Polkadot son ejemplos.
- **Atomic swap (HTLC)**: intercambio condicionado por un secreto y un tiempo límite que garantiza que ambas partes cumplen o ninguna lo hace.

- **Replay**: reejecución de un mensaje válido para obtener un efecto repetido, como acuñar dos veces; se evita con nonces y pruebas de consumo.
- **Finalidad**: momento en que un bloque se considera irreversible; su diferencia entre cadenas condiciona cuándo es seguro actuar sobre un mensaje.
- **Guardianes/validadores de puente**: conjunto externo que atestigua mensajes; su compromiso permite falsificar acuñaciones.
- **CCIP**: protocolo de mensajería de Chainlink con verificación y una red de riesgo independiente para transferencias cross-chain.
- **LayerZero**: enfoque de mensajería que separa el envío del mensaje de su verificación mediante componentes configurables.

🔬 Profundización

Los grandes hacks de puentes: anatomía de un patrón común

Los tres mayores incidentes de puentes de 2022 suman más de mil millones de dólares y comparten diagnóstico: la verificación del mensaje se degradó, en la práctica, a confiar en muy pocas partes o en ninguna.

Incidente	Año	Pérdida aprox.	Fallo raíz	Lección
Ronin (Axie Infinity)	2022	~624 M USD	El atacante comprometió 5 de las 9 claves de validadores (4 de Sky Mavis más una delegada) y firmó retiros falsos	Un multisig pequeño y correlacionado es un punto único de fallo; el hack tardó días en detectarse
Wormhole	2022	~326 M USD	La verificación de firmas en Solana usaba una función deprecada que permitió falsificar la comprobación y acuñar 120 000 wETH sin respaldo	La verificación es tan fuerte como su implementación; una dependencia obsoleta anula todo el diseño
Nomad	2022	~190 M USD	Una actualización dejó la raíz de confianza inicializada en cero, de modo que cualquier mensaje se daba por probado; cientos de imitadores copiaron el exploit	Un valor por defecto inseguro convirtió la verificación en un "acepta todo"; los errores de configuración también son criptográficos

El patrón común: en los tres casos el sistema *decía* verificar mensajes, pero la verificación efectiva se había reducido a un puñado de claves (Ronin), a una comprobación falsificable (Wormhole) o a nada (Nomad). Al modelar un puente, la pregunta correcta no es "¿verifica?" sino "¿qué es lo mínimo que hay que comprometer para que acepte un mensaje falso?".

Verificación por light client: el sync committee de Ethereum

Un light client on-chain sustituye a los firmantes externos por la verificación directa del consenso de la cadena de origen. En Ethereum, el mecanismo práctico es el *sync committee* introducido en Altair: un comité de 512 validadores seleccionados aleatoriamente que rota cada ~27 horas y firma cada cabecera de bloque con firmas BLS agregables. Un contrato light client desplegado en la cadena destino guarda el comité vigente, verifica la firma agregada de cada nueva cabecera (basta con que firmen 2/3 del comité) y actualiza el comité siguiente a partir de la propia cabecera firmada. Con una cabecera verificada, cualquier hecho de la cadena de origen —un depósito, un evento— se demuestra con una prueba de Merkle contra su raíz de estado.

El coste es real: verificar firmas BLS y mantener el estado del light client en cadena consume mucho más gas que comprobar k firmas de un multisig, y por eso varios proyectos comprimen esa verificación dentro de una prueba ZK (los llamados *ZK light clients*). A cambio, el supuesto de confianza se reduce de "estos n firmantes del puente son honestos" a "el consenso de Ethereum es honesto", que es exactamente el supuesto que el usuario ya aceptaba al usar Ethereum. IBC en Cosmos aplica el mismo principio entre cadenas Tendermint, donde la finalidad instantánea hace los light clients especialmente baratos.

Clasificación de la mensajería cross-chain

Protocolo	Quién verifica el mensaje	Supuesto de confianza	Madurez y ámbito
IBC (Cosmos)	Light client on-chain de la cadena de origen	El consenso de origen es honesto; sin terceros añadidos	En producción desde 2021 entre decenas de cadenas Tendermint; expansión fuera de Cosmos en curso
CCIP (Chainlink)	Redes de oráculos descentralizadas más una Risk Management Network independiente	Honestidad de mayorías en dos redes separadas entre sí	En producción desde 2023, orientado a adopción institucional y transferencias de valor
LayerZero	Componentes configurables por la aplicación (DVNs en v2) que atestiguan el mensaje	Depende de la configuración elegida: desde un solo verificador hasta comités múltiples	En producción y ampliamente integrado; la seguridad efectiva varía por aplicación

La tabla deja una conclusión incómoda: no existe "el estándar" de interoperabilidad, sino un espectro donde cada diseño intercambia coste, generalidad y confianza. La verificación por light client es el patrón oro en minimización de confianza, pero su coste y su acoplamiento al consenso de origen explican por qué los modelos intermedios dominan el mercado. El estado de adopción de cada protocolo cambia rápido: consúltalo en vivo en sus documentaciones y en agregadores independientes.

Por qué los puentes concentran los mayores robos del sector

No es casualidad ni mala suerte: es estructural. Un puente reúne tres condiciones que, juntas, lo convierten en el objetivo más rentable del ecosistema.

1. **Acumula valor en un punto.** En el esquema *lock-and-mint*, todo lo que se ha movido a la otra cadena está bloqueado en un contrato. Ese contrato es un solo objetivo con el valor de miles de usuarios.
2. **Suele verificar con firmantes, no con criptografía de consenso.** Si el puente acepta un mensaje porque *k* de *n* guardianes lo firmaron, comprometer esas *k* claves basta para acuñar de la nada.
3. **Es código nuevo y complejo** en un sitio donde el resto está muy auditado. La lógica de mensajería cross-chain no tiene décadas de escrutinio detrás.

Los tres patrones que se repiten en los incidentes públicos:

Patrón	Qué falla	Cómo se ve en el código
Claves comprometidas	El atacante consigue las firmas necesarias del conjunto verificador	Umbral bajo, o firmantes que comparten infraestructura y por tanto no son independientes
Verificación defectuosa	El contrato acepta como válida una prueba que no lo es	La comprobación no valida todos los campos, o hay un valor por defecto que pasa el filtro
Falta de anti-replay	Un mensaje legítimo se reejecuta y acuña dos veces	No hay `nonce` consumido, o el registro de mensajes procesados se puede reiniciar

La pregunta que ordena el análisis de cualquier puente, y que reemplaza a "¿es seguro?":

> ¿Qué tendría que ocurrir para que se acuñe un token en destino sin que exista el bloqueo correspondiente en origen?

La respuesta es su modelo de seguridad, dicho en una frase:

- "*Que se rompa el consenso de la cadena de origen*" → verificación por **light client**. El puente hereda la seguridad de la cadena, que es lo máximo alcanzable.
- "*Que se coludan k de n firmantes*" → verificación **externa**. La seguridad es la de esas k claves, sin importar el TVL ni la marca.
- "*Que el administrador del proxy lo decida*" → hay una **clave de actualización** que puede reemplazar toda la lógica. Es el escenario que hay que auditar primero, y el que más veces se pasa por alto.

> 💡 **En una frase:** todo puente añade supuestos de confianza; el análisis honesto

> consiste en enumerarlos, no en preguntar si es seguro.

▶ 🎓 **Si ya dominas esto** — el detalle operativo

🧪 Laboratorio guiado

Este módulo es un ejercicio de modelado de amenazas, sin código de repositorio. Consulta el índice de prácticas del curso en laboratorios.

1. Elige un puente real o de referencia y describe su flujo: origen, custodia, atestación del mensaje y acuñación en destino.
2. Dibuja el diagrama de actores y confía cada paso a alguien; marca dónde aparece una confianza añadida.

```
Vector                       | ¿Aplica? | Control / mitigación
-----------------------------+----------+---------------------
Validadores comprometidos    | ...      | ...
Replay de mensajes           | ...      | ...
Verificación incompleta      | ...      | ...
Actualización administrativa | ...      | ...
Diferencia de finalidad      | ...      | ...
Liquidez insuficiente        | ...      | ...
```

3. Para cada vector, indica cómo se detectaría un ataque y quién asume la pérdida.
4. Contrasta el diseño con uno basado en light client (IBC) y señala qué confianza desaparece.
5. Redacta una recomendación de un párrafo sobre si usar mensajería genérica (CCIP/LayerZero) o un puente dedicado.

📝 Reto verificable

Entrega el modelo de amenazas de un puente concreto: diagrama de flujo, tabla de los seis vectores con su mitigación y una conclusión que compare el diseño con una alternativa verificada por light client.

Criterio de aceptación: la tabla cubre los seis vectores (validadores comprometidos, replay, verificación incompleta, actualización administrativa, diferencia de finalidad y liquidez insuficiente), cada uno con un control explícito, y la conclusión identifica al menos una confianza añadida que la alternativa elimina.

⚠️ Errores frecuentes

Síntoma	Causa y cómo comprobarlo
Suponer que todos los puentes son igual de seguros	Se ignora el método de verificación; comprueba si usa light client o firmantes externos
Olvidar la protección anti-replay	Falta de nonce o de prueba de consumo; revisa si el mensaje puede reejecutarse
Actuar sobre un mensaje antes de la finalidad	La cadena de origen se reorganiza; verifica el tipo de finalidad de cada extremo
Confiar en una clave administrativa "temporal"	El upgrade admin permite drenar fondos; audita quién controla el proxy
Confundir liquidez con seguridad	Un puente con fondos puede seguir siendo vulnerable; separa TVL de modelo de confianza
Asumir que "cross-chain" equivale a "sin confianza"	Todo puente añade supuestos; enumera cada uno explícitamente

🛡️ Seguridad y ética

- Realiza todo el análisis en local o testnet; no transfieras fondos reales ni utilices claves privadas reales.
- No conectes wallets con activos a puentes o dApps durante el estudio, ni firmes mensajes que no comprendas.
- Reconoce que los puentes han causado las mayores pérdidas del sector; comunica el riesgo con honestidad, sin minimizarlo.
- Documenta cada supuesto de confianza; ocultarlo es una omisión ética, no un detalle técnico.
- Respeta las licencias y avisos de las cadenas empresariales (por ejemplo Fabric) al modelar escenarios permisionados.

🔗 Referencias

- Cosmos, documentación de IBC (Inter-Blockchain Communication) — https://ibc.cosmos.network/
- Polkadot Wiki, XCM (Cross-Consensus Messaging) — https://wiki.polkadot.network/docs/learn-xcm
- Chainlink, documentación de CCIP — https://docs.chain.link/ccip
- Hyperledger Fabric, documentación oficial — https://hyperledger-fabric.readthedocs.io/
- Fuente primaria: Cosmos IBC, especificación del protocolo y su verificación por light client — https://ibc.cosmos.network/

✅ Criterio de dominio

- Reconstruyes el flujo de un puente y señalas cada confianza añadida sin ayuda.
- Explicas cómo la finalidad y el replay afectan la seguridad de un mensaje cross-chain.
- Justificas cuándo un diseño por light client reduce riesgo frente a un conjunto de validadores externos.

🧭 Navegación

⬅️ Módulo 12 · Escalabilidad y capas 2 · 📚 Índice del currículo · ➡️ Módulo 14 · Privacidad y zero knowledge

14 · Privacidad y zero knowledge

> **Nivel:** Avanzado · ⏱️ **Duración estimada:** 180 min · **Fuente:** *Proofs, Arguments, and Zero-Knowledge* (Thaler) y ZKProof Community Reference
> ⬅️ Currículo · 📚 Bibliografía 🧭 ⬅️ **Anterior:** 13 · Interoperabilidad y ecosistemas · 📚 Índice · ➡️ **Siguiente:** 15 · Arquitectura avanzada 📖 Glosario de términos · 🌱 ¿Nuevo en esto? Empieza aquí

🎯 Objetivos

- Definir con precisión compromiso, witness, circuito, prover y verifier dentro de un sistema de conocimiento cero.
- Diferenciar SNARK y STARK en tamaño de prueba, tiempo, setup, supuestos criptográficos y resistencia post-cuántica.
- Explicar qué garantiza y qué no garantiza un trusted setup, y qué aportan los setups universales o transparentes.
- Diseñar conceptualmente una prueba de "soy mayor de 18 años" que no revele la fecha de nacimiento.
- Reconocer que ZK protege el enunciado, pero no elimina automáticamente metadatos ni problemas del mundo real.

📚 Resultados de aprendizaje

Al finalizar, el estudiante podrá:

1. **Explicar** las propiedades de completitud, solidez y conocimiento cero en términos de prover y verifier.
2. **Comparar** SNARK y STARK y justificar cuál conviene según setup, tamaño y supuestos.
3. **Diseñar** el enunciado de un circuito de credencial de edad identificando entradas públicas y privadas.
4. **Identificar** quién certifica el dato inicial y qué información sigue expuesta como metadato.
5. **Distinguir** entre setup confiable, universal y transparente y sus implicaciones de confianza.
6. **Evaluar** cómo se revoca una credencial ZK sin comprometer la privacidad del titular.

🗺️ Temas

#	Tema	Por qué importa
1	Prover, verifier y las tres propiedades	Fijan qué significa "probar sin revelar"
2	Circuito, witness y entradas públicas/privadas	Definen qué se demuestra y qué queda oculto
3	Compromisos (commitments)	Permiten fijar un valor sin revelarlo hasta después
4	Trusted setup vs. universal vs. transparente	Determina cuánta confianza inicial exige el sistema
5	SNARK vs. STARK	Compensan tamaño, tiempo, supuestos y post-cuántico
6	Credenciales selectivas (prueba de edad)	Caso canónico de minimizar la información revelada
7	Metadatos y desanonimización	ZK no oculta patrones de red, tiempos ni montos por defecto
8	Aplicaciones: rollups ZK, privacidad, identidad	Muestran el alcance real y sus límites regulatorios

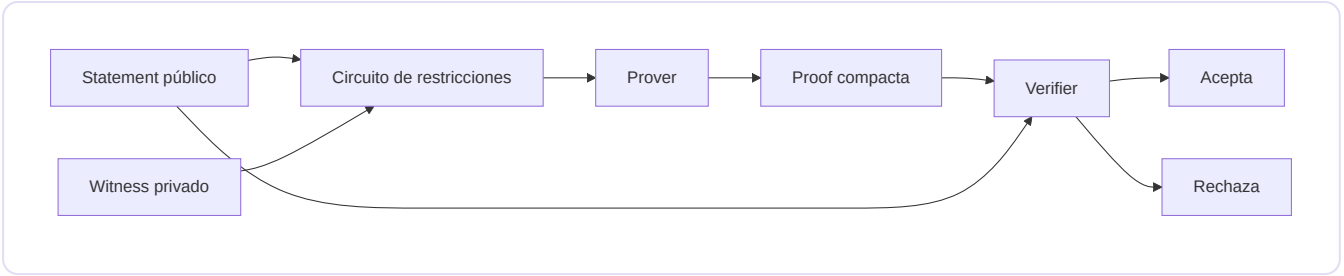
🧠 Modelo mental

Una prueba de conocimiento cero es como demostrar que conoces la contraseña de una puerta abriéndola frente a un testigo, sin pronunciar la contraseña. El prover convence al verifier de que una afirmación es cierta —"esta persona tiene más de 18 años"— entregando solo una prueba compacta, mientras que el dato sensible (la fecha de nacimiento) permanece como witness privado del circuito. La completitud asegura que las afirmaciones ciertas se aceptan, la solidez que las falsas no, y el conocimiento cero que nada más se filtra del enunciado.

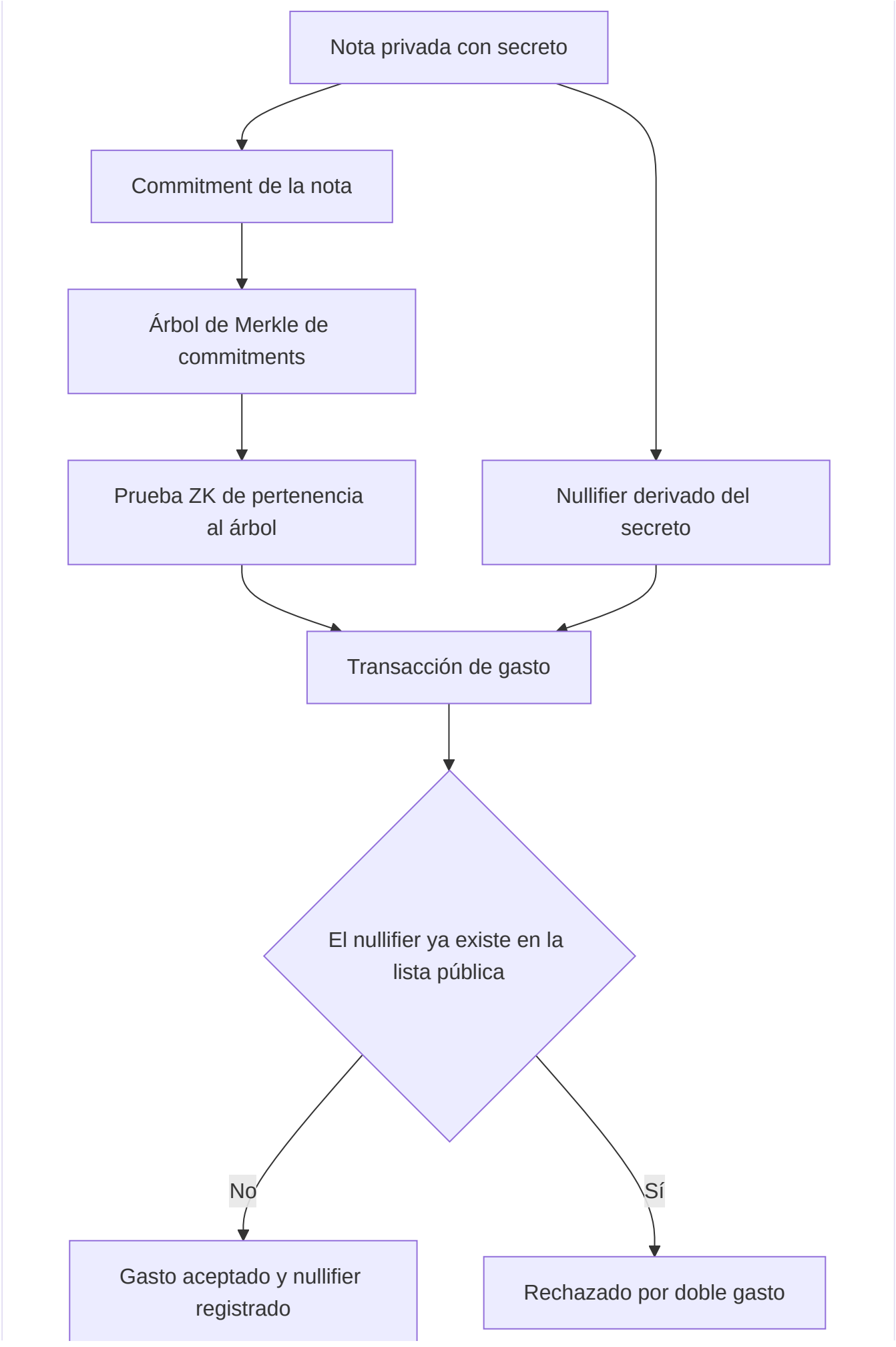
La analogía tiene un límite crucial: aunque no digas la contraseña, el testigo ve a qué hora llegaste, cuántas veces lo intentaste y qué puerta usaste. Igual ocurre con ZK: protege el contenido del enunciado, pero los metadatos —quién emitió la credencial, cuándo se usó, desde qué dirección— pueden seguir revelando información. Además, el sistema hereda un problema del mundo real: alguien debe certificar honestamente el dato inicial, y ZK no verifica esa verdad de origen.

🧩 Esquema visual

El recorrido de una prueba de conocimiento cero, desde el enunciado hasta el veredicto del verifier:



Esquema de nullifier para gastar una nota privada exactamente una vez, sin revelar cuál nota se gasta:


Nota privada con secreto
Commitment de la nota
Árbol de Merkle de commitments
Prueba ZK de pertenencia al árbol
Nullifier derivado del secreto
Transacción de gasto
El nullifier ya existe en la lista pública
No
Sí
Gasto aceptado y nullifier registrado
Rechazado por doble gasto

📖 Conceptos y definiciones

- **Zero knowledge**: propiedad por la cual una prueba convence de la verdad de un enunciado sin revelar nada más allá de su validez.
- **Prover**: parte que posee el witness y genera la prueba; carga el mayor coste computacional del sistema.
- **Verifier**: parte que comprueba la prueba con las entradas públicas; en cadena suele ser un contrato barato de ejecutar.
- **Circuito**: representación de la computación a probar como conjunto de restricciones que el witness debe satisfacer.
- **Witness**: entrada privada (por ejemplo la fecha de nacimiento) que satisface el circuito sin hacerse pública.
- **Compromiso (commitment)**: valor que fija un dato ocultándolo, con la posibilidad de abrirlo después de forma verificable.
- **Trusted setup**: fase que genera parámetros públicos; si los secretos ("toxic waste") no se destruyen, se podrían falsificar pruebas.
- **Setup universal/transparente**: alternativas que reutilizan el setup para muchos circuitos o lo eliminan usando solo aleatoriedad pública.
- **SNARK**: prueba sucinta y de verificación rápida; suele requerir setup y apoyarse en supuestos de curvas elípticas, no post-cuánticos.
- **STARK**: prueba transparente y post-cuántica basada en hashes, con pruebas mayores pero sin trusted setup.

🔬 Profundización

SNARK vs. STARK en números orientativos

Las cifras exactas dependen del esquema concreto (Groth16, PLONK, FRI…), del circuito y del hardware, pero los órdenes de magnitud marcan la decisión de diseño:

Dimensión	SNARK (p. ej. Groth16, PLONK)	STARK
Tamaño de prueba	Cientos de bytes (Groth16: ~128-200 bytes)	Decenas a cientos de KB
Verificación	Milisegundos; barata en cadena (unos pocos pairings)	Milisegundos a decenas de ms; más gas en cadena por el tamaño
Setup	Confiable por circuito (Groth16) o universal actualizable (PLONK, KZG)	Transparente: solo aleatoriedad pública, sin ceremonia
Supuestos criptográficos	Curvas elípticas con pairings; supuestos no estándar	Funciones hash resistentes a colisiones; supuestos mínimos
Resistencia post-cuántica	No: un computador cuántico rompería la curva	Sí, en la medida en que el hash resista
Coste del prover	Alto, pero pruebas pequeñas	Alto, con mejor paralelización y sin setup

Regla práctica: si la verificación en cadena debe ser lo más barata posible y se acepta una ceremonia, un SNARK con setup universal es la opción común; si la transparencia y el post-cuántico pesan más que el tamaño de prueba, un STARK (o un STARK envuelto en un SNARK final para abaratar la verificación, patrón usado por varios rollups) es la elección. Los sistemas de producción evolucionan rápido: contrasta los números del esquema concreto en su documentación.

Ceremonias de trusted setup: Powers of Tau y el modelo 1-de-N

Un trusted setup genera parámetros públicos a partir de un secreto que debe destruirse; si alguien lo conserva (el llamado *toxic waste*), puede falsificar pruebas indistinguibles de las válidas, sin romper nada más del sistema. Las ceremonias multi-participante mitigan este riesgo transformándolo en un supuesto *1-de-N honesto*: cada participante aporta su propia aleatoriedad secreta y la mezcla secuencialmente con la contribución acumulada; para comprometer el resultado, un atacante necesitaría que **todos** los participantes coludieran o fueran comprometidos, mientras que basta **uno solo** que destruya su secreto para que los parámetros sean seguros.

Powers of Tau es la ceremonia genérica más conocida: produce parámetros reutilizables por muchos circuitos (la "fase 1" común, seguida de una fase 2 específica por circuito en esquemas como Groth16). Ejemplos reales verificables: la ceremonia perpetua de Powers of Tau iniciada por la comunidad de Zcash y Ethereum acumuló cientos de contribuciones públicas y auditables, con participantes que llegaron a usar residuos radiactivos o rituales físicos de destrucción de hardware como evidencia teatral pero ilustrativa; la ceremonia KZG de Ethereum para EIP-4844 (2023) superó las 140 000 contribuciones, el mayor N de la historia, precisamente para que el supuesto "al menos uno fue honesto" resultara socialmente creíble. Los setups universales (PLONK) amortizan una sola ceremonia entre todos los circuitos futuros, y los sistemas transparentes (STARK) la eliminan por completo.

El nullifier: anti doble gasto sin revelar qué se gasta

En un protocolo de privacidad, las notas no se marcan como "gastadas" —hacerlo revelaría cuál se gastó—. El nullifier resuelve el dilema: es un valor determinista derivado del secreto de la nota, imposible de vincular con su commitment sin conocer ese secreto, pero único por nota. Ejemplo conceptual numerado:

1. Alicia crea una nota con secreto `s` y publica su commitment `C = H(s, valor)` en el árbol de Merkle del protocolo; nadie sabe que `C` es de Alicia.
2. Para gastar, Alicia calcula el nullifier `NF = H'(s)` con una función distinta, y genera una prueba ZK de que conoce un `s` tal que su commitment está en el árbol **y** que `NF` se deriva de ese mismo `s`.
3. El contrato verifica la prueba, comprueba que `NF` no figura en la lista pública de nullifiers, lo añade y libera el gasto. La prueba no revela cuál de los miles de commitments del árbol se usó.
4. Si Alicia intenta gastar la misma nota otra vez, el circuito la obliga a producir el mismo `NF = H'(s)`, que ya está registrado: la transacción se rechaza. El doble gasto se detecta sin haber desanonimizado ningún gasto legítimo.

El mismo patrón aparece fuera de la privacidad de pagos: los rollups ZK y los sistemas de identidad (por ejemplo, votación anónima o airdrops de un solo uso) usan nullifiers para garantizar "una sola vez por secreto" sin correlacionar acciones con identidades. El diseño fino importa: si el nullifier se deriva también de un contexto (un identificador de votación), la misma credencial puede usarse una vez *por contexto* sin que dos usos en contextos distintos sean vinculables.

Una prueba ZK contada sin matemáticas

La idea suena imposible: *demostrar que sabes algo sin revelar qué sabes*. La analogía clásica —la cueva de Ali Babá— explica el mecanismo mejor que cualquier fórmula.

La cueva. Un túnel circular con una puerta cerrada al fondo que solo se abre con una palabra secreta. Desde la entrada, el túnel se bifurca en dos caminos, A y B, que se juntan en la puerta.

El protocolo:

1. Tú entras y tomas el camino que quieras. La otra persona **no ve cuál**.
2. Desde la entrada, grita al azar: "¡sal por A!".
3. Si sabes la palabra, sales por A siempre — cruzando la puerta si hiciera falta. Si no la sabes, solo puedes salir por donde entraste.

Con una ronda, alguien sin el secreto acierta con probabilidad **1/2**. Repítelo:

```
 1 ronda  → 1/2      = 50 %      de engañar
10 rondas → 1/2^10   ≈ 0,1 %
20 rondas → 1/2^20   ≈ 0,0001 %
```

Y ahí están las tres propiedades, sin una sola fórmula:

- **Completitud:** si sabes la palabra, siempre pasas la prueba.
- **Solidez:** si no la sabes, la probabilidad de colar se hace despreciable con las repeticiones. No cero: *despreciable*. Toda prueba ZK es probabilística.
- **Conocimiento cero:** quien observa ve salidas correctas por caminos aleatorios. **Podría haber grabado ese mismo vídeo sin conocer el secreto**, poniéndose de acuerdo de antemano — por eso la transcripción no le sirve para convencer a un tercero, y por eso no filtra nada.

Qué cambia en la versión real. Los SNARK sustituyen las rondas interactivas por una sola prueba (la transformación de Fiat–Shamir, que usa un hash como si fuera el gritador aleatorio) y el "secreto" es un **witness** que satisface un circuito de restricciones.

Y el límite que hay que tener presente: la prueba demuestra que *conoces un valor que satisface el circuito*. Si el circuito dice "esta fecha de nacimiento está firmada por una autoridad y es anterior a 2007", la prueba no garantiza que la fecha sea cierta: garantiza que **alguien la certificó**. La confianza no desaparece, se traslada al certificador.

> 💡 **En una frase:** una prueba ZK convence de que una afirmación es cierta sin revelar por qué. No convierte en verdad un dato falso: solo prueba que satisface el circuito que escribiste.

▶ 🎓 **Si ya dominas esto** — donde se decide el diseño

🧪 Laboratorio guiado

Este módulo es un diseño conceptual de circuito, sin código de repositorio. Consulta el índice de prácticas del curso en laboratorios.

1. Define el enunciado exacto a probar: "la persona nació antes de una fecha umbral" sin revelar la fecha real.
2. Separa las señales del circuito en públicas y privadas, y anota qué recibe el verifier.

```
Elemento del circuito         | Tipo      | Visible para el verifier
------------------------------+-----------+-------------------------
Fecha de nacimiento           | privada   | no
Fecha umbral (hoy - 18 años) | pública   | sí
Firma del emisor de la credencial | pública/compromiso | sí (compromiso)
Resultado (edad >= 18)        | pública   | sí
```

3. Identifica quién certifica el dato inicial (el emisor de la credencial) y cómo el circuito verifica su firma sin exponer la fecha.
4. Enumera qué metadatos siguen filtrando información (emisor, momento de uso, dirección) y cómo se revocaría la credencial.

📝 Reto verificable

Entrega el diseño conceptual completo de la prueba de mayoría de edad: enunciado, tabla de señales públicas/privadas, actor que certifica el dato, metadatos residuales y un mecanismo de revocación.

Criterio de aceptación: el documento indica explícitamente qué dato queda oculto (la fecha de nacimiento), qué recibe el verifier, quién certifica el dato inicial, al menos dos metadatos que siguen expuestos y un método de revocación que no revela la identidad del titular.

⚠️ Errores frecuentes

Síntoma	Causa y cómo comprobarlo
Creer que ZK anonimiza todo	Solo oculta el enunciado; revisa qué metadatos (tiempos, direcciones) siguen visibles
Ignorar el trusted setup	Si el toxic waste no se destruye, se falsifican pruebas; comprueba si el setup es transparente
Confundir witness con entrada pública	El witness debe ser privado; verifica qué señales expone el circuito al verifier
Suponer que ZK valida la verdad de origen	El circuito prueba una relación, no que el emisor certificó bien; identifica al certificador
Elegir SNARK/STARK por moda	Depende de setup, tamaño y post-cuántico; contrasta requisitos reales del caso
Olvidar la revocación	Una credencial sin revocación queda válida para siempre; diseña el mecanismo desde el inicio

🛡️ Seguridad y ética

- Todo el diseño es conceptual y se realiza en local; no uses datos personales reales, fondos ni claves privadas reales.
- No proceses fechas de nacimiento u otros datos sensibles de personas reales; usa valores ficticios.
- Comunica con honestidad los límites de ZK: protege el enunciado, no elimina metadatos ni riesgos de correlación.
- Considera el marco legal de las credenciales de identidad; la privacidad técnica no exime del cumplimiento normativo.
- Documenta el modelo de confianza del setup; ocultar un trusted setup es una omisión relevante para el usuario.

🔗 Referencias

- Thaler, J., *Proofs, Arguments, and Zero-Knowledge* — https://people.cs.georgetown.edu/jthaler/ProofsArgsAndZK.html
- Least Authority, *The MoonMath Manual to zk-SNARKs* — https://github.com/LeastAuthority/moonmath-manual
- ZKProof, Community Reference — https://zkproof.org/
- circom, documentación del lenguaje de circuitos — https://docs.circom.io/
- Fuente primaria: Goldwasser, Micali y Rackoff, *The Knowledge Complexity of Interactive Proof-Systems* (1985), definición original de conocimiento cero.

✅ Criterio de dominio

- Enuncias las tres propiedades de una prueba ZK y las aplicas al caso de la mayoría de edad sin ayuda.
- Comparas SNARK y STARK y eliges uno justificando setup, tamaño y post-cuántico.
- Identificas los metadatos residuales y el mecanismo de revocación de una credencial ZK.

🧭 Navegación

⬅️ Módulo 13 · Interoperabilidad y ecosistemas · 📚 Índice del currículo · ➡️ Módulo 15 · Arquitectura avanzada

15 · Arquitectura avanzada

> **Nivel:** Avanzado · ⏱️ **Duración estimada:** 180 min · **Fuente:** ERC-4337 / EIP-7702 (abstracción de cuenta) e investigación de Flashbots (MEV) ⬅️ Currículo · 📚 Bibliografía 🧭 ⬅️ **Anterior:** 14 · Privacidad y zero knowledge · 📚 Índice · ➡️ **Siguiente:** 16 · Infraestructura y operación de nodos 📖 Glosario de términos · 🌱 ¿Nuevo en esto? Empieza aquí

🎯 Objetivos

- Integrar la abstracción de cuenta con ERC-4337 (UserOperations, bundlers, paymasters) y EIP-7702 para EOAs en Pectra (2025).
- Diseñar sistemas actualizables con patrones proxy (UUPS, Transparent) cuidando el storage layout y sus riesgos.
- Analizar el MEV (front-running, sandwich) y las respuestas de mercado como PBS y mev-boost.
- Redactar un documento de arquitectura que cubra amenazas, invariantes, costos, gobernanza, privacidad, legalidad y operación.
- Argumentar cuándo la decisión más madura es reducir o eliminar componentes blockchain.

📚 Resultados de aprendizaje

Al finalizar, el estudiante podrá:

1. **Explicar** el flujo de una UserOperation en ERC-4337 y en qué se diferencia de EIP-7702.
2. **Diseñar** un contrato actualizable seguro justificando el patrón proxy y el storage layout.
3. **Identificar** vectores de MEV en un flujo de usuario y proponer mitigaciones.
4. **Elaborar** un documento de arquitectura con activos, actores, amenazas e invariantes explícitos.
5. **Evaluar** tokenomics de un diseño con un simulador de suministro y contrastarlo con sus objetivos.
6. **Decidir** con criterio cuándo una alternativa no-blockchain resuelve mejor el problema.

🗺️ Temas

#	Tema	Por qué importa
1	ERC-4337 (UserOps, bundlers, paymasters)	Habilita cuentas programables y patrocinio de gas sin cambiar el protocolo
2	EIP-7702 (EOAs con código en Pectra)	Da capacidades de smart account a cuentas existentes desde 2025
3	Proxies y upgrades (UUPS, Transparent)	Permiten evolucionar contratos; un error de storage puede corromper el estado
4	MEV: front-running y sandwich	El orden de las transacciones es un activo que se explota
5	PBS y mev-boost	Separan proponer de construir bloques para acotar el poder del validador
6	Tokenomics y economía de mecanismos	Alinea incentivos; un mal diseño rompe el sistema aunque el código sea correcto
7	Observabilidad y respuesta a incidentes	Sin monitoreo no hay detección ni contención de un fallo
8	Cumplimiento y decisión de no usar blockchain	La madurez incluye reconocer cuándo blockchain no aporta

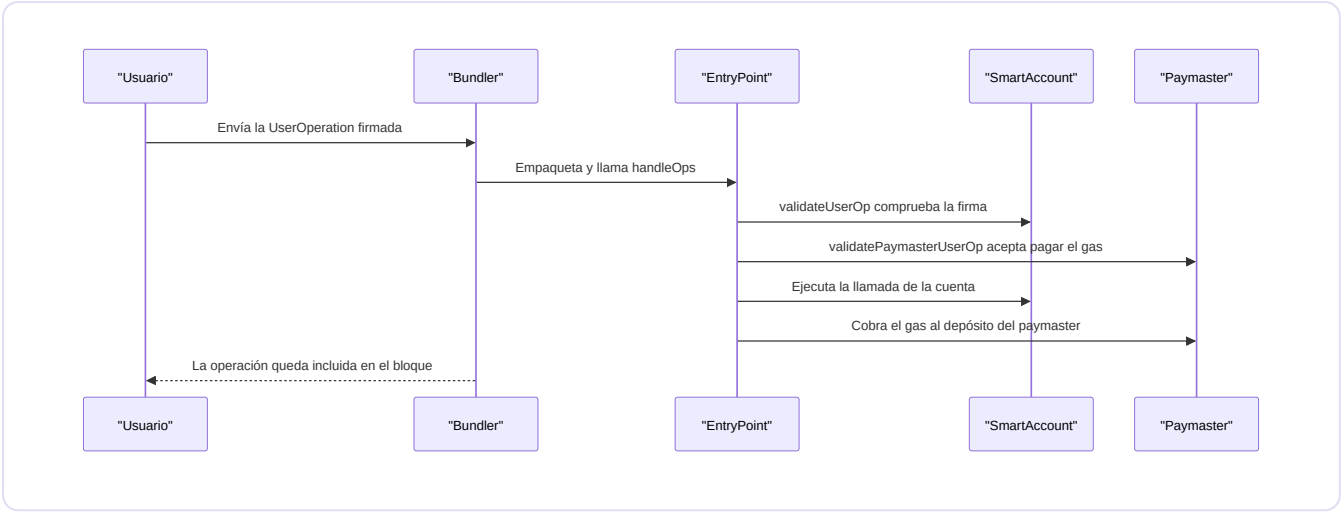
🧠 Modelo mental

Piensa en la arquitectura avanzada como el plano de un edificio, no como el ladrillo. La abstracción de cuenta cambia las cerraduras (quién puede firmar y cómo se paga la entrada), los proxies son las reformas que permiten cambiar habitaciones sin demoler la estructura, y el MEV es el portero que decide en qué orden entran los visitantes y puede cobrar por adelantar a unos frente a otros. El documento de arquitectura es la memoria del proyecto: qué se construye, para quién, con qué riesgos y qué se hace cuando algo falla.

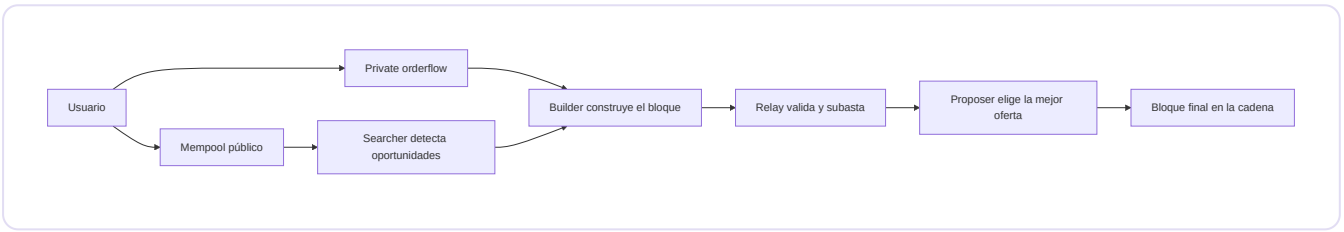
La analogía se queda corta en un punto esencial: a diferencia de un edificio, aquí muchas veces la mejor decisión de diseño es construir menos. Añadir un componente blockchain introduce costes, superficie de ataque y rigidez de actualización; la madurez técnica se demuestra sabiendo cuándo una base de datos, una firma o un servicio convencional resuelven el problema con menos riesgo. El plano correcto puede ser el que retira ladrillos.

🧩 Esquema visual

Flujo de una UserOperation en ERC-4337, con patrocinio de gas por un paymaster:



La cadena de suministro del MEV bajo PBS: cada eslabón compite por capturar valor del orden de las transacciones.



📖 Conceptos y definiciones

- **Abstracción de cuenta**: capacidad de que la lógica de validación de una cuenta sea programable, en lugar de fija como en una EOA clásica.
- **UserOperation**: intención firmada que un bundler empaqueta y envía en ERC-4337, procesada por un contrato EntryPoint.
- **Paymaster**: contrato que puede patrocinar el gas de una UserOperation, habilitando pagos en tokens o gas gratuito condicionado.
- **EIP-7702**: mecanismo de Pectra (2025) que permite a una EOA delegar en código de contrato, acercándola a una smart account.
- **Proxy UUPS/Transparent**: patrón que separa lógica y almacenamiento para permitir upgrades; exige preservar el storage layout.
- **MEV**: valor extraíble por reordenar, insertar o censurar transacciones dentro de un bloque; incluye front-running y sandwich.
- **PBS (proposer-builder separation)**: separación entre quien propone el bloque y quien lo construye, para limitar la extracción de MEV.
- **mev-boost**: software que implementa un mercado de construcción de bloques externo para validadores de Ethereum.
- **Invariante**: propiedad que el sistema debe cumplir siempre (por ejemplo "el suministro total nunca disminuye salvo por quema").
- **Tokenomics**: diseño económico del token (emisión, distribución, incentivos) que determina la sostenibilidad del sistema.

🔬 Profundización

ERC-4337 frente a EIP-7702: dos caminos hacia la cuenta programable

ERC-4337 (2023) construyó la abstracción de cuenta *sin tocar el protocolo*: un mempool alternativo de UserOperations, bundlers que las empaquetan y un contrato EntryPoint que orquesta validación y ejecución. EIP-7702 (Pectra, mayo de 2025) atacó el problema desde el protocolo: una EOA existente puede firmar una autorización de delegación que hace que su dirección ejecute el código de un contrato, conservando su clave y su dirección de siempre.

Dimensión	ERC-4337	EIP-7702
Tipo de cuenta	Contrato smart account nuevo, con dirección propia	La EOA de siempre, con código delegado
Cambio de protocolo	Ninguno; infraestructura fuera del protocolo	Sí; nuevo tipo de transacción en Pectra (2025)
Migración del usuario	Debe mover activos a la cuenta nueva	Ninguna; conserva dirección e historial
Batching de llamadas	Sí, nativo en la cuenta	Sí, vía el código delegado
Session keys y políticas	Sí, con lógica de validación arbitraria	Sí, según el contrato delegado
Sponsorship de gas	Paymasters vía EntryPoint	Compatible: una 7702-EOA puede actuar como cuenta 4337
Riesgo característico	Complejidad del EntryPoint y de los bundlers	Delegar en un contrato malicioso entrega la cuenta entera

No son rivales sino complementarios: el diseño previsto es que las EOA con 7702 deleguen en implementaciones de smart account compatibles con 4337, unificando ambos mundos. Las cifras de adopción (cuentas 4337 activas, delegaciones 7702) cambian mes a mes: consúltalo en vivo en paneles como los de BundleBear en Dune.

MEV en números: anatomía de un sandwich

Un sandwich explota la tolerancia al deslizamiento (slippage) de un swap visible en el mempool. Ejemplo numérico simplificado:

1. Alicia envía un swap de 100 000 USDC por ETH en un AMM, con un slippage máximo del 1%: acepta recibir como mínimo el 99% del precio actual.
2. Un searcher lo ve en el mempool y compra ETH justo antes (front-run), empujando el precio al alza dentro del margen que Alicia toleró.

3. El swap de Alicia se ejecuta al peor precio permitido: recibe ~1% menos de ETH, es decir, hasta ~1 000 USDC de valor cedido.
4. El searcher vende inmediatamente después (back-run) el ETH comprado, capturando la diferencia menos el gas y el pago al builder por la posición en el bloque.

El valor no aparece de la nada: sale del slippage de Alicia. Las mitigaciones atacan cada eslabón: el *private orderflow* (enviar la transacción directamente a un builder o a un RPC protegido) evita exponerla en el mempool público; las *batch auctions* tipo CoW Protocol liquidan muchas órdenes a un precio uniforme por lote, eliminando la ventaja del orden intra-bloque; y MEV-Share invierte el juego devolviendo al usuario parte del valor que su flujo genera. En paralelo, PBS con mev-boost no elimina el MEV, pero lo saca de las manos del validador individual y lo convierte en un mercado competitivo de builders, más observable y menos discrecional.

Runbook de un upgrade seguro

Un upgrade de contrato es una operación de producción con usuarios y fondos en juego; improvisar es la principal causa de incidentes autoinfligidos. Secuencia mínima:

1. **Proponer**: publicar el código nuevo, su auditoría o revisión, el diff del storage layout y la motivación del cambio; abrir la propuesta a la gobernanza correspondiente.
2. **Timelock público**: encolar la ejecución en un timelock en cadena (por ejemplo, de 48 horas a varios días) para que el cambio sea inevitablemente visible antes de aplicarse.
3. **Comunicar**: anunciar por los canales oficiales qué cambia, cuándo y qué debe hacer un usuario que no esté de acuerdo; el silencio convierte un upgrade legítimo en indistinguible de un ataque.
4. **Ventana de salida**: garantizar que durante el timelock los usuarios pueden retirar fondos o revocar aprobaciones si rechazan el cambio; sin salida real, la gobernanza es cosmética.
5. **Ejecución multisig**: ejecutar desde un multisig con umbral y firmantes públicos, verificando que el calldata ejecutado coincide byte a byte con lo propuesto y encolado.
6. **Verificación post-upgrade**: comprobar en cadena la nueva dirección de implementación, correr los invariantes sobre el estado migrado, verificar el código en el explorador y monitorizar métricas y eventos anómalos durante las primeras horas, con un plan de contención listo por si algo falla.

Un sándwich de MEV, contado en números

El MEV se entiende cuando se ve el dinero moverse. Sigamos el caso más común: alguien intenta comprar un token y un bot le extrae valor sin robarle nada en sentido técnico.

La víctima envía: comprar 100 000 USDC de TOKEN, con una tolerancia de deslizamiento del 3 %.

Ese último número es la puerta. Le está diciendo al mundo: *acepto pagar hasta un 3 % peor de lo que veo ahora*.

El bot lo ve en el mempool y construye tres transacciones seguidas:

```
1. [BOT compra]     sube el precio del pool hasta el borde del 3 %
2. [VÍCTIMA compra] se ejecuta al precio empeorado — pero dentro de su tolerancia,
                    así que NO revierte y para ella "funcionó"
3. [BOT vende]      deshace su posición al precio que la víctima acaba de crear
```

El resultado, con números redondos:

```
La víctima recibe ~3 % menos TOKEN del que habría recibido sin el bot
  3 % de 100 000 USDC  ≈  3 000 USDC de valor extraído
Menos el gas del bot y su pago al constructor del bloque → beneficio neto para el bot
```

Lo incómodo del asunto: no hubo hackeo, ni bug, ni contrato malicioso. Todo funcionó como está escrito. El valor extraído sale de un parámetro que la víctima eligió y probablemente no entendía.

Qué reduce la exposición, y qué no:

Medida	Efecto real
Bajar el deslizamiento al 0,5 %	Reduce mucho lo extraíble: el margen del sándwich es literalmente ese número
Enviar por un **RPC privado**	La transacción no pasa por el mempool público, así que el bot no la ve venir
Operar en pools con liquidez profunda	Mover el precio cuesta más, y el ataque deja de compensar
"Poner más gas"	**No ayuda**: el bot puja por posición igual y suele pujar mejor

> 💡 **En una frase:** el deslizamiento no es un ajuste técnico, es cuánto autorizas a que te extraigan. El sándwich se cobra exactamente esa cifra.

▶ 🎓 **Si ya dominas esto** — el MEV como capa de mercado

🧪 Laboratorio guiado

> 🧪 Estas prácticas están catalogadas y **resueltas paso a paso** en el catálogo de laboratorios.

Ejecuta el simulador de suministro para la parte de tokenomics y luego integra los hallazgos en el documento de arquitectura. El proyecto integrador vive en `capstone` y en `projects/community-funding`.

1. Lanza el simulador de tokenomics y registra las curvas de emisión y suministro resultantes.

```
pnpm lab:tokenomics
```

2. Varía los parámetros de emisión y observa el efecto sobre el suministro total y los incentivos.
3. Documenta el patrón de upgrade elegido (UUPS o Transparent) y verifica que el storage layout se preserva entre versiones.
4. Traza el flujo de una UserOperation (ERC-4337) y contrasta el modelo con una delegación EIP-7702 desde una EOA.
5. Analiza dónde aparece MEV en el flujo de usuario y qué mitigación aplicarías (por ejemplo envío privado o subasta de bloques).

📝 Reto verificable

Entrega un documento de arquitectura del proyecto integrador que incluya: problema y usuarios, decisión blockchain vs. alternativas, componentes y límites de confianza, activos/actores/amenazas, invariantes, costos y escalabilidad, administración y gobernanza, privacidad y aspectos legales, y pruebas/despliegue/monitoreo/retiro.

Criterio de aceptación: el documento contiene las nueve secciones citadas, define al menos tres invariantes verificables, justifica explícitamente la decisión de usar (o no) blockchain frente a alternativas, y adjunta los resultados del simulador `pnpm lab:tokenomics` que respaldan el diseño económico.

⚠️ Errores frecuentes

Síntoma	Causa y cómo comprobarlo
Upgrade que corrompe el estado	Cambio en el storage layout; compara las variables de almacenamiento entre versiones
Confundir ERC-4337 con EIP-7702	Uno usa UserOps vía EntryPoint, el otro delega desde una EOA; revisa el flujo de cada uno
Ignorar el MEV en el diseño	Se asume orden neutral; simula un sandwich sobre el flujo de swap
Paymaster sin límites	Patrocinio ilimitado se explota; verifica cuotas y validación del paymaster
Tokenomics sin invariantes	El incentivo se rompe silenciosamente; define y prueba invariantes de suministro
Usar blockchain por defecto	No se evaluó la alternativa; documenta por qué una base de datos no bastaba

🛡️ Seguridad y ética

- Trabaja en local o testnet; no despliegues con fondos reales ni utilices claves privadas reales en el laboratorio.
- No incluyas datos personales reales en el documento de arquitectura ni en el simulador.
- Declara los límites de confianza y las claves administrativas; ocultar un poder de upgrade es una omisión ética.
- Reconoce el impacto del MEV en los usuarios y prioriza diseños que reduzcan la extracción a su costa.
- Considera el cumplimiento legal y la privacidad desde el diseño, no como un añadido posterior.

🔗 Referencias

- ERC-4337, Account Abstraction Using Alt Mempool — https://eips.ethereum.org/EIPS/eip-4337
- EIP-7702, Set EOA account code (Pectra) — https://eips.ethereum.org/EIPS/eip-7702
- Flashbots, investigación sobre MEV — https://writings.flashbots.net/
- OpenZeppelin, Upgrades Plugins — https://docs.openzeppelin.com/upgrades-plugins/
- Voshmgir, S., *Token Economy* — https://tokeneconomy.co/
- Fuente primaria: Daian et al., *Flash Boys 2.0* — https://arxiv.org/abs/1904.05234

✅ Criterio de dominio

- Explicas el flujo de ERC-4337 y su diferencia con EIP-7702 sin apoyo.

- Diseñas un upgrade seguro razonando el storage layout y el patrón proxy elegido.
- Produces un documento de arquitectura con invariantes verificables y una decisión justificada sobre usar o no blockchain.

🧭 Navegación

⬅️ Módulo 14 · Privacidad y zero knowledge · 📚 Índice del currículo · ➡️ Módulo 16 · Infraestructura y operación de nodos

16 · Infraestructura y operación de nodos

> **Nivel:** Avanzado-Producción · ⏱️ **Duración estimada:** 180 min · **Fuente:** documentación de clientes de nodo (ethereum.org, Geth, Lighthouse) y guías de operación de EthStaker
> ⬅️ Currículo · 📚 Bibliografía 🧭 ⬅️ **Anterior:** 15 · Arquitectura avanzada · 📚 Índice · ➡️ **Siguiente:** 17 · Blockchain en la empresa: valor, casos y costos 📖 Glosario de términos · 🌱 ¿Nuevo en esto? Empieza aquí

Hasta aquí el programa habló de protocolos; este módulo habla de **discos, memoria, máquinas y facturas**. Qué hardware necesita cada tipo de nodo, dónde corre físicamente (casa, datacenter o nube), con qué comandos se levanta y cómo se opera sin caerse.

🎯 Objetivos

- Dimensionar el hardware real (CPU, RAM, NVMe, red) que exige cada tipo de nodo.
- Comparar las cuatro opciones físicas de despliegue: casa, colocation, nube y nodo gestionado.
- Levantar un nodo de desarrollo con Docker y consultarlo por JSON-RPC.
- Diseñar una topología de nodos redundante para servir RPC en producción.
- Ubicar dónde viven físicamente las claves (hardware wallet, HSM, MPC, multisig).

📚 Resultados de aprendizaje

Al finalizar, el estudiante podrá:

1. **Especificar** la máquina para un full node de Bitcoin o Ethereum y justificar cada componente.
2. **Explicar** por qué las IOPS del disco —y no la CPU— son el cuello de botella de un nodo.
3. **Levantar** un cliente de ejecución en modo desarrollo con Docker y verificarlo con `cast`.
4. **Presupuestar** a orden de magnitud un despliegue en nube (instancia, disco, egreso, réplica).
5. **Diseñar** una topología con dos zonas de disponibilidad y contingencia externa.
6. **Distinguir** la infraestructura del nodo de la infraestructura de firma de claves.

🗺️ Temas

#	Tema	Por qué importa
1	Requisitos de hardware por tipo de nodo	Un presupuesto errado mata el proyecto antes de empezar
2	IOPS, NVMe y por qué el disco manda	Es la causa número uno de nodos que nunca sincronizan
3	Ejecución + consenso en Ethereum post-Merge	Son dos procesos con secreto JWT compartido, no uno
4	Casa vs. colocation vs. nube vs. gestionado	Cada opción es un modelo de amenaza y de costo distinto
5	Nube en concreto: instancias, discos, egreso	Las facturas reales salen del egreso y los snapshots
6	Docker y orquestación de flotas	La unidad de despliegue de la industria
7	Monitoreo y alertas del nodo	Peers, distancia a la punta y disco libre: lo que despierta a un operador
8	Infraestructura de firma: HSM, MPC, multisig	Las claves nunca viven en el nodo

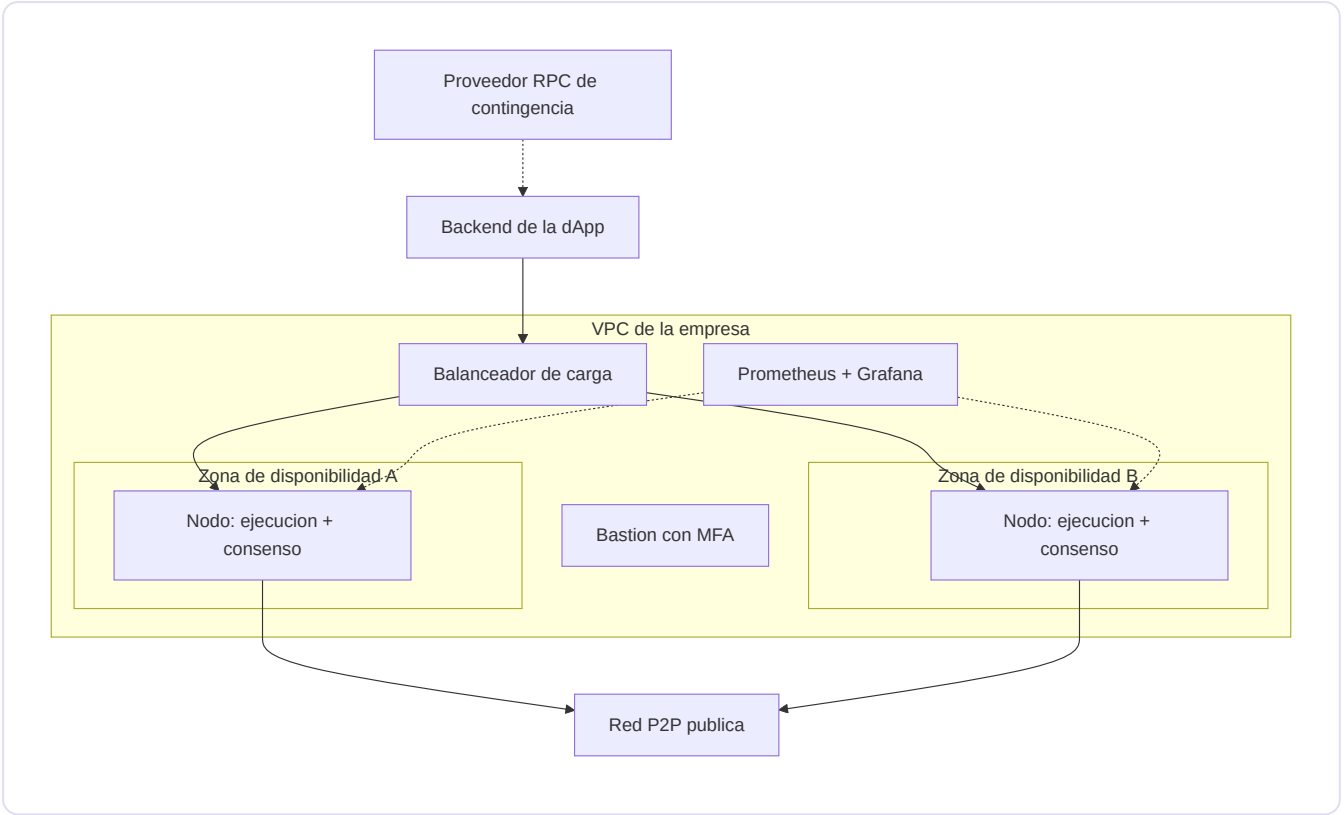
🧠 Modelo mental

Un nodo es una **bodega con un mostrador**: la bodega (el disco) recibe camiones de historia todos los días y el mostrador (la API RPC) atiende consultas. Si la bodega es lenta, la fila de camiones crece y nunca te pones al día; si el mostrador se expone a la calle sin control, cualquiera te lo revienta a consultas. La empresa seria tiene dos bodegas en barrios distintos y un proveedor externo por si ambas fallan.

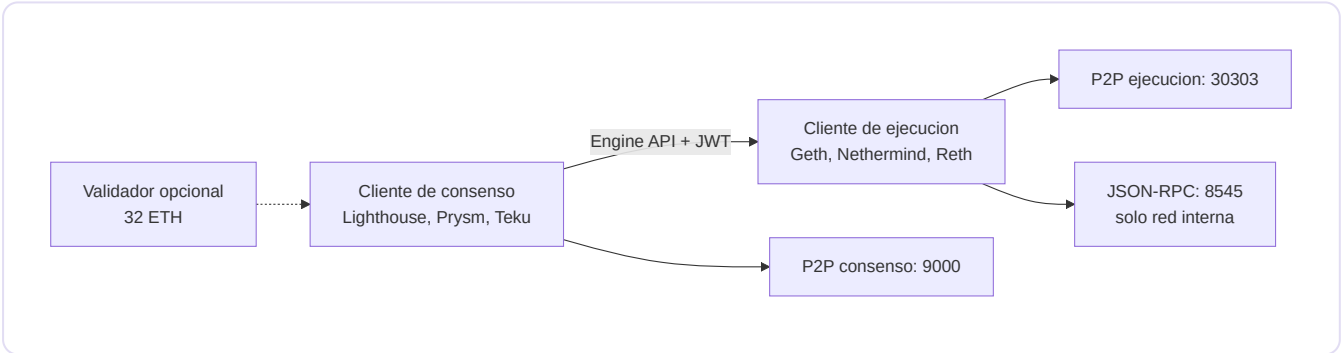
El límite de la analogía: a diferencia de una bodega, el nodo **verifica** cada camión criptográficamente — correr tu propio nodo es lo que te libera de confiar en el mostrador de otro.

🧩 Esquema visual

Topología mínima seria para servir RPC propio en producción:



Anatomía de un nodo Ethereum post-Merge — dos procesos, un secreto compartido:



📖 Conceptos y definiciones

- **IOPS**: operaciones de entrada/salida por segundo del disco; un nodo hace millones de lecturas aleatorias, por eso NVMe es obligatorio y HDD imposible.
- **Full node**: verifica y almacena el estado reciente de la cadena (~700 GB Bitcoin, 1,2-2 TB Ethereum, orientativo — consúltalo en vivo).
- **Archive node**: conserva todos los estados históricos; con Erigon/Reth ronda 2,5-3 TB, con clientes clásicos mucho más.
- **Checkpoint sync**: arranque del cliente de consenso desde un estado finalizado confiable, en minutos en vez de días.
- **Snap sync**: sincronización de la ejecución descargando el estado reciente en lugar de reproducir toda la historia.
- **JWT secret**: secreto compartido entre cliente de ejecución y de consenso para autenticar la Engine API.

- **Egreso**: tráfico saliente de la nube; se factura aparte y domina el costo de servir RPC público.
- **Nodo gestionado**: RPC como servicio (Alchemy, Infura, QuickNode); rapidez de arranque a cambio de dependencia y límites de tasa.
- **HSM / MPC**: hardware o cómputo multiparte donde viven las claves de firma; el nodo nunca custodia fondos.
- **Home staking**: validador operado en casa (mini-PC, UPS, fibra); aporta descentralización real a la red.

🔬 Profundización

La tabla que define el presupuesto

Nodo	CPU	RAM	Disco (orientativo)	Red	Nota
Bitcoin full	2-4 núcleos	4-8 GB	~700 GB SSD	50+ GB/mes	Descarga inicial: días
Ethereum full	4-8 núcleos	16-32 GB	1,2-2 TB **NVMe**	25+ Mbps	Ejecución + consenso
Ethereum archive	8-16 núcleos	32-64 GB	2,5-3 TB NVMe (Erigon/Reth)	25+ Mbps	Para indexación histórica
Validador Ethereum	4 núcleos	16-32 GB	2 TB NVMe	estable	+ 32 ETH por validador
Solana validator	12+ núcleos	256+ GB	varios TB NVMe separados	1+ Gbps	Otra liga; verifica requisitos vivos

Nube en números

Para **un** nodo Ethereum full: AWS `i4i.2xlarge` (8 vCPU, 64 GB, NVMe local 1,9 TB) o `m7i.2xlarge` + EBS `gp3` con IOPS aprovisionadas; GCP `n2-standard-8` + Local SSD; Azure serie `L` optimizada en almacenamiento. Orden de magnitud: **150-800 USD/mes** por nodo según instancia y disco — y produción seria duplica por la réplica en otra zona. Partidas que los presupuestos olvidan: egreso, snapshots de 2 TB y el proveedor externo de contingencia. Verifica en las calculadoras oficiales de cada nube.

Diversidad de clientes: por qué importa

Si un solo cliente de ejecución concentra la supermayoría y tiene un bug, la red entera puede finalizar un estado inválido. Elegir cliente minoritario (Nethermind, Besu, Reth; Lighthouse, Teku, Nimbus) es una decisión de ingeniería **y** de salud de la red — métricas vivas en https://clientdiversity.org/.

Qué pasa, minuto a minuto, cuando un validador se cae

La lista de comprobación dice "ten un SAI". Esta sección explica **qué te cuesta no tenerlo**, que es lo que hace que la gente lo compre.

Un validador de Ethereum tiene un trabajo continuo: **atestiguar** en cada época (unos 6,4 minutos) que ve la cadena correcta. Si está apagado, no atestigua.

Tiempo caído	Qué ocurre	Coste aproximado
Un corte de segundos entre épocas	Nada: no había atestación en juego	0
Minutos	Se pierden atestaciones. La penalización por *no* atestiguar es **similar en magnitud** a la recompensa que habrías ganado	Dejas de ganar y pierdes algo parecido: el doble de daño que "solo no cobrar"
Horas o días	Sigue el goteo, pero la cadena finaliza igual porque la mayoría está en línea	Pérdida lineal y lenta; recuperable
Caída masiva (>1/3 de la red a la vez)	Se activa la **fuga de inactividad**: la penalización crece de forma cuadrática hasta que los ausentes pierden peso suficiente para que la cadena vuelva a finalizar	Aquí sí se destruye capital rápido

La lección que cambia decisiones: **para un validador doméstico, estar caído no es una emergencia**. Pierdes poco a poco y lo recuperas al volver. Lo que sí destruye capital es el **doble voto** — dos máquinas firmando con la misma clave a la vez — porque eso es *slashing*, se penaliza deliberadamente y no se recupera.

De ahí la regla que parece contraintuitiva y es la correcta:

> Ante la duda, **apaga**. Nunca arranques un segundo validador "por si acaso el primero está caído". Un validador apagado pierde céntimos; dos validadores encendidos con la misma clave pierden el depósito.

Esto reordena la lista de prioridades: la contingencia no es "tener otra máquina lista para arrancar", es "tener la certeza de que solo una está firmando".

El presupuesto que casi nadie calcula bien

Comparemos nodo propio contra RPC gestionado con números, porque la intuición falla en las dos direcciones.

Nodo de ejecución + consenso en la nube, uso interno:

Partida	Estimación mensual	Nota
VM (8 vCPU, 32 GB)	150–250 USD	El cálculo que todo el mundo hace
Disco NVMe de 4 TB	300–500 USD	La partida que se subestima: el disco cuesta más que la máquina
Egreso de datos	20–200 USD	**La sorpresa**: no aparece en la calculadora de la VM
Snapshots / respaldo	40–80 USD	
Total	**≈ 500–1 000 USD**	Sin contar el tiempo de la persona que lo opera

RPC gestionado: desde gratis (con límites de tasa) hasta 50–500 USD/mes según volumen.

La conclusión honesta no es "el nodo propio es caro", sino: **el nodo propio casi nunca se paga por precio; se paga por lo que compra.**

- **Soberanía:** nadie puede censurar tus consultas ni cerrarte la cuenta.
- **Privacidad:** un proveedor externo ve todas tus consultas, y de ahí se deduce qué direcciones te importan y cuándo.
- **Verificación propia:** un nodo completo comprueba las reglas por sí mismo. Confiar en un RPC es confiar en que quien te responde no miente.

Si tu proyecto no necesita ninguna de las tres, el RPC gestionado es la decisión racional, y decirlo así es más profesional que montar infraestructura por costumbre. Lo que **no** es defendible es depender de un único proveedor sin plan B: eso no es ahorrar, es tener un punto único de fallo que no controlas.

> 💡 **En una frase:** el disco y el egreso, no la CPU, deciden la factura; y el nodo propio se justifica por soberanía, privacidad y verificación, no por precio.

▶ 🎓 **Si ya dominas esto** — lo que decide en operación real

🧪 Laboratorio guiado

> 🧪 Estas prácticas están catalogadas y **resueltas paso a paso** en el catálogo de laboratorios.

El laboratorio usa un nodo real en **modo desarrollo** (cadena efímera, sin sincronizar terabytes) para practicar la operación exacta de producción. Requiere Docker y Foundry.

1. Levanta un cliente de ejecución real (Geth) en modo dev:

```
docker run -d --name geth-dev -p 127.0.0.1:8545:8545 \
  ethereum/client-go:stable --dev --http --http.addr 0.0.0.0 \
  --http.api eth,net,web3
```

2. Opéralo por JSON-RPC igual que un nodo de producción:

```
cast chain-id --rpc-url http://127.0.0.1:8545
cast block-number --rpc-url http://127.0.0.1:8545
cast rpc eth_syncing --rpc-url http://127.0.0.1:8545
cast rpc net_peerCount --rpc-url http://127.0.0.1:8545
```

3. Observa los logs como lo haría un operador y detén el nodo limpiamente:

```
docker logs --tail 20 geth-dev
docker stop geth-dev && docker rm geth-dev
```

4. Compara con Anvil (`anvil` + los mismos comandos `cast`): misma interfaz RPC, distinto motor — esa uniformidad es la que permite cambiar de proveedor sin tocar la dApp.
5. Consulta los requisitos vivos de un full node en https://ethereum.org/developers/docs/nodes-and-clients/run-a-node/ y anota disco y RAM vigentes.

📝 Reto verificable

Una empresa te pide **RPC propio con disponibilidad ante la caída de una zona** para su dApp en una L2. Entrega un documento de una página con: (a) topología (diagrama con dos zonas, balanceador y contingencia externa), (b) especificación de máquina por nodo, (c) presupuesto mensual con precios citados de la calculadora del proveedor elegido y fecha de consulta, y (d) tres alertas de monitoreo con su umbral.

Criterio de aceptación: el diagrama no expone el puerto RPC a internet; el presupuesto incluye réplica, egreso y snapshots (no solo la VM); cada alerta indica métrica, umbral y acción del operador.

⚠️ Errores frecuentes

Síntoma	Causa y cómo comprobarlo
El nodo "nunca termina de sincronizar"	Disco sin IOPS suficientes (HDD o SSD de red lento); mide latencia de disco y compara con NVMe
Factura de nube el triple de lo esperado	Egreso y snapshots no presupuestados; revisa el desglose de facturación, no la calculadora de VM
El validador queda offline en cada corte de luz	Sin UPS ni reinicio automático; prueba el ciclo de energía completo
"Me hackearon el nodo"	Puerto 8545 expuesto a internet; el RPC va por red interna o autenticado, solo el P2P es público
Todo funciona hasta que cae el proveedor RPC	Sin contingencia: configura fallback (nodo propio ↔ proveedor externo) y pruébalo apagando uno
Pánico por "perder el nodo"	El nodo es reconstruible; lo irrecuperable son las claves — están en HSM/MPC/hardware wallet, no en el nodo

🛡️ Seguridad y ética

- El nodo del laboratorio corre en modo dev local: jamás expongas `--http.addr 0.0.0.0` en una máquina con IP pública sin firewall.
- Las claves de validador y de tesorería se generan y respaldan **fuera** del nodo (ceremonia documentada, respaldo físico); el nodo es desechable, las claves no.
- Concentrar validadores en dos o tres nubes daña la descentralización que da valor a la red; la diversidad geográfica y de cliente es parte de la ética del operador.
- Respeta los términos de las redes públicas: un nodo que abusa del P2P o replica spam degrada el servicio de todos.

🔗 Referencias

- ethereum.org — *Run a node* y *Nodes and clients*: https://ethereum.org/developers/docs/nodes-and-clients/run-a-node/
- Geth — documentación oficial: https://geth.ethereum.org/docs
- Lighthouse Book (Sigma Prime): https://lighthouse-book.sigmaprime.io/
- Bitcoin Core — requisitos de full node: https://bitcoin.org/en/full-node
- EthStaker — guías de staking y hardware: https://ethstaker.org/
- Diversidad de clientes: https://clientdiversity.org/
- Calculadoras: AWS https://calculator.aws/, GCP https://cloud.google.com/products/calculator, Azure https://azure.microsoft.com/pricing/calculator/

✅ Criterio de dominio

- Especificas y presupuestas la máquina de un full node justificando disco, RAM y red con fuentes vivas.
- Levantas y consultas un nodo por JSON-RPC con Docker y `cast` sin exponer el RPC.
- Explicas dónde viven las claves y por qué el nodo es desechable pero las claves no.

🧭 Navegación

⬅️ Módulo 15 · Arquitectura avanzada · 📚 Índice del currículo · ➡️ Módulo 17 · Blockchain en la empresa

17 · Blockchain en la empresa: valor, casos y costos

> **Nivel:** Avanzado-Producción · ⏱️ **Duración estimada:** 150 min · **Fuente:** informes del BIS y el WEF, casos públicos documentados y *The Blockchain and the New Architecture of Trust* (Werbach) ⬅️ Currículo · 📚 Bibliografía 🧭 ⬅️ **Anterior:** 16 · Infraestructura y operación de nodos · 📚 Índice · ➡️ **Siguiente:** 18 · Implementación empresarial end-to-end 📖 Glosario de términos · 🌱 ¿Nuevo en esto? Empieza aquí

Este módulo responde la pregunta del directorio: **¿qué gana la empresa con esto, en qué se usa, cuánto cuesta llevarlo a cabo y qué servicios existen para no construir todo?** Con casos reales verificables — éxitos y fracasos — porque el criterio profesional se entrena con ambos.

🎯 Objetivos

- Identificar los cinco beneficios reales que una blockchain aporta a una empresa y su mecanismo.
- Mapear los usos con tracción por sector y distinguirlos de los sobreprometidos.
- Analizar casos de éxito y fracaso documentados extrayendo la lección de cada uno.
- Presupuestar a orden de magnitud un proyecto empresarial (equipo, auditoría, infraestructura, servicios).
- Evaluar los servicios del mercado (custodia, nodo gestionado, KYT, auditoría) y cuándo contratarlos.

📚 Resultados de aprendizaje

Al finalizar, el estudiante podrá:

1. **Explicar** ante un directorio, sin jerga, qué gana la empresa y qué mecanismo produce ese beneficio.
2. **Clasificar** un caso de uso como de tracción real, emergente o humo, con evidencia.
3. **Extraer** la lección de un caso fallido y aplicarla a una propuesta nueva.
4. **Construir** un presupuesto de proyecto con partidas completas y precios citados en vivo.
5. **Seleccionar** servicios de terceros con criterios de build vs. buy justificados.
6. **Anticipar** las consideraciones regulatorias, reputacionales y de gobernanza del caso.

🗺️ Temas

#	Tema	Por qué importa
1	Los cinco beneficios reales y su mecanismo	Sin mecanismo identificable, el beneficio es marketing
2	Usos por sector: tracción vs. promesa	Dónde hay resultados operativos y dónde solo pilotos
3	Casos de éxito documentados	Liquidación institucional, bonos digitales, fondos tokenizados, remesas
4	Fracasos instructivos	TradeLens, Libra/Diem, ASX: la tecnología rara vez es la causa
5	Tokenización de activos reales (RWA)	La tendencia 2024-2026 con adopción institucional medible
6	El mapa de servicios del mercado	Custodia, RPC, KYT, auditoría: qué se compra y a quién
7	Costos asociados del proyecto	Partidas completas, no solo el desarrollo
8	Consideraciones: regulación, reputación, gobernanza	Lo que decide si el proyecto vive tras el piloto
9	Cómo explicarlo a clientes y no técnicos	El mejor caso de negocio muere si el directorio no lo entiende

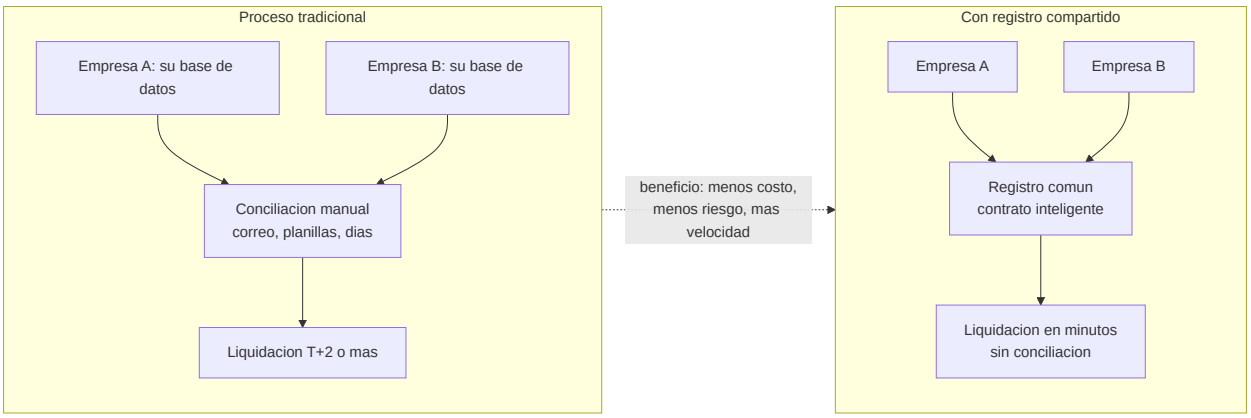
🧠 Modelo mental

Piensa en la blockchain empresarial como un **notario compartido y programable entre organizaciones que no se tienen plena confianza**. Donde hoy dos empresas concilian planillas por correo durante días, un registro común liquidado en minutos elimina la conciliación: ese es el mecanismo del beneficio, no la palabra "blockchain".

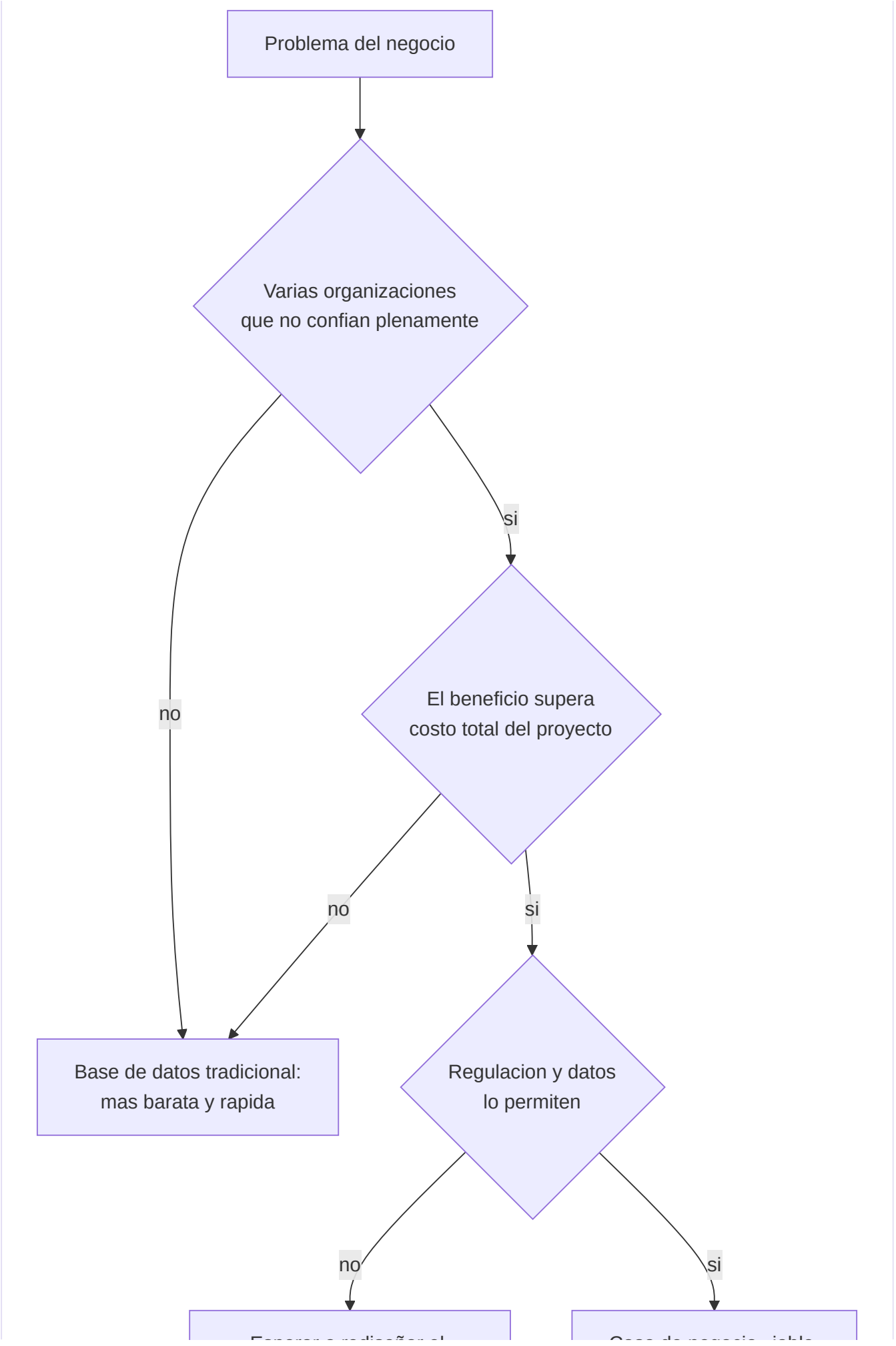
El límite de la analogía: un notario da fe, pero no ejecuta; los contratos inteligentes además **ejecutan** (liquidan, reparten, bloquean), y eso convierte errores de negocio en pérdidas automáticas — por eso los costos de auditoría y límites operativos son parte del caso de negocio, no un extra.

🧩 Esquema visual

Dónde aparece el beneficio: el antes y el después de una liquidación entre empresas.



Flujo de decisión del caso de negocio:


Problema del negocio
Varias organizaciones
que no confian plenamente
no
si
El beneficio supera
costo total del proyecto
no
si
Base de datos tradicional:
mas barata y rapida
Regulacion y datos
lo permiten
no
si

Esperar o rediseñar el alcance

Caso de negocio viable: elegir red y servicios

📖 Conceptos y definiciones

- **Conciliación**: proceso de cuadrar registros entre organizaciones; su eliminación es el beneficio empresarial más medible.
- **Liquidación atómica (DvP)**: entrega contra pago en una sola transacción; elimina el riesgo de contraparte del intervalo.
- **Tokenización / RWA**: representar activos del mundo real (fondos, bonos, facturas) como tokens con reglas programadas.
- **Stablecoin**: token anclado a moneda fiat; el vehículo de pagos y remesas con más tracción del ecosistema.
- **Custodio regulado**: empresa licenciada que guarda claves por terceros (Fireblocks, BitGo, bancos custodios).
- **KYT (Know Your Transaction)**: análisis de transacciones contra listas y patrones ilícitos; obligación según jurisdicción.
- **Red permisionada**: participantes autorizados (Fabric, Besu, Corda); privacidad a cambio de gobernanza de consorcio.
- **Piloto vs. producción**: el piloto prueba tecnología; producción exige modelo de negocio, gobernanza y cumplimiento — la mayoría de los fracasos ocurre en ese salto.
- **Costo total (TCO)**: equipo + auditoría + infraestructura + servicios + cumplimiento + operación, no solo el desarrollo.

🔬 Profundización

Qué ganan las empresas: beneficio → mecanismo → evidencia

Beneficio	Mecanismo	Caso que lo evidencia
Liquidación más rápida y barata	Registro común elimina conciliación; DvP atómico	Pagos institucionales tipo JPMorgan Kinexys; liquidación de stablecoins de Visa
Nuevos productos financieros	Activos programables 24/7 con distribución global	Fondo tokenizado BUIDL de BlackRock; bonos digitales del BEI y Siemens
Menos riesgo de contraparte	El contrato ejecuta; nadie custodia unilateralmente el intervalo	DvP en bonos digitales
Auditabilidad compartida	Historial verificable por todas las partes y el regulador	Reportes sobre registros comunes
Acceso a rieles globales de pago	Stablecoins como liquidación transfronteriza en minutos	Corredores de remesas en LatAm; verifica volúmenes en vivo

Casos de estudio: éxito y fracaso, con lección

Caso	Resultado	Lección
Kinexys (JPMorgan) — pagos y repo intradía	En producción con volumen institucional	Empezó por un problema interno medible: liquidez intradía
BEI / Siemens — bonos digitales	Emisiones reales liquidadas on-chain	El regulador participó desde el diseño, no al final
BlackRock BUIDL — fondo tokenizado	Adopción institucional verificable	RWA gana cuando el activo ya es financiero y el beneficio es distribución/liquidez
Remesas con stablecoins (LatAm)	Tracción real en corredores caros	El beneficio (costo y velocidad) es visible para el usuario final
TradeLens (IBM/Maersk) — supply chain	Cerrado en 2023	La tecnología funcionó; falló la gobernanza: los competidores no querían la plataforma del rival
Libra/Diem (Meta)	Cancelado por presión regulatoria	Sin viabilidad regulatoria no hay proyecto, por grande que sea el patrocinador
ASX CHESS (Australia) — post-trade	Reemplazo abandonado en 2022 tras años	Migrar un sistema crítico nacional exige gestión de proyecto impecable, no solo DLT

El mapa de servicios: qué se contrata y a quién

Servicio	Qué resuelve	Ejemplos	Cuándo contratarlo
Nodo/RPC gestionado	Acceso a la red sin operar nodos	Alchemy, Infura, QuickNode	Siempre al inicio; nodo propio al crecer (módulo 16)
Custodia / MPC	Claves institucionales con póliza y licencia	Fireblocks, BitGo, custodios bancarios	Cuando hay fondos de terceros o tesorería relevante
KYT / analítica	Cumplimiento y monitoreo de fondos	Chainalysis, TRM, Elliptic	Obligatorio según actividad y jurisdicción
Auditoría de contratos	Revisión externa pre-lanzamiento	Firmas especializadas + contests	Siempre antes de mainnet; se agenda con meses
Tokenización como servicio	Emisión regulada de RWA	Securitize y equivalentes locales	Cuando el activo exige registro regulado
Rollup/red como servicio	Cadena propia sin equipo de protocolo	Conduit, Caldera y similares	Casos que justifican appchain (revisa el módulo 12)

Costos asociados: el presupuesto completo

Partidas para un proyecto mediano de 6 meses (órdenes de magnitud del mercado — **consulta precios en vivo**): el **equipo** (6-8 personas) domina el costo; **auditoría externa** 30.000-150.000+ USD según alcance; **infraestructura** 500-5.000+ USD/mes (módulo 16); **custodia** fijo mensual + variable; **KYT/cumplimiento** suscripción anual; **gas** marginal en L2 post-EIP-4844, relevante en L1. El error clásico: presupuestar solo el desarrollo y descubrir auditoría y cumplimiento a mitad de camino.

El caso de negocio, con la cuenta hecha

Un comité no aprueba "innovación": aprueba un número. Hagamos la cuenta de un caso concreto —**conciliación de facturas entre una empresa y sus 40 proveedores**— porque el método sirve para cualquier otro.

Paso 1 — cuantificar el dolor actual. Sin esto, no hay conversación:

```
40 proveedores × 300 facturas/mes    = 12 000 facturas
Discrepancias que exigen intervención humana (3 %)  =    360 casos/mes
Tiempo medio por caso                =     25 min
                                       ──────────────
                                        150 h/mes  ≈  1 persona a tiempo completo
Coste cargado (~35 €/h)              ≈  5 250 €/mes  ≈  63 000 €/año
```

Paso 2 — el coste total, no el del desarrollo. Aquí es donde mueren los casos mal hechos:

Partida	Año 1	Recurrente
Desarrollo (contratos + integración)	120 000 €	—
Auditoría de seguridad	40 000 €	15 000 €/año
Infraestructura (nodos, RPC, monitorización)	12 000 €	12 000 €/año
Cumplimiento y asesoría legal	25 000 €	10 000 €/año
Operación (parte de una persona)	30 000 €	30 000 €/año
Total	**227 000 €**	**67 000 €/año**

Paso 3 — la comparación honesta. El ahorro son 63 000 €/año y el coste recurrente 67 000 €/año. **El caso no se sostiene**: nunca se amortizan los 227 000 € iniciales, porque ni siquiera cubre su propia operación.

Y esa es la conclusión correcta. Un análisis que siempre dice que sí no es un análisis.

Qué habría cambiado el resultado:

- **Más volumen.** Con 400 proveedores en vez de 40, el ahorro sube a ~630 000 €/año y el coste apenas se mueve. Los proyectos de conciliación se justifican por escala.
- **Un beneficio adicional cuantificable**, como reducir el plazo de pago y con ello el capital circulante inmovilizado.
- **Una alternativa más barata que funcione igual.** Y aquí la pregunta incómoda: una API compartida con registros firmados y un tercero neutral resolvería buena parte del problema por una fracción del coste. Si la respuesta es que sí, el caso honesto es *no usar blockchain*.

> 💡 **En una frase:** el caso de negocio no lo decide la tecnología, lo decide la escala. Y un análisis serio tiene que poder terminar en "no".

▶ 🎓 **Si ya dominas esto** — lo que decide en un comité real

🗣️ Cómo explicarlo a clientes y personas que no conocen el tema

El mejor caso de negocio muere si la contraparte no lo entiende. Reglas de comunicación probadas en el sector (versión ampliada y para imprimir en la guía de comunicación):

El discurso de 30 segundos

> "Hoy, cuando su empresa y sus socios mueven [dinero/activos/registros], cada uno lleva su propia planilla y cuadrarlas toma días y errores. Esto es un **registro único compartido que nadie puede alterar por su cuenta** y que ejecuta las reglas que todos acordaron, automáticamente. Resultado: lo que hoy tarda días, tarda minutos, y nadie discute los números."

Ni una sola vez dice "blockchain", "cripto", "token" ni "descentralización". Primero el problema y el resultado; la tecnología después, y solo si la preguntan.

Traducción de jerga: qué decir en lugar de qué

No digas	Di	Por qué funciona
"Blockchain / DLT"	"un registro compartido que nadie puede alterar solo"	Describe la propiedad, no la tecnología
"Smart contract"	"reglas acordadas que se ejecutan solas"	El valor es la automatización sin árbitro
"Token / tokenizar"	"una representación digital del activo, transferible en minutos"	Ancla al activo que ya conocen
"Wallet / llaves privadas"	"su firma digital; la custodia se gestiona como en un banco"	Evita el pánico de la seed phrase
"Gas / L2 / rollup"	"costo por operación, hoy de centavos"	El detalle técnico no aporta a la decisión
"Descentralizado"	"sin depender de un intermediario único"	Beneficio concreto en vez de ideología

Analogías que funcionan (y su límite)

- **Libro de contabilidad notariado entre empresas**: útil para conciliación; aclara que además *ejecuta* (el notario no ejecuta).
- **El contenedor marítimo**: un estándar común que abarata mover valor entre organizaciones, como el contenedor abarató mover carga.
- **Correo electrónico del dinero**: transferir valor como se transfiere un correo, sin oficinas de por medio; aclara que aquí no hay "deshacer envío" — por eso los límites y controles.

Manejo de objeciones frecuentes

Objeción del cliente	Respuesta honesta
"¿Esto no es lo de las criptos que se desplomaron?"	"Eso fue especulación con activos volátiles. Esto usa la misma infraestructura para un problema operativo suyo: conciliación y liquidación. No compramos criptomonedas."
"¿Es legal?"	"Regulado y en uso por bancos centrales y gestoras como BlackRock. En Chile lo cubre la Ley Fintech; el diseño incluye cumplimiento desde el día uno."
"¿Y si se cae o nos hackean?"	"Los riesgos existen y se gestionan con límites, custodia profesional y auditorías; aquí está el plan y su costo — está dentro del presupuesto que le presento."
"¿Por qué no una base de datos?"	"Si una sola organización controlara los datos, una base de datos sería mejor y más barata. Aquí participan varias que no comparten dueño: ese es exactamente el caso donde esto gana."

Regla final: **nunca prometas rentabilidad ni uses el precio de las criptomonedas como argumento** — destruye la credibilidad del caso operativo y puede tener implicancias legales.

🧪 Laboratorio guiado

> 🧪 Estas prácticas están catalogadas y **resueltas paso a paso** en el catálogo de laboratorios.

Análisis de caso con plantilla, sobre fuentes primarias:

1. Elige un caso de la tabla (éxito o fracaso) y localiza dos fuentes primarias (anuncio oficial, informe del regulador, documentación del producto).
2. Completa la ficha: problema de negocio → mecanismo del beneficio → evidencia medible → costos visibles → riesgos y consideraciones → estado actual (verifícalo en vivo).
3. Aplica el flujo de decisión del esquema visual: ¿habrías aprobado el proyecto con la información de su época?
4. Contrasta con la matriz del módulo 00: ¿este caso necesitaba blockchain o una base de datos?
5. Presenta la ficha en una página, como si fuera para un comité de inversión.

📝 Reto verificable

Elabora el **caso de negocio** de una empresa real o ficticia de tu sector que evalúa blockchain: beneficio con mecanismo y cifra estimada, caso comparable citado, presupuesto completo por partidas (con fuentes y fecha de consulta), servicios a contratar vs. construir, tres riesgos con mitigación y la alternativa sin blockchain evaluada honestamente.

Acompáñalo del **discurso de 30 segundos** adaptado a ese cliente, sin jerga.

Criterio de aceptación: el documento cabe en dos páginas; cada cifra tiene fuente y fecha; incluye al menos un caso comparable de éxito y uno de fracaso con su lección; la alternativa "base de datos tradicional" tiene análisis real, no de paja; el discurso no usa ninguno de los términos de la columna "No digas".

⚠️ Errores frecuentes

Síntoma	Causa y cómo comprobarlo
"El beneficio es que es blockchain"	No hay mecanismo identificado; exige la cadena beneficio→mecanismo→evidencia
El piloto muere al pasar a producción	Faltó modelo de negocio y gobernanza (lección TradeLens); revisa quién paga y quién manda
Presupuesto sin auditoría ni cumplimiento	Solo se costeó desarrollo; contrasta contra la tabla de partidas completa
"Los usuarios manejarán sus wallets"	En retail/banca la custodia suele ser de la empresa; valida con usuarios reales
Caso construido sobre cifras de marketing	TVL o TPS sin fuente; exige métricas verificables y fecha
Ignorar al regulador hasta el final	Lección Libra/Diem; incorpóralo desde el diseño (en Chile: Ley Fintech 21.521 y CMF)

🛡️ Seguridad y ética

- No presentes proyecciones de retorno como garantías: este programa no da asesoría financiera y un caso de negocio honesto declara sus supuestos.
- Cita las fuentes de cada caso; los números de casos reales cambian — verifica el estado actual antes de usarlos en una decisión.
- Considera el impacto en usuarios finales: custodia, privacidad de datos transaccionales y reversibilidad de errores son decisiones éticas, no solo técnicas.
- El cumplimiento (KYT, reportes) protege a la empresa y al ecosistema; diseñarlo desde el inicio es más barato que remediarlo.

🔗 Referencias

- Werbach, *The Blockchain and the New Architecture of Trust* (MIT Press).
- BIS — informes sobre tokenización y dinero digital: https://www.bis.org/
- World Economic Forum — informes de adopción blockchain: https://www.weforum.org/
- MiCA — Reglamento (UE) 2023/1114: https://eur-lex.europa.eu/eli/reg/2023/1114/oj
- Chile — regulación y tributación en el programa: documento local
- Lectura extendida del programa: Industria · Blockchain para empresas y Modelos de negocio

✅ Criterio de dominio

- Explicas beneficio→mecanismo→evidencia de al menos tres casos reales sin recurrir a marketing.
- Presupuestas un proyecto con todas las partidas y fuentes citadas.
- Extraes y aplicas la lección de al menos dos fracasos documentados.

🧭 Navegación

⬅️ Módulo 16 · Infraestructura y operación de nodos · 📚 Índice del currículo · ➡️ Módulo 18 · Implementación empresarial

18 · Implementación empresarial end-to-end

> **Nivel:** Avanzado-Producción · ⏱️ **Duración estimada:** 180 min · **Fuente:** prácticas públicas de integración del sector financiero y documentación de los componentes citados
⬅️ Currículo · 📚 Bibliografía 🧭 ⬅️ **Anterior:** 17 · Blockchain en la empresa: valor, casos y costos · 📚 Índice · ➡️ **Siguiente:** 🎓 Proyecto final 📖 Glosario de términos · 🌱 ¿Nuevo en esto? Empieza aquí

El módulo final antes del capstone responde la pregunta que los diagramas conceptuales esquivan: **¿cómo se conecta esto con el ERP, el core bancario y los sistemas que la empresa ya tiene?** Con qué piezas, en qué ambientes, con qué equipo, en cuántos meses — y lo ensamblas en miniatura, ejecutable, con las piezas reales de este repositorio.

🎯 Objetivos

- Diseñar la arquitectura de siete capas de una solución empresarial sobre blockchain.
- Decidir componente a componente entre construir y comprar, con criterios explícitos.
- Planificar los cuatro ambientes (desarrollo, QA, pre-producción, producción) y qué valida cada uno.
- Estructurar un proyecto realista de seis meses con entregables verificables por fase.
- Ensamblar el patrón completo en local: contrato, firma, indexador e interfaz.

📚 Resultados de aprendizaje

Al finalizar, el estudiante podrá:

1. **Dibujar** la arquitectura completa, del usuario al bloque, ubicando cada componente y su rol.
2. **Justificar** cada decisión build vs. buy con volumen, riesgo y costo.
3. **Explicar** por qué la blockchain nunca es la base de datos de lectura de las pantallas.
4. **Planificar** ambientes y ceremonias para que el día del lanzamiento no haya "primeras veces".
5. **Ejecutar** el pipeline local completo del repositorio como maqueta de la arquitectura.
6. **Operar** con separación de deberes: quién despliega, quién firma, quién monitorea.

🗺️ Temas

#	Tema	Por qué importa
1	Las siete capas de la solución	Solo una es la blockchain; el resto es ingeniería de sistemas
2	El middleware: construir, simular, firmar	El corazón que aplica negocio y compliance antes de tocar la red
3	Firma y custodia: KMS, MPC, multisig	Dónde viven las claves y quién puede usarlas
4	Indexador y base de lectura	Las pantallas no leen de la cadena
5	Build vs. buy por componente	Se construye lo que diferencia, se compra el commodity
6	Ambientes y ceremonias	La primera firma multisig no puede ser el día del lanzamiento
7	El plan de seis meses	Fases con entregables verificables y equipo realista
8	Operación segura: límites, runbooks, post-mortems	Un bug con límites es un incidente; sin límites, una catástrofe

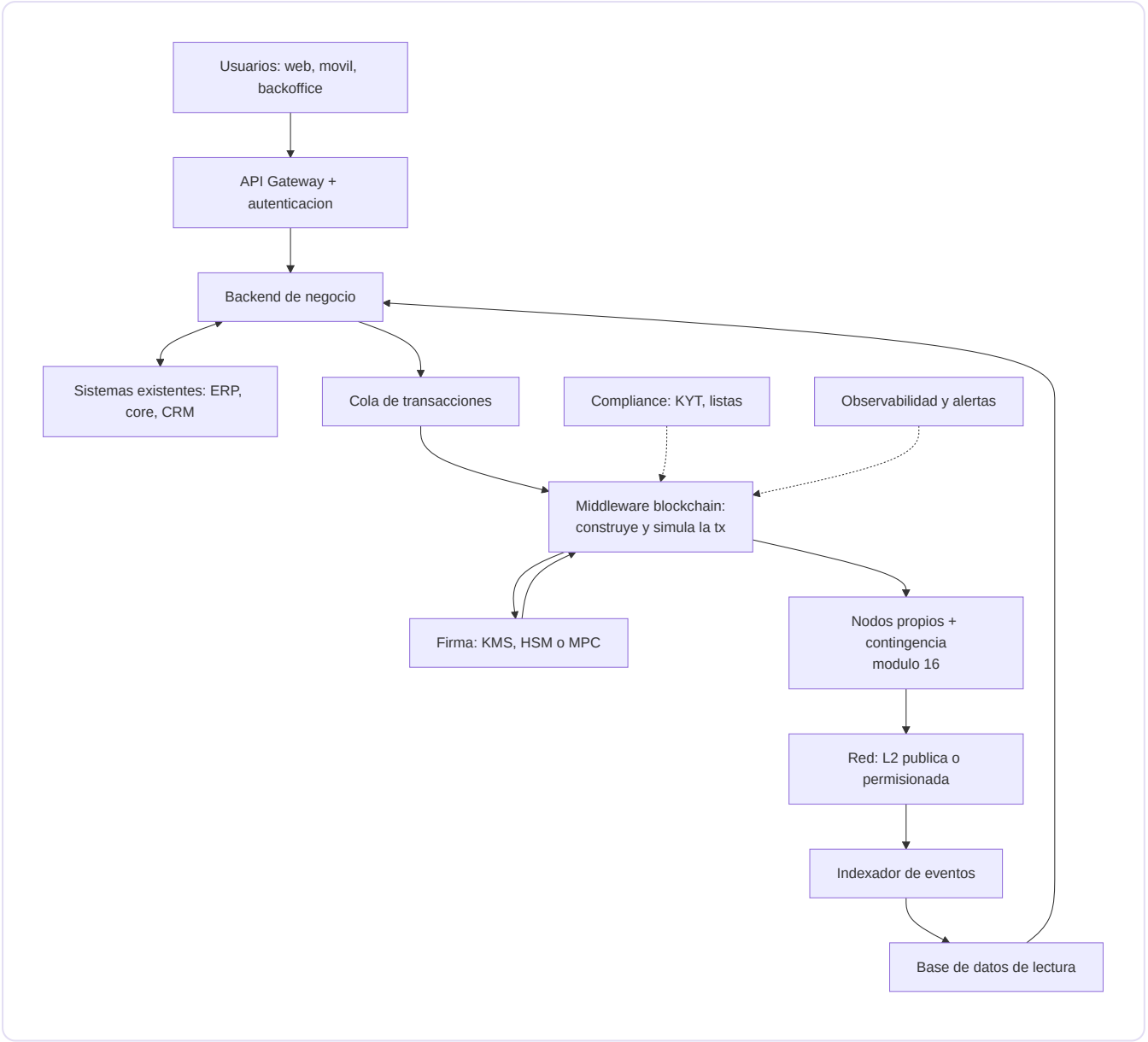
🧠 Modelo mental

La solución empresarial es un **aeropuerto**: la pista (la blockchain) es indispensable pero pequeña comparada con todo lo demás — torre de control (middleware), aduana (compliance), mangas y cintas (integraciones con el ERP), seguridad (firma y custodia) y paneles de vuelos (indexador y pantallas). Los pasajeros nunca pisan la pista: usan la terminal. Tus usuarios quizá nunca vean una wallet: usan la aplicación de siempre.

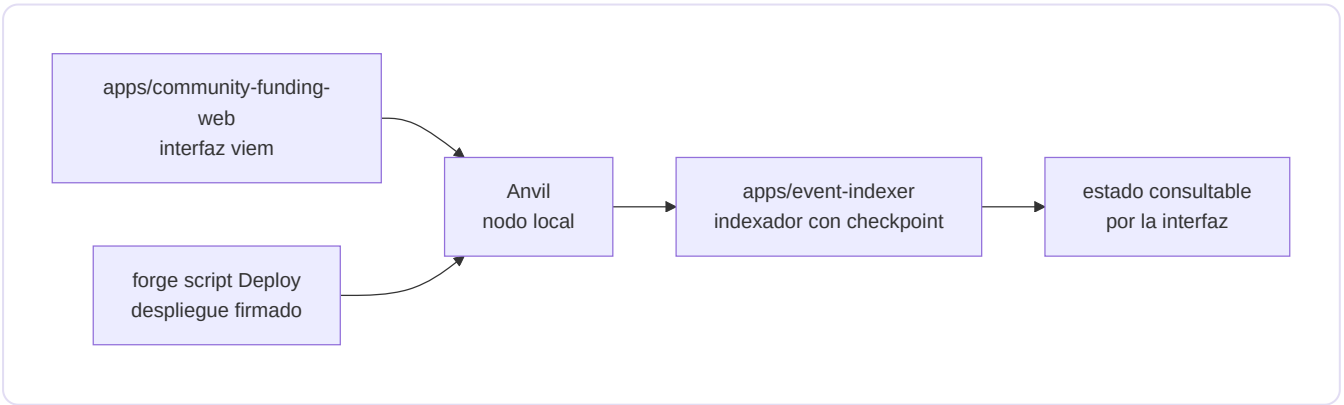
Límite de la analogía: en el aeropuerto un vuelo se puede cancelar; en la cadena, lo despegado despegó — por eso simulación previa, límites y timelocks son parte del diseño, no mejoras posteriores.

🧩 Esquema visual

La arquitectura completa, de la pantalla al bloque:



El mismo patrón, en miniatura ejecutable, con las piezas de este repositorio:



📖 Conceptos y definiciones

- **Middleware blockchain**: servicio interno que construye transacciones, las **simula** (`eth_call`), aplica reglas de negocio/compliance y solo entonces solicita la firma.

- **Cola de transacciones**: desacopla el negocio de la red; si el gas se dispara o la red se congestiona, las operaciones esperan con política de reintento.
- **Servicio de firma**: KMS/HSM (clave en hardware con política y auditoría), MPC (clave repartida entre máquinas) o multisig on-chain (política en el contrato).
- **Base de datos de lectura**: réplica consultable del estado, alimentada por el indexador; las pantallas leen de aquí, jamás de un `eth_call` por clic.
- **Guarded launch**: lanzamiento con límites (caps por transacción, por día, por contrato) que se amplían con evidencia de operación.
- **Ceremonia**: procedimiento presencial documentado (generación de claves, alta de firmante) con actas, testigos y respaldo físico.
- **Separación de deberes**: quien despliega no firma; ninguna persona sola mueve fondos (política M-de-N).
- **Runbook**: guion paso a paso de un incidente (pausar, contactar custodio, comunicar), ensayado antes de producción.

🔬 Profundización

Build vs. buy, componente a componente

Componente	Construir	Comprar	Criterio
Nodos / RPC	Flota propia (módulo 16)	Alchemy, Infura, QuickNode	Volumen, privacidad, SLA
Firma / custodia	HSM propio + política	Fireblocks, BitGo, custodio regulado	Licencias, monto, seguro
Indexación	Indexador propio (como el del repo)	The Graph, proveedores de datos	Complejidad y latencia
Contratos	Equipo propio + auditoría	Plantillas auditadas (OpenZeppelin)	Cuán estándar es el caso
Compliance / KYT	Integración propia	Chainalysis, TRM, Elliptic	Obligación regulatoria
Monitoreo on-chain	Alertas propias	Tenderly, Defender, Forta	Minutos de reacción requeridos

Ambientes: qué valida cada uno

Ambiente	Red	Claves	Qué se valida
Desarrollo	Anvil local	de prueba, conocidas	Lógica y flujo end-to-end
Integración / QA	Testnet (Sepolia)	de prueba gestionadas	Integración con ERP, indexador, colas
Pre-producción	Testnet + fork de mainnet	estructura real, sin fondos	Ceremonias, runbooks, límites
Producción	Mainnet / L2 / permisionada	KMS/MPC/multisig reales	Lanzamiento acotado con monitoreo

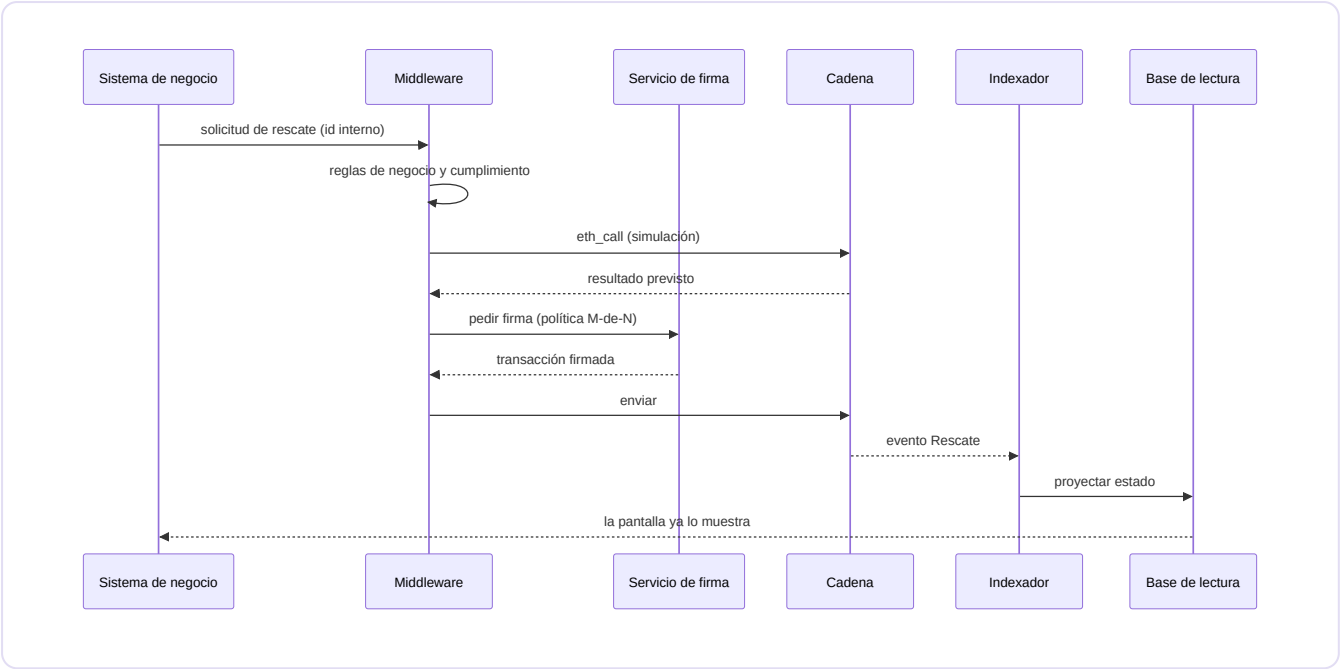
El plan de seis meses (equipo: arquitecto, 2 de contratos, 2 backend, DevOps, PM)

Fase	Semanas	Entregable verificable
1 · Descubrimiento	1-4	Matriz del módulo 00 respondida con evidencia; elección de red; ADRs
2 · Diseño	5-8	Spec con invariantes, threat model, plan de custodia e integración
3 · Construcción	9-16	Contratos probados y fuzzeados; middleware + firma; todo en testnet
4 · Endurecimiento	17-20	Auditoría externa, correcciones verificadas, pre-producción completa
5 · Lanzamiento acotado	21-24	Mainnet con caps, monitoreo y runbook ensayado
6 · Operación	25+	Ampliación gradual de límites, post-mortems, métricas de negocio

Las fases 1-2 son las más baratas y las más determinantes: los fracasos del módulo 17 (TradeLens, ASX) se gestaron ahí, no en el código.

Una operación real, paso a paso, y dónde se rompe cada una

La lista de comprobación dice "usa un middleware". Aquí se ve **qué pasa exactamente si no lo usas**, siguiendo una operación de negocio corriente: *un cliente solicita el rescate de 10 000 unidades de un fondo tokenizado*.



Y ahora el mismo recorrido, con lo que falla al saltarse cada paso:

Paso	Qué aporta	Si te lo saltas
Reglas antes de firmar	Aplica límites, listas y horarios en un solo sitio	La regla acaba duplicada en cada pantalla que puede iniciar la operación, y basta olvidarla en una para que se escape
Simulación (`eth_call`)	Predice el resultado sin gastar	Se firman transacciones que revierten: se paga el gas, no hay efecto, y el usuario ve un cobro sin resultado
Cola de transacciones	Desacopla el negocio de la red	Si el gas se dispara o el RPC cae, la petición del cliente falla en su cara en vez de esperar y reintentar
Servicio de firma con política	Ninguna persona sola mueve fondos	La clave acaba en una variable de entorno de un servidor, y quien acceda a ese servidor es dueño del fondo
Indexador + base de lectura	Las pantallas leen a velocidad de base de datos	Cada clic dispara un `eth_call`; la interfaz va lenta y se cae entera cuando cae el proveedor RPC
Idempotencia por id interno	La misma solicitud no se ejecuta dos veces	Un reintento tras un timeout ejecuta el rescate **dos veces**. La cadena no tiene forma de saber que era el mismo

El último es el más caro y el menos citado. En un sistema clásico, un doble apunte se corrige con un asiento inverso. Aquí, la segunda transferencia es tan válida y tan definitiva como la primera: **la corrección exige que la contraparte colabore**.

> 💡 **En una frase:** el middleware no es una capa de arquitectura por elegancia — es el sitio donde se decide, se simula y se registra *una sola vez* lo que en la cadena no se

> puede deshacer.

Los números de un lanzamiento con límites

"Guarded launch" suena a consigna hasta que se le ponen cifras. La idea es simple: **acotar cuánto puedes perder mientras el sistema todavía no tiene historial**, y ampliar los límites con evidencia, no con optimismo.

Fase	Duración típica	Tope por operación	Tope diario	Qué la cierra
Piloto interno	2–4 semanas	1 000	5 000	Cero incidentes y el runbook ensayado de verdad
Clientes seleccionados	4–8 semanas	10 000	100 000	Volumen real sostenido sin intervención manual
Apertura	—	según negocio	según negocio	Auditoría cerrada y monitorización con alertas probadas

Dos reglas que hacen que esto funcione y sin las cuales es teatro:

1. **Los topes viven en el contrato, no en la interfaz.** Un límite en la pantalla lo esquiva cualquiera que llame al contrato directamente. Si el tope no está en el código, no es un tope: es una sugerencia.
2. **Ampliar exige evidencia escrita**, no la sensación de que va bien. Fija de antemano qué métrica autoriza el siguiente escalón (operaciones sin incidente, tiempo desde el último fallo, cobertura de la auditoría) para que la decisión no dependa de la presión comercial del momento.

El cálculo que convence a un comité: si el tope diario es 100 000 y el peor caso es un fallo que drena el máximo antes de que alguien reaccione, **la pérdida máxima está acotada a esa cifra**. Sin topes, la respuesta a "¿cuánto podemos perder?" es "todo lo que haya en el contrato", y esa respuesta no se puede llevar a un consejo.

▶ 🎓 **Si ya dominas esto** — lo que se descubre en el primer incidente

🧪 Laboratorio guiado

> 🧪 Estas prácticas están catalogadas y **resueltas paso a paso** en el catálogo de laboratorios.

Ensambla la arquitectura de referencia **en miniatura y ejecutable**, con las piezas reales del repositorio (guía detallada en despliegue local):

1. Levanta el "nodo de la empresa" (capa de acceso a red):

```
anvil --chain-id 31337
```

2. Despliega el contrato con un script firmado (capa de contratos + firma):

```
cd projects/community-funding
forge script script/Deploy.s.sol:DeployCommunityFunding \
  --rpc-url "$RPC_URL" --broadcast
```

3. Arranca el indexador y verifica su checkpoint (capa de datos de lectura):

```
pnpm test:indexer
```

4. Construye la interfaz y conéctala al contrato (capa de canal):

```
pnpm build:web
```

5. Recorre el diagrama de siete capas y marca qué pieza del repo cumple cada rol — y cuáles faltan para producción real (cola, KYT, HSM): esa brecha es exactamente lo que separa la maqueta de un despliegue empresarial.

📝 Reto verificable

Escribe el **documento de arquitectura** de una implementación empresarial para el caso de negocio que construiste en el módulo 17: diagrama de siete capas adaptado, tabla build vs. buy con justificación por componente, plan de ambientes con qué valida cada uno, plan de fases con entregables, y la operación segura (límites iniciales, política de firma M-de-N, tres runbooks nombrados).

Criterio de aceptación: el diagrama ubica firma e indexador correctamente (la firma antes del RPC; las pantallas leen de la base de lectura); cada "comprar" cita al menos dos proveedores; el plan no tiene "primeras veces" en producción; los límites del lanzamiento tienen números concretos.

⚠️ Errores frecuentes

Síntoma	Causa y cómo comprobarlo
"La blockchain reemplazará nuestra base de datos"	Conviven: estado compartido on-chain, lectura y negocio en sistemas clásicos; revisa el diagrama
Pantallas lentas que "leen de la cadena"	Falta indexador + base de lectura; ningún `eth_call` por clic de usuario
El lanzamiento se retrasa por la auditoría	Se agendó tarde; las firmas serias se reservan desde la fase de diseño
La primera firma multisig falla en producción	No hubo pre-producción con ceremonia ensayada
Un bug drena más de lo tolerable	Sin caps de guarded launch; los límites se definen antes de mainnet
"Multi-región lo vemos después"	La contingencia RPC y la réplica se diseñan el día uno (módulo 16)

🛡️ Seguridad y ética

- Separación de deberes real: quien despliega no es quien firma; documenta la política M-de-N y pruébala.
- Los firmadores nunca son accesibles desde internet; red privada, mTLS y bastion con MFA.
- Límites como airbag: caps por transacción y por día desde el día uno, ampliados solo con evidencia.
- Comunicación honesta de incidentes: pausar, informar y publicar post-mortem — el runbook del programa es el punto de partida.
- En el laboratorio, todo es local y con claves de prueba; ninguna clave real sale jamás de su HSM/MPC.

🔗 Referencias

- OpenZeppelin — Contracts y Defender: https://docs.openzeppelin.com/
- Safe — multisig para tesorerías: https://docs.safe.global/
- AWS KMS — firma con secp256k1: https://docs.aws.amazon.com/kms/
- Fireblocks — arquitectura MPC: https://www.fireblocks.com/platforms/mpc-wallet/
- Trail of Bits — *Building Secure Contracts*: https://secure-contracts.com/
- BIS — tokenización y liquidación institucional: https://www.bis.org/
- Lectura extendida del programa: Industria · Ciclo de vida de un proyecto

✅ Criterio de dominio

- Dibujas y defiendes la arquitectura de siete capas ubicando firma, indexador y contingencia.
- Ejecutas la maqueta local completa y nombras qué le falta para producción real.
- Tu plan de proyecto no contiene "primeras veces" en producción.

🧭 Navegación

⬅️ Módulo 17 · Blockchain en la empresa · 📚 Índice del currículo · ➡️ 🎓 Proyecto final

Industria

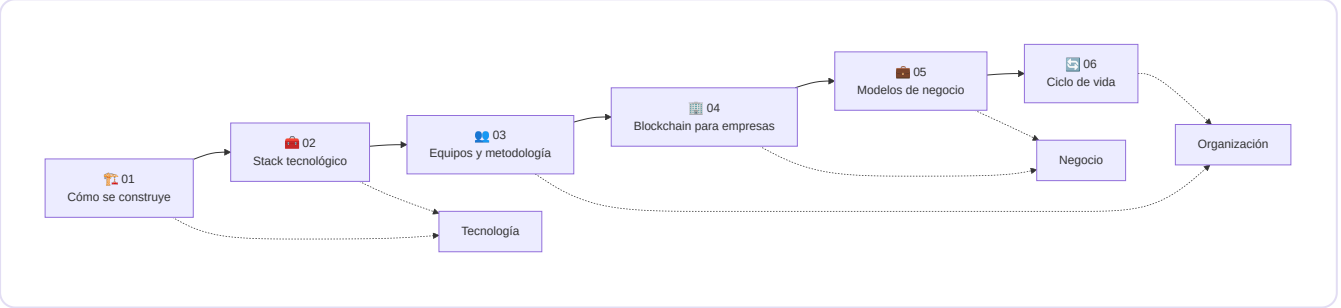
🏭 Industria: blockchain como profesión y negocio

🏠 Volver al programa · 📚 Currículo · 📖 Bibliografía

El currículo enseña **cómo funciona** la tecnología; esta sección enseña **cómo funciona la industria**: cómo se construye una blockchain en la práctica, con qué stack, cómo se organizan y comunican los equipos, qué valor real aporta a las empresas, cómo se gana dinero en el ecosistema y cómo se lleva un proyecto de la idea a producción.

Los seis documentos son de nivel profesional: casos reales verificables —incluidos los **fracasos instructivos**—, cifras marcadas para consultar en vivo y fuentes primarias.

Mapa de la sección



Documentos

#	Documento	Pregunta que responde	Audiencia
01	Cómo se construye una blockchain	¿Qué hay dentro de un nodo y qué vías existen para lanzar una red?	Ingeniería y arquitectura
02	Stack tecnológico	¿Qué tecnologías usa el ecosistema en cada capa y cómo elegirlas?	Desarrollo y CTO
03	Equipos, roles y metodología	¿Quién hace qué y cómo se coordinan los equipos del sector?	Líderes técnicos, PM y empleo
04	Blockchain para empresas	¿Qué beneficios reales aporta y qué casos funcionaron o fracasaron?	Decisores y consultores
05	Modelos de negocio	¿Cómo se gana dinero de verdad en este ecosistema?	Fundadores y analistas
06	Ciclo de vida de un proyecto	¿Cómo se va de la idea a mainnet y a la operación responsable?	Equipos de producto

Cómo usar esta sección

Esta sección es la **lectura profesional extendida** del programa. La versión con laboratorios y retos verificables vive en el currículo: los módulos 16 · Infraestructura y nodos, 17 · Blockchain en la empresa y 18 · Implementación empresarial.

- **Si vienes del currículo:** léela tras el módulo 09 (Seguridad); da contexto de industria a los módulos avanzados.
- **Si eres decisor o consultor:** empieza por 04 · Blockchain para empresas y 05 · Modelos de negocio.
- **Si buscas empleo en el sector:** 03 · Equipos y roles mapea los puestos y sus habilidades; el currículo te da la ruta técnica.
- **Si vas a lanzar un protocolo:** 06 · Ciclo de vida es tu checklist, junto al capstone.
- **Si tienes que vender o explicar la idea:** la guía para explicar blockchain a personas no técnicas te da el discurso, la traducción de jerga y el manejo de objeciones.

🧭 Navegación

🏠 Programa · 📚 Currículo · ➡️ 01 · Cómo se construye una blockchain

🏗️ Cómo se construye una blockchain

> **Audiencia:** Ingenieros y arquitectos · ⏱️ **Lectura:** 25 min · **Fuentes:** docs de ethereum.org, OP Stack, Cosmos SDK y Polkadot SDK ⬅️ Industria · 🏠 Programa · 📚 Bibliografía

---

🧬 Anatomía de un nodo: las capas que todo el mundo confunde

Una blockchain en producción no es "un programa": es una red de nodos donde cada nodo apila varias capas con responsabilidades separadas. Entenderlas por separado es el primer filtro entre una conversación de marketing y una de ingeniería:

Capa	Responsabilidad	Tecnologías típicas
Red P2P	Descubrimiento de pares, gossip de bloques y transacciones	devp2p (Ethereum ejecución), libp2p (consenso Ethereum, Cosmos, Polkadot)
Consenso	Acordar el orden canónico de bloques y su finalidad	Gasper (PoS Ethereum), CometBFT, BABE/GRANDPA, Snow (Avalanche)
Ejecución	Aplicar transacciones y transicionar el estado	EVM, CosmWasm, runtimes WASM de Polkadot, SVM
Almacenamiento	Persistir estado, historial y pruebas (tries de Merkle-Patricia, etc.)	LevelDB, PebbleDB, MDBX (Erigon/Reth)
RPC / API	Exponer lectura y envío de transacciones a aplicaciones	JSON-RPC, gRPC, WebSockets

Cada capa falla de forma distinta: un bug de P2P aísla nodos, uno de consenso parte la red en dos, y uno de ejecución corrompe el estado. Los post-mortems públicos del ecosistema casi siempre se clasifican en una de estas capas.

🔀 Consenso y ejecución separados: Ethereum post-Merge

Desde The Merge (septiembre de 2022), Ethereum es Proof of Stake y cada nodo completo ejecuta **dos clientes** que se comunican por la Engine API:

- **Clientes de consenso** (beacon chain, atestaciones, finalidad): Prysm, Lighthouse, Teku, Nimbus, Lodestar.
- **Clientes de ejecución** (EVM, mempool, estado): Geth, Nethermind, Besu, Erigon, Reth.

Esta separación no es cosmética. Permitió The Merge sin detener la red, y habilita que cada capa evolucione a su ritmo: Dencun (2024) introdujo blobs con EIP-4844 para abaratar la publicación de datos de rollups, y Pectra (2025) sumó, entre otros, EIP-7702 para mejorar la experiencia de cuentas.

La **diversidad de clientes** es un requisito operativo, no una preferencia estética. Los umbrales que importan:

- Con **>33 %** de la red en un cliente con bug, la red puede perder finalidad temporalmente.
- Con **>50 %**, ese cliente puede imponer una cadena incorrecta como la más larga.
- Con **>66 %**, puede **finalizar** una cadena inválida: revertirla implica slashing masivo de validadores honestos que la atestiguaron, un escenario catastrófico sin salida limpia.

El sitio de referencia para ver la distribución actual es clientdiversity.org (las cifras cambian; consúltalo en vivo antes de citarlas). Para un operador profesional, la lección es directa: elegir cliente minoritario es una contribución activa a la seguridad de la red, y toda infraestructura seria corre al menos dos implementaciones distintas.

🛤️ Las 4 vías reales para "construir una blockchain" en 2025/2026

Casi nadie que dice "construimos una blockchain" escribió consenso o criptografía propios. En la práctica hay cuatro caminos, con perfiles de costo y soberanía muy distintos:

Vía	Qué es	Ejemplos de stack	Soberanía	Costo operativo	Cuándo tiene sentido
(a) Fork / cliente de una L1	Reutilizar el código de una L1 existente con parámetros propios	Forks de Geth/Reth, forks históricos de Bitcoin	Alta	Muy alto: heredas todo el mantenimiento y los upgrades upstream	Casi nunca para productos; útil para investigación
(b) Rollup con un stack	L2 que publica datos/pruebas en una L1	OP Stack, Arbitrum Orbit, ZK Stack, Polygon CDK; RaaS como Conduit o Caldera	Media (dependes de la L1 y del stack)	Medio: sequencer, batcher, infra; el RaaS lo reduce más	Producto que necesita throughput y hereda seguridad de Ethereum
(c) Appchain con SDK	L1 soberana construida con un framework	Cosmos SDK + CometBFT, Polkadot SDK (Substrate), subnets de Avalanche	Alta	Alto: necesitas un conjunto de validadores propio y coordinar upgrades	Lógica que no cabe en un contrato y comunidad capaz de validar
(d) Red permisionada	Red privada entre entidades identificadas	Hyperledger Fabric, Besu (privado), Corda	Total dentro del consorcio	Medio: infra empresarial clásica	Consorcios con requisitos regulatorios y sin token público

Los stacks de rollup redujeron radicalmente la barrera de entrada tras EIP-4844: publicar datos en blobs es órdenes de magnitud más barato que en calldata, aunque el precio de blobs es un mercado propio y fluctúa (consúltalo en vivo). El matiz honesto: la mayoría de los rollups en producción hoy siguen operando con **sequencer centralizado** y ruedas de entrenamiento (training wheels); L2BEAT documenta el estado real de cada uno.

📐 Decisiones de diseño que definen el proyecto

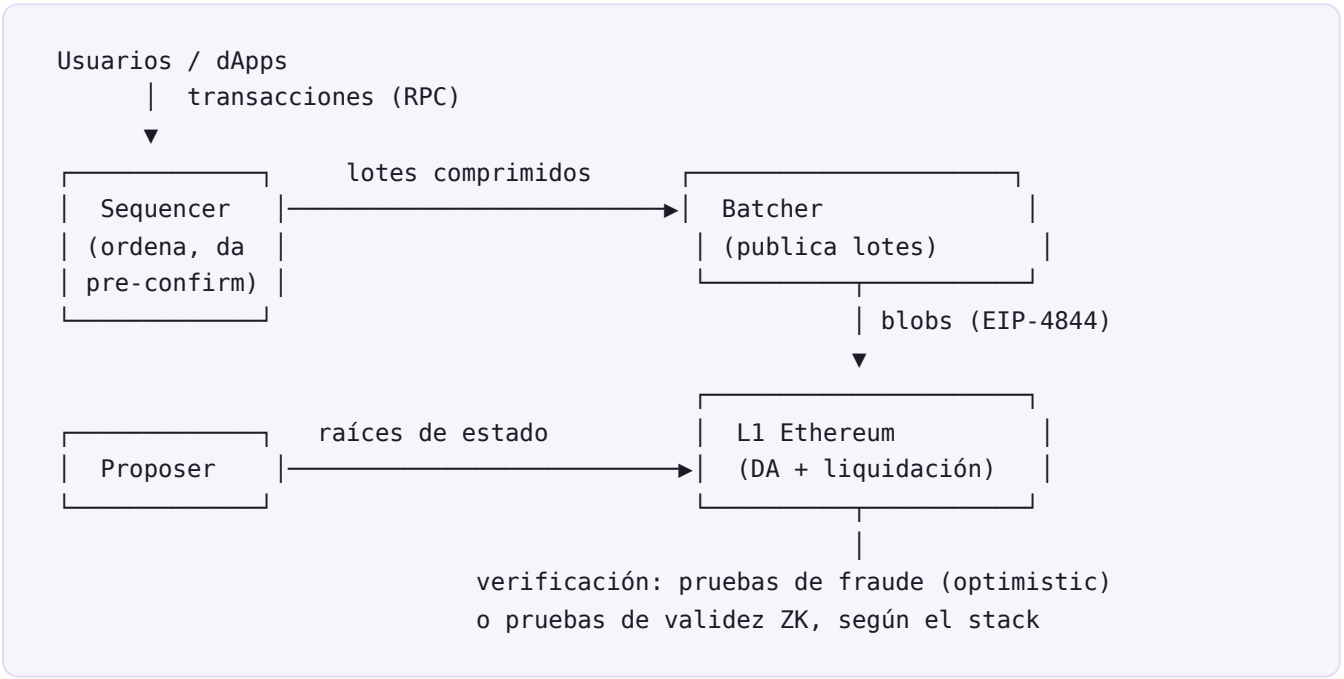
Elegir vía es solo la primera decisión. Estas cinco definen el riesgo y la economía del sistema:

1. **Disponibilidad de datos (DA):** ¿Ethereum blobs, un comité (DAC), o una capa externa como Celestia o EigenDA? Más barato suele significar menos garantías de que los datos estarán disponibles para reconstruir el estado.
2. **Secuenciador:** centralizado (rápido, censurable, punto único de fallo), compartido, o descentralizado (aún inmaduro en producción). ¿Existe una vía de escape (forced inclusion) vía L1?
3. **Gas token:** ¿ETH, el token propio, o abstracción de gas? Un token propio crea tesorería pero añade fricción y riesgo regulatorio.

4. **Gobernanza del protocolo:** ¿quién puede actualizar los contratos del bridge y del rollup? Multisig, timelock, gobernanza por token: cada opción es un vector de ataque distinto.
5. **Bridges:** históricamente la categoría con mayores pérdidas del ecosistema (Ronin y Wormhole en 2022 suman más de mil millones de USD). Minimizar bridges custom es una decisión de seguridad, no de pereza.

Ninguna de estas decisiones es reversible gratis: cambiar de capa de DA o descentralizar un sequencer ya en producción es un proyecto de ingeniería en sí mismo, con migración de usuarios incluida. Documentarlas como ADRs desde el día uno (el formato se practica en el índice de ADRs del programa) evita re-litigar cada decisión con cada incorporación al equipo.

🗺️ Arquitectura de un rollup, en bloque



El flujo completo — de la transacción del usuario a la finalidad en L1 — puede tardar minutos (rollups ZK) o mantener una ventana de disputa de días (optimistic). Las pre-confirmaciones del sequencer son una promesa, no finalidad.

🚫 Qué NO construir uno mismo

Regla de la industria, repetida en todo manual serio (incluido *Mastering Blockchain* de Bashir): **no implementes tu propia criptografía ni tu propio consenso desde cero**.

- Las primitivas criptográficas (curvas, hashes, firmas BLS, sistemas de pruebas ZK) requieren años de revisión académica y auditorías; un error no produce una excepción, produce fondos robados en silencio.

- Los protocolos de consenso tienen espacios de fallo enormes (particiones, equivocación, ataques de largo alcance) que solo emergen bajo adversarios reales.
- Los frameworks citados existen precisamente para que reutilices consenso y criptografía revisados y te concentres en la lógica de aplicación.

El historial de proyectos que ignoraron esta regla es un catálogo de exploits. La innovación defendible casi nunca está en la capa de consenso: está en el producto.

🚀 Del devnet al mainnet: cómo se lanza de verdad

Ningún equipo serio pasa del repositorio al mainnet en un paso. La secuencia estándar del sector, visible en los lanzamientos públicos de los últimos años, es:

1. **Devnet interna:** red efímera controlada por el equipo, se destruye y recrea a diario; sirve para validar la configuración de génesis y los flujos de upgrade.
2. **Testnet pública:** validadores externos, faucet, explorador; aquí aparecen los problemas de coordinación que la devnet nunca muestra (nodos desactualizados, clocks desincronizados, peers maliciosos).
3. **Auditorías y ejercicios de caos:** auditoría de los contratos del bridge y de los módulos custom; simulacros de partición de red y de pérdida de claves antes de que ocurran con fondos reales.
4. **Génesis de mainnet:** ceremonia de génesis (en appchains, coordinación explícita de validadores), límites conservadores iniciales (caps de depósito, pausas activables) y descentralización progresiva documentada.
5. **Operación continua:** el lanzamiento no termina el proyecto; lo convierte en un servicio 24/7 con SLOs, guardias y runbooks (véase operación e incidentes).

Ethereum mismo ensaya cada hard fork en varias testnets (y en shadow forks del mainnet) antes de activarlo; es el estándar de referencia de cómo se coordina un upgrade en una red que no puede detenerse.

💸 Costos operativos reales de mantener una red

El costo de lanzar es bajo; el de operar, no. Presupuesta desde el diseño:

- **Validadores / sequencers:** hardware, hosting redundante, monitoreo 24/7, gestión de claves (HSM o servicios de firma remota). En una appchain, además, incentivos suficientes para que terceros validen.
- **Upgrades coordinados:** cada hard fork exige coordinar a todos los operadores de nodos; en redes pequeñas eso es una llamada, en redes públicas es un proceso de meses (Ethereum lo ejecuta con all-core-devs calls y varias testnets).
- **Infraestructura de soporte:** exploradores de bloques, endpoints RPC públicos, indexadores, faucets de testnet, documentación. Nada de esto viene gratis con el stack.

- **Seguridad continua:** auditorías por cada upgrade, bug bounty permanente, respuesta a incidentes (véase operación e incidentes).
- **El fracaso silencioso más común:** appchains técnicamente correctas que mueren por no poder pagar validadores ni retener un equipo core; el cementerio de cadenas Cosmos y de rollups sin tracción es amplio y bien documentado.

Un orden de magnitud honesto: operar un rollup propio "a mano" exige un equipo de plataforma dedicado; un RaaS lo reduce a una factura mensual más el costo variable de publicar datos en L1 (que depende del mercado de blobs — consúltalo en vivo). Una appchain con validadores externos añade el costo permanente de mantener alineados los incentivos de terceros. Compara siempre contra la línea base: desplegar contratos en una L2 existente, cuyo costo operativo marginal es cercano a cero.

Para practicar estas piezas con las manos, el catálogo de laboratorios del programa incluye ejercicios de nodos y despliegue, y el currículo ordena los prerrequisitos.

⚠️ Errores y mitos frecuentes

Mito o error	Realidad
"Necesitamos nuestra propia blockchain"	En la mayoría de los casos un contrato en una L2 existente resuelve el problema con una fracción del costo y del riesgo.
"Un fork de Geth y listo"	Heredas cada CVE y cada hard fork upstream para siempre; sin equipo de protocolo dedicado, el fork se pudre.
"Los rollups son tan seguros como Ethereum"	Heredan DA y liquidación, pero añaden riesgos propios: sequencer centralizado, multisigs de upgrade, bridges. L2BEAT clasifica estos supuestos por proyecto.
"PoS significa que ya no hay mineros, todo lo demás es igual"	El cambio a PoS reestructuró la arquitectura completa del nodo (dos clientes, Engine API, finalidad económica) y las suposiciones de seguridad.
"Más TPS = mejor blockchain"	El TPS aislado ignora DA, descentralización de validadores y costo de verificación; es la métrica favorita del marketing precisamente porque es fácil de inflar.
"La red permisionada nos da blockchain sin sus problemas"	Si todos los participantes confían en un operador, una base de datos replicada suele ser más simple, barata y rápida.
"Lanzar la cadena es el hito final"	Es el inicio de una operación 24/7: upgrades, guardias, incidentes y costos recurrentes que superan con creces el costo de desarrollo.
"Nuestro consenso propio nos diferencia"	Salvo equipos de investigación con años de revisión por pares, el consenso casero es un pasivo de seguridad, no una ventaja competitiva.

🔗 Referencias

- Nodos y clientes de Ethereum: https://ethereum.org/developers/docs/nodes-and-clients/
- Arquitectura del OP Stack: https://docs.optimism.io/
- Documentación del Cosmos SDK: https://docs.cosmos.network/
- Polkadot SDK (Substrate): https://docs.polkadot.com/
- Diversidad de clientes de Ethereum: https://clientdiversity.org/
- Estado real de los L2 (riesgos y training wheels): https://l2beat.com/
- Imran Bashir, *Mastering Blockchain* (Packt) — capítulos de consenso y arquitectura de nodos.

🧭 Navegación

⬅️ Índice de industria · ➡️ 02 · Stack tecnológico

🧰 El stack tecnológico del ecosistema

> **Audiencia:** Desarrolladores y CTOs · ⏱️ **Lectura:** 25 min · **Fuentes:** documentación oficial de cada herramienta ⬅️ Industria · 🏠 Programa · 📚 Bibliografía

---

🗂️ El stack por capas: mapa completo

El "stack web3" no es una lista de logos: es un conjunto de capas con responsabilidades claras, cada una con dos o tres opciones dominantes y una cola larga de alternativas. Esta tabla es el mapa; las secciones siguientes son el criterio:

Capa	Para qué sirve	Opciones dominantes (2025/2026)	Nota honesta
Protocolo (L1/L2)	Dónde vive y se ejecuta el contrato	Ethereum, L2s (Arbitrum, OP Mainnet, Base), Solana, y appchains	La elección condiciona todo lo demás: lenguaje, tooling, usuarios
Lenguajes de contratos	Escribir la lógica on-chain	Solidity, Vyper (EVM); Rust/Anchor (Solana); Cairo (Starknet); Move (Aptos/Sui)	Solidity concentra la mayoría del empleo y de las auditorías disponibles
Frameworks de desarrollo	Compilar, testear, desplegar	Foundry, Hardhat	Foundry domina en testing/fuzzing; Hardhat mantiene ecosistema de plugins JS
Librerías cliente	Hablar con la cadena desde apps	viem, wagmi, ethers.js (JS/TS); web3.py (Python)	viem/wagmi son el estándar de facto en frontends nuevos; ethers sigue omnipresente en código existente
Wallets y firma	Custodia de claves y aprobación de transacciones	MetaMask, Rabby, Safe (multisig), WalletConnect, hardware wallets (Ledger, Trezor)	Toda tesorería de equipo seria vive en un Safe, no en una EOA
Acceso a nodos / RPC	Leer estado y enviar transacciones	Nodo propio; Alchemy, Infura, QuickNode	Comodidad del proveedor vs. dependencia y privacidad: trade-off real, ver abajo
Indexación y datos	Consultar datos históricos y agregados	The Graph (subgraphs), Dune, indexer propio	Leer eventos vía RPC no escala; la indexación es infraestructura obligatoria
Oráculos	Traer datos externos on-chain	Chainlink, Pyth	Un oráculo mal elegido es la vulnerabilidad completa del protocolo
Almacenamiento	Datos grandes fuera de la cadena	IPFS, Arweave, Filecoin	IPFS no garantiza persistencia sin pinning; Arweave cobra por permanencia
Seguridad y análisis	Encontrar bugs antes que los atacantes	Slither, Echidna, fuzzing de Foundry, librerías OpenZeppelin, auditorías externas	Las herramientas no sustituyen la auditoría; la complementan
Monitoreo y operación	Ver y reaccionar en producción	Tenderly, OpenZeppelin Defender, Forta	Los contratos no emiten logs a un APM: el monitoreo hay que diseñarlo
Testing y CI	Calidad continua	forge test en CI (GitHub Actions), cobertura, gates de análisis estático	El estándar del sector es CI en verde + fuzzing como mínimo, no opcional

⛓️ Lenguajes y frameworks: dónde está el centro de gravedad

Solidity sigue siendo el centro de gravedad del empleo y del conocimiento acumulado (auditorías, patrones, exploits documentados). **Vyper** aporta una sintaxis minimalista tipo Python con menos superficie de sorpresas, a costa de ecosistema más pequeño — y su incidente de 2023 (bug del compilador que afectó a pools de Curve) recordó que el riesgo de compilador existe en todos los lenguajes. Fuera de la EVM: **Rust con Anchor** en Solana, **Cairo** en Starknet y **Move** en Aptos/Sui, cada uno con modelos de recursos y de cuentas distintos que no se transfieren automáticamente desde la mentalidad EVM.

En frameworks, la industria consolidó dos opciones:

- **Foundry** (forge, cast, anvil): tests en Solidity, fuzzing y pruebas de invariantes integradas, velocidad de compilación notable. Es la elección por defecto de los equipos de protocolo actuales y la de este programa.
- **Hardhat:** tests en TypeScript, ecosistema maduro de plugins, buena integración con tooling JS existente. Sigue siendo común en equipos con base frontend fuerte y en proyectos anteriores a 2023.

Muchos repos serios usan ambos: Foundry para tests e invariantes, Hardhat para scripting de despliegue heredado. No es una guerra religiosa; es una decisión de equipo.

🔌 RPC: nodo propio o proveedor, el trade-off que nadie puede evitar

Toda aplicación necesita hablar con un nodo. Las dos rutas:

- **Proveedor gestionado (Alchemy, Infura, QuickNode):** cero operación, APIs mejoradas (trazas, webhooks), SLAs. Costos: dependencia de un tercero que ve todas tus consultas (privacidad), puede aplicar rate limits o censura, y es un punto único de fallo — las caídas históricas de proveedores grandes han tumbado "medio ecosistema" de golpe porque demasiadas apps compartían el mismo endpoint.
- **Nodo propio:** soberanía y privacidad completas, sin límites de terceros; a cambio, operación 24/7, discos NVMe grandes, sincronización y upgrades en cada hard fork.

La práctica profesional en producción es **redundancia**: múltiples proveedores detrás de un router de RPC con failover, o proveedor + nodo propio de respaldo. Tratar el endpoint RPC como dependencia crítica única es un incidente esperando fecha.

📊 Datos, indexación y oráculos

La cadena es pésima base de datos de lectura. El patrón estándar:

1. **Eventos** bien diseñados en los contratos (indexed donde corresponde).
2. **Subgraph** en The Graph (o indexer propio con viem + base de datos) que transforma eventos en entidades consultables vía GraphQL.
3. **Dune** para analítica SQL exploratoria y dashboards públicos.

Para datos externos (precios, clima, resultados), los **oráculos** son la frontera de confianza más delicada del diseño: **Chainlink** (redes descentralizadas de nodos, feeds push) y **Pyth** (datos de primera parte de exchanges y market makers, modelo pull) dominan. La historia de DeFi está llena de protocolos técnicamente correctos drenados por manipulación de un oráculo débil (spot price de un pool ilíquido como oráculo es el clásico). La elección y configuración del oráculo es una decisión de seguridad de primer orden, no un detalle de integración.

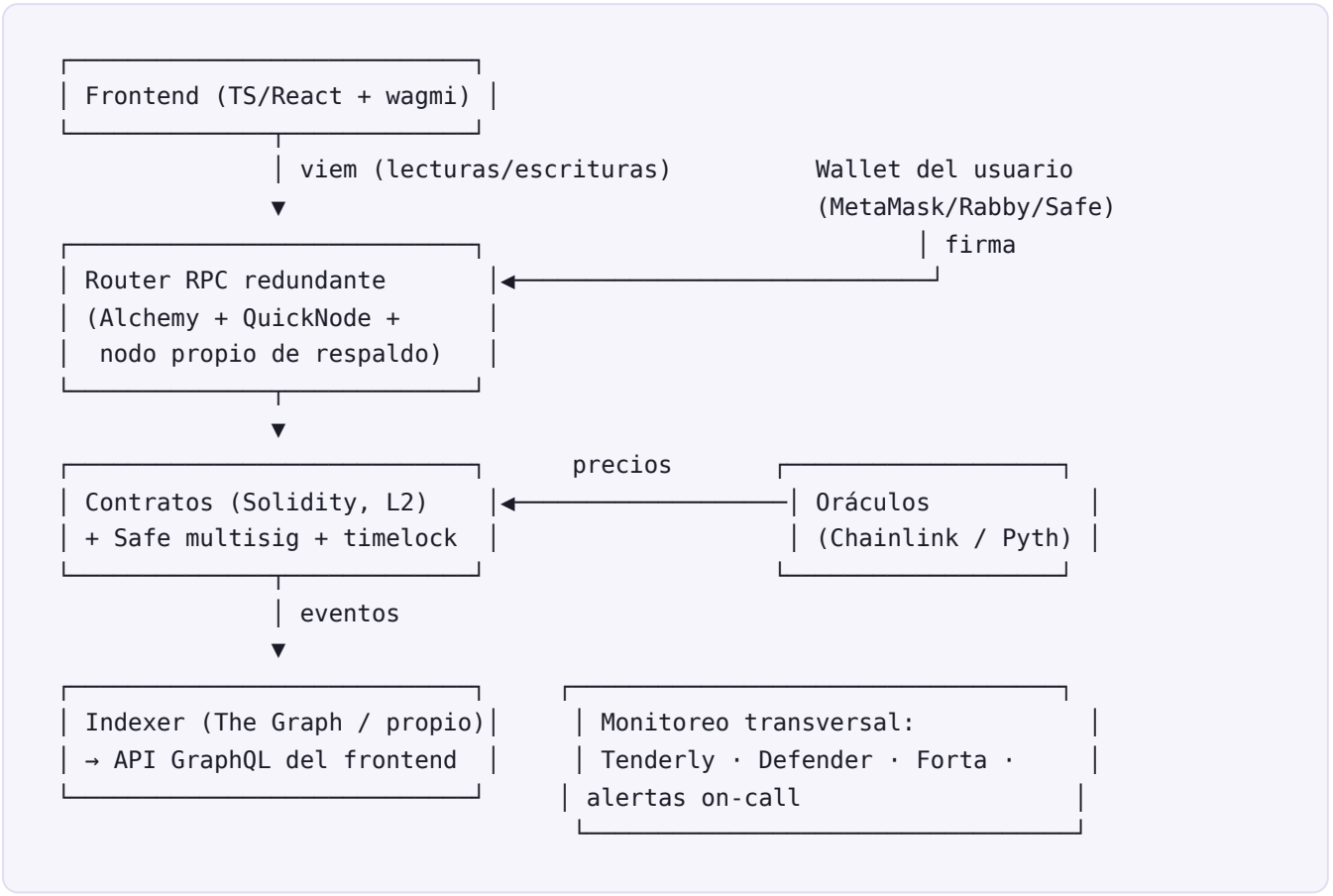
🛡️ Seguridad, monitoreo y operación

El pipeline mínimo de un equipo profesional:

- **Estático:** Slither en CI en cada PR; revisar findings, no silenciarlos en bloque.
- **Dinámico:** fuzzing e invariantes con Foundry y/o Echidna — las propiedades ("la suma de balances es igual al supply") encuentran clases de bugs que los tests unitarios no ven.
- **Dependencias:** OpenZeppelin Contracts como base auditada en lugar de reimplementar ERC-20, control de acceso o upgrades a mano.
- **Auditoría externa** antes de mainnet, y **bug bounty** después (Immunefi es el estándar del sector).
- **En producción:** Tenderly para simulación y debugging de transacciones, OpenZeppelin Defender para automatizar operaciones administrativas con multisig, Forta para detección de anomalías en tiempo real.

El matiz honesto: nada de esto garantiza ausencia de exploits — protocolos auditados por firmas de primer nivel han sido drenados igualmente. El objetivo es reducir probabilidad y ganar capacidad de respuesta, no comprar certeza.

🖼️ Una dApp en producción, en bloque



Nótese que la mitad de las cajas no son "blockchain": son infraestructura web clásica al servicio de dos o tres contratos. Esa proporción es representativa de los proyectos reales.

🧪 Criterios de selección de herramientas

Antes de adoptar cualquier pieza del stack, un CTO responsable pregunta:

Criterio	Pregunta concreta
Madurez	¿Años en producción, protocolos grandes que dependan de ella, historial de incidentes?
Auditorías	¿El código de la herramienta/librería está auditado? ¿Las auditorías son públicas?
Licencia	¿MIT/Apache, o una Business Source License que restringe forks comerciales?
Lock-in	¿Qué cuesta migrar? ¿Formatos abiertos, APIs estándar, o propietario?
Comunidad	¿Issues respondidos, releases regulares, más de un mantenedor activo, financiamiento sostenible?

La última fila importa más de lo que parece: el ecosistema tiene un historial de herramientas excelentes abandonadas cuando se agotó el financiamiento del equipo. Dependencia crítica sin plan B es deuda operativa.

🚀 Stack mínimo recomendado para empezar

El stack de este programa es deliberadamente el camino más transitado del sector, para que cada hora invertida transfiera directamente a un empleo real:

- **Solidity + Foundry** para contratos, tests, fuzzing e invariantes.
- **viem + TypeScript** para scripts e integración cliente.
- **pnpm** como gestor de paquetes del monorepo.
- **GitHub Actions** para CI (compilación, tests, análisis estático).

Ese stack se ejercita de punta a punta en el currículo y en el catálogo de laboratorios. La recomendación profesional es dominar un stack completo en profundidad antes de coleccionar herramientas: la amplitud llega sola con los proyectos.

⚠️ Errores y mitos frecuentes

Mito o error	Realidad
"Necesito aprender todas las herramientas"	Los equipos contratan profundidad en un stack (típicamente Solidity+Foundry+viem), no un bingo de logos.
"Con Infura ya tengo infraestructura descentralizada"	Un único proveedor RPC es un punto central de fallo y de vigilancia; la descentralización de la cadena no se hereda automáticamente.
"IPFS guarda mis archivos para siempre"	Sin pinning (propio o pagado) el contenido puede desaparecer; permanencia real es el modelo de Arweave, y también se paga.
"Pasó Slither, está seguro"	El análisis estático encuentra patrones conocidos; la lógica de negocio rota, los fallos económicos y de oráculo requieren fuzzing, revisión y auditoría.
"Uso el precio del pool como oráculo, es lo mismo"	El spot de un pool es manipulable en una transacción (flash loans); es la causa raíz de una fracción enorme de los exploits de DeFi.
"ethers.js está muerto"	Sigue en una cantidad enorme de código en producción; viem/wagmi son el default para código nuevo, pero leer ethers es habilidad laboral real.
"El monitoreo lo agregamos después del lanzamiento"	Después del lanzamiento ya custodias fondos; sin monitoreo, el primer aviso de un exploit suele llegar por redes sociales.

🔗 Referencias

- Foundry Book: https://book.getfoundry.sh/
- viem: https://viem.sh/
- The Graph: https://thegraph.com/docs/
- Chainlink: https://docs.chain.link/
- OpenZeppelin: https://docs.openzeppelin.com/

- Solidity: https://docs.soliditylang.org/
- Documentación para desarrolladores de ethereum.org: https://ethereum.org/developers/docs/
- wagmi: https://wagmi.sh/
- Safe (multisig): https://docs.safe.global/

🧭 Navegación

⬅️ 01 · Cómo se construye una blockchain · Índice de industria · ➡️ 03 · Equipos, roles y metodología

👥 Equipos, roles y metodología de trabajo

> **Audiencia:** Líderes técnicos, PMs y quienes buscan empleo en el sector · ⏱️ **Lectura:** 20 min · **Fuentes:** proceso EIP, foros de gobernanza y prácticas públicas de equipos del ecosistema ⬅️ Industria · 🏠 Programa · 📚 Bibliografía

---

🧩 Los roles reales del sector

El organigrama de un equipo blockchain se parece al de una empresa de software con tres diferencias: la seguridad es un rol de primera línea (no un consultor trimestral), existe gente dedicada a diseñar incentivos económicos, y varios roles trabajan de cara a comunidades públicas. La tabla resume los roles que aparecen en las ofertas reales del sector:

Rol	Responsabilidad principal	Habilidades clave	Nota de mercado
Protocol engineer	Clientes de nodo, consenso, redes P2P, upgrades de protocolo	Go/Rust, sistemas distribuidos, criptografía aplicada	Pocos puestos, barrera alta, muy bien pagados
Smart contract engineer	Diseño e implementación de contratos y sus tests	Solidity/Vyper, Foundry, patrones de seguridad, gas	El rol de entrada más común al sector
Security engineer / auditor	Revisión de código, fuzzing, modelado de amenazas, auditorías	Exploits históricos, invariantes, Slither/Echidna	Demanda crónica superior a la oferta
Frontend web3	Interfaces que firman transacciones reales	TypeScript, React, viem/wagmi, UX de wallets	Un bug de frontend puede firmar la transacción equivocada: no es "solo UI"
Infra / DevOps	Nodos, RPC, indexers, CI/CD, observabilidad	Kubernetes, Terraform, operación de nodos 24/7	Transferible casi 1:1 desde DevOps tradicional
Researcher	Consenso, criptografía, MEV, diseño de mecanismos	Matemáticas, papers, prototipos	Concentrado en fundaciones y equipos core
Tokenomics / mechanism designer	Incentivos, emisión, gobernanza económica	Teoría de juegos, modelado, finanzas	Escaso; a menudo el fundador lo hace mal solo
Product manager	Priorización con restricciones únicas (inmutabilidad, gobernanza)	PM clásico + entender qué NO se puede iterar	Los ciclos de release no se parecen a Web2
DevRel	Documentación, ejemplos, soporte a integradores	Escribir código y comunicarlo en público	Puerta de entrada frecuente para perfiles mixtos
Business development	Integraciones, partnerships, listados	Red de contactos del ecosistema	En cripto, las integraciones son también técnicas
Legal / compliance	Regulación por jurisdicción, estructura de entidades	MiCA (UE), normativa de valores por país	Involucrado desde el diseño del token, no después

📡 Cómo se comunican los equipos: remoto, async y en público

La cultura dominante del sector se formó en proyectos open source globales y se nota:

- **Remote-first y asincronía real:** los equipos están repartidos en todos los husos horarios; la escritura clara sustituye a la reunión. Una propuesta que no está escrita, con contexto y trade-offs, en la práctica no existe.
- **Trabajo en público:** el código en GitHub, las decisiones de protocolo en foros abiertos (Ethereum Magicians para EIPs; los foros de gobernanza de Optimism y Arbitrum para sus DAOs), la conversación diaria en Discord. Un candidato puede leer las discusiones internas reales de la mayoría de los protocolos antes de la entrevista — y los buenos entrevistadores esperan que lo haya hecho.
- **Reputación por contribución:** en un ecosistema donde todo es público, el historial de PRs, propuestas y post-mortems escritos pesa tanto como el currículum.

El matiz honesto: "trabajo en público" también significa que los errores son públicos. Los equipos maduros lo asumen con post-mortems firmados; los inmaduros borran mensajes de Discord. La diferencia se nota y el mercado la recuerda.

Para quien busca empleo, esto es una ventaja asimétrica frente a Web2: el trabajo del equipo al que aspiras es legible de antemano. Antes de una entrevista con un equipo de protocolo, es razonable (y esperado) haber leído:

- Sus últimos PRs relevantes y cómo revisan código en GitHub.
- Las propuestas activas en su foro de gobernanza y las objeciones que recibieron.
- Su último post-mortem, si lo hay: dice más de la cultura que cualquier página de "valores".

📜 RFC y EIP: la columna vertebral de la coordinación técnica

El mecanismo central de coordinación del ecosistema Ethereum es el proceso **EIP** (Ethereum Improvement Proposal), definido en EIP-1. Su ciclo de vida formal:

```
Idea ──▶ Draft ──▶ Review ──▶ Last Call ──▶ Final
            │          │           │
            │          │           └─ ventana mínima de 14 días para
            │          │              objeciones de última hora
            │          └─ discusión pública (Ethereum Magicians),
            │             revisión de editores, iteración
            └─ el autor escribe la especificación completa;
               sin spec escrita no hay discusión seria

(estados adicionales: Stagnant si se abandona, Withdrawn si se retira)
```

Lo relevante para un profesional no es memorizar los estados sino la filosofía: **la especificación escrita precede al código**, cualquiera puede objetar en público, y una propuesta "Final" es un contrato social que decenas de equipos independientes

implementan sin que nadie pueda obligarlos. EIP-4844 y EIP-7702 recorrieron exactamente este camino antes de llegar a Dencun (2024) y Pectra (2025).

Dentro de las empresas del sector, el equivalente son los **design docs** y los **ADRs** (Architecture Decision Records): el mismo principio — decisión escrita, revisada y archivada — a escala de equipo. Este programa lo practica en su propio índice de ADRs.

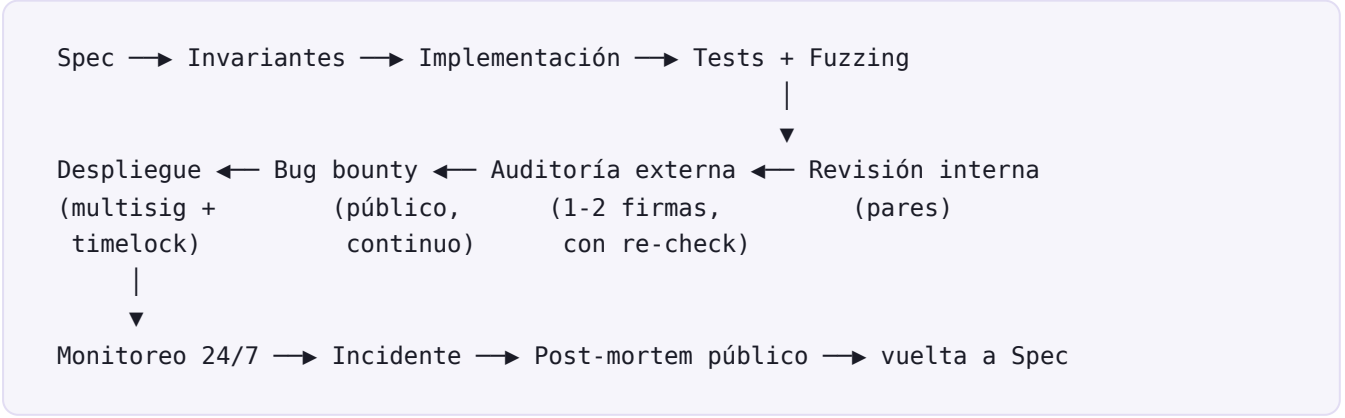
🐢 Por qué "move fast and break things" no aplica aquí

En Web2, un bug en producción se corrige con un deploy; el costo del error es acotado y reversible. En contratos inteligentes, el código desplegado es inmutable por defecto y custodia fondos: **el costo del error es instantáneo, público e irreversible**. La historia del sector — de The DAO (2016) a los cientos de millones drenados cada año que documentan las firmas de seguridad — es la factura de equipos que trasladaron hábitos de Web2 sin adaptarlos.

La metodología estándar que la industria destiló de esos fracasos:

1. **Especificación:** qué hace el sistema, qué no hace, quién puede hacer qué.
2. **Invariantes:** propiedades que deben cumplirse siempre ("nadie retira más de lo que depositó"), escritas antes que el código.
3. **Implementación** contra la spec, con librerías auditadas (OpenZeppelin) como base.
4. **Pruebas y fuzzing:** unitarias, de integración, fuzzing e invariant testing (Foundry/Echidna); cobertura alta como piso, no como meta.
5. **Revisión interna** por pares que no escribieron el código.
6. **Auditoría externa:** una o dos firmas independientes; los findings se corrigen y se re-auditan. Es un gate obligatorio, no un sello de marketing.
7. **Bug bounty** público (típicamente vía Immunefi) antes y después del despliegue.
8. **Despliegue controlado:** multisig (Safe) + timelock para toda función administrativa; límites iniciales conservadores.
9. **Monitoreo y respuesta:** alertas en tiempo real, runbooks de pausa/mitigación, guardias.

🔄 El pipeline, en bloque



El ciclo completo de un protocolo serio, de spec a mainnet, se mide en meses, y las auditorías de firmas reconocidas se reservan con semanas o meses de antelación (los precios varían mucho; consúltalo en vivo con las firmas). Planificar el calendario sin contar la auditoría es el error de PM más común del sector.

Dos observaciones prácticas sobre el pipeline:

- **No es cascada pura:** dentro de cada etapa se itera con normalidad; lo que no se negocia es saltarse un gate. Un cambio de una línea después de la auditoría reabre la auditoría, y los equipos que lo "olvidaron" protagonizan varios de los exploits mejor documentados.
- **El pipeline es también un artefacto de contratación:** preguntarle a un equipo cómo es su camino de spec a mainnet distingue en una conversación a los equipos profesionales de los que improvisan. Las etapas de pruebas e invariantes de este pipeline se practican en los laboratorios del programa, sobre el stack descrito en el currículo.

🗳️ Ceremonias y coordinación: del standup a la gobernanza

Las ceremonias ágiles clásicas existen (standups async, sprints, retros), pero el sector añade capas de coordinación sin equivalente en Web2:

- **All-core-devs calls de Ethereum:** llamadas públicas y grabadas donde los equipos de clientes — empresas independientes entre sí — acuerdan el contenido y calendario de cada hard fork. Es el ejemplo más visible de coordinación técnica multi-equipo sin autoridad central: nadie puede ordenar, solo convencer.
- **Gobernanza off-chain:** foros (propuestas, temperatura), snapshot votes sin gas. Ahí se decide la dirección.
- **Gobernanza on-chain:** votos con token que ejecutan cambios reales en contratos, generalmente con timelock. Ahí se ejecuta — y el equipo debe diseñar qué está bajo gobernanza y qué no.
- **Coordinación de upgrades:** un cambio de protocolo exige que operadores de nodos independientes actualicen a tiempo; la comunicación pública (blogs, foros, Discord) es parte del trabajo de ingeniería, no del de marketing.

Para un PM o líder técnico que llega de Web2, la diferencia operativa es que una parte de los stakeholders no son empleados: son delegados de una DAO, operadores anónimos de nodos o usuarios con derecho a voto. Los plazos se negocian en público.

🌐 Diferencias concretas con Web2

Dimensión	Web2 típico	Sector blockchain
Gate de release	QA + feature flags	Auditoría externa obligatoria + bug bounty
Corrección de errores	Deploy inmediato	Gobernanza/multisig/timelock; a veces imposible sin migración
Incident response	Statuspage y comunicación controlada	Incidente visible on-chain en tiempo real; los atacantes y los usuarios lo ven a la vez
Post-mortems	Internos, a veces públicos	Públicos por norma del sector; su calidad define la reputación del equipo
Roadmap	Privado hasta el anuncio	Discutido en foros públicos con la comunidad
Stakeholders	Management y clientes	Además: DAO, validadores, holders, auditores, reguladores

La respuesta a incidentes merece estudio propio — el programa la desarrolla en operación e incidentes — pero la regla cultural es clara: en un sistema transparente, ocultar un incidente es imposible y intentarlo destruye más valor que el exploit mismo.

⚠️ Errores y mitos frecuentes

Mito o error	Realidad
"Es una startup: iteramos rápido y arreglamos después"	Con código inmutable que custodia fondos, "después" puede no existir; el proceso spec→auditoría→timelock es el mínimo profesional.
"La auditoría garantiza que no habrá exploits"	Protocolos auditados caen cada año; la auditoría reduce riesgo y demuestra diligencia, no compra certeza.
"Los roles son como en Web2 con otro nombre"	Auditores, mechanism designers y DevRel de protocolo no tienen equivalente directo; y el frontend firma transacciones irreversibles.
"La gobernanza es teatro, decide el equipo"	A veces es cierto — y es un riesgo señalado públicamente; en protocolos maduros la DAO ha revertido decisiones del equipo core. La honestidad exige mirar caso por caso.
"Para entrar al sector necesito años de cripto"	Los equipos valoran fundamentos sólidos de ingeniería + evidencia pública (contribuciones, CTFs, auditorías practicadas); el contexto se aprende en meses.
"Discord es ruido, lo importante está en Jira"	Las decisiones reales de los protocolos abiertos se cocinan en foros y llamadas públicas; ignorarlos es trabajar a ciegas.

🔗 Referencias

- EIP-1, propósito y proceso de los EIPs: https://eips.ethereum.org/EIPS/eip-1
- Ethereum Magicians (foro de discusión de EIPs): https://ethereum-magicians.org/

- Trail of Bits, *Building Secure Contracts*: https://secure-contracts.com/
- a16z crypto (investigación y prácticas del sector): https://a16zcrypto.com/
- Comunidad Ethereum, all-core-devs y contribución: https://ethereum.org/community/
- Foro de gobernanza de Optimism: https://gov.optimism.io/
- Immunefi (bug bounties del sector): https://immunefi.com/

🧭 Navegación

⬅️ 02 · Stack tecnológico · Índice de industria · ➡️ 04 · Blockchain para empresas

🏢 Blockchain para empresas

> **Audiencia:** Decisores, consultores y arquitectos empresariales · ⏱️ **Lectura:** 25 min · **Fuentes:** informes públicos de adopción y casos documentados ⬅️ Industria · 🏠 Programa · 📚 Bibliografía

---

🎯 Qué aporta de verdad (y qué es humo)

Una blockchain **no** hace que los datos sean "verdaderos", ni abarata por sí sola una base de datos, ni elimina intermediarios por arte de magia. Lo que sí aporta, cuando el caso de uso es el correcto, es un conjunto acotado de beneficios verificables:

- **Coordinación entre organizaciones que no confían plenamente entre sí:** un registro compartido con reglas ejecutables elimina la pregunta "¿de quién es la copia buena?".
- **Liquidación más rápida y programable:** pasar de T+2 a liquidación atómica (entrega contra pago en la misma transacción) reduce riesgo de contraparte y capital inmovilizado.
- **Auditabilidad compartida:** todas las partes ven el mismo historial firmado criptográficamente; el regulador puede tener un nodo observador.
- **Activos programables:** un bono que paga cupones solo, una stablecoin con lógica de cumplimiento embebida, un fondo cuyas participaciones se transfieren 24/7.
- **Reducción de conciliación:** si dos bancos comparten el registro, desaparecen los procesos batch de "cuadrar" sistemas que hoy consumen equipos enteros de back office.

Lo que suele ser humo: "blockchain para trazabilidad" cuando el problema real es que alguien miente al ingresar los datos (el problema del oráculo no se resuelve con hashes), "blockchain interna" de una sola empresa (eso es una base de datos con pasos extra) y proyectos donde el incentivo para que los competidores compartan infraestructura nunca existió. Kevin Werbach lo resume bien: la blockchain es una **nueva arquitectura de confianza**, no un reemplazo universal de las existentes.

📊 Casos reales: éxitos operativos y fracasos instructivos

La mejor vacuna contra el marketing es estudiar casos con estado verificable. Esta tabla mezcla deliberadamente éxitos y fracasos, porque ambos enseñan:

Caso	Sector	Estado (a 2025-2026)	Lección principal
JPMorgan **Kinexys** (ex Onyx / JPM Coin)	Pagos y liquidación institucional	✅ Operativo; procesa miles de millones de USD diarios en pagos entre clientes institucionales	La liquidación interbancaria programable funciona cuando un actor con escala la opera y hay caso de negocio claro
Visa liquidación con stablecoins (USDC)	Pagos	✅ Operativo; liquidación de obligaciones con adquirentes sobre redes públicas	Las redes públicas pueden integrarse a infraestructura de pagos tradicional de forma incremental
Bonos digitales del **BEI** (con Goldman, Santander, SocGen) y bono de **Siemens**	Mercado de capitales	✅ Emisiones reales completadas, en volúmenes aún modestos	La emisión nativa digital reduce plazos de liquidación; el cuello de botella es legal y de mercado, no técnico
BlackRock BUIDL (fondo tokenizado, vía Securitize)	Gestión de activos / RWA	✅ Operativo; uno de los mayores fondos tokenizados del mundo (cifras: consúltalo en vivo)	La tokenización de fondos de mercado monetario encontró demanda real como colateral on-chain
Remesas y pagos con stablecoins en LatAm (Bitso, corredores México–EE. UU. y otros)	Remesas	✅ Operativo y creciendo; volúmenes relevantes (consúltalo en vivo)	Donde el sistema tradicional es caro y lento, las stablecoins compiten por precio y velocidad reales
TradeLens (IBM + Maersk)	Cadena de suministro	❌ Cerrado en 2023 pese a funcionar técnicamente	El problema fue el modelo de negocio y la gobernanza: los competidores de Maersk no querían subirse a la plataforma de Maersk
Libra / Diem (Meta)	Pagos / stablecoin	❌ Cancelado (activos vendidos en 2022) por presión regulatoria frontal	Una stablecoin global lanzada por una big tech era políticamente inviable; el riesgo regulatorio puede ser terminal
Reemplazo de **CHESS** en ASX (bolsa australiana)	Post-trade	❌ Proyecto DLT abandonado en 2022 tras años y cientos de millones invertidos	Reescribir infraestructura crítica nacional sobre tecnología inmadura y con mala gestión de proyecto es un riesgo compuesto
Pilotos de supply chain y trazabilidad (numerosos, 2017-2021)	Varios	⚠️ La mayoría no pasó de PoC	Sin incentivo económico para que todos los participantes carguen datos veraces, el registro compartido no aporta valor

El patrón de los fracasos casi nunca es tecnológico: es **gobernanza del consorcio** (¿quién manda?, ¿quién paga?, ¿por qué mi competidor operaría mi plataforma?), **modelo de negocio** ausente o **riesgo regulatorio** subestimado.

🏭 Sectores: tracción real vs. sobrepromesa

Sector	Tracción a 2025-2026	Comentario honesto
Pagos, liquidación y stablecoins	🟢 Alta	El caso de uso más probado; marcos regulatorios avanzando en varias jurisdicciones
Tokenización de activos (RWA)	🟢 Alta y creciente	Fondos de mercado monetario, deuda y crédito privado lideran; volúmenes en vivo en agregadores públicos
Mercado de capitales (bonos, repos)	🟡 Media	Emisiones reales pero volumen marginal frente al mercado tradicional
Cadena de suministro	🔴 Baja	Cementerio de pilotos; el problema del oráculo y los incentivos siguen sin resolverse
Identidad digital	🟡 Emergente	Estándares (credenciales verificables) maduran, adopción masiva aún pendiente
Salud	🔴 Baja	Regulación de datos y sistemas legacy hacen el caso de uso muy difícil

💎 Tokenización de activos del mundo real (RWA)

La tendencia dominante de 2024-2026 en el segmento empresarial es la **tokenización de activos del mundo real**: representar en una red blockchain instrumentos financieros tradicionales (cuotas de fondos, bonos, crédito privado, depósitos). El atractivo es concreto: liquidación atómica, operación 24/7, uso como colateral programable y fraccionamiento. BUIDL de BlackRock es el caso emblemático, pero el ecosistema incluye a Franklin Templeton, emisores de deuda tokenizada y bancos con depósitos tokenizados. Dos advertencias profesionales: primero, el token es tan bueno como el **vínculo legal** con el activo subyacente (custodia, jurisdicción, derechos del tenedor); segundo, las cifras de mercado cambian rápido — consúltalas en vivo en agregadores como RWA.xyz o DefiLlama antes de citarlas en un informe.

🔀 Pública, permisionada o L2: dónde se está moviendo la industria

La conversación de 2016-2020 era "pública vs. permisionada". La de 2024-2026 es distinta: el costo de las L2 públicas cayó drásticamente tras Dencun/EIP-4844 (2024), y buena parte de la industria financiera gira hacia **redes públicas con privacidad selectiva** o hacia L2/redes propias ancladas en infraestructura pública.

Criterio	Hyperledger Fabric	Besu (permisionada)	Corda	Rollup público (L2 de Ethereum)
Modelo	Permisionada, canales privados	EVM permisionada o pública	Punto a punto entre pares	Pública, ejecución barata anclada a L1
Privacidad	Alta (por diseño)	Media (configurable)	Alta (solo las partes ven el acuerdo)	Baja por defecto; en desarrollo (pruebas ZK, pools privados)
Interoperabilidad con DeFi/RWA público	Nula	Posible si es EVM	Baja	Nativa
Gobernanza	Consorcio (el punto débil histórico)	Consorcio u operador	Operador de red	Protocolo público + secuenciador
Costo por transacción	Infraestructura propia	Infraestructura propia	Infraestructura propia	Muy bajo post-EIP-4844 (consúltalo en vivo)
Cuándo elegirla	Datos sensibles entre pocos actores conocidos	Consorcio que quiere compatibilidad EVM	Acuerdos bilaterales financieros	Activos que se benefician de liquidez y composabilidad públicas

La lección de la década: las redes permisionadas resuelven la privacidad pero heredan el problema de gobernanza del consorcio (ver TradeLens); las públicas resuelven la neutralidad pero exigen resolver privacidad y cumplimiento. No hay respuesta única — hay trade-offs.

🧮 Matriz de decisión: cuándo sí y cuándo no

Coherente con el criterio del módulo 00 del programa (ver currículo): la pregunta no es "¿puedo usar blockchain?" sino "¿este problema necesita un registro compartido entre partes que no se confían?".

- ✅ **Considera blockchain si:** hay múltiples organizaciones escribiendo en el registro, no existe (o no conviene) un intermediario neutral, se necesita liquidación programable o el activo gana valor por ser transferible 24/7 entre terceros.
- ❌ **No la uses si:** una sola entidad controla los datos, los participantes ya confían en un operador central barato, el dato crítico nace fuera de la cadena y nadie puede verificarlo (problema del oráculo), o el volumen/latencia requerido excede lo razonable para la red elegida.
- ⚠️ **Señal de alerta:** si el proyecto sigue teniendo sentido reemplazando "blockchain" por "base de datos compartida con API", entonces eso es lo que deberías construir.

🗺️ Hoja de ruta de adopción por fases

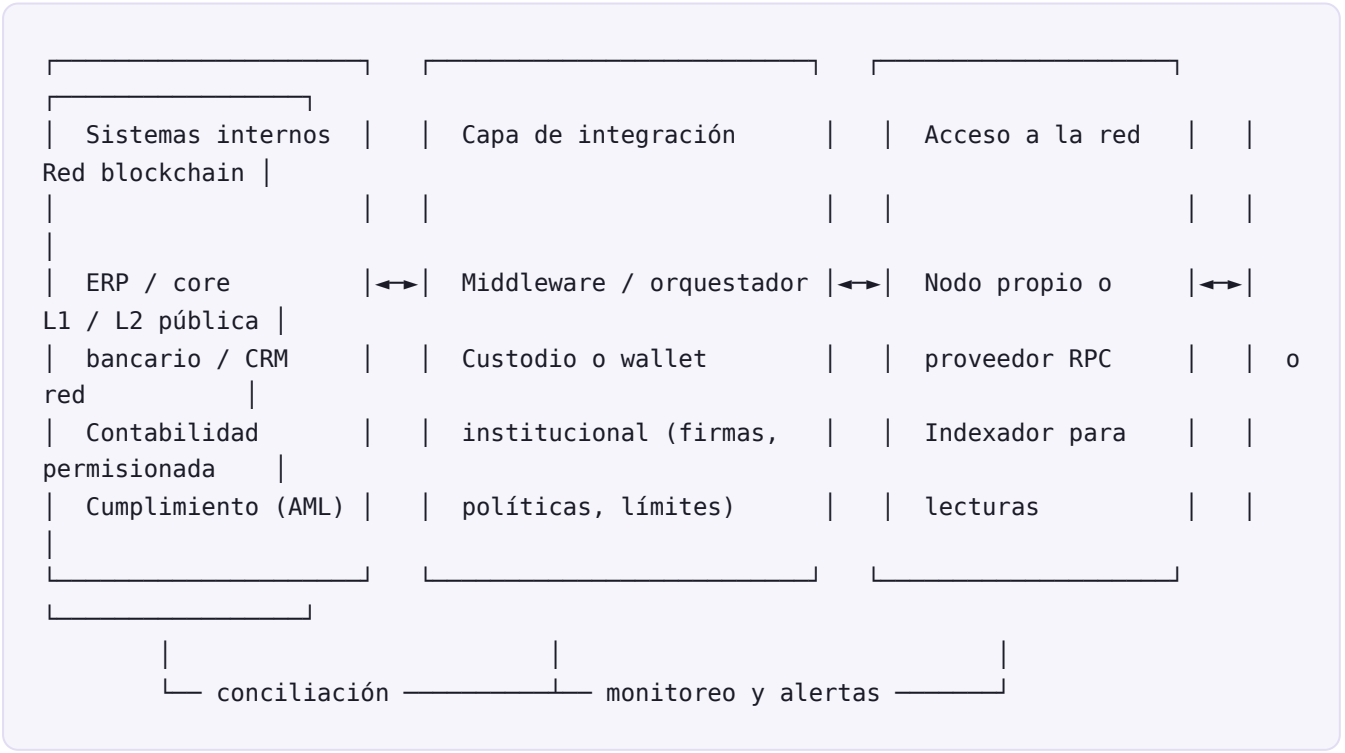
Cuando la matriz da "sí", la adopción sensata avanza por puertas de decisión, no por *big bang*:

1. **Exploración (semanas):** caso de negocio cuantificado (¿qué costo de conciliación, liquidación o capital inmovilizado se reduce y en cuánto?), análisis regulatorio preliminar por jurisdicción y mapa de partes interesadas internas: legal, riesgo, TI, negocio.
2. **PoC técnico (meses):** prototipo sobre testnet o red de prueba, con datos sintéticos. El objetivo es aprender las restricciones reales de integración y custodia, no impresionar al directorio con una demo.
3. **Piloto con alcance real:** volumen limitado, contrapartes reales, custodia y cumplimiento en configuración de producción. Aquí es donde mueren los proyectos con gobernanza no resuelta — y es mejor que mueran aquí que en producción.
4. **Producción acotada:** límites de monto, monitoreo continuo, plan de reversa al proceso tradicional y expansión gradual contra métricas acordadas de antemano.

Dos disciplinas transversales separan a los equipos serios: definir desde la fase 1 los **criterios de cancelación** (qué evidencia haría abandonar el proyecto — la mayoría de los pilotos zombis de 2017-2021 nunca los escribió) y presupuestar la **operación continua** desde el inicio, no solo la construcción.

🏗️ Arquitectura de integración empresarial

Una empresa no "se conecta a la blockchain" desde el ERP directamente. La integración real tiene capas, cada una con responsables y controles distintos:



Decisiones clave de esta arquitectura: ¿custodia propia o custodio regulado?, ¿nodo propio o RPC de terceros (y con qué SLA)?, ¿cómo se reflejan los eventos on-chain en la contabilidad interna? La capa de custodia y firmas es donde se concentra el riesgo operacional — merece el mismo rigor que la tesorería.

💰 Costos totales y riesgos

El costo del contrato inteligente suele ser la parte menor del presupuesto. Un caso empresarial serio presupuesta:

Partida	Comentario
Desarrollo	Contratos + backend de integración + frontend; el contrato es la punta del iceberg
Auditorías de seguridad	Una o más firmas externas; costo significativo y no opcional (rangos: consúltalo en vivo)
Integración con legacy	Frecuentemente la partida más cara: ERP, core bancario, conciliación contable
Cumplimiento	Licencias, asesoría legal por jurisdicción, KYC/AML, reporting al regulador
Operación continua	Nodos/RPC, monitoreo, gestión de claves, respuesta a incidentes, actualizaciones

Regla de oro presupuestaria: si el plan financiero termina en el go-live, no hay plan. La operación (claves, monitoreo, cumplimiento continuo, upgrades) es un costo permanente comparable al de cualquier sistema financiero crítico, y la decisión *build vs. buy* (nodo propio vs. RPC gestionado, custodia propia vs. custodio regulado) debe evaluarse partida por partida, no por ideología.

Riesgos que un decisor debe tener en el radar:

- **Regulatorio:** en la UE, MiCA está vigente desde 2024 y regula emisores de criptoactivos y proveedores de servicios (CASP); en Chile, la Ley Fintech 21.521 (2023) somete a los proveedores de servicios de criptoactivos a registro y supervisión de la CMF — detalle local en regulación y tributación en Chile. Operar en varias jurisdicciones multiplica el trabajo legal.
- **Reputacional:** un incidente de seguridad o la asociación con actores dudosos del ecosistema daña la marca más allá del monto perdido.
- **Técnico:** bugs en contratos son pérdidas directas y frecuentemente irreversibles; la inmutabilidad corta hacia ambos lados.
- **Gobernanza del consorcio:** el asesino silencioso de los proyectos permisionados; si no está resuelto en papel antes de escribir código, el proyecto ya está en riesgo.

⚠️ Errores y mitos frecuentes

Mito o error	Realidad
"Blockchain garantiza que los datos son verdaderos"	Garantiza integridad e historial del registro, no la veracidad del dato de origen (problema del oráculo)
"Ahorraremos eliminando intermediarios"	Se sustituyen intermediarios por otros (custodios, oráculos, operadores de red) y por costos nuevos de cumplimiento
"Empecemos con un piloto y la gobernanza se verá después"	TradeLens funcionó técnicamente y murió por gobernanza; es la primera conversación, no la última
"Permisionada = segura y sin riesgo regulatorio"	Sigue habiendo riesgo operacional, de claves y, según el activo, plena regulación financiera
"El éxito del piloto predice producción"	La mayoría de los PoC de 2017-2021 nunca escaló; producción exige modelo de negocio, no demo
"Las empresas serias solo usan redes privadas"	La tendencia 2024-2026 es la inversa: instituciones emitiendo y liquidando sobre redes públicas y L2

🔗 Referencias

- Reglamento MiCA (EUR-Lex): https://eur-lex.europa.eu/eli/reg/2023/1114/oj
- Bank for International Settlements — investigación sobre tokenización y dinero digital: https://www.bis.org/
- World Economic Forum — informes de blockchain y activos digitales: https://www.weforum.org/
- Kevin Werbach, *The Blockchain and the New Architecture of Trust* (MIT Press): https://mitpress.mit.edu/9780262547161/the-blockchain-and-the-new-architecture-of-trust/
- Securitize — infraestructura del fondo tokenizado BUIDL: https://securitize.io/
- Hyperledger Fabric — documentación oficial: https://hyperledger-fabric.readthedocs.io/
- RWA.xyz — datos en vivo de activos tokenizados: https://www.rwa.xyz/

🧭 Navegación

⬅️ 03 · Equipos, roles y metodología · Índice de industria · ➡️ 05 · Modelos de negocio

💼 Modelos de negocio del ecosistema

> **Audiencia:** Fundadores, analistas y product managers · ⏱️ **Lectura:** 25 min · **Fuentes:** documentación pública de protocolos y literatura de tokenomics ⬅️ Industria · 🏠 Programa · 📚 Bibliografía

---

🧭 De dónde sale el dinero realmente

Separemos dos preguntas que el marketing mezcla: **cómo genera ingresos un negocio del ecosistema** y **cómo captura valor un token**. Son problemas distintos y confundirlos ha quemado miles de millones. Los negocios del sector cobran por lo mismo que cualquier otro: por transacciones facilitadas, por infraestructura alquilada, por riesgo gestionado o por conocimiento aplicado. El token, cuando existe, es un mecanismo adicional de coordinación y captura de valor — a veces funciona, a veces es pura fricción.

Un filtro profesional útil antes de analizar cualquier proyecto: ¿quién paga, cuánto, por qué, y seguiría pagando si el precio del token fuera cero? Si la respuesta a la última pregunta es "nadie", el "modelo de negocio" es en realidad exposición al ciclo de mercado.

📊 Tabla de modelos: quién cobra, a quién y por qué

Modelo	Ejemplos	Cómo genera ingresos	Riesgos del modelo
Protocolo DeFi (intercambio)	Uniswap, Curve	Comisión por swap pagada por traders; hoy fluye mayormente a proveedores de liquidez; el "fee switch" para el protocolo es decisión de gobernanza	Competencia feroz por comisiones; la gobernanza puede tardar años en activar captura de valor
Protocolo DeFi (préstamos)	Aave, Morpho	Spread entre interés pagado por prestatarios y recibido por depositantes; comisiones de liquidación	Riesgo de deuda incobrable en caídas bruscas; parámetros de riesgo mal calibrados
L2 / secuenciador	Arbitrum, Base, OP Mainnet	Margen entre el gas cobrado a usuarios y el costo de publicar datos en L1 — muy visible desde EIP-4844 (2024), que abarató la disponibilidad de datos con blobs	Compresión de márgenes por competencia entre L2; presión por descentralizar el secuenciador (y compartir el ingreso)
Staking y validación	Lido, staking institucional (Coinbase, Figment)	Comisión porcentual sobre las recompensas de staking de los clientes	Riesgo de slashing, concentración criticada por la comunidad, regulación del staking como servicio
Infraestructura RPC / nodos	Alchemy, Infura, QuickNode	Suscripción SaaS por acceso a nodos y APIs; niveles gratuitos como embudo	Comoditización; los clientes grandes migran a nodos propios
Indexación de datos	The Graph, indexadores propietarios	Pago por consulta o suscripción; en The Graph, mercado de curación e indexación con token	Alternativas centralizadas más simples compiten bien en la práctica
Oráculos	Chainlink	Los protocolos pagan por datos (suscripción o pago por actualización); servicios adicionales como VRF y CCIP	Concentración de dependencia sistémica; negociación opaca de contratos enterprise
Exchanges y custodia	Binance, Coinbase, custodios regulados	Spreads y comisiones de trading, listado, custodia institucional con tarifa sobre activos	Regulatorio (el mayor); ciclos de volumen; competencia de DEX
Seguridad como servicio	Trail of Bits, OpenZeppelin, Immunefi	Auditorías por proyecto, retainers, plataformas de bug bounty con comisión, monitoreo	Escasez de talento senior; responsabilidad reputacional por incidentes post-auditoría

Modelo	Ejemplos	Cómo genera ingresos	Riesgos del modelo
Wallets	MetaMask, Rabby, wallets móviles	Comisión sobre swaps integrados, acuerdos con on-ramps, funciones premium	Monetizar sin erosionar confianza es difícil; competencia de wallets de exchanges
Tooling open core	Foundry (gratuito) vs. Tenderly, Hardhat + servicios	Núcleo abierto gratuito + plataforma de pago (simulación, monitoreo, CI especializado)	El estándar abierto puede canibalizar el producto de pago
Consultoría e integración	Consultoras especializadas, big four	Tarifa por proyecto/hora para llevar empresas a producción	Depende del ciclo de interés corporativo; difícil de escalar

Nota sobre los números: los ingresos reales de muchos de estos actores son públicos y verificables on-chain o en agregadores — pero cambian semana a semana. Cita siempre la fuente y la fecha: **consúltalo en vivo** en DefiLlama (fees/revenue) y Token Terminal.

🪙 El token como modelo de negocio

Un token puede cumplir funciones reales: asegurar la red (staking), coordinar gobernanza, alinear incentivos tempranos o dar derecho a flujos de caja. La pregunta profesional es cuándo **captura valor** y cuándo es solo fricción añadida para financiar el proyecto:

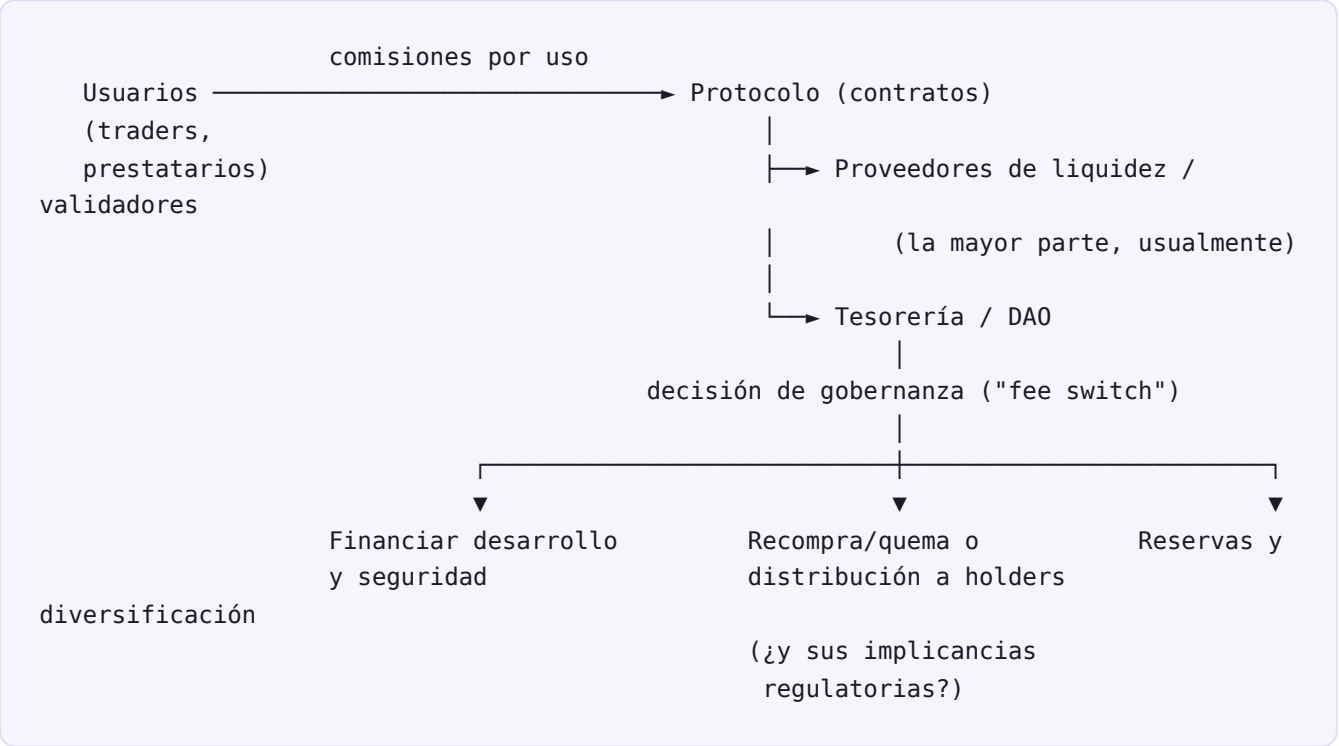
- **Captura valor** cuando existe un vínculo mecánico entre uso del protocolo y demanda del token: quema de comisiones, distribución de ingresos a stakers, colateral obligatorio. Y cuando ese vínculo sobrevive escrutinio legal.
- **Es fricción** cuando el producto funcionaría igual con ETH o stablecoins y el token solo existe porque financió la ronda: el usuario debe adquirir un activo volátil extra para usar el servicio, y la presión vendedora de emisiones e inversores supera cualquier demanda orgánica. La taxonomía clásica es menos nítida en la práctica de lo que sugieren las categorías:

Categoría	Promesa nominal	Realidad frecuente
Utility token	Acceso o pago dentro del servicio	Si el servicio acepta stablecoins, la "utilidad" es sustituible y la demanda es especulativa
Governance token	Derecho a votar decisiones del protocolo	Sobre una tesorería con ingresos, se parece funcionalmente a un derecho económico — y los reguladores lo saben
Security token	Valor negociable declarado, con cumplimiento	La categoría más honesta y la menos usada, porque asume el costo regulatorio de frente

La **"fat protocol thesis"** (Joel Monegro, USV, 2016) sostenía que en blockchain el valor se acumularía en la capa de protocolo y no en las aplicaciones — lo inverso a internet.

Como hipótesis histórica fue influyente y parcialmente correcta para las L1. Las críticas actuales son sólidas: las comisiones migran hacia donde hay usuarios (aplicaciones y frontends como los de Uniswap o los wallets capturan cada vez más), las L2 compiten los márgenes de ejecución a la baja, y "valor de mercado del token" no es lo mismo que "valor capturado sostenibleMente". Úsala como referencia conceptual, no como ley.

💸 Flujo de valor en un protocolo con comisiones



El eslabón punteado — de la tesorería al holder — es el más frágil: es donde la gobernanza se politiza y donde un token puede empezar a parecer un valor negociable ante el regulador.

📈 Métricas: qué mirar y qué no creerse

- **TVL (valor total bloqueado):** mide capital depositado, no ingresos ni usuarios. Es inflable con incentivos (capital mercenario que se va cuando terminan las emisiones) y doble-contable entre protocolos apilados. Útil como proxy de confianza, inútil como métrica de éxito por sí sola.
- **Fees vs. revenue:** *fees* es lo que pagan los usuarios; *revenue* es la porción que retiene el protocolo/tesorería. Un protocolo puede generar millones en fees y cero revenue (Uniswap durante años, con el fee switch apagado). DefiLlama y Token Terminal desglosan ambos — consúltalo en vivo.
- **Emisiones netas:** ingresos menos el costo de los incentivos en token propio. Muchos protocolos "rentables" en fees son deficitarios netos cuando se descuenta lo que emiten para atraer ese uso.

- **Usuarios activos y retención:** las direcciones no son personas (sybils, bots); busca cohortes y retención, no wallets acumuladas.
- **P/F y P/S (precio sobre fees o revenue):** útiles para comparar protocolos entre sí, siempre que se use la misma definición de ingreso; sin esa disciplina son marketing con decimales.
- **Concentración:** pocos usuarios o integraciones generando la mayoría de los fees es riesgo de cliente, igual que en cualquier SaaS.

🌱 Sostenibilidad: quién sobrevivió los inviernos

Los ciclos bajistas (2018-2019, 2022-2023) fueron el mejor auditor de modelos de negocio. Sobrevivieron los que cobraban por algo con demanda real e ingresos en activos sólidos: exchanges e infraestructura con suscripciones, protocolos de intercambio y préstamo con uso orgánico, firmas de seguridad, emisores de stablecoins (cuyo modelo — rendimiento del colateral — resultó ser de los más rentables del sector).

Patrones comunes de los sobrevivientes:

- Ingresos denominados en activos que no dependen del propio token (ETH, stablecoins, moneda fiat).
- Costos operativos que se pueden recortar en mercado bajista sin matar el producto.
- Tesorería diversificada antes del invierno, no durante.
- Un servicio por el que alguien pagaría aunque no existiera expectativa de apreciación de ningún token.

Murieron los **ponzinomics**: modelos cuyo ingreso dependía de la entrada de nuevos participantes comprando el token. El caso de estudio es el *play-to-earn* de 2021 (Axie Infinity como emblema): la "economía" pagaba a jugadores con un token cuya demanda eran otros jugadores nuevos; cuando el flujo de entrada se frenó, el modelo colapsó en meses. Misma mecánica en protocolos de "rendimiento" tipo Terra/Anchor: subsidios presentados como modelo de negocio. La señal de alerta es siempre la misma: rendimientos cuyo origen nadie puede explicar sin mencionar la llegada de nuevos compradores.

⚖️ Consideraciones regulatorias del modelo

El diseño del modelo de negocio y del token determina el perímetro regulatorio:

- **EE. UU. (referencia conceptual):** el test de *Howey* pregunta si hay inversión de dinero en una empresa común con expectativa de ganancia derivada del esfuerzo de terceros. Un token vendido prometiendo que el equipo lo valorizará encaja incómodamente bien. La aplicación concreta ha variado con los cambios de administración — el criterio conceptual sigue siendo la referencia.

- **UE:** MiCA (vigente desde 2024) clasifica criptoactivos (EMT, ART, otros), exige libro blanco y autorización a emisores y CASP, y regula especialmente a los emisores de stablecoins. Diseñar el token con MiCA en la mesa es más barato que rediseñarlo después.
- **Chile:** la Ley Fintech 21.521 (2023) incorpora los criptoactivos al perímetro de la CMF; los proveedores de servicios sobre criptoactivos requieren registro y autorización según el servicio.
- Regla práctica: si el modelo depende de vender el token a inversionistas minoristas con promesa implícita de apreciación, el riesgo regulatorio **es** el modelo de negocio.

🧪 Casos de estudio: la teoría en la práctica

Uniswap y el "fee switch": valor generado no es valor capturado

Uniswap ha facilitado un volumen acumulado de billones de USD y ha generado miles de millones en comisiones — pagadas casi íntegramente a los proveedores de liquidez, no al protocolo ni al token UNI. Durante años la gobernanza debatió activar el *fee switch* (desviar una fracción de las comisiones a la tesorería o a los holders) sin ejecutarlo, en parte por el riesgo de que esa distribución acerque a UNI a la definición de valor negociable, y en parte por el temor a que los LPs migren a competidores. Lección doble: la captura de valor es una decisión de gobernanza con costo regulatorio y competitivo, no una consecuencia automática del uso; y un token puede coexistir años con un producto exitoso sin recibir un centavo de sus flujos. Estado actual del debate: consúltalo en vivo en el foro de gobernanza del protocolo.

Secuenciadores de L2 después de EIP-4844: un negocio de márgenes visibles

Antes de Dencun (2024), el mayor costo operativo de una L2 era publicar datos en Ethereum. Los *blobs* de EIP-4844 redujeron ese costo drásticamente, y el margen del secuenciador — gas cobrado a usuarios menos costo de disponibilidad de datos en L1 — se volvió observable on-chain para cualquier analista. Es uno de los modelos más limpios del sector: ingreso por servicio prestado, costos medibles, sin depender del precio de un token. Sus riesgos también son claros: la competencia entre L2 comprime las tarifas al usuario, y la presión por descentralizar el secuenciador implicará, tarde o temprano, compartir ese ingreso.

Staking líquido: comisión recurrente sobre un flujo estructural

Lido cobra un porcentaje de las recompensas de staking que genera para sus usuarios: ingresos recurrentes, denominados en el activo de la red, con costos operativos acotados. Es probablemente el modelo de negocio más simple y robusto del ecosistema post-Merge — y a la vez el más criticado, por la concentración de participación que acumula sobre la seguridad de Ethereum. Los servicios institucionales de staking replican el modelo con

comisiones mayores a cambio de cumplimiento, custodia y soporte. La lección: los mejores modelos del sector cobran una fracción de un flujo estructural de la red, no apuestan a la apreciación de un token propio.

🧰 Checklist para evaluar cualquier modelo del sector

- ¿Quién paga, cuánto y a cambio de qué servicio concreto?
- ¿Los ingresos existen sin contar emisiones ni apreciación del token propio?
- ¿Qué fracción de los *fees* llega realmente al protocolo (revenue) y quién decide eso?
- ¿El modelo sobrevivió al menos un ciclo bajista completo, o solo conoce mercados alcistas?
- ¿Qué pasa con los ingresos si la actividad especulativa cae 80 %? (les pasó a todos en 2022)
- ¿El token es necesario para el servicio o es un instrumento de financiamiento con fricción añadida?
- ¿Qué clasificación regulatoria tendría el modelo en cada jurisdicción donde opera?

⚠️ Errores y mitos frecuentes

Mito o error	Realidad
"Si el producto se usa, el token sube"	No necesariamente: sin mecanismo de captura (quema, distribución, colateral), uso y precio pueden divergir durante años
"TVL alto = protocolo exitoso"	El TVL mide depósitos, no ingresos; puede ser capital mercenario alquilado con emisiones
"Los fees del protocolo son ingresos del protocolo"	La mayoría va a LPs/validadores; el revenue del protocolo puede ser cero con el fee switch apagado
"El token alinea incentivos por definición"	Un token mal diseñado desalinea: presión vendedora de insiders contra usuarios que necesitan estabilidad
"APY alto = buen negocio"	Pregunta de dónde sale el rendimiento; si la respuesta es "de las emisiones del token", es un subsidio, no un negocio
"La fat protocol thesis demuestra que hay que invertir en L1"	Es una hipótesis de 2016 con contraejemplos crecientes: frontends, wallets y aplicaciones capturan cada vez más valor
"Ser DAO nos exime de la regulación de valores"	Los reguladores miran la función económica, no la etiqueta organizacional

🔗 Referencias

- Shermin Voshmgir, *Token Economy* (texto abierto): https://github.com/Token-Economy-Book/EnglishVersion
- DefiLlama — TVL, fees y revenue en vivo: https://defillama.com/
- Token Terminal — métricas financieras de protocolos: https://tokenterminal.com/

- Joel Monegro, "Fat Protocols" (USV, 2016): https://www.usv.com/writing/2016/08/fat-protocols/
- Documentación de Uniswap (fees y gobernanza): https://docs.uniswap.org/
- Reglamento MiCA (EUR-Lex): https://eur-lex.europa.eu/eli/reg/2023/1114/oj

---

🧭 Navegación

⬅️ 04 · Blockchain para empresas · Índice de industria · ➡️ 06 · Ciclo de vida de un proyecto

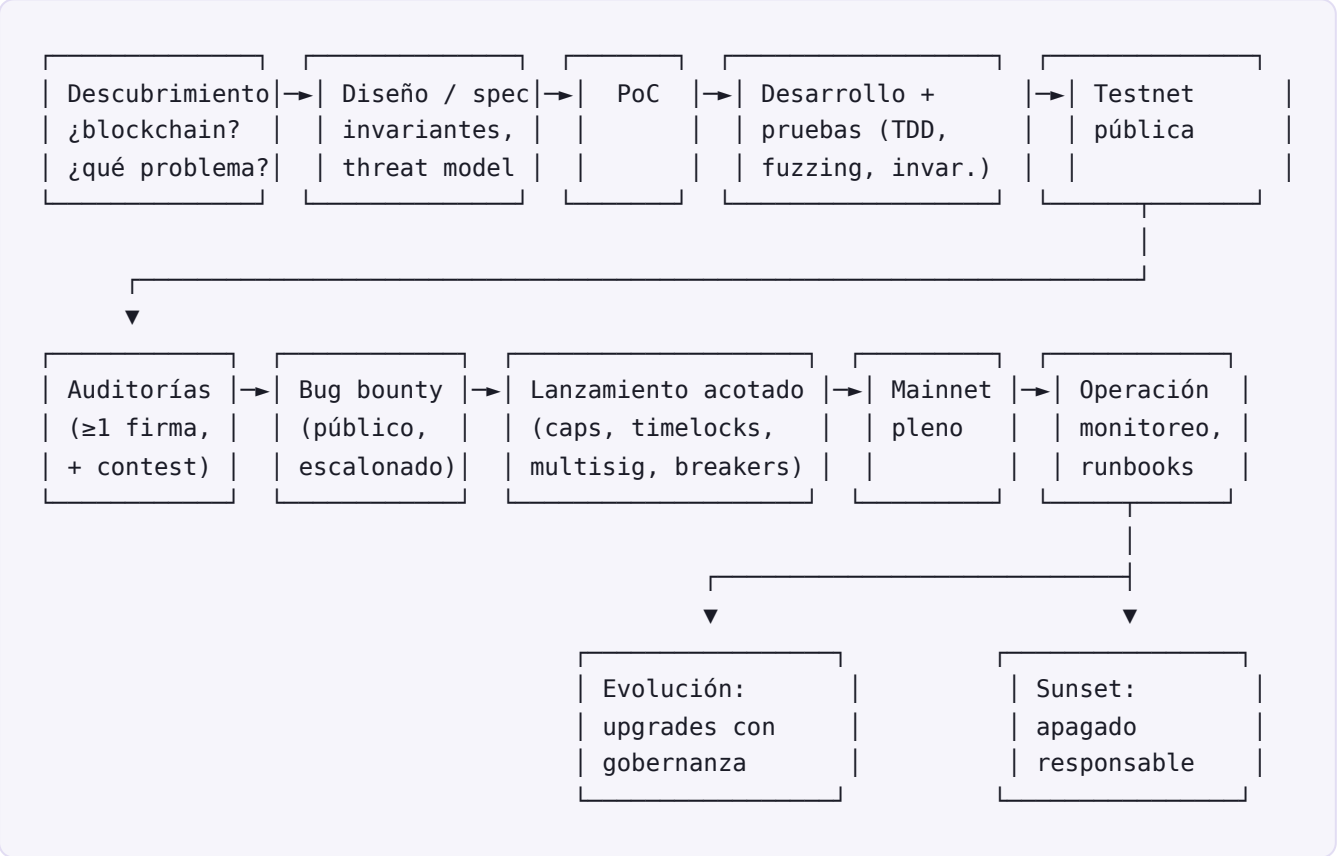
🔄 Ciclo de vida de un proyecto blockchain

> **Audiencia:** Equipos que llevan un protocolo de la idea a producción · ⏱️ **Lectura:** 25 min · **Fuentes:** prácticas públicas de despliegue y operación del ecosistema ⬅️ Industria · 🏠 Programa · 📚 Bibliografía

---

🗺️ El pipeline completo

Un protocolo serio no "se lanza": atraviesa un pipeline con puertas de calidad explícitas. Saltarse una fase no ahorra tiempo — traslada el costo a producción, donde los errores se pagan en fondos de usuarios. El pipeline de referencia:



Cada flecha es una **puerta de calidad** con criterio de salida explícito:

Fase	Criterio de salida (no se avanza sin esto)
Descubrimiento	Decisión documentada de que blockchain aporta valor; criterios de cancelación escritos
Diseño / spec	Spec aprobada, invariantes enumeradas, threat model revisado por alguien externo al equipo
PoC	Riesgos técnicos principales validados o descartados; decisión go/no-go registrada
Desarrollo + pruebas	CI verde, fuzzing e invariantes sin hallazgos abiertos, cobertura revisada, código congelado
Testnet pública	Flujos completos ejecutados por terceros; scripts de despliegue idénticos a los de mainnet
Auditorías	Hallazgos críticos y altos corregidos y re-verificados por el auditor; informe publicado
Bug bounty	Programa activo con pagos proporcionales al valor que se custodiará
Lanzamiento acotado	Caps, timelocks, multisig y monitoreo operativos y ensayados antes del primer depósito
Mainnet pleno	Evidencia de operación estable bajo caps; plan de subida de límites aprobado

🔍 Descubrimiento y diseño

Descubrimiento. La primera puerta es la más barata de cruzar en la dirección correcta: ¿este problema necesita blockchain? Aplica la matriz de decisión del módulo 00 del programa (ver currículo): múltiples partes que escriben, ausencia razonable de intermediario confiable, valor en la liquidación programable. Si la respuesta es no, el mejor entregable del proyecto es un documento que lo diga.

Diseño. Antes de escribir Solidity se escribe la **especificación**: qué hace el sistema, qué no hace, y — crucial — qué debe ser siempre verdad. Los entregables de esta fase:

- **Spec funcional** con casos de uso y flujos de fondos explícitos.
- **Invariantes del sistema** ("la suma de balances internos es igual al balance del contrato", "nadie retira más de lo depositado más su rendimiento"): serán las propiedades que el fuzzing y las pruebas de invariantes atacarán después.
- **Threat model:** actores, capacidades (¿qué puede hacer un atacante con flash loans?, ¿y el propio equipo con las claves de admin?), superficies de ataque, supuestos de confianza documentados.
- **Tokenomics**, si aplica, con simulaciones de emisiones y presión vendedora — no una hoja de cálculo optimista.
- **ADRs** (Architecture Decision Records) para cada decisión estructural: por qué upgradeable o no, qué oráculo, qué L2. El formato del programa está en el índice de ADRs.

🛠️ Desarrollo, pruebas y testnet

El estándar de la industria para contratos serios es el que practica este programa (ver currículo): **TDD con Foundry**, pruebas unitarias por función, **fuzzing** sobre entradas y **pruebas de invariantes** sobre las propiedades del diseño. Cobertura alta es necesaria pero no suficiente: la pregunta es si las invariantes correctas están escritas. A esto se suman análisis estático (Slither), revisión interna cruzada y un CI que bloquea cualquier merge con pruebas en rojo.

Testnet pública (Sepolia u Holesky para el ecosistema Ethereum; consulta las redes vigentes porque rotan): despliega con los mismos scripts que usarás en mainnet — el despliegue es código y se prueba como código. Sé honesto con lo que una testnet demuestra y lo que no:

- ✅ Demuestra: que el despliegue funciona, que las integraciones responden, que los flujos completos son ejecutables por terceros.
- ❌ No demuestra: comportamiento bajo valor económico real (en testnet nadie ataca porque no hay botín), condiciones de congestión y MEV reales, ni la conducta de usuarios adversariales. Las *incentivized testnets* mitigan esto solo parcialmente.

🛡️ Auditorías y bug bounty

Auditoría externa. Cómo elegir firma: historial público de informes, experiencia en tu tipo de protocolo (un AMM no se audita como un bridge), y disponibilidad real de sus auditores senior. Qué cubre: revisión manual experta del código congelado contra vulnerabilidades conocidas y lógica de negocio, en un tiempo acotado. Qué **no** cubre: código que cambies después del informe, dependencias fuera del alcance, riesgo económico/de oráculos si no se contrató específicamente, y en general la garantía de ausencia de bugs — una auditoría es una opinión experta con presupuesto finito, no un certificado. Costos: varían por alcance y firma en órdenes de magnitud — consúltalo en vivo.

La práctica madura para protocolos que custodian valor significativo es **defensa en profundidad**: dos auditorías independientes más un *contest* público (Code4rena, Sherlock), porque cientos de ojos con incentivos encuentran clases de bugs distintas a las que encuentra un equipo de tres auditores.

Bug bounty. Antes de mainnet, publica un programa (Immunefi es el estándar del sector) con severidades y pagos escalonados. La estructura típica:

- **Crítico:** robo o congelamiento directo de fondos de usuarios; paga el máximo del programa, frecuentemente como porcentaje de los fondos en riesgo con un tope.
- **Alto:** pérdida acotada o robo de fondos no depositados por usuarios (por ejemplo, de la tesorería del protocolo).
- **Medio/bajo:** griefing, denegación de servicio, desviaciones sin pérdida directa.

- **Informativo:** hallazgos de calidad de código sin impacto explotable.

El bounty debe ser proporcional al valor custodiado — un bounty máximo de 50 000 USD protegiendo 500 millones invita al atacante a elegir la otra opción. Y debe pagarse rápido y sin regateos: la reputación de pago del programa es lo que hace que el próximo investigador reporte en vez de explotar.

🚀 Estrategias de lanzamiento

El lanzamiento no es binario. Las tres estrategias arquetípicas:

Criterio	Inmutable sin admin	Upgradeable con timelock + multisig	Guarded launch progresivo
Filosofía	"El código es la ley"; máxima confianza del usuario	Capacidad de corregir y evolucionar con transparencia	Exposición limitada mientras se gana evidencia en producción
Riesgo si hay un bug	No hay corrección posible: migración total o pérdida	Corregible dentro de la ventana del timelock	Acotado por los caps: el peor caso tiene techo conocido
Riesgo de gobernanza/claves	Mínimo (no hay llaves que robar)	El multisig y el proceso de upgrade son superficie de ataque	Intermedio; los controles temporales deben expirar de verdad
Confianza que exige al usuario	Solo en el código auditado	En el código y en los firmantes del multisig	En el código, el equipo y el plan de descentralización
Ejemplo de uso típico	Primitivas simples y maduras (estilo Uniswap v2 core)	Protocolos de préstamo y sistemas complejos	Casi todo lanzamiento serio nuevo desde 2021
Trade-off central	Seguridad de gobernanza a cambio de rigidez total	Flexibilidad a cambio de un vector de ataque nuevo	Complejidad operacional a cambio de riesgo acotado

Herramientas del **guarded launch**: caps de depósito por usuario y globales que suben gradualmente, **timelocks** en toda función administrativa (el usuario puede salir antes de que un cambio se aplique), **multisig** con firmantes independientes y umbral razonable, **circuit breakers** (pausas automáticas ante anomalías: retiros masivos, desviación de oráculo) y feature flags para activar módulos por etapas. Cada control temporal necesita fecha de expiración o plan de remoción: un "lanzamiento guardado" permanente es simplemente un protocolo centralizado.

Checklist mínima antes del primer depósito real:

- El bytecode desplegado corresponde exactamente al commit auditado (verificación reproducible).

- Contratos verificados públicamente en el explorador; direcciones oficiales publicadas y firmadas.
- Multisig probado con una transacción real; firmantes ensayaron el procedimiento de emergencia.
- Monitoreo y alertas activos **antes** del lanzamiento, no después del primer susto.
- Runbook de pausa/incidente ensayado en simulacro, con tiempos medidos.
- Programa de bug bounty publicado y canal de divulgación responsable visible.

📡 Operación

Mainnet es el comienzo del trabajo, no el final. La operación seria incluye:

- **Monitoreo on-chain:** alertas sobre las invariantes del diseño (si una se rompe en producción, quieres saberlo en segundos, no en el post-mortem), transacciones anómalas, salud de oráculos y colas de retiro. Herramientas del ecosistema: Tenderly, Forta, OpenZeppelin Defender.
- **Runbooks de incidentes:** quién puede pausar, con qué firma, en cuánto tiempo, y qué se comunica mientras tanto. El programa tiene una guía dedicada: operación e incidentes. Un runbook que nunca se ensayó es una hipótesis, no un plan.
- **Gestión de claves y multisig ops:** firmantes con hardware wallets dedicadas, verificación independiente de cada transacción propuesta (los firmantes que firman lo que les ponen delante son el eslabón débil — así cayeron varios multisigs), rotación documentada y simulacros.
- **Comunicación pública:** los mejores equipos publican post-mortems técnicos completos tras cada incidente (causa raíz, línea de tiempo, fondos afectados, correcciones). Esa transparencia — practicada por equipos como los de Compound o Lido ante incidentes — es hoy la norma de los protocolos respetados y una ventaja reputacional medible.

🔁 Evolución y sunset

Evolución. Los upgrades pasan por gobernanza: propuesta pública, revisión (idealmente auditoría del diff), votación o aprobación multisig, timelock, ejecución. Las migraciones de versión (v1 → v2) suelen convivir años: v1 inmutable sigue operando mientras la liquidez migra por incentivos, no por coerción. Deprecar exige avisos amplios, herramientas de migración y mantener la capacidad de retiro indefinidamente.

Sunset. Apagar un protocolo responsablemente es una fase de diseño, no una improvisación: desactivar depósitos nuevos, garantizar retiros permanentes, renunciar a permisos administrativos o transferirlos a la comunidad, y documentar el estado final. Aquí importa la decisión de diseño tomada años antes: los contratos **pausables** permiten un cierre ordenado pero exigen confianza en quien pausa; los **inmutables** no se pueden apagar — quedarán en la cadena para siempre, con o sin frontend. Si tu protocolo

custodia fondos, "el equipo se disolvió" no puede implicar "los fondos quedaron atrapados".

Este ciclo completo — de la matriz de decisión al plan de sunset — es exactamente lo que se practica a escala en el proyecto final del programa: capstone.

⚠️ Errores y mitos frecuentes

Mito o error	Realidad
"La auditoría garantiza que el contrato es seguro"	Es una revisión experta acotada en tiempo y alcance; protocolos auditados pierden fondos cada año. Por eso: defensa en profundidad
"Funcionó estable en testnet, mainnet será igual"	En testnet no hay valor que robar ni MEV real; la ausencia de ataques no es evidencia de seguridad
"Somos descentralizados, no necesitamos operación"	Alguien monitorea, alguien propone upgrades, alguien responde incidentes; descentralizado significa responsabilidades distribuidas, no inexistentes
"Auditamos una vez y ya"	Cada cambio de código posterior al informe está sin auditar; el hash auditado y el desplegado deben coincidir
"El multisig nos protege"	Solo si los firmantes verifican independientemente lo que firman; un multisig con firmantes ciegos es teatro de seguridad
"Lanzamos sin caps para no frenar el crecimiento"	El costo de un cap es crecimiento diferido; el costo de no tenerlo es el TVL entero en el peor caso
"El bug bounty es opcional si ya auditamos"	Es la única capa que sigue activa en producción y alinea a los que encuentran bugs para reportarlos en vez de explotarlos

🔗 Referencias

- Trail of Bits, *Building Secure Contracts*: https://secure-contracts.com/
- OpenZeppelin Defender — operación y monitoreo: https://docs.openzeppelin.com/defender/
- Immunefi — bug bounties y clasificación de severidades: https://immunefi.com/
- Code4rena — contests de auditoría competitiva: https://code4rena.com/
- Foundry Book — scripts de despliegue y pruebas: https://book.getfoundry.sh/
- ethereum.org — redes y testnets: https://ethereum.org/developers/docs/networks/

🧭 Navegación

⬅️ 05 · Modelos de negocio · Índice de industria · 🏠 Programa

Laboratorios

🧪 Catálogo de 50 prácticas

⬅️ Cuaderno de laboratorios · 📚 Currículo · 🏠 Programa

Las 50 prácticas del programa, agrupadas por bloque. Cada fila enlaza al **módulo** del currículo que la sustenta y a su **resolución explicada** (cómo se resuelve, el comando, la salida esperada y el error común).

Cómo moverte: elige una práctica → abre su **Resolución** para ver el paso a paso → vuelve al **módulo** para la teoría. Las marcadas **auto** traen verificación ejecutable (`pnpm lab:*` o `node --test`); el resto produce una evidencia revisable con rúbrica.

Marca	Significado
auto	Tiene verificación ejecutable incluida
Módulo	Enlace al módulo del currículo que da la teoría
Resolución	Enlace a la guía con el paso a paso y la salida esperada

Prácticas 01–10 · Fundamentos

📖 Resolución explicada del bloque: 01-foundations.md · Teoría: módulos 01–03.

#	Práctica	Nivel	Evidencia	Módulo	Resolución
01	Matriz blockchain vs. base tradicional	inicial	ADR	00	ver
02	Historia anotada de sistemas de dinero digital	inicial	línea temporal	00	ver
03	Propiedades y límites de SHA-256	inicial	**auto**	01	ver
04	Cadena de hashes manipulada	inicial	**auto**	01	ver
05	Árbol y raíz de Merkle	inicial	**auto**	01	ver
06	Prueba de inclusión Merkle	inicial	**auto**	01	ver
07	Firma y verificación Ed25519	inicial	**auto**	01	ver
08	Amenazas de custodia de claves	inicial	threat model	01	ver
09	Propagación P2P con retrasos	inicial	**auto**	02	ver
10	Partición y reconciliación	inicial	**auto**	02	ver

Prácticas 11–20 · Consenso y Bitcoin

📖 Resolución explicada del bloque: 02-consensus-bitcoin.md · Teoría: módulos 03–04.

#	Práctica	Nivel	Evidencia	Módulo	Resolución
11	Proof of Work y dificultad	inicial	**auto**	03	ver
12	Comparación PoW, PoS y BFT	inicial	matriz	03	ver
13	Construcción de una mini blockchain	intermedio	**auto**	03	ver
14	Detección de bloque alterado	intermedio	**auto**	03	ver
15	Selección de UTXO	intermedio	**auto**	04	ver
16	Comisión y cambio Bitcoin	intermedio	**auto**	04	ver
17	Bitcoin Core en regtest	intermedio	transcript	04	ver
18	Wallet y direcciones regtest	intermedio	transcript	04	ver
19	Crear y confirmar una transacción regtest	intermedio	txid local	04	ver
20	Multisig/descriptor en regtest	intermedio	política	04	ver

Prácticas 21–30 · Desarrollo EVM

📖 Resolución explicada del bloque: 03-evm-development.md · Teoría: módulos 05–07.

#	Práctica	Nivel	Evidencia	Módulo	Resolución
21	Anatomía de una transacción pública	intermedio	informe	05	ver
22	Selector ABI	intermedio	**auto**	05	ver
23	Codificación de calldata	intermedio	**auto**	05	ver
24	Eventos y topics	intermedio	**auto**	05	ver
25	Storage, memory y calldata	intermedio	medición	05	ver
26	Estimación y comparación de gas	intermedio	tabla	05	ver
27	Vault: depósito y retiro	intermedio	**auto**	06	ver
28	Vault: fuzzing e invariantes	intermedio	**auto**	06	ver
29	Cliente de lectura con viem	intermedio	**auto**	07	ver
30	Flujo seguro de conexión de wallet	intermedio	interfaz	07	ver

Prácticas 31–40 · Profesional y seguridad

📖 Resolución explicada del bloque: 04-professional-security.md · Teoría: módulos 07–10.

#	Práctica	Nivel	Evidencia	Módulo	Resolución
31	Estados de una transacción	intermedio	**auto**	07	ver
32	ERC-20 con roles	profesional	**auto**	08	ver
33	Allowance y permit	profesional	**auto**	08	ver
34	ERC-721 y metadatos	profesional	contrato	08	ver
35	Indexador de eventos	profesional	**auto**	10	ver
36	Oráculo y dato obsoleto	profesional	**auto**	10	ver
37	Reentrancia	profesional	**auto** · exploit + fix	09	ver
38	Control de acceso	profesional	**auto** · exploit + fix	09	ver
39	Manipulación de oráculo	profesional	**auto** · exploit + fix	09	ver
40	Repetición de firmas	profesional	**auto** · exploit + fix	09	ver

Prácticas 41–50 · Avanzado y capstone

📖 Resolución explicada del bloque: 05-advanced-capstone.md · Teoría: módulos 09–15 y capstone.

#	Práctica	Nivel	Evidencia	Módulo	Resolución
41	Front-running y commit-reveal	profesional	**auto** · commit-reveal	09	ver
42	Colisión de storage en proxy	profesional	**auto** · storage layout	09	ver
43	Auditoría completa del Vault	profesional	reporte	09	ver
44	Multisig y timelock	profesional	política	11	ver
45	Propuesta y voto DAO	profesional	**auto**	11	ver
46	Comparación de rollups	avanzado	ADR	12	ver
47	Modelo de amenazas de puente	avanzado	threat model	13	ver
48	Prueba ZK conceptual	avanzado	diseño	14	ver
49	Simulación de emisión y concentración	avanzado	**auto**	15	ver
50	Capstone y defensa técnica	avanzado	producto	Capstone	ver

🧭 Navegación

🧪 Cuaderno de laboratorios · 📚 Currículo · 🎓 Capstone · 🏠 Programa

Guías 01–10 · Fundamentos

Este cuaderno cubre criptografía aplicada y sistemas distribuidos: hashes, cadenas encadenadas, Merkle, firmas Ed25519 y propagación P2P. Acompaña a los módulos criptografía, sistemas distribuidos y consenso.

⬅️ Cuaderno de laboratorios · 🧪 Catálogo · 📚 Currículo

Cada práctica exige una entrada de bitácora con hipótesis, procedimiento, resultado, explicación y límite de lo demostrado. La bitácora es la evidencia; el comando solo la produce.

#	Práctica	Tipo	Comando/entrega
01	Matriz blockchain vs. base tradicional	concepto	ADR con tres matrices
02	Historia anotada del dinero digital	concepto	línea temporal de 10 hitos
03	Propiedades y límites de SHA-256	auto	`node -e` con `sha256`
04	Cadena de hashes manipulada	auto	`pnpm lab:hash`
05	Árbol y raíz de Merkle	auto	`pnpm lab:merkle`
06	Prueba de inclusión Merkle	auto	`node --test merkle-proof.test.mjs`
07	Firma y verificación Ed25519	auto	`node digital-signature.mjs`
08	Amenazas de custodia de claves	concepto	threat model
09	Propagación P2P con retrasos	concepto	simulación de 5 nodos
10	Partición y reconciliación	concepto	informe de rama

01 · Blockchain o base tradicional

- **Objetivo:** justificar cuándo una blockchain aporta sobre una base tradicional.
- **Cómo se resuelve:**
  1. Toma los tres casos (salud, puntos internos, pagos internacionales) y aplica el ADR 001.
  2. Para cada uno completa seis ejes: autoridad, quiénes escriben, privacidad, corrección de errores, costo y estrategia de salida.
  3. Decide por caso: si hay un único escritor confiable y la privacidad es dura, una base tradicional gana; blockchain solo justifica su costo cuando hay múltiples escritores sin confianza mutua.
- **Estructura de la respuesta:** tres matrices de 6 filas y una línea de decisión por caso ("elijo X porque el eje autoridad/salida pesa más").

- **Criterio de aceptación:** cada decisión nombra el eje que la determinó, no solo la conclusión.
- **Error común:** elegir blockchain "porque es moderno" → falta el eje de autoridad que lo descarta.

02 · Historia anotada del dinero digital

- **Objetivo:** relacionar cada avance con el problema previo que resolvió.
- **Cómo se resuelve:**
  1. Ordena diez hitos desde firmas digitales y dinero electrónico hasta Bitcoin, Ethereum y rollups.
  2. Para cada hito cita una fuente primaria (paper o especificación original).
  3. Redacta una frase que enlace el hito con el defecto que corrige del anterior (p. ej. doble gasto → PoW).
- **Estructura de la respuesta:** tabla de 10 filas con `año | hito | fuente | problema que resuelve`.
- **Criterio de aceptación:** cada hito explica qué problema anterior aborda, no solo qué es.
- **Error común:** listar tecnologías sueltas → se pierde la cadena causa-efecto que se evalúa.

03 · Propiedades y límites de SHA-256

- **Objetivo:** observar determinismo y efecto avalancha, y ubicar los límites del hash.
- **Cómo se resuelve:** la función `sha256` de `hash-chain.mjs` envuelve `createHash("sha256")`. Calcula el hash de dos mensajes que difieren en un solo carácter y compara.

```
node -e "import('./labs/01-cryptography/hash-chain.mjs').then(m=>
{console.log(m.sha256('hola'));console.log(m.sha256('holA'))})"
```

```
4d186321c1a7f0f354b297e8914ab240...   # 64 hex, siempre igual para 'hola'
a7c4f...totalmente distinto...          # un solo bit de entrada cambia ~la mitad de
los bits
```

- Cada salida es de 256 bits (64 caracteres hex) e idéntica en cada corrida: determinismo.
- Cambiar un carácter reescribe cerca de la mitad de los bits: efecto avalancha, sin correlación con el original.
- **Criterio de aceptación:** distingue integridad (hash) de contraseña (KDF con sal) y de cifrado (reversible con clave).

- **Error común:** llamar "cifrado" al hash → SHA-256 no es reversible, no hay clave que lo deshaga.

04 · Cadena de hashes manipulada

- **Objetivo:** demostrar que alterar un registro rompe toda la cadena posterior.
- **Cómo se resuelve:** `buildHashChain` enlaza cada registro con `previousHash` y `verifyHashChain` recomputa. Ejecuta el lab, luego edita un registro en tu bitácora y vuelve a verificar.

```
pnpm lab:hash
```

```
┌─────────┬───────┬──────────────────┬──────────────────────┬──────────┐
│ (index) │ index │ data             │ previousHash         │ hash     │
├─────────┼───────┼──────────────────┼──────────────────────┼──────────┤
│ 0       │ 0     │ 'A paga 2 a B' │ '000...0' (64)     │ '9f..' │
│ 1       │ 1     │ 'B paga 1 a C' │ '9f..' (hash #0) │ 'c3..' │
└─────────┴───────┴──────────────────┴──────────────────────┴──────────┘
Cadena válida: true
```

- `previousHash` del bloque 0 es 64 ceros (génesis); el del bloque 1 es exactamente el `hash` del bloque 0.
- Si cambias `data` del registro 0 sin reminar, su `hash` recalculado ya no coincide y `verifyHashChain` devuelve `false`.
- **Criterio de aceptación:** explica por qué recalcular solo el hash local no basta: el enlace `previousHash` del siguiente delata la alteración.
- **Error común:** "recalculé el hash y sigue válido" → olvidaste que el bloque siguiente guarda el hash viejo.

05 · Árbol y raíz de Merkle

- **Objetivo:** resumir un conjunto de transacciones en una sola raíz.
- **Cómo se resuelve:** `merkleRoot` en `merkle-tree.mjs` hashea cada hoja y combina pares por nivel hasta quedar una raíz. Compara la raíz con 3, 4 y 5 hojas.

```
pnpm lab:merkle
```

```
{
  transactions: [ 'A→B:2', 'B→C:1', 'C→D:0.5' ],
  root: '<64 caracteres hex>'
}
```

- Con número impar de hojas, la última se duplica (`level.push(level.at(-1))`) para poder emparejar.
- La raíz es un único hash de 64 hex: cambiar cualquier hoja cambia la raíz por completo.
- **Criterio de aceptación:** explica la duplicación de hoja impar de esta implementación y su efecto en la raíz.
- **Error común:** esperar raíces iguales con 3 y 4 hojas → el número de hojas y la duplicación cambian el árbol.

06 · Prueba de inclusión Merkle

- **Objetivo:** demostrar que un elemento pertenece a la raíz sin revelar los demás.
- **Cómo se resuelve:** `buildProof` recolecta los hermanos con su lado (`left` / `right`); `verifyProof` recompone hasta la raíz. La prueba incluida verifica el índice 2 de `["a","b","c","d","e"]`.

```
node --test labs/01-cryptography/merkle-proof.test.mjs
```

```
✔ verifica inclusión sin revelar todos los elementos
# tests 1
# pass 1
```

- `verifyProof("c", proof, root)` es `true`; `verifyProof("C", proof, root)` es `false`: un solo bit invalida la prueba.
- La prueba solo demuestra pertenencia; no revela el valor de las otras hojas ni su orden.
- **Criterio de aceptación:** identifica qué NO demuestra la prueba (contenido de las demás hojas, posición absoluta).
- **Error común:** invertir `left` / `right` al recomponer → la raíz no cuadra aunque el elemento sí pertenezca.

07 · Firma y verificación Ed25519

- **Objetivo:** separar clave, control de clave e identidad legal.
- **Cómo se resuelve:** `signMessage` en `digital-signature.mjs` genera un par Ed25519, firma un mensaje y expone `verify`. Prueba a alterar el mensaje verificado.

```
node labs/01-cryptography/digital-signature.mjs
```

```
Mensaje original: true
Mensaje alterado: false
La firma autentica una clave, no una identidad legal.
```

- `verify("Autorizo la práctica local")` es `true` porque es el mensaje firmado; cualquier otro texto da `false`.
- La firma prueba que quien firmó controlaba la clave privada, no quién es esa persona.
- **Criterio de aceptación:** distingue "esta clave firmó" de "esta persona autorizó legalmente".
- **Error común:** asumir identidad → una firma válida solo vincula a una clave, transferible o robable.

08 · Amenazas de custodia de claves

- **Objetivo:** modelar riesgos de custodia en tres esquemas de wallet.
- **Cómo se resuelve:**
  1. Modela wallet móvil, hardware wallet y multisig.
  2. Por esquema lista activos, amenazas, controles y plan de recuperación.
  3. Cubre explícitamente pérdida, phishing, malware y coerción física.
- **Estructura de la respuesta:** tabla `esquema | activo | amenaza | control | recuperación` con las cuatro amenazas presentes en cada fila relevante.
- **Criterio de aceptación:** incluye las cuatro amenazas y un plan de recuperación por esquema.
- **Error común:** olvidar coerción ("llave de $5") → el multisig con umbral geográfico es su mitigación.

09 · Propagación P2P con retrasos

- **Objetivo:** distinguir propagar un mensaje de acordar un estado.
- **Cómo se resuelve:**
  1. Representa cinco nodos y entrega el mismo mensaje con latencias distintas por enlace.
  2. Registra el orden en que cada nodo lo recibe y su estado intermedio.
  3. Muestra un instante en que los nodos difieren aunque todos acabarán viendo el mensaje.
- **Estructura de la respuesta:** tabla de orden de recepción por nodo + una frase que separe propagación de consenso.

- **Criterio de aceptación:** diferencia propagación (llegar a todos) de consenso (acordar un orden).
- **Error común:** creer que recibir el mensaje ya es acordarlo → falta la regla de decisión.
- **Verificación ejecutable:** `pnpm lab:p2p` simula la red con latencias y `node --test labs/02-consensus/p2p-propagation.test.mjs` comprueba que cada nodo se queda con el camino más rápido y que dos bloques simultáneos parten la red.

10 · Partición y reconciliación

- **Objetivo:** razonar seguridad y vivacidad ante una partición de red.
- **Cómo se resuelve:**
  1. Divide la red 3/2 y deja que cada lado produzca propuestas en paralelo.
  2. Reconecta y reconcilia eligiendo una rama según una regla explícita (p. ej. más trabajo acumulado).
  3. Decide qué prioriza tu regla: seguridad (no bifurcar) o vivacidad (seguir avanzando).
- **Estructura de la respuesta:** informe con las dos ramas, la regla de reconciliación y la decisión seguridad/vivacidad.
- **Criterio de aceptación:** explica cuándo una rama puede descartarse sin violar seguridad.
- **Error común:** descartar la rama minoritaria sin regla → reorganización arbitraria, no consenso.
- **Verificación ejecutable:** `pnpm lab:particion` reproduce el corte y la reconciliación; `node --test labs/02-consensus/partition-reconciliation.test.mjs` comprueba que gana la cadena con más trabajo (aunque sea más corta) y que las transacciones del lado perdedor quedan huérfanas.

🧭 Navegación

- Siguiente: Guías 11–20 · Consenso y Bitcoin
- Cuaderno de laboratorios · Catálogo · Currículo

Guías 11–20 · Consenso y Bitcoin

Este cuaderno lleva del consenso (Proof of Work y sus alternativas) al modelo UTXO de Bitcoin y a un nodo local en regtest. Acompaña a los módulos consenso y Bitcoin.

⬅️ Cuaderno de laboratorios · 🧪 Catálogo · 📚 Currículo

Trabaja siempre en regtest: sin fondos ni direcciones reales. La bitácora registra hipótesis, comando exacto, resultado y el límite de lo demostrado.

#	Práctica	Tipo	Comando/entrega
11	Proof of Work y dificultad	auto	`pnpm lab:pow 4`
12	Comparación PoW, PoS y BFT	concepto	matriz de 5 ejes
13	Construcción de una mini blockchain	proyecto	`mini-chain.mjs` extendido
14	Detección de bloque alterado	auto	`node --test mini-chain.test.mjs`
15	Selección de UTXO	auto	`pnpm lab:utxo`
16	Comisión y cambio Bitcoin	auto	`node --test utxo-selection.test.mjs`
17	Bitcoin Core en regtest	transcript	altura y hash local
18	Wallet y direcciones regtest	transcript	dos wallets descriptor
19	Crear y confirmar una transacción	txid local	mempool → confirmada
20	Multisig/descriptor en regtest	política	descriptor 2-de-3 + PSBT

11 · Proof of Work y dificultad

- **Objetivo:** relacionar dificultad con el número esperado de intentos.
- **Cómo se resuelve:** `proof-of-work.mjs` itera `nonce` hasta que `sha256(payload:nonce)` empiece con `difficulty` ceros. Corre dificultades 2 a 5, tres veces cada una.

```
pnpm lab:pow 4
```

```
{
  difficulty: 4,
  nonce: <número de intentos hasta el hash válido>,
  hash: '0000<...>',           # empieza con 4 ceros hex
  elapsedMs: <milisegundos>
}
```

- Cada cero hexadecimal adicional multiplica por ~16 los intentos esperados; `nonce` sube en ese orden de magnitud.
- El `hash` siempre empieza con tantos ceros como `difficulty`; el tiempo varía por corrida aunque el promedio crezca.
- La dificultad está tope en 6 (`Math.min(..., 6)`) para no colgar la máquina.
- **Criterio de aceptación:** relaciona dificultad con intentos esperados, no con un tiempo exacto.
- **Error común:** concluir "tardó X, entonces la dificultad Y es lenta" → un solo dato; el trabajo es probabilístico.

12 · Comparación PoW, PoS y BFT

- **Objetivo:** comparar familias de consenso por sus garantías, no por marketing.
- **Cómo se resuelve:**
  1. Construye una matriz con cinco ejes: resistencia Sybil, finalidad, penalización, participación y gobierno.
  2. Llena PoW, PoS y BFT en cada eje con una afirmación verificable.
  3. Contrasta finalidad probabilística (PoW) con finalidad económica/absoluta (PoS/BFT).
- **Estructura de la respuesta:** tabla 3×5 con una nota por celda.
- **Criterio de aceptación:** no usa TPS como único criterio de comparación.
- **Error común:** ordenar solo por velocidad → ignora finalidad y coste de ataque.

13 · Construcción de una mini blockchain

- **Objetivo:** entender por qué una cadena minada localmente aún no es una red segura.
- **Cómo se resuelve:** `mini-chain.mjs` mina cada bloque enlazando `previousHash` y exige un prefijo de ceros. Extiéndelo con timestamp y una regla de ajuste de dificultad.
  - Agrega `timestamp` al objeto que hashea `mine`.
  - Añade una regla que suba/baje `difficulty` según el tiempo entre bloques.
  - Verifica que `isValid()` siga en `true` tras tus cambios.
- **Estructura de la respuesta:** diff del archivo + una nota que explique qué le falta para ser segura (sin red P2P ni consenso distribuido, un solo minero puede reescribirla).
- **Criterio de aceptación:** documenta por qué, pese a minar, no es una red segura.
- **Error común:** creer que minar solo da seguridad → sin nodos independientes no hay a quién resistir.

14 · Detección de bloque alterado

- **Objetivo:** demostrar que alterar un bloque invalida la cadena.

- **Cómo se resuelve:** el test mina, valida, cambia una transacción del bloque 1 y vuelve a validar.

```
node --test labs/03-mini-chain/mini-chain.test.mjs
```

```
✔ mina y detecta manipulación
# tests 1
# pass 1
```

- Antes de alterar, `isValid()` es `true`; tras cambiar `transactions[0]`, el hash recomputado no coincide y devuelve `false`.
- **Reto de bitácora:** remina el bloque alterado y observa que el bloque siguiente sigue apuntando al hash viejo, así que también se invalida.
- **Criterio de aceptación:** explica el costo acumulado de reminar toda la cola y el papel del consenso externo.
- **Error común:** reminar solo el bloque tocado → el enlace del siguiente ya no cuadra.

15 · Selección de UTXO

- **Objetivo:** seleccionar entradas suficientes para cubrir monto más comisión.
- **Cómo se resuelve:** `selectUtxos` en `utxo-selection.mjs` ordena de menor a mayor y acumula hasta cubrir `target + fee`, devolviendo el cambio.

```
pnpm lab:utxo
```

```
{
  selected: [ { id: 'a:0', value: 8000 }, { id: 'b:1', value: 12000 } ],
  total: 20000,
  target: 17000,
  fee: 1000,
  change: 2000        # total - target - fee
}
```

- Ordena ascendente y toma UTXOs hasta que `total >= target + fee`; con 8000 y 12000 llega a 20000 y no necesita el de 30000.
- El `change` es lo que vuelve a tu wallet; `total - target - fee`.
- **Reto:** implementa una variante de mayor a menor y compara el cambio y el número de entradas.
- **Criterio de aceptación:** conserva la identidad entradas = salidas + comisión (`total = target + fee + change`).

- **Error común:** olvidar sumar `fee` al objetivo → la transacción no cubre la comisión y quedaría inválida.

16 · Comisión y cambio Bitcoin

- **Objetivo:** ver que la comisión depende del tamaño (entradas), no del monto enviado.
- **Cómo se resuelve:** el test verifica la conservación con dos UTXOs y rechaza fondos insuficientes.

```
node --test labs/04-bitcoin/utxo-selection.test.mjs
```

```
✔ conserva entradas = salidas + comisión
✔ rechaza fondos insuficientes
# tests 2
# pass 2
```

- La primera prueba comprueba `total === target + fee + change`; la segunda que sin fondos `selectUtxos` lanza error.
- **Reto de bitácora:** calcula tres transacciones con 1, 2 y 3 entradas y observa que más entradas encarecen la comisión aunque el monto sea igual.
- **Criterio de aceptación:** explica por qué el monto transferido no determina por sí solo la comisión.
- **Error común:** ligar comisión al monto → la comisión escala con el peso en bytes (número de inputs).

17 · Bitcoin Core en regtest

- **Objetivo:** levantar un nodo local aislado y leer su estado.
- **Cómo se resuelve:**
  1. Sigue la guía de `labs/04-bitcoin-regtest/README.md` para arrancar `bitcoind` en regtest.
  2. Consulta versión y `getblockchaininfo`; mina bloques con `generatetoaddress`.
  3. Registra versión, altura y hash del último bloque local.
- **Estructura de la respuesta:** transcript con los tres datos (versión, altura, hash).
- **Criterio de aceptación:** no usa direcciones ni fondos reales; todo ocurre en regtest.
- **Error común:** confundir la red → verifica que `chain` sea `regtest`, nunca `main`.

18 · Wallet y direcciones regtest

- **Objetivo:** crear wallets descriptor y entender la madurez coinbase.
- **Cómo se resuelve:**
    1. Crea dos wallets descriptor y genera direcciones en cada una.
    2. Mina hacia una y observa que la recompensa coinbase no es gastable hasta 100 confirmaciones.
    3. Exporta y guarda los descriptors como respaldo.
- **Estructura de la respuesta:** transcript con las dos wallets, direcciones y una nota de backup.
- **Criterio de aceptación:** explica la madurez coinbase (100 bloques) y el backup de descriptors.
- **Error común:** intentar gastar la coinbase recién minada → aún inmadura.

19 · Crear y confirmar una transacción regtest

- **Objetivo:** recorrer el ciclo de una transacción de mempool a confirmada.
- **Cómo se resuelve:**
    1. Transfiere entre las dos wallets e inspecciona la mempool antes de minar.
    2. Mina un bloque y confirma la transacción.
    3. Registra txid, entradas, salidas, comisión y cambio.
- **Estructura de la respuesta:** txid local + desglose de inputs/outputs/fee/change.
- **Criterio de aceptación:** la transacción pasa de mempool a confirmada con los datos registrados.
- **Error común:** leer la comisión antes de minar y darla por final → puede cambiar hasta confirmarse.

20 · Multisig/descriptor en regtest

- **Objetivo:** construir una política 2-de-3 y demostrar que una sola clave no basta.
- **Cómo se resuelve:**
    1. Crea un descriptor multisig 2-de-3 y una PSBT que gaste de él.
    2. Firma con una sola clave e intenta finalizar: falla.
    3. Firma con la segunda clave y finaliza correctamente.
- **Estructura de la respuesta:** política del descriptor + evidencia de que una firma no finaliza y dos sí.
- **Criterio de aceptación:** demuestra que una clave no puede finalizar y documenta la recuperación.
- **Error común:** perder un descriptor sin backup → sin él no se reconstruye la política de gasto.

🧭 Navegación

- Anterior: Guías 01–10 · Fundamentos
- Siguiente: Guías 21–30 · EVM y desarrollo
- Cuaderno de laboratorios · Catálogo · Currículo

Guías 21–30 · EVM y desarrollo

Este cuaderno entra en la máquina virtual de Ethereum: selectores, calldata, eventos, gas y el primer contrato con Foundry. Acompaña a los módulos Ethereum y EVM y Solidity y Foundry.

> ⬅️ Cuaderno de laboratorios · 🧪 Catálogo · 📚 Currículo

Aquí separas los hechos on-chain (bytes, gas, eventos) de las inferencias de identidad. La bitácora registra el compilador, el optimizer y la medición exacta.

#	Práctica	Tipo	Comando/entrega
21	Anatomía de una transacción pública	concepto	informe explorador + RPC
22	Selector ABI	auto	`pnpm lab:abi`
23	Codificación de calldata	auto	`pnpm lab:abi` + decode manual
24	Eventos y topics	auto	descomposición de `Transfer`
25	Storage, memory y calldata	medición	tres variantes + gas
26	Estimación y comparación de gas	tabla	`forge test --gas-report`
27	Vault: depósito y retiro	Foundry	`forge test -vv`
28	Vault: fuzzing e invariantes	Foundry	fuzz + invariant
29	Cliente de lectura con viem	TypeScript	`createPublicClient`
30	Flujo seguro de conexión de wallet	interfaz	dApp con validación de red

21 · Anatomía de una transacción pública

- **Objetivo:** leer una transacción real separando datos de identidad.
- **Cómo se resuelve:**
  1. Elige una transacción en un explorador y recupera el mismo hash por RPC (`eth_getTransactionByHash`).
  2. Identifica `from`, `to`, `value`, `input` (calldata), `gas` y estado.
  3. Marca qué es hecho on-chain y qué es una inferencia (un `from` no es una identidad legal).
- **Estructura de la respuesta:** informe con los campos crudos + una columna "hecho vs. inferencia".
- **Criterio de aceptación:** separa hechos on-chain de inferencias de identidad.
- **Error común:** llamar "dueño" a una dirección → solo sabes qué clave firmó.

22 · Selector ABI

- **Objetivo:** obtener el selector de 4 bytes de una función.
- **Cómo se resuelve:** `abi-selector.mjs` mapea firmas canónicas a sus 4 bytes y explica que la EVM usa Keccak-256.

```
pnpm lab:abi
```

```
transfer(address,uint256) 0xa9059cbb
balanceOf(address) 0x70a08231
approve(address,uint256) 0x095ea7b3
totalSupply() 0x18160ddd
EVM usa Keccak-256, que no es idéntico al SHA3-256 estandarizado.
```

- El selector es los primeros 4 bytes del Keccak-256 de la firma canónica (sin nombres de parámetro ni espacios).
- Verificación adicional: `node --test labs/05-evm/abi-selector.test.mjs` comprueba `transfer` → `0xa9059cbb` y `balanceOf` → `0x70a08231`.
- Una firma no mapeada lanza error pidiendo calcular Keccak-256 a mano.
- **Criterio de aceptación:** explica por qué Keccak-256 (EVM) no es idéntico al SHA3-256 estandarizado (distinta constante de padding).
- **Error común:** usar SHA3-256 de librería y obtener otro selector → no es el Keccak original.

23 · Codificación de calldata

- **Objetivo:** construir la calldata completa de `transfer(address,uint256)`.
- **Cómo se resuelve:**
  1. Toma el selector `0xa9059cbb` de la práctica 22 (`pnpm lab:abi`).
  2. Codifica cada argumento como una palabra de 32 bytes: la dirección rellenada a la izquierda con ceros y el monto en hex a 32 bytes.
  3. Concatena `selector + palabra_dirección + palabra_monto` y decodifícalo de vuelta para comprobar.

```
pnpm lab:abi
```

```
0xa9059cbb                                                          # 4 bytes: selector
000000000000000000000000<20 bytes de dirección>                    # palabra 1: address
(padded)
0000000000000000000000000000000000000000000000000000000000000064  # palabra 2: uint256
= 100
```

- La calldata mide `4 + 32*n` bytes; aquí 4 + 32 + 32 = 68 bytes.
- Decodificar es partir por longitudes fijas: 4 bytes de selector y luego palabras de 32.
- **Criterio de aceptación:** identifica selector y las dos palabras de 32 bytes y decodifica el monto.
- **Error común:** no rellenar la dirección a 32 bytes → el decodificador lee campos corridos.

24 · Eventos y topics

- **Objetivo:** descomponer un log en firma, `topics` indexados y `data`.
- **Cómo se resuelve:**
  1. Toma un evento `Transfer(address indexed from, address indexed to, uint256 value)`.
  2. `topic[0]` es el Keccak-256 de la firma; los parámetros `indexed` van en `topic[1..]`; lo no indexado va en `data`.
  3. Explica qué puede filtrar un nodo eficientemente (los topics) y qué obliga a leer `data`.
- **Estructura de la respuesta:** tabla `topic0 | topic1 (from) | topic2 (to) | data (value)` con la separación justificada.
- **Criterio de aceptación:** explica qué campos permiten filtrado eficiente y por qué (indexación de topics).
- **Error común:** poner `value` como indexed en el análisis → no lo está; vive en `data`.

25 · Storage, memory y calldata

- **Objetivo:** medir el impacto de la ubicación de datos en gas.
- **Cómo se resuelve:**
  1. Implementa tres variantes de una operación sobre arrays: leyendo de `storage`, copiando a `memory` y usando `calldata`.
  2. Mide el gas de cada una con Foundry (`forge test --gas-report`).
  3. Explica vida útil y mutabilidad: `storage` persiste y es caro, `memory` es temporal, `calldata` es de solo lectura y barato.
- **Estructura de la respuesta:** tabla `variante | ubicación | gas | mutable?`.
- **Criterio de aceptación:** relaciona el gas medido con la ubicación y su mutabilidad.
- **Error común:** copiar `calldata` a `memory` sin necesidad → gasto extra evitable.

26 · Estimación y comparación de gas

- **Objetivo:** comparar costos entre patrones equivalentes.
- **Cómo se resuelve:**
  1. Compara acceso frío vs. caliente (EIP-2929), escribir cero vs. no cero, y emitir evento vs. escribir storage.
  2. Usa `forge test --gas-report` y registra la versión del compilador y del optimizer.
  3. Interpreta: el primer `SLOAD` / `SSTORE` a un slot es más caro (frío) que los siguientes (caliente).
- **Estructura de la respuesta:** tabla comparativa con la config de compilación anotada.
- **Criterio de aceptación:** registra versión del compilador y estado del optimizer junto a cada medición.
- **Error común:** comparar medidas con optimizer distinto → los números no son comparables.

27 · Vault: depósito y retiro

- **Objetivo:** trazar el ciclo depósito → retiro en un contrato con Foundry.
- **Cómo se resuelve:**
  1. En `labs/06-solidity-vault`, ejecuta las pruebas con verbosidad para ver las trazas.
  2. Sigue cómo el depósito actualiza el balance interno y el retiro lo descuenta antes de transferir.
  3. Anota los eventos emitidos en cada paso.

```
forge test -vv
```

```
[PASS] test_Deposit() ...
[PASS] test_Withdraw() ...
Traza: Deposit(usuario, monto) → balance += monto ; Withdraw → balance -= monto →
transfer
```

- **Criterio de aceptación:** la traza muestra el orden efecto-interacción y los eventos coherentes.
- **Error común:** transferir antes de actualizar el balance → abre la puerta a reentrancia (ver práctica 37).

28 · Vault: fuzzing e invariantes

- **Objetivo:** validar propiedades que deben cumplirse ante cualquier entrada.
- **Cómo se resuelve:**
  1. Define el rango de inputs y los supuestos (`vm.assume`) para el fuzzer.
  2. Formula dos invariantes, p. ej. "la suma de balances nunca excede el balance del contrato".
  3. Ejecuta las pruebas de fuzz e invariantes y confirma que fallan si rompes la propiedad a propósito.
- **Estructura de la respuesta:** dos invariantes con asserts significativos y el reporte de corridas del fuzzer.
- **Criterio de aceptación:** una prueba sin asserts útiles no cuenta como evidencia.
- **Error común:** invariante trivial (`true == true`) → no prueba nada.

29 · Cliente de lectura con viem

- **Objetivo:** leer estado de un contrato y manejar fallos de red.
- **Cómo se resuelve:**
  1. Crea un `createPublicClient` de viem apuntando al RPC del proyecto.
  2. Lee un dato de la campaña (p. ej. total recaudado) con `readContract` .
  3. Maneja explícitamente tres fallos: RPC caído, contrato inexistente y chain ID incorrecto.
- **Estructura de la respuesta:** script TypeScript con los tres caminos de error controlados.
- **Criterio de aceptación:** lee el dato y degrada con un mensaje claro en cada fallo.
- **Error común:** asumir que el RPC siempre responde → sin manejo, la app se rompe en silencio.
- **Verificación ejecutable:** `pnpm lab:montos` y `node --test labs/07-dapps/token-amounts.test.mjs` comprueban la conversión según los `decimals` reales del token y que nunca se trunca en silencio.

30 · Flujo seguro de conexión de wallet

- **Objetivo:** conectar una wallet mostrando contexto antes de firmar.
- **Cómo se resuelve:**
  1. Conecta la dApp del proyecto y muestra red, contrato, cuenta, monto y una simulación de la operación.
  2. Compara el chain ID conectado con el esperado.
  3. Rechaza la firma si la red no coincide, con un mensaje explícito.
- **Estructura de la respuesta:** interfaz que expone el contexto y bloquea la firma en red incorrecta.
- **Criterio de aceptación:** rechaza firmar si la red no coincide con la esperada.

- **Error común:** firmar sin validar la red → la transacción se envía a la cadena equivocada.

🧭 Navegación

- Anterior: Guías 11–20 · Consenso y Bitcoin
- Siguiente: Guías 31–40 · Profesional y seguridad
- Cuaderno de laboratorios · Catálogo · Currículo

Guías 31–40 · Profesional y seguridad

Este cuaderno pasa de construir a atacar y defender: tokens con roles, allowance, oráculos y las vulnerabilidades clásicas con su exploit y su corrección. Acompaña a los módulos tokens y seguridad.

⬅️ Cuaderno de laboratorios · 🧪 Catálogo · 📚 Currículo

Cada vulnerabilidad se demuestra dos veces: un PoC que la explota y un parche con prueba de regresión. Sin exploit reproducible no hay evidencia.

#	Práctica	Tipo	Comando/entrega
31	Estados de una transacción	máquina de estados	diagrama de 8 estados
32	ERC-20 con roles	Foundry	pruebas de `labs/08-protocols`
33	Allowance y permit	threat model	aprobación exacta + permit
34	ERC-721 y metadatos	contrato	metadata mutable/inmutable
35	Indexador de eventos	servicio	`apps/event-indexer` reanudable
36	Oráculo y dato obsoleto	pruebas	rango + circuit breaker
37	Reentrancia	exploit + fix	atacante + CEI/guard + regresión
38	Control de acceso	exploit + fix	toma de control + two-step
39	Manipulación de oráculo	exploit + fix	spot vs. TWAP
40	Repetición de firmas	exploit + fix	dominio + nonce + deadline

31 · Estados de una transacción

- **Objetivo:** modelar el ciclo de vida completo de una transacción.
- **Cómo se resuelve:**
  1. Define ocho estados: preparado, simulado, esperando firma, rechazado, enviado, reemplazado, confirmado y revertido.
  2. Traza las transiciones válidas (p. ej. enviado → reemplazado por fee-bump; enviado → revertido).
  3. Marca los estados terminales y los reintentables.
- **Estructura de la respuesta:** máquina de estados con transiciones etiquetadas.
- **Criterio de aceptación:** incluye reemplazo (RBF) y revertido como caminos distintos de confirmado.
- **Error común:** tratar "enviado" como final → aún puede reemplazarse o revertir.
- **Verificación ejecutable:** `pnpm lab:tx` imprime la máquina de estados y `node --test labs/07-dapps/tx-lifecycle.test.mjs` comprueba que desde `pendiente` hay cuatro

finales posibles y que ningún estado se queda sin salida.

32 · ERC-20 con roles

- **Objetivo:** verificar suministro, roles y transferencia de un token.
- **Cómo se resuelve:**
  1. Ejecuta las pruebas del token en `labs/08-protocols`.
  2. Comprueba el suministro inicial, que solo el rol autorizado acuña/quema y que las transferencias respetan balances.
  3. Anota qué ocurre cuando una cuenta sin rol intenta una acción privilegiada (revert).
- **Estructura de la respuesta:** salida de pruebas + tabla `acción | rol requerido | resultado sin rol`.
- **Criterio de aceptación:** verifica suministro, roles y transferencia con pruebas que pasan.
- **Error común:** conceder mint a cualquiera → inflación no controlada.

33 · Allowance y permit

- **Objetivo:** demostrar el riesgo de la aprobación ilimitada y su mitigación.
- **Cómo se resuelve:**
  1. Aprueba `type(uint256).max` a un contrato y muestra que puede vaciar el saldo aprobado en cualquier momento.
  2. Propón tres controles: aprobación exacta, revocación (`approve(0)`) y `permit` con nonce y deadline.
  3. Explica por qué `permit` evita una transacción previa de aprobación.
- **Estructura de la respuesta:** threat model con activo, amenaza (allowance persistente) y los tres controles.
- **Criterio de aceptación:** propone aprobación exacta, revocación y permit con nonce/deadline.
- **Error común:** aprobar ilimitado "por comodidad" → la allowance sobrevive a la operación.
- **Verificación ejecutable:** `node --test labs/07-dapps/token-amounts.test.mjs` comprueba que una allowance infinita expone el saldo entero y que lo recomendable es autorizar exactamente lo necesario.

34 · ERC-721 y metadatos

- **Objetivo:** decidir qué representa realmente un NFT y dónde vive su metadata.
- **Cómo se resuelve:**
  1. Diseña dos variantes: metadata mutable (URI cambiable) e inmutable (hash on-chain o IPFS fijo).

    2. Analiza disponibilidad (¿quién sirve la imagen?), propiedad intelectual y qué otorga el token.
    3. Concluye qué garantiza on-chain y qué depende de un servidor externo.
- **Estructura de la respuesta:** contrato + nota que separe "el token" de "el contenido apuntado".
- **Criterio de aceptación:** explica disponibilidad, propiedad intelectual y qué representa el token.
- **Error común:** creer que poseer el NFT es poseer la imagen → solo posees el apuntador.

35 · Indexador de eventos

- **Objetivo:** construir un indexador que reanude sin perder ni duplicar eventos.
- **Cómo se resuelve:**
    1. Ejecuta `apps/event-indexer` y deja que persista el último bloque procesado.
    2. Detén el proceso, genera nuevos eventos y reanuda.
    3. Verifica que retoma desde el último bloque persistido, sin releer ni saltar.
- **Estructura de la respuesta:** servicio con checkpoint + evidencia de reanudación idempotente.
- **Criterio de aceptación:** reanuda desde el último bloque persistido sin duplicar.
- **Error común:** guardar el checkpoint antes de procesar → se pierden eventos si cae en medio.

36 · Oráculo y dato obsoleto

- **Objetivo:** proteger un consumidor de precios inválidos u obsoletos.
- **Cómo se resuelve:**
    1. Prueba cuatro casos: precio válido, cero, obsoleto (timestamp viejo) y actualización no autorizada.
    2. Añade validación de rango y un circuit breaker que rechace datos fuera de banda o vencidos.
    3. Confirma que el consumidor revierte ante cada caso inválido.
- **Estructura de la respuesta:** pruebas de los cuatro casos + la lógica de rango y breaker.
- **Criterio de aceptación:** rechaza cero, obsoleto y no autorizado; acepta solo el válido en rango.
- **Error común:** confiar en el último precio sin mirar su timestamp → dato obsoleto usado como fresco.

37 · Reentrancia

- **Objetivo:** reproducir un drenaje por reentrancia y corregirlo.
- **Cómo se resuelve:**

    1. Escribe un contrato atacante local cuyo `receive` vuelva a llamar a `withdraw`.
    2. Reproduce el drenaje del saldo antes de que el balance se actualice.
    3. Aplica Checks-Effects-Interactions y/o un guard de reentrancia; añade una prueba de regresión.
- **Estructura de la respuesta:** PoC del drenaje + parche + test que falla sin el fix y pasa con él.
- **Criterio de aceptación:** el exploit drena antes del fix y la regresión lo bloquea después.
- **Error común:** poner el guard pero seguir transfiriendo antes de actualizar estado → sigue vulnerable a otras rutas.

38 · Control de acceso

- **Objetivo:** tomar control de un contrato mal protegido y corregirlo.
- **Cómo se resuelve:**
    1. Identifica una función privilegiada sin verificación de rol y toma el control (cambia el owner).
    2. Corrige con verificación de rol y una transferencia de propiedad en dos pasos (propose/accept).
    3. Verifica que la transferencia en dos pasos impide capturas por dirección errónea.
- **Estructura de la respuesta:** exploit de toma de control + fix con two-step ownership.
- **Criterio de aceptación:** verifica la transferencia en dos pasos en la corrección.
- **Error común:** `transferOwnership` en un paso a una dirección equivocada → propiedad perdida.

39 · Manipulación de oráculo

- **Objetivo:** mostrar cómo un precio spot manipulable rompe un protocolo.
- **Cómo se resuelve:**
    1. Modela un préstamo que valora colateral con el precio spot de un pool.
    2. Manipula el spot con un swap grande (préstamo temporal) y extrae valor.
    3. Compara mitigaciones: TWAP y fuente externa, con las nuevas confianzas que introduce cada una.
- **Estructura de la respuesta:** simulación del ataque + comparación spot vs. TWAP vs. externo.
- **Criterio de aceptación:** compara TWAP y fuente externa con sus nuevos supuestos de confianza.
- **Error común:** usar el spot de un pool de baja liquidez → barato de mover en un bloque.

40 · Repetición de firmas

- **Objetivo:** impedir que una firma válida se reutilice.
- **Cómo se resuelve:**
  1. Intenta repetir una firma en el mismo contrato, en otro contrato y en otro chain ID.
  2. Observa qué repeticiones tienen éxito sin protección.
  3. Añade separación de dominio (EIP-712), nonce y deadline; vuelve a intentar los tres casos.
- **Estructura de la respuesta:** exploit de replay + fix con dominio, nonce y deadline.
- **Criterio de aceptación:** la aceptación exige dominio, nonce y deadline verificados.
- **Error común:** firmar sin chain ID en el dominio → la firma se replica entre redes.

🧭 Navegación

- Anterior: Guías 21–30 · EVM y desarrollo
- Siguiente: Guías 41–50 · Avanzado y capstone
- Cuaderno de laboratorios · Catálogo · Currículo

Guías 41–50 · Avanzado y capstone

Este cuaderno cierra el programa: front-running, proxies, auditoría, gobernanza, rollups, puentes, ZK, tokenomics y el proyecto final con defensa técnica. Acompaña al módulo arquitectura avanzada y al capstone.

⬅️ Cuaderno de laboratorios · 🧪 Catálogo · 📚 Currículo

Aquí integras todo lo anterior en decisiones de arquitectura y en un producto defendible. Cada entrega nombra sus supuestos de confianza y sus límites.

#	Práctica	Tipo	Comando/entrega
41	Front-running y commit-reveal	simulación	mempool local + commit-reveal
42	Colisión de storage en proxy	informe	layouts V1/V2 + verificación
43	Auditoría completa del Vault	reporte	plantilla de auditoría
44	Multisig y timelock	política	2-de-3 + demora + emergencia
45	Propuesta y voto DAO	simulación	`SimpleGovernor` completo
46	Comparación de rollups	ADR	optimista vs. ZK, 6 ejes
47	Modelo de amenazas de puente	threat model	lock/mint + burn/release
48	Prueba ZK conceptual	diseño	mayoría de edad
49	Simulación de emisión y concentración	auto	`pnpm lab:tokenomics`
50	Capstone y defensa técnica	producto	todas las puertas del capstone

41 · Front-running y commit-reveal

- **Objetivo:** reproducir un front-run y mitigarlo con commit-reveal.
- **Cómo se resuelve:**
  1. En una mempool local, observa una respuesta en claro y cópiala con más fee para adelantarla.
  2. Implementa commit-reveal: primero envías `hash(respuesta+sal)`, luego revelas.
  3. Identifica qué metadata sigue visible (dirección, timing, tamaño) pese al commit.
- **Estructura de la respuesta:** simulación del adelantamiento + esquema commit-reveal y su metadata residual.
- **Criterio de aceptación:** identifica la metadata aún visible tras el commit.
- **Error común:** revelar sin sal → el commit es adivinable por fuerza bruta.

42 · Colisión de storage en proxy

- **Objetivo:** demostrar cómo un upgrade desplaza slots y corrompe estado.
- **Cómo se resuelve:**
  1. Compara los layouts de storage de V1 y V2 de un contrato tras un proxy.
  2. Introduce una variable nueva en medio de V2 y muestra el slot desplazado leyendo un valor corrupto.
  3. Agrega una verificación previa a upgrades (comparar layouts o usar gaps de storage).
- **Estructura de la respuesta:** informe con ambos layouts, el slot desplazado y la verificación propuesta.
- **Criterio de aceptación:** demuestra el slot desplazado y añade la verificación previa al upgrade.
- **Error común:** insertar variables en medio del layout → todo lo posterior se corre un slot.

43 · Auditoría completa del Vault

- **Objetivo:** producir un reporte de auditoría estructurado.
- **Cómo se resuelve:**
  1. Usa `assessments/audit-report-template.md` sobre el contrato Community Funding.
  2. Define alcance, invariantes esperadas y prueba cada una con un PoC.
  3. Clasifica hallazgos por severidad y documenta el riesgo residual tras las correcciones.
- **Estructura de la respuesta:** reporte con alcance, invariantes, PoCs y riesgo residual.
- **Criterio de aceptación:** incluye alcance, invariantes, PoC y riesgo residual.
- **Error común:** listar hallazgos sin PoC → no se distingue riesgo real de teórico.

44 · Multisig y timelock

- **Objetivo:** diseñar una gobernanza operativa resistente a pérdida y compromiso.
- **Cómo se resuelve:**
  1. Diseña un 2-de-3 con demora (timelock) para acciones sensibles.
  2. Añade cancelación de propuestas en cola y una vía de emergencia acotada.
  3. Simula la pérdida de un firmante y el compromiso de otro; muestra que el sistema sobrevive a uno.
- **Estructura de la respuesta:** política con umbral, demora, cancelación y emergencia + los dos escenarios simulados.
- **Criterio de aceptación:** simula pérdida y compromiso de firmante y el sistema resiste.
- **Error común:** emergencia sin límites → se convierte en puerta trasera.

45 · Propuesta y voto DAO

- **Objetivo:** recorrer el ciclo completo de gobernanza on-chain.
- **Cómo se resuelve:**
  1. Ejecuta el protocolo `SimpleGovernor`: crea una propuesta.
  2. Vota, alcanza el quórum y espera el periodo de votación/timelock.
  3. Ejecuta la propuesta aprobada y verifica el efecto.
- **Estructura de la respuesta:** simulación con propuesta, voto, quórum, espera y ejecución.
- **Criterio de aceptación:** cubre propuesta, voto, quórum, espera y ejecución.
- **Error común:** ejecutar antes de cumplir el timelock → la propuesta no procede.

46 · Comparación de rollups

- **Objetivo:** decidir entre un rollup optimista y uno ZK con criterios técnicos.
- **Cómo se resuelve:**
  1. Compara ambos en seis ejes: seguridad heredada, disponibilidad de datos, secuenciador, sistema de prueba, retiro y mecanismo de escape.
  2. Contrasta ventana de fraude (optimista) contra validez inmediata (ZK).
  3. Redacta una decisión (ADR) según un caso de uso concreto.
- **Estructura de la respuesta:** ADR con la matriz de 6 ejes y la decisión justificada.
- **Criterio de aceptación:** cubre los seis ejes y decide según el caso, no por moda.
- **Error común:** elegir por TPS anunciado → ignora finalidad y salida de emergencia.

47 · Modelo de amenazas de puente

- **Objetivo:** enumerar los riesgos de un puente cross-chain.
- **Cómo se resuelve:**
  1. Modela los dos flujos: lock/mint (bloquear origen, acuñar destino) y burn/release.
  2. Enumera amenazas: conjunto de validadores, replay, finalidad divergente, upgrade malicioso y liquidez.
  3. Asigna un control a cada amenaza.
- **Estructura de la respuesta:** threat model con los dos flujos y la tabla amenaza→control.
- **Criterio de aceptación:** enumera validadores, replay, finalidad, upgrade y liquidez.
- **Error común:** asumir finalidad igual en ambas cadenas → un reorg en origen invalida el mint.

48 · Prueba ZK conceptual

- **Objetivo:** diseñar una prueba de conocimiento cero de mayoría de edad.
- **Cómo se resuelve:**

1. Define el problema: probar edad ≥ 18 sin revelar la fecha exacta.
2. Identifica emisor (quién certifica), witness (dato privado), señales públicas (el umbral) y revocación.
3. Analiza qué metadata puede filtrar el flujo aunque la prueba sea de conocimiento cero.

- **Estructura de la respuesta:** diseño con emisor, witness, señales públicas, revocación y metadata.
- **Criterio de aceptación:** identifica emisor, witness, señales públicas, revocación y metadata.
- **Error común:** exponer la fecha como señal pública → deja de ser conocimiento cero.

49 · Simulación de emisión y concentración

- **Objetivo:** proyectar suministro y reparto bajo emisión compuesta.
- **Cómo se resuelve:** `simulate` en `supply-simulator.mjs` valida que las asignaciones sumen 1 y proyecta `initial * (1 + inflación)^año`, repartiendo por tenedor.

```
pnpm lab:tokenomics
```

```
┌─────────┬──────┬─────────┬──────────────┐
│ (index) │ year │ supply  │ holders      │
├─────────┼──────┼─────────┼──────────────┤
│ 0       │ 0    │ 1000000 │ [Object]     │  # community 500000, team 200000, treasury
300000
│ 1       │ 1    │ 1030000 │ [Object]     │  # supply × 1.03 cada año
│ ...     │ ...  │ ...     │ [Object]     │
│ 5       │ 5    │ 1159274 │ [Object]     │
└─────────┴──────┴─────────┴──────────────┘
```

- `supply` crece de forma compuesta: 1.000.000 × 1,03^año; el año 5 ronda 1.159.274.
- Cada tenedor recibe `supply × su share`; las proporciones (50/20/30) se mantienen aunque el suministro crezca.
- Si las asignaciones no suman 1, `simulate` lanza `"Las asignaciones deben sumar 1"`.
- Verificación: `node --test labs/15-tokenomics/supply-simulator.test.mjs` comprueba que con 100 al 10% en 2 años el suministro es 121 y que un reparto inválido lanza error.
- **Reto:** cambia emisión y asignación; calcula concentración (share del mayor tenedor) y dilución de los demás.
- **Criterio de aceptación:** relaciona la inflación con la dilución y el reparto con la concentración.

- **Error común:** asignaciones que no suman 1 → la simulación aborta antes de proyectar.

50 · Capstone y defensa técnica

- **Objetivo:** integrar todo el programa en un producto defendible.
- **Cómo se resuelve:**
  1. Entrega el paquete completo: problema, ADR, contratos, interfaz, pruebas, auditoría, operación, costos y defensa.
  2. Aplica todas las puertas de `capstone/README.md` antes de presentar.
  3. Prepara la defensa técnica: justifica cada decisión con evidencia de las prácticas previas.
- **Estructura de la respuesta:** producto con los nueve componentes y las puertas del capstone superadas.
- **Criterio de aceptación:** aplica todas las puertas de `capstone/README.md`.
- **Error común:** entregar código sin ADR ni costos → falta la justificación que exige la defensa.

🧭 Navegación

- Anterior: Guías 31–40 · Profesional y seguridad
- Cuaderno de laboratorios · Catálogo · Currículo

Decisiones de arquitectura (ADR)

Registros de decisiones de arquitectura (ADRs)

⬅️ Volver al programa · 📚 Currículo

¿Qué es un ADR?

Un ADR (*Architecture Decision Record*) es un documento breve que captura **una** decisión de arquitectura: el contexto que la provocó, las opciones evaluadas, la elección tomada, sus consecuencias y las señales que obligarían a reconsiderarla. No es documentación exhaustiva: es la memoria de *por qué* el sistema es como es.

¿Por qué esta industria los usa tanto?

En el ecosistema blockchain los ADRs pesan más que en el software tradicional por dos razones:

- **Coordinación asíncrona y distribuida.** Los equipos de protocolos y DAOs trabajan repartidos por el mundo, muchas veces sin empresa central. Un ADR permite que cualquiera entienda una decisión sin asistir a la reunión donde se tomó (los EIPs de Ethereum y los ADRs del Cosmos SDK siguen esta lógica).
- **Código inmutable o carísimo de cambiar.** Un contrato desplegado no se parchea con un `git push`. Cuando revertir cuesta millones o es directamente imposible, documentar la decisión *antes* de ejecutarla deja de ser burocracia y pasa a ser gestión de riesgo.

Los 6 ADRs del programa

Cada ADR modela una decisión recurrente al diseñar un sistema con (o sin) blockchain. Son **guías educativas**: muestran cómo razonar, no imponen una respuesta única.

ADR	Pregunta que responde	Decisión por defecto del programa
001 · Blockchain vs. base de datos	¿Este problema necesita blockchain o basta una base de datos?	Base de datos, salvo que se cumplan criterios estrictos
002 · Pública vs. permisionada	Si hay blockchain, ¿red pública o de consorcio?	Pública (L2) con privacidad selectiva
003 · Datos on-chain vs. off-chain	¿Qué datos van a la cadena y cuáles fuera?	On-chain solo lo que otros contratos verifican; PII jamás
004 · Inmutabilidad vs. upgrades	¿Contrato inmutable o actualizable?	Inmutable con *guarded launch*; UUPS + timelock si el dominio lo exige
005 · L1, L2 o appchain	¿Dónde desplegar: L1, rollup general o cadena propia?	L2 rollup de propósito general
006 · Token propio	¿Emitir un token o usar activos existentes?	No emitir token salvo mecanismo de valor claro

Cómo escribir tu propio ADR

Plantilla resumida en 6 puntos:

1. **Título y estado.** Un número secuencial, un título en forma de decisión y un estado (`propuesto`, `aceptado`, `reemplazado por ADR-NNN`).
2. **Contexto.** El problema real y las restricciones (técnicas, regulatorias, de negocio) en 1-2 párrafos. Sin contexto, la decisión parece arbitraria en seis meses.
3. **Opciones.** Al menos dos alternativas reales, comparadas con los mismos criterios. Si solo hay una opción, no hay decisión que registrar.
4. **Decisión.** Qué se elige y el razonamiento que inclinó la balanza. Una frase clara, no un ensayo.
5. **Consecuencias.** Lo bueno *y* lo malo que se acepta. Un ADR sin consecuencias negativas es marketing, no ingeniería.
6. **Señales para reconsiderar.** Qué cambio del entorno (costos, regulación, madurez tecnológica) invalidaría la decisión. Esto convierte el ADR en un documento vivo.

Para profundizar, consulta la bibliografía del programa.

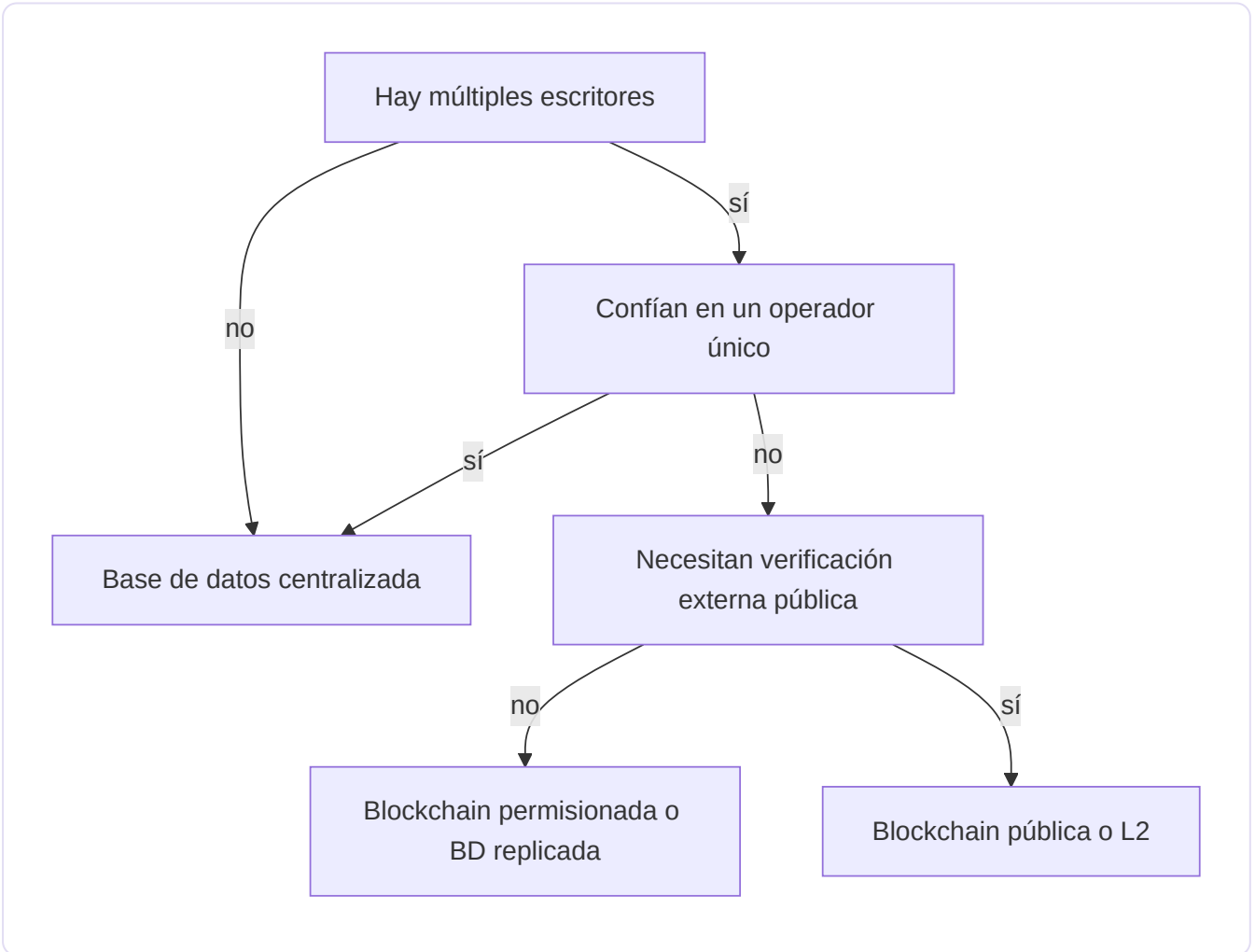
ADR-001 · ¿Blockchain o base de datos?

> **Estado:** guía educativa · **Ámbito:** arquitectura de datos · ⬅️ Índice de ADRs

Contexto

Varios participantes necesitan registrar y consultar cambios sobre un mismo conjunto de datos. La pregunta de fondo no es "¿podemos usar blockchain?" sino "¿está justificado el costo del consenso compartido?". Una blockchain replica cada escritura en todos los nodos, sacrifica rendimiento y privacidad, y complica la corrección de errores; a cambio elimina la necesidad de confiar en un operador único. Si esa confianza ya existe (o puede establecerse por contrato legal), una base de datos tradicional es casi siempre la respuesta correcta.

Este ADR formaliza el árbol de decisión que el programa presenta en el módulo 00: la mayoría de los proyectos que evalúan blockchain terminan, correctamente, en una base de datos.



Opciones

Criterio	BD centralizada	BD replicada multi-org	Blockchain permisionada	Blockchain pública / L2
Confianza requerida	Total en un operador	Alta entre organizaciones	Media: consorcio conocido	Mínima: reglas del protocolo
Verificabilidad externa	Nula (auditoría a posteriori)	Baja	Media: solo miembros	Alta: cualquiera verifica
Corrección de errores	Trivial (UPDATE)	Coordinada	Posible por gobernanza	Muy difícil o imposible
Privacidad	Total	Alta	Configurable	Baja por defecto
Costo por escritura	Muy bajo	Bajo	Medio (operación de nodos)	Variable: consúltalo en vivo

Criterios de decisión

- ¿Hay **múltiples escritores** que no confían plenamente entre sí ni en un tercero neutral?
- ¿Necesitan actores **externos** (usuarios, reguladores, competidores) verificar el registro sin pedir permiso?
- ¿Es aceptable que los **errores no puedan corregirse** con un simple UPDATE, sino con transacciones compensatorias?
- ¿Los datos pueden ser **públicos o seudónimos**, o hay información sensible que jamás debería replicarse?
- ¿El valor de eliminar al intermediario **supera el costo** en rendimiento, operación y gestión de claves?

Si respondes "no" a las dos primeras preguntas, la decisión ya está tomada: base de datos.

Decisión educativa

El programa recomienda por defecto **una base de datos tradicional** (con firmas digitales y logs de auditoría si hace falta integridad demostrable). Solo cuando se cumplen simultáneamente los criterios de múltiples escritores sin confianza plena y verificabilidad externa se justifica una blockchain; y en ese caso, pública o L2 antes que permisionada (ver ADR-002).

La razón pedagógica: aprender a decir "aquí no hace falta blockchain" es la primera competencia de un arquitecto serio en este campo.

Consecuencias

Positivas:

- Se evita complejidad accidental: consenso, gas, wallets y gobernanza solo cuando aportan valor.
- Los proyectos que sí usan blockchain lo hacen con una justificación defendible ante negocio y auditoría.

Negativas:

- Se renuncia a la verificabilidad pública en los casos "grises", donde podría aportar confianza de marca.
- Migrar después de una BD a una blockchain es costoso: hay que diseñar la salida de datos desde el inicio si el caso puede evolucionar.

Señales para reconsiderar

- Aparecen nuevos escritores externos que no aceptan al operador actual como árbitro.
- Un regulador o cliente exige pruebas de integridad verificables por terceros en tiempo real.
- El costo de transacción en L2 baja tanto (post EIP-4844 ya cayó de forma drástica) que anclar hashes públicos se vuelve marginal.

Referencias

- NIST, *Blockchain Technology Overview* (NISTIR 8202): https://nvlpubs.nist.gov/nistpubs/ir/2018/NIST.IR.8202.pdf
- Wüst y Gervais, *Do you need a Blockchain?*: https://eprint.iacr.org/2017/375.pdf
- Ethereum.org, *What is Ethereum?*: https://ethereum.org/en/what-is-ethereum/

ADR-002 · ¿Red pública o permisionada?

> **Estado:** guía educativa · **Ámbito:** selección de red · ⬅️ Índice de ADRs

Contexto

Decidido que el caso justifica una blockchain (ADR-001), toca elegir el tipo de red. Una red pública maximiza verificabilidad, neutralidad y participación abierta; una permisionada controla identidad, privacidad y gobernanza a cambio de reintroducir confianza en un club cerrado. "Privada" no implica confidencialidad automática: hay que definir con precisión quién ve, quién valida y quién administra.

El péndulo de la industria es instructivo. Entre 2016 y 2020 los consorcios permisionados (Hyperledger Fabric, Quorum, Corda) dominaron la agenda empresarial; buena parte fracasó no por tecnología sino por **gobernanza**: nadie quería pagar la infraestructura común ni ceder el control. El cierre de TradeLens (IBM/Maersk, 2023) es la señal canónica. Desde entonces el movimiento real —incluida la banca tradicional con fondos tokenizados y stablecoins reguladas— apunta a **redes públicas con privacidad selectiva** (pruebas de conocimiento cero, datos cifrados con anclaje público), abaratadas además por las L2 tras EIP-4844.

Opciones

Criterio	L1 pública	L2 pública (rollup)	Permisionada de consorcio	Híbrida (permisionada anclada a pública)
Verificabilidad externa	Máxima	Alta (hereda de la L1)	Solo miembros	Parcial: compromisos públicos
Privacidad por defecto	Baja	Baja	Alta	Alta dentro, prueba pública fuera
Costo por transacción	Alto en congestión	Bajo post EIP-4844 (consúltalo en vivo)	Costo fijo de operación de nodos	Suma de ambos
Gobernanza	Del protocolo (neutral)	Equipo del rollup + protocolo	Del consorcio (riesgo de cartel)	Doble capa, compleja
Ejemplos	Ethereum	Arbitrum, OP Mainnet, zkSync	Hyperledger Besu, Fabric	Cadenas empresariales con anclaje a Ethereum

Criterios de decisión

- ¿Los verificadores relevantes son **abiertos e imprevisibles** (usuarios, reguladores, mercado) o un grupo cerrado y estable?
- ¿La confidencialidad requerida puede resolverse con **cifrado, ZK o datos off-chain** manteniendo la red pública?
- ¿Existe un consorcio con **incentivos y financiamiento sostenibles** para operar infraestructura común durante años?
- ¿Se necesita **composabilidad** con activos y protocolos existentes (stablecoins, DeFi, identidad)?
- ¿Qué exige el marco regulatorio: control de acceso estricto o transparencia auditable?

Decisión educativa

El programa recomienda por defecto **redes públicas, típicamente una L2 sobre Ethereum**, con privacidad selectiva cuando el dominio lo exige. Las permisionadas se estudian (Besu es útil para entender el stack) pero se tratan como excepción justificable solo con un consorcio real, financiado y con gobernanza resuelta *antes* de escribir código.

Consecuencias

Positivas:

- Se hereda seguridad, liquidez, herramientas y talento del ecosistema público.
- No hay que resolver el problema más difícil de los consorcios: quién manda y quién paga.

Negativas:

- La privacidad requiere diseño explícito (nunca es gratis en una red pública).
- Se depende de la hoja de ruta y la gobernanza de la L2 elegida (ver ADR-005).

Señales para reconsiderar

- Regulación sectorial que prohíba explícitamente registrar compromisos en redes públicas.
- Un consorcio con contrato de financiamiento plurianual firmado y gobernanza neutral operativa.
- Madurez de privacidad ZK insuficiente para el requisito concreto de confidencialidad.

Referencias

- Hyperledger Besu (documentación): https://besu.hyperledger.org/
- Anuncio de discontinuación de TradeLens (Maersk, 2022): https://www.maersk.com/news/articles/2022/11/29/maersk-and-ibm-to-discontinue-tradelens
- Ethereum.org, capas 2: https://ethereum.org/en/layer-2/
- BIS, *The tokenisation continuum*: https://www.bis.org/publ/bisbull72.htm

ADR-003 · ¿Qué datos van on-chain y cuáles off-chain?

> **Estado:** guía educativa · **Ámbito:** arquitectura de datos · ⬅️ Índice de ADRs

Contexto

Guardar datos on-chain es lo más caro que puede hacer un contrato: cada byte de storage se replica en todos los nodos para siempre. Además, lo escrito en una red pública es efectivamente imposible de borrar, lo que choca de frente con normativas como el GDPR y su derecho al olvido. La regla general del programa: **on-chain va solo lo que otros contratos deben verificar o lo que requiere disponibilidad consensuada; todo lo demás vive fuera, con un compromiso (hash) en la cadena si hace falta integridad.**

Conviene recordar qué prueba cada cosa: un hash on-chain prueba *integridad* (que el dato no cambió), no *disponibilidad* (que alguien lo siga sirviendo) ni *veracidad* (que fuera cierto al registrarse). Desde EIP-4844 (2024), Ethereum ofrece además *blobs*: datos temporales baratos pensados para rollups, que expiran en unas semanas y no son storage permanente.

Opciones

Criterio	Storage on-chain	Eventos on-chain	Blobs (EIP-4844)	IPFS / Arweave	BD tradicional
Legible por contratos	Sí	No	No	No	No
Permanencia	Perpetua	Perpetua en nodos de archivo	Semanas	Depende de pinning / dotación	Bajo tu control
Costo	Muy alto	Bajo	Muy bajo	Bajo	Mínimo
Borrable (GDPR)	No	No	Expira solo	Difícil	Sí
Uso típico	Balances, permisos, hashes	Historial para indexadores	Datos de rollups	Metadatos, imágenes, documentos	PII, datos mutables

Criterios de decisión

Tabla de referencia dato → dónde → por qué:

Dato	Dónde	Por qué
Saldos, propiedad, permisos	Storage on-chain	Otros contratos lo verifican en cada transacción
Hash o raíz Merkle de un documento	Storage on-chain	Compromiso de integridad barato y verificable
Historial de operaciones	Eventos	Los indexadores (The Graph, etc.) lo reconstruyen sin costo de storage
Contenido (imágenes, JSON, documentos)	IPFS o Arweave	Direccionado por contenido; la cadena guarda solo el CID
Datos personales (PII)	BD tradicional, jamás on-chain	Derecho al olvido: lo inmutable es incompatible con lo borrable
Datos de disponibilidad de un rollup	Blobs	Baratos y suficientes: solo se necesitan durante la ventana de disputa

Preguntas que inclinan la balanza:

- ¿Algún **contrato** necesita leer este dato para tomar decisiones? Si no, no va a storage.
- ¿El dato identifica, directa o indirectamente, a una **persona**? Entonces jamás on-chain, ni siquiera cifrado (el cifrado se rompe con el tiempo; la cadena no olvida).
- ¿Basta con probar **integridad**? Publica el hash y guarda el dato donde puedas gobernarlo.
- ¿Quién garantiza la **disponibilidad** off-chain y qué pasa si desaparece?

Decisión educativa

El programa enseña a diseñar con **minimalismo on-chain**: estado verificable y compromisos en la cadena, eventos para el historial, IPFS/Arweave para contenido y bases tradicionales para todo dato personal o mutable. El GDPR se trata como criterio duro, no negociable: ninguna PII toca la cadena en ningún ejercicio.

Consecuencias

Positivas:

- Costos de gas órdenes de magnitud menores y contratos más simples de auditar.
- Cumplimiento normativo posible por diseño, no como parche posterior.

Negativas:

- La arquitectura se fragmenta: hay que operar y monitorear la capa off-chain (pinning, índices, backups).
- La verdad "completa" ya no vive en un solo lugar; la disponibilidad off-chain es un riesgo que se gestiona, no que desaparece.

Señales para reconsiderar

- Cambios regulatorios que redefinan qué cuenta como dato personal (por ejemplo, direcciones de wallet vinculables).
- Abaratamiento drástico del storage on-chain o nuevas capas de disponibilidad de datos (el costo real: consúltalo en vivo).
- Que un contrato nuevo necesite verificar datos que hoy están solo en eventos: habría que promoverlos a storage.

Referencias

- EIP-4844, *Shard Blob Transactions*: https://eips.ethereum.org/EIPS/eip-4844
- EDPB, directrices sobre blockchain y GDPR: https://www.edpb.europa.eu/our-work-tools/documents/public-consultations/2025/guidelines-022025-processing-personal-data_en
- Documentación de IPFS: https://docs.ipfs.tech/
- Arweave: https://www.arweave.org/

ADR-004 · ¿Contrato inmutable o actualizable?

> **Estado:** guía educativa · **Ámbito:** ciclo de vida de contratos · ⬅️ Índice de ADRs

Contexto

Un contrato inmutable maximiza la promesa central de la plataforma —las reglas no cambian— pero convierte cualquier bug en permanente. Un contrato actualizable permite reparar fallos y evolucionar, pero reintroduce exactamente lo que la blockchain prometía eliminar: un administrador con poder para cambiar las reglas. La decisión no es técnica sino de **distribución de poder**: quién puede cambiar qué, con qué demora y con qué visibilidad.

Los mecanismos de upgrade añaden además riesgos técnicos propios: colisiones de layout de storage entre versiones, inicializadores ejecutables por cualquiera si se olvidan, y la llave del admin como punto único de fallo (la mayoría de los grandes exploits de puentes fueron compromisos de llaves, no bugs de lógica).

Opciones

Criterio	Inmutable puro	Proxy Transparent	Proxy UUPS	Módulos / Diamond (EIP-2535)	Migración por redeploy
Reparación de bugs	Imposible	Sí, vía admin	Sí, vía lógica de upgrade	Sí, por facetas	Sí, con migración de usuarios
Riesgo de admin key	Nulo	Alto	Alto	Alto y granular	Nulo entre versiones
Riesgo de storage collision	N/A	Medio	Medio	Alto	N/A
Complejidad de auditoría	Baja	Media	Media	Alta	Baja por versión
Costo de gas	Base	+1 delegatecall	+1 delegatecall	+lookup por faceta	Base

Criterios de decisión

- ¿El dominio exige poder **responder a incidentes** (custodia de fondos de terceros, cumplimiento regulatorio)?
- ¿Puede el protocolo tolerar un **bug congelado** mitigado con pausas, límites o migración voluntaria?
- ¿Quién controla la llave de upgrade: EOA, multisig, timelock, gobernanza? ¿Los usuarios pueden **salir antes** de que un upgrade los afecte?
- ¿El equipo tiene la disciplina operativa para gestionar layouts de storage entre versiones (herramientas como el plugin de upgrades de OpenZeppelin)?
- ¿Qué comunica al mercado: "código es ley" o "confía en nuestro multisig"?

Decisión educativa

El programa recomienda por defecto **contratos inmutables con *guarded launch***: lanzamiento con límites de depósito, circuit breakers (pausa acotada) y despliegue progresivo, retirando esos poderes con el tiempo. Cuando el dominio exige capacidad de reparación (custodia significativa, requisitos regulatorios), la alternativa aceptada es **UUPS + timelock público + multisig**, documentando quién firma, cuánto dura la demora y cómo sale un usuario que no acepta el upgrade.

Transparent se estudia como patrón histórico; Diamond solo se recomienda cuando el tamaño del contrato lo hace inevitable, por su costo de auditoría.

Consecuencias

Positivas:

- Los ejercicios del programa fuerzan a diseñar bien desde el inicio: no hay "ya lo parcheamos luego".
- Cuando hay upgrades, el poder administrativo queda explícito, demorado y auditable.

Negativas:

- Un bug serio en un contrato inmutable obliga a redeploy y migración, con costo reputacional y de coordinación.
- El timelock demora también las correcciones de emergencia: hay que combinar con pausas de alcance mínimo.

Señales para reconsiderar

- El contrato pasa a custodiar valor de terceros a una escala donde "no poder reparar" es indefendible.

- La gobernanza madura (participación real, timelock respetado) y puede asumir upgrades con legitimidad.
- Aparecen exigencias regulatorias de intervención (congelamiento, listas) incompatibles con la inmutabilidad pura.

Referencias

- OpenZeppelin, *Proxy Upgrade Pattern*: https://docs.openzeppelin.com/upgrades-plugins/proxies
- EIP-1822 (UUPS): https://eips.ethereum.org/EIPS/eip-1822
- EIP-2535 (Diamonds): https://eips.ethereum.org/EIPS/eip-2535
- Trail of Bits, *Contract upgrade anti-patterns*: https://blog.trailofbits.com/2018/09/05/contract-upgrade-anti-patterns/

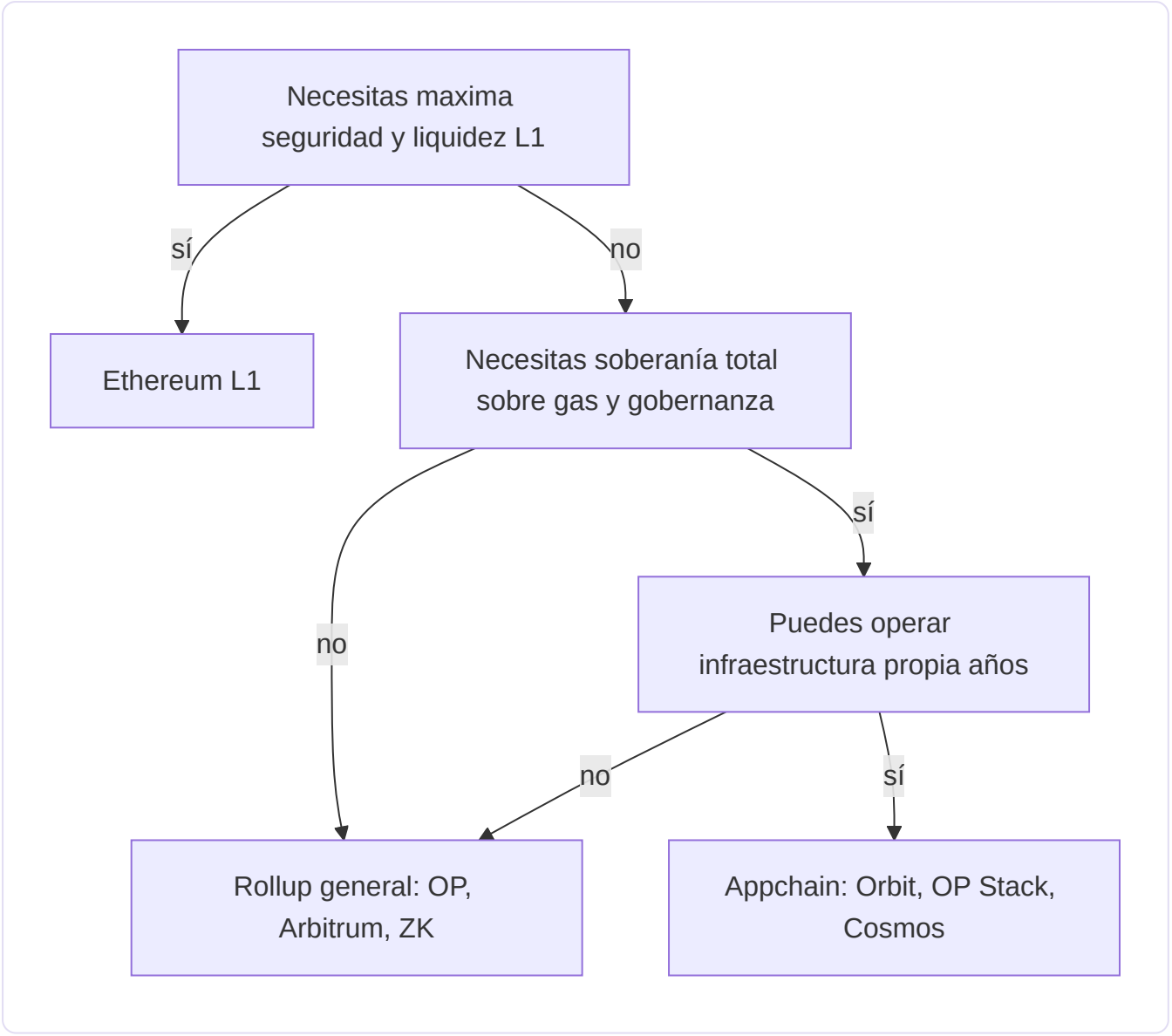
ADR-005 · ¿L1, L2 o appchain?

> **Estado:** guía educativa · **Ámbito:** capa de despliegue · ⬅️ Índice de ADRs

Contexto

Elegida una red pública (ADR-002), falta decidir la capa: desplegar en la L1 de Ethereum, en un rollup de propósito general (Arbitrum, OP Mainnet, zkSync y similares), levantar una appchain propia (Cosmos SDK, Arbitrum Orbit, OP Stack) o usar una sidechain con su propia seguridad. Desde EIP-4844 (2024) los rollups publican sus datos en blobs y el costo por transacción en L2 cayó drásticamente (a menudo a centavos o menos; el valor exacto: consúltalo en vivo), lo que desplazó el defecto razonable de L1 a L2 para casi cualquier aplicación nueva.

El error clásico es decidir solo por TPS. Los criterios que de verdad separan las opciones son la seguridad heredada, la disponibilidad de datos, quién opera el secuenciador, la calidad del puente y del mecanismo de escape, y cuánta carga operativa asume tu equipo.


Necesitas maxima seguridad y liquidez L1
sí
no
Ethereum L1
Necesitas soberanía total sobre gas y gobernanza
sí
no
Puedes operar infraestructura propia años
no
sí
Rollup general: OP, Arbitrum, ZK
Appchain: Orbit, OP Stack, Cosmos

Opciones

Criterio	L1 Ethereum	L2 rollup general	Appchain propia	Sidechain
Seguridad	Máxima (validadores de Ethereum)	Heredada de L1 vía pruebas	La que tú construyas	Propia, usualmente menor
Costo por tx	Alto en congestión	Bajo post EIP-4844	Marginal, pero pagas la infraestructura	Bajo
Soberanía (gas, gobernanza, MEV)	Nula	Baja	Total	Alta
Liquidez y composabilidad	Máxima	Alta y creciente	Aislada: depende de puentes	Media
Carga operativa	Ninguna	Ninguna	Validadores/secuenciador propios, upgrades, monitoreo	Alta
Madurez verificable	N/A	Etapas 0-2 de L2BEAT	Depende de ti	Variable

Criterios de decisión

- ¿Cuánto vale cada transacción? Liquidaciones de alto valor toleran fees de L1; interacciones frecuentes no.
- ¿Necesitas **composabilidad síncrona** con protocolos existentes (DeFi, stablecoins)? Eso ancla a L1 o a una L2 grande.
- ¿Requieres **soberanía real**: tu token como gas, ordenamiento propio, gobernanza independiente? Solo la appchain la da.
- ¿Tu equipo puede **operar infraestructura** (secuenciador, nodos, upgrades) durante años, o es distracción del producto?
- ¿Qué etapa L2BEAT tiene el rollup candidato (Stage 0/1/2)? ¿Existe salida forzada (*escape hatch*) si el secuenciador censura?

Decisión educativa

El programa recomienda por defecto **desplegar en una L2 rollup de propósito general** con buena posición en L2BEAT: seguridad heredada de Ethereum, costos bajos, herramientas idénticas a L1 y cero carga operativa. L1 queda para lo que exige máxima seguridad o composabilidad con liquidez profunda; la appchain, solo cuando la soberanía es requisito de producto y hay equipo para sostenerla. Las sidechains sin herencia de seguridad se estudian, pero no se recomiendan para valor significativo.

Consecuencias

Positivas:

- Los proyectos del programa se despliegan con costos de centavos sin renunciar a la seguridad de Ethereum.
- Se aprende una sola toolchain (EVM) reutilizable en L1, L2 y OP Stack/Orbit.

Negativas:

- Dependencia del secuenciador (hoy centralizado en casi todas las L2) y de la gobernanza del rollup.
- Fragmentación: liquidez y usuarios repartidos entre L2s, con puentes como superficie de riesgo.

Señales para reconsiderar

- El rollup elegido se estanca en Stage 0 o su hoja de descentralización no avanza.
- Los costos o límites de la L2 general se vuelven el cuello de botella del producto (señal de appchain).
- Cambios en la disponibilidad de datos (más blobs, danksharding completo) que reordenen los costos: consúltalo en vivo.

Referencias

- L2BEAT (riesgo y etapas de rollups): https://l2beat.com/
- Ethereum.org, rollups: https://ethereum.org/en/developers/docs/scaling/
- Vitalik Buterin, *The different types of layer 2s*: https://vitalik.eth.limo/general/2023/10/31/l2types.html
- EIP-4844: https://eips.ethereum.org/EIPS/eip-4844

ADR-006 · ¿Emitir un token propio?

> **Estado:** guía educativa · **Ámbito:** diseño económico · ⬅️ Índice de ADRs

Contexto

Casi todo proyecto blockchain enfrenta la tentación de emitir su propio token: financia el desarrollo, crea comunidad y "es lo que todos hacen". Pero un token es un producto financiero vivo que exige diseño de emisión y distribución, gestión de liquidez, gobernanza y —sobre todo— exposición regulatoria permanente. La pregunta correcta no es "¿podemos emitir un token?" sino "¿hay una función que un activo existente (ETH, una stablecoin) o una simple base de permisos no resuelva mejor?".

Si la única función del token es financiar el proyecto, eso tiene nombre: una venta de valores no registrada, con el riesgo jurídico y de incentivos que implica. La historia reciente está llena de tokens de gobernanza sin captura de valor real que cotizan como pasivo reputacional del equipo que los emitió.

Opciones

Criterio	Sin token (stablecoin/ETH)	Token de gobernanza	Token de utilidad	Puntos off-chain primero
Captura de valor	En el producto, vía ingresos	Solo si la gobernanza controla flujos reales	Solo si el uso exige el token de verdad	Diferida: opcionalidad total
Riesgo regulatorio	Mínimo	Medio-alto (según derechos)	Medio (test de Howey caso a caso)	Bajo mientras no sean transferibles
Costo operativo	Bajo	Alto: gobernanza, liquidez, tesorería	Alto: liquidez y UX de compra	Muy bajo
Fricción de usuario	Mínima	Media	Alta (comprar el token para usar el producto)	Nula
Reversibilidad	Total	Casi nula tras el listado	Casi nula	Total: se puede migrar o cancelar

Criterios de decisión

- ¿Existe **captura de valor real**: el token recibe flujos, derechos o utilidad que crecen con el uso del protocolo?
- ¿Pasaría un análisis tipo **test de Howey** (inversión de dinero, empresa común, expectativa de ganancia por esfuerzo ajeno)? En la UE, ¿qué categoría de **MiCA** aplicaría y con qué obligaciones (white paper, autorización)?
- ¿Quién provee y sostiene la **liquidez**? Un token ilíquido es peor que ninguno; uno líquido convierte al equipo en gestor de mercado.
- ¿La utilidad propuesta sobrevive a la pregunta "¿y si esto se paga en stablecoin?"? Si sí, el token es fricción, no función.
- ¿El equipo acepta la **responsabilidad permanente** (legal, fiscal, de comunicación) que un token cotizado impone?

Decisión educativa

El programa recomienda por defecto **no emitir token**: cobrar en stablecoins o ETH y gestionar permisos con mecanismos simples. Si el diseño sugiere que un token podría tener sentido, el camino recomendado es **puntos off-chain primero**, midiendo si el mecanismo funciona antes de asumir la irreversibilidad de un activo transferible. Solo se justifica emitir cuando existe un mecanismo de captura de valor claro y defendible, coherente con lo trabajado en los módulos 08 y 17 del currículo.

Consecuencias

Positivas:

- Los proyectos se evalúan por su producto, no por la especulación sobre su token.
- Se evita la exposición regulatoria más severa del ecosistema y la carga de gestionar un mercado.

Negativas:

- Se renuncia a la vía de financiamiento y de incentivos tempranos que un token bien diseñado puede ofrecer.
- Migrar de puntos a token más tarde exige diseño cuidadoso para no defraudar expectativas creadas.

Señales para reconsiderar

- El protocolo genera flujos reales y una gobernanza madura podría dirigirlos: un token con derechos sobre esos flujos deja de ser humo.
- Claridad regulatoria en la jurisdicción objetivo (por ejemplo, registro viable bajo MiCA) que reduzca el riesgo a niveles gestionables.

- El mecanismo probado con puntos off-chain demuestra que la transferibilidad añade valor y no solo especulación.

Referencias

- SEC, *Framework for “Investment Contract” Analysis of Digital Assets*: https://www.sec.gov/corpfin/framework-investment-contract-analysis-digital-assets
- Reglamento MiCA (UE) 2023/1114: https://eur-lex.europa.eu/eli/reg/2023/1114/oj
- a16z crypto, *Defining tokens*: https://a16zcrypto.com/posts/article/defining-tokens/
- Variant, *Productive assets and token value*: https://variant.fund/articles/

Documentación de referencia

📚 Bibliografía y fuentes

⬅️ Volver al programa · 📚 Currículo · 🔗 Recursos oficiales

Todo el contenido de este programa es **original en su redacción** y se apoya en la literatura de referencia del área. Aquí se listan las obras y fuentes primarias que sustentan cada módulo. **No se reproduce el contenido de los libros**: las referencias apuntan a las obras para que profundices en la fuente.

Cuando una obra tiene varias ediciones, usa **la más reciente**: el ecosistema cambia rápido y las ediciones nuevas corrigen detalles de protocolo. Las fechas de las especificaciones (EIPs, upgrades) están indicadas para que distingas lo estable de lo que aún evoluciona.

🧪 Cómo se valida este contenido (y qué NO garantiza)

Pregunta legítima: si el material es original y no reproduce los libros, **¿cómo sabes que lo que afirma es cierto?** La respuesta honesta es que hay dos tipos de afirmación en el programa y **se validan de forma distinta**.

1. Lo que se demuestra ejecutando

La mayor parte del contenido técnico no te pide que confíes en nadie: **lo puedes correr**. Si el módulo 01 afirma que una prueba de Merkle de 8 hojas necesita 3 hashes, hay un test que lo comprueba; si el módulo 09 afirma que cierta línea habilita una reentrancia, hay un exploit que drena el contrato y un fix que lo resiste.

Afirmación de	Se comprueba con
Hashes, Merkle, firmas	`pnpm test` — pruebas de `labs/01-cryptography`
Prueba de trabajo y dificultad	`pnpm test` — `labs/02-consensus`
Conservación de valor en UTXO	`pnpm test` — `labs/04-bitcoin`
Codificación ABI y selectores	`pnpm test` — `labs/05-evm`
Reentrancia, control de acceso, oráculo, replay, front-running, colisión de storage	`forge test` en `security-challenges/` — cada reto trae su **exploit** y su **fix**
Contratos de bóveda, gobernanza y financiamiento	`forge test` en `labs/` y `projects/`

Son **80 pruebas automatizadas** (60 de Node y 20 de Foundry) que la CI ejecuta en cada cambio. Una afirmación que se contradiga con el código hace fallar el build. Eso es más

fuerte que una cita: no apela a la autoridad de un autor, se comprueba.

Ese número tampoco es una promesa: `pnpm check` cuenta las pruebas del repositorio y falla si esta página deja de coincidir con la realidad.

2. Lo que se apoya en fuentes

El resto —historia, decisiones de diseño, cifras del ecosistema, casos de empresa, regulación— no se puede ejecutar. Ahí la garantía es la **trazabilidad**:

- Cada módulo **declara su fuente** en la cabecera (`**Fuente:**`).
- Cada módulo cierra con **Referencias** enlazando a la fuente primaria (mínimo 3 enlaces; los módulos van de 3 a 9).
- `pnpm check` **falla** si un módulo no declara fuente, no tiene referencias enlazadas o no aparece en la tabla de abajo.
- Un workflow revisa **semanalmente** que esos enlaces siguen vivos y abre un issue si alguno muere. Una fuente que ya no se puede consultar deja de ser una fuente.

Qué NO garantiza esto

Sé explícito sobre el límite: que el material cite una obra **no significa que un tercero haya certificado que la interpretación sea fiel a ella**. Este programa no tiene revisión académica por pares. Si vas a apoyar una decisión técnica o de negocio en algo que leas aquí:

- **Contrástalo con la fuente primaria enlazada** — para eso están los enlaces.
- Desconfía especialmente de **cifras y estados del ecosistema**: cambian rápido. El material marca las fechas de cada hito por esa razón.
- Si encuentras un error, abre un issue: corregir el material es una contribución tan válida como añadirlo.

🔗 Qué obra sustenta cada módulo

Cada módulo declara su fuente en la cabecera. Esta tabla invierte esa relación: te dice **dónde se usa cada obra**, para que puedas ir del libro al módulo o del módulo al libro. Los enlaces apuntan a la fuente oficial —cuando la obra tiene una edición legalmente gratuita, se enlaza esa.

Módulo	Obra que lo sustenta
00 · Orientación	Bashir — *Mastering Blockchain* · Werbach — *The Blockchain and the New Architecture of Trust*
01 · Criptografía	Aumasson — *Serious Cryptography* · Katz, Lindell — *Introduction to Modern Cryptography*
02 · Sistemas distribuidos	Cachin, Guerraoui, Rodrigues — *Reliable and Secure Distributed Programming* · Tanenbaum, van Steen — *Distributed Systems* (PDF gratuito)
03 · Consenso	Nakamoto — whitepaper de Bitcoin · Castro, Liskov — *Practical Byzantine Fault Tolerance*
04 · Bitcoin	Antonopoulos — *Mastering Bitcoin* (libre) · *Mastering the Lightning Network* (libre)
05 · Ethereum y EVM	Antonopoulos, Wood — *Mastering Ethereum* (libre) · Wood — *Yellow Paper*
06 · Solidity y Foundry	Documentación de Solidity · The Foundry Book
07 · dApps	ethereum.org — Developers · documentación de viem
08 · Tokens	EIPs de Ethereum · OpenZeppelin Contracts
09 · Seguridad	Trail of Bits — *Building Secure Contracts* · ConsenSys — *Smart Contract Best Practices*
10 · Oráculos e indexación	Chainlink docs · The Graph docs
11 · DAO y gobernanza	OpenZeppelin Governor · Compound Governance
12 · Escalabilidad	Buterin — *An Incomplete Guide to Rollups* · L2BEAT
13 · Interoperabilidad	Cosmos IBC · Polkadot XCM
14 · Privacidad y ZK	Thaler — *Proofs, Arguments, and Zero-Knowledge* (libre) · ZKProof Community Reference
15 · Arquitectura avanzada	ERC-4337 y EIP-7702 · Flashbots — investigación
16 · Infraestructura y nodos	ethereum.org — Nodos y clientes · EthStaker
17 · Empresa	Informes del BIS y del WEF · Werbach — *The Blockchain and the New Architecture of Trust*
18 · Implementación	Prácticas públicas de integración del sector financiero y documentación de cada componente citado

> **Obras libres.** *Mastering Bitcoin*, *Mastering Ethereum*, *Mastering the Lightning Network*, *Distributed Systems*, *Proofs, Arguments, and Zero-Knowledge* y el *MoonMath Manual* tienen edición legalmente gratuita: puedes seguir el programa completo sin comprar un solo libro.

Libros de referencia por área

Área	Obras de referencia
Fundamentos y panorama	Bashir — *Mastering Blockchain* (4.ª ed.) · Narayanan, Bonneau, Felten, Miller, Goldfeder — *Bitcoin and Cryptocurrency Technologies* (Princeton) · Werbach — *The Blockchain and the New Architecture of Trust*
Criptografía	Aumasson — *Serious Cryptography* (2.ª ed.) · Ferguson, Schneier, Kohno — *Cryptography Engineering* · Katz, Lindell — *Introduction to Modern Cryptography* (3.ª ed.)
Sistemas distribuidos	Cachin, Guerraoui, Rodrigues — *Introduction to Reliable and Secure Distributed Programming* · Tanenbaum, van Steen — *Distributed Systems* · Kleppmann — *Designing Data-Intensive Applications*
Consenso	Lamport — *The Part-Time Parliament* (Paxos) · Castro, Liskov — *Practical Byzantine Fault Tolerance* · Nakamoto — *Bitcoin: A Peer-to-Peer Electronic Cash System* · Buterin, Griffith — *Casper the Friendly Finality Gadget*
Bitcoin	Antonopoulos — *Mastering Bitcoin* (3.ª ed.) · Antonopoulos, Osuntokun, Pickhardt — *Mastering the Lightning Network*
Ethereum / EVM	Antonopoulos, Wood — *Mastering Ethereum* · Wood — *Ethereum Yellow Paper* · documentación de ethereum.org
Solidity / Foundry	Documentación de Solidity · The Foundry Book · OpenZeppelin Contracts
Seguridad de contratos	Trail of Bits — *Building Secure Contracts* · ConsenSys — *Smart Contract Best Practices* · SWC Registry · *Damn Vulnerable DeFi*
Oráculos e indexación	Chainlink — documentación y whitepaper 2.0 · The Graph — docs · IPFS — docs
DAO y gobernanza	OpenZeppelin Governor · Compound Governance · Voshmgir — *Token Economy*
Escalabilidad / L2	Buterin — *An Incomplete Guide to Rollups* · ethereum.org — Escalado · L2BEAT
Interoperabilidad	Cosmos IBC · Polkadot XCM · Chainlink CCIP
Privacidad y ZK	Thaler — *Proofs, Arguments, and Zero-Knowledge* · *The MoonMath Manual to zk-SNARKs* · ZKProof — Community Reference
Arquitectura avanzada	ERC-4337 y EIP-7702 (abstracción de cuenta) · Flashbots — investigación MEV · OpenZeppelin — Proxies y Upgrades

Fuentes primarias (whitepapers y especificaciones)

- **Bitcoin** — Nakamoto, *Bitcoin: A Peer-to-Peer Electronic Cash System* (2008): https://bitcoin.org/bitcoin.pdf
- **Ethereum** — Buterin, *Ethereum Whitepaper*: https://ethereum.org/whitepaper/ · Wood, *Yellow Paper*: https://ethereum.github.io/yellowpaper/paper.pdf

- **Estándares (EIP/ERC)** — https://eips.ethereum.org/ — en especial ERC-20, ERC-721, ERC-1155, ERC-4626 (bóvedas), ERC-2612 (permit), ERC-4337 y EIP-7702 (abstracción de cuenta), EIP-1559 (comisiones) y EIP-4844 (blobs).
- **Registro de vulnerabilidades** — SWC Registry: https://swcregistry.io/ · Solodit / informes de auditoría públicos.

Actualidad del ecosistema (hitos recientes)

El contenido se mantiene al día con la hoja de ruta de Ethereum y del ecosistema. Hitos relevantes que el material tiene en cuenta:

- **The Merge** (2022) — Ethereum pasó de Proof of Work a Proof of Stake.
- **Shapella** (2023) — habilitó el retiro de *staking*.
- **Dencun / EIP-4844** (marzo 2024) — *blobs* de datos que abarataron drásticamente los rollups (proto-danksharding).
- **Pectra / EIP-7702** (2025) — mejoras de abstracción de cuenta para EOAs y más capacidad de *blobs*.
- **Roadmap en curso** — danksharding completo, *statelessness* (árboles Verkle), separación proponente-constructor (PBS) y *restaking*. Se presentan como **dirección**, no como hechos consumados.

> ⚠️ Verifica siempre contra la fuente primaria. Las cifras concretas (TPS, comisiones, número de validadores, TVL de un puente) cambian a diario; este material enseña los **principios** y señala explícitamente qué datos debes consultar en vivo.

🧭 Navegación

📚 Currículo · 🔗 Recursos oficiales · 🗺️ Roadmap · 🏠 Inicio

Glosario del programa

⬅️ Volver al programa · 📚 Currículo · 📖 Bibliografía

Glosario de referencia de los términos que usan los módulos 00-18, los laboratorios y el capstone. Se privilegia el término en español con el anglicismo entre paréntesis cuando el sector lo usa de forma dominante. Para profundizar en cada tema, consulta recursos-oficiales.md.

Fundamentos y criptografía

- **Cadena de bloques (blockchain)**: registro replicado y ordenado de transacciones cuya integridad se garantiza enlazando bloques con hashes.
- **Bloque**: conjunto ordenado de transacciones y metadatos enlazado criptográficamente al bloque anterior.
- **Hash**: huella determinista de longitud fija producida por una función criptográfica; cualquier cambio en la entrada altera la salida.
- **Preimagen**: entrada original de una función hash; la resistencia a preimagen impide recuperarla desde el hash.
- **Colisión**: dos entradas distintas que producen el mismo hash; una función segura hace su búsqueda impracticable.
- **Árbol de Merkle (Merkle tree)**: estructura que permite probar la inclusión de un dato en un conjunto con pruebas logarítmicas.
- **Prueba de Merkle (Merkle proof)**: conjunto mínimo de hashes hermanos que demuestra que una hoja pertenece a la raíz.
- **Firma digital**: prueba de que el titular de una clave privada autorizó un mensaje, verificable con la clave pública.
- **Criptografía de curva elíptica (ECC)**: familia de esquemas de clave pública usada por Bitcoin y Ethereum (secp256k1) y por BLS en consenso.
- **Clave privada / clave pública**: par asimétrico; la privada firma y jamás se comparte, la pública identifica y verifica.
- **Frase semilla (seed phrase)**: secuencia mnemónica (BIP-39) de la que se derivan de forma determinista todas las claves de una billetera.
- **Billetera (wallet)**: herramienta que administra claves y firma transacciones; no "guarda monedas" literalmente, las monedas viven en el registro.
- **Nodo**: software que valida, almacena o comunica datos de la red.
- **Consenso**: reglas para acordar el estado válido entre participantes que no confían entre sí.
- **Prueba de trabajo (proof of work, PoW)**: consenso que exige gasto computacional verificable para proponer bloques.

- **Prueba de participación (proof of stake, PoS)**: consenso que selecciona proponentes según capital depositado en garantía, con penalizaciones (slashing).
- **Finalidad (finality)**: garantía de que un bloque no será revertido; puede ser probabilística (PoW) o económica/absoluta (PoS con checkpoints).
- **Tolerancia a fallas bizantinas (BFT)**: capacidad de un sistema de acordar estado correcto aunque una fracción de nodos actúe de forma arbitraria o maliciosa.

Bitcoin

- **UTXO**: salida de transacción no gastada; el modelo contable de Bitcoin, donde el "saldo" es la suma de UTXO controlados por tus claves.
- **Nonce**: valor usado una vez; en minería PoW es el campo que se itera para encontrar un hash válido, en Ethereum ordena las transacciones de una cuenta.
- **Dificultad**: parámetro que ajusta cada 2016 bloques cuán difícil es encontrar un bloque válido, apuntando a ~10 minutos por bloque.
- **Halving**: reducción a la mitad del subsidio por bloque cada 210 000 bloques (~4 años); el último ocurrió en abril de 2024.
- **Mempool**: conjunto de transacciones válidas pendientes de inclusión en un bloque; cada nodo tiene su propia vista.
- **SegWit**: actualización de 2017 que separó las firmas (witness) del cuerpo de la transacción, corrigiendo la maleabilidad y habilitando más capacidad.
- **Taproot**: actualización de 2021 que introdujo firmas Schnorr y scripts más privados y eficientes.
- **Script**: lenguaje de condiciones de gasto de Bitcoin, deliberadamente no Turing-completo.
- **Lightning Network**: red de canales de pago fuera de cadena (off-chain) para pagos rápidos y baratos liquidados en Bitcoin.
- **Regtest**: modo de red local de Bitcoin Core para desarrollo, donde tú generas los bloques (lo usa el laboratorio `04-bitcoin-regtest`).

Ethereum y EVM

- **EVM (Ethereum Virtual Machine)**: máquina virtual determinista que ejecuta el bytecode de los contratos en cada nodo.
- **Gas**: unidad que mide trabajo computacional en la EVM; cada opcode tiene un costo y cada transacción un límite.
- **Tarifa base (base fee)**: componente de la tarifa por gas que se ajusta por bloque según demanda y se quema, introducido por EIP-1559.
- **Propina (priority fee / tip)**: componente de la tarifa que va al proponente del bloque para priorizar la transacción.
- **EIP-1559**: reforma del mercado de tarifas de 2021 que introdujo base fee quemada más propina, haciendo las tarifas más predecibles.

- **Cuenta EOA (externally owned account)**: cuenta controlada por una clave privada, sin código propio.
- **Cuenta de contrato**: cuenta con código y almacenamiento, controlada por su propia lógica.
- **Calldata**: datos de entrada de una llamada a contrato; zona de solo lectura y la más barata para pasar argumentos.
- **Storage / memory**: almacenamiento persistente del contrato (caro, sobrevive entre transacciones) frente a memoria volátil de la ejecución (barata, se descarta al terminar).
- **ABI (Application Binary Interface)**: especificación para codificar llamadas y respuestas de contratos EVM; los primeros 4 bytes del calldata son el selector de función.
- **Opcode**: instrucción elemental de la EVM (por ejemplo `SSTORE`, `CALL`, `REVERT`).
- **Revert**: aborto de la ejecución que deshace todos los cambios de estado de la llamada y devuelve el gas no consumido.
- **Evento (log)**: registro barato emitido por un contrato, legible fuera de cadena pero inaccesible desde contratos.
- **Nodo de ejecución / nodo de consenso**: desde The Merge (2022), un nodo completo de Ethereum combina un cliente de ejecución (Geth, Nethermind, Reth) y uno de consenso (Lighthouse, Prysm, Teku).
- **Sincronización por checkpoint (checkpoint sync)**: arranque de un nodo de consenso desde un estado finalizado reciente en lugar de reproducir toda la historia.
- **MEV (maximal extractable value)**: valor extraíble reordenando, insertando o censurando transacciones dentro de un bloque.

Contratos y desarrollo

- **Contrato inteligente (smart contract)**: programa determinista desplegado en la red cuya ejecución no depende de un operador central.
- **Solidity**: lenguaje dominante para contratos EVM, compilado a bytecode.
- **Foundry**: kit de herramientas para desarrollo EVM (`forge`, `cast`, `anvil`); el estándar de pruebas de este programa.
- **Prueba de propiedad (fuzz test)**: prueba que ejecuta una función con miles de entradas aleatorias buscando violar aserciones.
- **Prueba de invariantes (invariant test)**: prueba que verifica que una propiedad global del sistema se mantiene tras secuencias arbitrarias de llamadas.
- **Fork de red (fork testing)**: pruebas contra una copia local del estado real de una red (`forge test --fork-url`).
- **Proxy / actualizabilidad (upgradeability)**: patrón que separa almacenamiento y lógica para poder reemplazar la implementación; añade riesgos propios (colisión de storage, inicializadores).
- **Interfaz / herencia**: mecanismos de Solidity para componer contratos y cumplir estándares ERC.

- **Verificación de código fuente**: publicación del código en un explorador para que cualquiera compruebe que coincide con el bytecode desplegado.
- **Cuenta abstracta (account abstraction, ERC-4337)**: estándar que permite cuentas inteligentes con validación programable, patrocinio de gas y recuperación, sin cambiar el protocolo.
- **EIP-7702**: mejora (activa desde Pectra, 2025) que permite a una EOA delegar temporalmente su comportamiento a código de contrato.

Tokens y DeFi

- **ERC-20**: estándar de tokens fungibles (saldos, `transfer`, `approve`).
- **ERC-721**: estándar de tokens no fungibles (NFT), un identificador único por token.
- **ERC-1155**: estándar multi-token que combina fungibles y no fungibles en un solo contrato.
- **ERC-4626**: estándar de bóvedas (vaults) tokenizadas que reciben un activo y emiten participaciones; base de mucha contabilidad DeFi.
- **Stablecoin**: token diseñado para mantener paridad con un activo (habitualmente USD), con respaldo fiat, cripto-colateralizado o algorítmico (este último con historial de fracasos, como UST en 2022).
- **AMM (automated market maker)**: intercambio que fija precios con una fórmula sobre reservas (por ejemplo $x \cdot y = k$) en lugar de un libro de órdenes.
- **Pool de liquidez**: reservas depositadas por usuarios que habilitan el intercambio en un AMM a cambio de comisiones.
- **Pérdida impermanente (impermanent loss)**: costo de oportunidad de proveer liquidez cuando los precios de los activos del pool divergen.
- **Préstamo relámpago (flash loan)**: préstamo sin colateral que debe devolverse dentro de la misma transacción; herramienta legítima y vector de ataque frecuente.
- **Oráculo**: mecanismo que lleva datos externos (precios, resultados) al entorno on-chain; su manipulación es una causa recurrente de exploits.
- **TWAP (time-weighted average price)**: precio promedio ponderado por tiempo, usado para resistir manipulación puntual de oráculos.
- **Gobernanza on-chain**: toma de decisiones de un protocolo mediante votación con tokens y ejecución automática de propuestas.
- **Quorum**: participación mínima requerida para que una votación de gobernanza sea válida.
- **Timelock**: contrato que impone una demora obligatoria entre aprobar una acción y ejecutarla, dando tiempo a reaccionar.
- **DAO**: organización cuyo tesoro y reglas se administran mediante contratos y gobernanza distribuida.
- **TVL (total value locked)**: valor total depositado en un protocolo; métrica de adopción (consúltala en vivo en DefiLlama), no de seguridad.
- **RWA (real-world assets)**: activos del mundo real (deuda, inmuebles, fondos) representados como tokens on-chain.

- **Tokenomics**: diseño económico de un token: emisión, distribución, incentivos y sumideros de demanda.

Seguridad

- **Reentrancia (reentrancy)**: ataque en que un contrato externo vuelve a llamar a la función víctima antes de que actualice su estado; mitigada con el patrón checks-effects-interactions y guardas.
- **Checks-effects-interactions**: orden defensivo: validar, actualizar estado propio y solo al final interactuar con contratos externos.
- **Desbordamiento (overflow/underflow)**: exceso del rango numérico; Solidity ≥0.8 revierte por defecto, pero `unchecked` lo reintroduce.
- **Front-running**: adelantarse a una transacción visible en el mempool para obtener ventaja; caso particular del MEV.
- **Ataque sándwich**: front-run más back-run alrededor de un intercambio para extraer valor del deslizamiento (slippage) de la víctima.
- **Control de acceso**: restricción de funciones sensibles a roles autorizados; su ausencia o mala configuración es una causa principal de pérdidas.
- **Multifirma (multisig)**: esquema que exige M-de-N firmas para ejecutar una acción; estándar mínimo para administrar contratos con fondos.
- **MPC (multi-party computation)**: técnica que reparte una clave entre varias partes de modo que ninguna la conozca completa; usada en custodia institucional.
- **HSM (hardware security module)**: hardware certificado que genera y protege claves sin exponerlas jamás en memoria de propósito general.
- **Auditoría**: revisión experta del código antes del despliegue; reduce riesgo, no lo elimina.
- **Recompensa por vulnerabilidades (bug bounty)**: programa que paga por reportes responsables (por ejemplo, vía Immunefi).
- **Lanzamiento protegido (guarded launch)**: despliegue gradual con límites de depósito, pausas y monitoreo intensivo antes de abrir el protocolo por completo.
- **Interruptor de pausa (circuit breaker / pause)**: facultad de detener funciones críticas ante un incidente; ver operacion-incidentes.md.
- **Runbook**: procedimiento operativo escrito y ensayado para responder a un tipo de incidente.
- **Modelo de amenazas**: análisis sistemático de quién puede atacar qué, cómo y con qué impacto.
- **KYT (know your transaction)**: monitoreo de transacciones para detectar fondos vinculados a actividades ilícitas; contraparte transaccional del KYC.

Escalabilidad e interoperabilidad

- **L1 / L2**: capa base con su propio consenso frente a sistema que escala apoyándose en la seguridad de una capa base.

- **Rollup**: L2 que ejecuta transacciones fuera de la L1 y publica en ella los datos y las pruebas necesarias para verificarlas.
- **Rollup optimista (optimistic rollup)**: asume validez y permite disputarla durante una ventana con pruebas de fraude (fraud proofs); ejemplos: Arbitrum, Optimism.
- **Rollup de validez (ZK rollup)**: demuestra cada lote con una prueba de validez (validity proof) verificada on-chain; ejemplos: zkSync, Starknet, Scroll.
- **Prueba de fraude / prueba de validez**: mecanismos opuestos de seguridad de rollups: castigar estados inválidos a posteriori frente a impedirlos a priori.
- **Disponibilidad de datos (data availability, DA)**: garantía de que los datos de un rollup están publicados y cualquiera puede reconstruir su estado.
- **Blob (EIP-4844, proto-danksharding)**: espacio de datos temporal y barato introducido en 2024 para que los rollups publiquen datos sin usar calldata permanente.
- **Canal de estado (state channel)**: acuerdo fuera de cadena entre partes que solo liquida on-chain al abrir y cerrar.
- **Cadena lateral (sidechain)**: cadena independiente conectada a otra por un puente, con su propia seguridad (no hereda la de la L1).
- **Puente (bridge)**: mecanismo para mover activos o mensajes entre cadenas; históricamente el componente más explotado del ecosistema.
- **Cliente ligero (light client)**: verificador que valida cabeceras y pruebas sin almacenar todo el estado; base de los puentes con menos confianza.
- **Compromiso (commitment)**: valor que fija datos sin revelarlos y puede abrirse después (por ejemplo, una raíz de Merkle o un compromiso de Pedersen).
- **Fragmentación de liquidez**: dispersión de activos y usuarios entre múltiples L2 y cadenas, con costos de experiencia y capital.

Privacidad y ZK

- **Prueba de conocimiento cero (ZK proof)**: prueba criptográfica de que una afirmación es cierta sin revelar los datos que la sustentan.
- **SNARK**: prueba ZK sucinta y de verificación barata; muchas construcciones requieren una ceremonia de configuración confiable (trusted setup).
- **STARK**: prueba ZK sin configuración confiable y post-cuántica en sus supuestos, a costa de pruebas más grandes.
- **Circuito**: representación aritmética del cómputo que se quiere demostrar en un sistema ZK.
- **Testigo (witness)**: entradas privadas del circuito que el probador conoce y no revela.
- **Anulador (nullifier)**: valor único que impide usar dos veces la misma nota o compromiso privado sin revelar cuál se usó.
- **Mixer / pool de privacidad**: protocolo que rompe el vínculo entre depósitos y retiros; con implicaciones regulatorias serias (caso Tornado Cash).

- **Identidad autosoberana / credencial verificable**: identidad digital cuyo titular controla qué atributos demuestra, a menudo con pruebas selectivas ZK.

Empresa e infraestructura

- **Cadena permisionada**: red donde los validadores y a veces los lectores requieren autorización; modelo de Hyperledger Fabric o Besu en consorcios.
- **Consorcio**: grupo de organizaciones que operan una red permisionada compartida con gobernanza contractual.
- **Tokenización**: representación de un activo o derecho como token, con el desafío central de que el vínculo legal off-chain sea exigible.
- **Custodia**: guarda de claves por un tercero regulado; alternativa a la autocustodia con otros riesgos y obligaciones.
- **RPC (remote procedure call)**: interfaz por la que aplicaciones consultan nodos (propios o de proveedores como Alchemy/Infura).
- **Indexador**: servicio que transforma eventos on-chain en datos consultables (por ejemplo, The Graph o un indexador propio).
- **IOPS**: operaciones de entrada/salida por segundo; el disco (NVMe) suele ser el cuello de botella real al operar nodos, más que la CPU.
- **Nodo de archivo (archive node)**: nodo que conserva todos los estados históricos, con requisitos de disco muy superiores a un nodo completo.
- **SLA / observabilidad**: compromisos de disponibilidad y la instrumentación (métricas, logs, alertas) para cumplirlos; ver operacion-incidentes.md.
- **MiCA**: reglamento europeo de mercados de criptoactivos, en aplicación plena desde diciembre de 2024; referencia regulatoria para emisores y proveedores de servicios.

Cómo usar este glosario

- Los cuestionarios de evaluación asumen que manejas estos términos al nivel de la definición dada.
- Si un término te resulta opaco, el módulo correspondiente del currículo lo desarrolla con laboratorios.
- Las cifras asociadas (tarifas, TVL, número de validadores) cambian constantemente: consúltalas en vivo en las fuentes de recursos-oficiales.md.

🗣️ Cómo explicar blockchain a clientes y personas que no conocen el tema

⬅️ Volver al programa · 🏢 Módulo 17 · Blockchain en la empresa · 💼 Industria · Modelos de negocio

El mejor caso de negocio muere si la contraparte no lo entiende. Esta guía reúne, en un solo lugar, el método de comunicación que usa el sector para explicar blockchain a un directorio, un cliente o un familiar **sin jerga y sin prometer nada que no se pueda cumplir**. Es material del módulo 17, extraído aquí como referencia rápida.

La regla de oro

Primero el problema y el resultado; la tecnología después, y solo si la preguntan. Nadie compra "blockchain": compran que lo que hoy tarda días tarde minutos, que nadie discuta los números, que un proceso caro se abarate. La palabra "blockchain" puede no aparecer nunca en una buena explicación.

🎤 El discurso de 30 segundos

> "Hoy, cuando su empresa y sus socios mueven [dinero / activos / registros], cada uno lleva su propia planilla y cuadrarlas toma días y errores. Esto es un **registro único compartido que nadie puede alterar por su cuenta** y que ejecuta las reglas que todos acordaron, automáticamente. Resultado: lo que hoy tarda días, tarda minutos, y nadie discute los números."

Ni una vez dice "blockchain", "cripto", "token" ni "descentralización". Adáptalo al dolor concreto del cliente antes de decir una sola palabra técnica.

🔄 Traducción de jerga: qué decir en lugar de qué

No digas	Di	Por qué funciona
"Blockchain / DLT"	"un registro compartido que nadie puede alterar solo"	Describe la propiedad útil, no la tecnología
"Smart contract"	"reglas acordadas que se ejecutan solas"	El valor es la automatización sin árbitro
"Token / tokenizar"	"una representación digital del activo, transferible en minutos"	Ancla al activo que ya conocen
"Wallet / llaves privadas"	"su firma digital; la custodia se gestiona como en un banco"	Evita el pánico de la seed phrase
"Gas / L2 / rollup"	"costo por operación, hoy de centavos"	El detalle técnico no aporta a la decisión
"Descentralizado"	"sin depender de un intermediario único"	Beneficio concreto en vez de ideología
"Minería / validadores"	"la red que mantiene el registro seguro"	Aparta un debate que no viene al caso
"Inmutable"	"queda registrado y no se puede borrar ni falsificar"	Enfatiza la garantía, no el término

🧩 Analogías que funcionan (y su límite)

- **Libro de contabilidad notariado entre empresas** — útil para conciliación. *Límite:* aclara que además **ejecuta** (un notario da fe, no ejecuta).
- **El contenedor marítimo** — un estándar común que abarató mover carga entre empresas; aquí abarata mover valor. *Límite:* no todo activo "cabe" ni conviene mover.
- **El correo electrónico del dinero** — transferir valor como se transfiere un correo, sin oficinas de por medio. *Límite:* aquí **no hay "deshacer envío"**; por eso existen los límites y controles.

Elige **una** analogía y menciona su límite: prometer de más destruye la credibilidad.

🛡️ Manejo de objeciones frecuentes

Objeción del cliente	Respuesta honesta
"¿Esto no es lo de las criptos que se desplomaron?"	"Eso fue especulación con activos volátiles. Esto usa la misma infraestructura para un problema operativo suyo: conciliación y liquidación. No compramos criptomonedas."
"¿Es legal?"	"Regulado y en uso por bancos centrales y gestoras como BlackRock. En Chile lo cubre la Ley Fintech; el diseño incluye cumplimiento desde el día uno."
"¿Y si se cae o nos hackean?"	"Los riesgos existen y se gestionan con límites, custodia profesional y auditorías. Aquí está el plan y su costo; está dentro del presupuesto que le presento."
"¿Por qué no una base de datos normal?"	"Si una sola organización controlara los datos, una base de datos sería mejor y más barata. Aquí participan varias que no comparten dueño: ese es exactamente el caso donde esto gana."
"¿Necesito criptomonedas para esto?"	"No necesariamente. Muchas soluciones empresariales usan monedas estables o ni siquiera manejan cripto de cara al usuario."
"Esto suena a moda pasajera"	"La especulación va y viene; la infraestructura de liquidación la están adoptando bancos e instituciones de forma sostenida. Le muestro casos en producción."

🚦 Lo que nunca debes hacer

- **Nunca prometas rentabilidad** ni uses el precio de las criptomonedas como argumento: destruye la credibilidad del caso operativo y puede tener implicancias legales.
- **No uses miedo a "quedarse atrás"** como palanca; usa el problema concreto del cliente.
- **No escondas los riesgos ni los costos**: preséntalos con su mitigación y su precio. Un caso honesto convence más y protege la relación.
- **No adornes con jerga** para parecer experto: la claridad es la señal de dominio.

🎯 Practícalo

El reto verificable del módulo 17 te pide construir un caso de negocio y acompañarlo del discurso de 30 segundos, sin usar ninguno de los términos de la columna "No digas". Grábate explicándolo a alguien sin formación técnica: si lo entiende y sabe por qué le conviene, lo lograste.

🧭 Navegación

🏠 Programa · 🏢 Módulo 17 · 🏭 Industria · 📚 Currículo

Mejores prácticas

⬅️ Volver al programa · 📚 Currículo · 🔐 Seguridad y auditoría

Buenas prácticas de desarrollo, prueba y operación de sistemas blockchain. Cada práctica se describe con **qué** es, **por qué** importa y **cómo verificar** que se cumple. La profundización con exploits reales está en el módulo 09 · Seguridad y auditoría.

Diseño

Decisiones tomadas antes de escribir la primera línea de código.

Práctica	Qué	Por qué	Cómo verificar
Justificar la descentralización	Documentar por qué se necesita consenso e inmutabilidad	Evita usar blockchain por moda	ADR con alternativas descartadas
Minimizar estado on-chain	Almacenar y calcular fuera de la cadena cuando se pueda	Storage y cómputo cuestan gas real	Revisar variables y bucles por costo
Definir invariantes primero	Escribir las propiedades que nunca deben romperse	Guían el diseño y las pruebas	Lista de invariantes en el repo
Separar responsabilidades	Aislar lógica, permisos, tesorería y actualización	Reduce el radio de impacto de un fallo	Módulos y roles distintos

Seguridad de contratos

Práctica	Qué	Por qué	Cómo verificar
Checks-Effects-Interactions	Validar, actualizar estado y **luego** llamar al exterior	Neutraliza reentrancia	Revisión y test de reentrancia
Control de acceso	Mínimo privilegio, roles explícitos, admin en dos pasos	Limita abuso interno y errores	Test por rol; `onlyRole` cubierto
Reentrancy guards	Candado en funciones con llamadas externas	Defensa en profundidad ante CEI incompleto	Test que simula el reingreso
Pull over push	El beneficiario retira; el contrato no empuja fondos	Un receptor malicioso no bloquea a otros	Patrón de retiro comprobado con test
No reinventar criptografía	Usar estándares auditados (OpenZeppelin, libs probadas)	Implementar cripto desde cero introduce fallos	Dependencias auditadas y fijadas
Evitar bucles no acotados	No iterar sobre colecciones de tamaño arbitrario	Un bucle grande agota el gas y bloquea	Revisión de bucles; límites explícitos

Pruebas

Tipo	Qué prueba	Herramienta	Cómo verificar
Unitaria	Una función en aislamiento	`forge test`	Casos felices y de error
Fuzz	Entradas aleatorias contra una propiedad	Foundry fuzz	Miles de runs sin contraejemplo
Invariante	Propiedades globales tras secuencias de acciones	Foundry invariant	Invariantes se mantienen
Fork	Comportamiento contra estado real de una red	`forge test --fork-url`	Integración con protocolos reales

- **Cobertura por comportamiento, no por porcentaje:** un 100% de líneas no garantiza que se probaron los estados adversos (pausa, oráculo obsoleto, rol comprometido).
- **Una auditoría reduce riesgo; no demuestra ausencia de errores.**

Gestión de claves

Práctica	Qué	Por qué	Cómo verificar
Nunca en el repositorio	Claves y secretos fuera de git	Un secreto commiteado se compromete	Escaneo de secretos en CI
`.env` fuera de control de versiones	Cargar secretos desde el entorno local	Evita fugas accidentales	`.gitignore` cubre `.env`
HSM / MPC en producción	Custodia en hardware o firma multiparte	Elimina el punto único de compromiso	Claves críticas nunca en texto plano
Claves de prueba desechables	Usar las claves que muestra Anvil solo local	Aislar experimentos de fondos reales	No reutilizar claves entre entornos

Operación

Práctica	Qué	Por qué	Cómo verificar
Multisig	Firma de N-de-M para acciones críticas	Ningún individuo actúa solo	Safe u equivalente configurado
Timelock	Retraso obligatorio en cambios sensibles	Da tiempo a reaccionar ante abuso	Delay verificable on-chain
Monitoreo	Vigilar eventos, balances, roles e invariantes	Detectar anomalías temprano	Alertas y panel activos
Runbooks	Procedimientos de incidente y divulgación	Reducir el tiempo de respuesta	Ver operación e incidentes

Simula actualizaciones y respuestas a incidentes **antes** de producción.

Calidad de código y CI

Práctica	Qué	Herramienta	Cómo verificar
Formato y linters	Estilo consistente y automatizado	`forge fmt`, linters	CI falla si hay desviaciones
Análisis estático	Detectar patrones peligrosos	Slither, Semgrep	Reporte sin hallazgos críticos
Fuzzing dirigido	Buscar contraejemplos económicos	Echidna, Foundry	Propiedades sin violación
Integración continua	Ejecutar todo en cada cambio	GitHub Actions	Build en verde, escaneo de secretos

Dependencias

- **Fijar versiones (pin):** compilador y librerías en versiones exactas y estables.

- **Auditar antes de adoptar:** revisar y preferir dependencias mantenidas y auditadas.
- **Mínima superficie:** cada dependencia amplía el área de ataque; incluir solo lo necesario.

Experiencia de usuario

Práctica	Qué	Por qué	Cómo verificar
Mostrar contexto antes de firmar	Red, dirección abreviada, monto, gas y consecuencias	El usuario firma con información completa	Revisar la pantalla de confirmación
Distinguir firma de tx	Separar firma de mensaje de firma de transacción	Un mensaje mal presentado puede autorizar valor	Probar ambos flujos en la UI
Aprobaciones acotadas	Evitar `approve` ilimitado por defecto	Limita el daño si el gastador se compromete	Verificar el monto exacto aprobado
Protección ante ataques	Mitigar front-running, slippage y phishing	El usuario opera en un entorno adversarial	Límites de slippage y dominio verificado

Priorización

No todas las prácticas tienen el mismo retorno. Cuando el tiempo es limitado, prioriza en este orden:

1. **Seguridad de contratos** — un fallo aquí es irreversible y con pérdida directa.
2. **Gestión de claves** — un secreto filtrado anula cualquier otra defensa.
3. **Pruebas** — sin evidencia reproducible no sabes si lo anterior funciona.
4. **Operación** — controla el riesgo una vez en producción.
5. **Calidad y dependencias** — sostienen todo lo demás a largo plazo.

Recursos relacionados

- 09 · Seguridad y auditoría
- Modelo de amenazas del proyecto
- Operación e incidentes

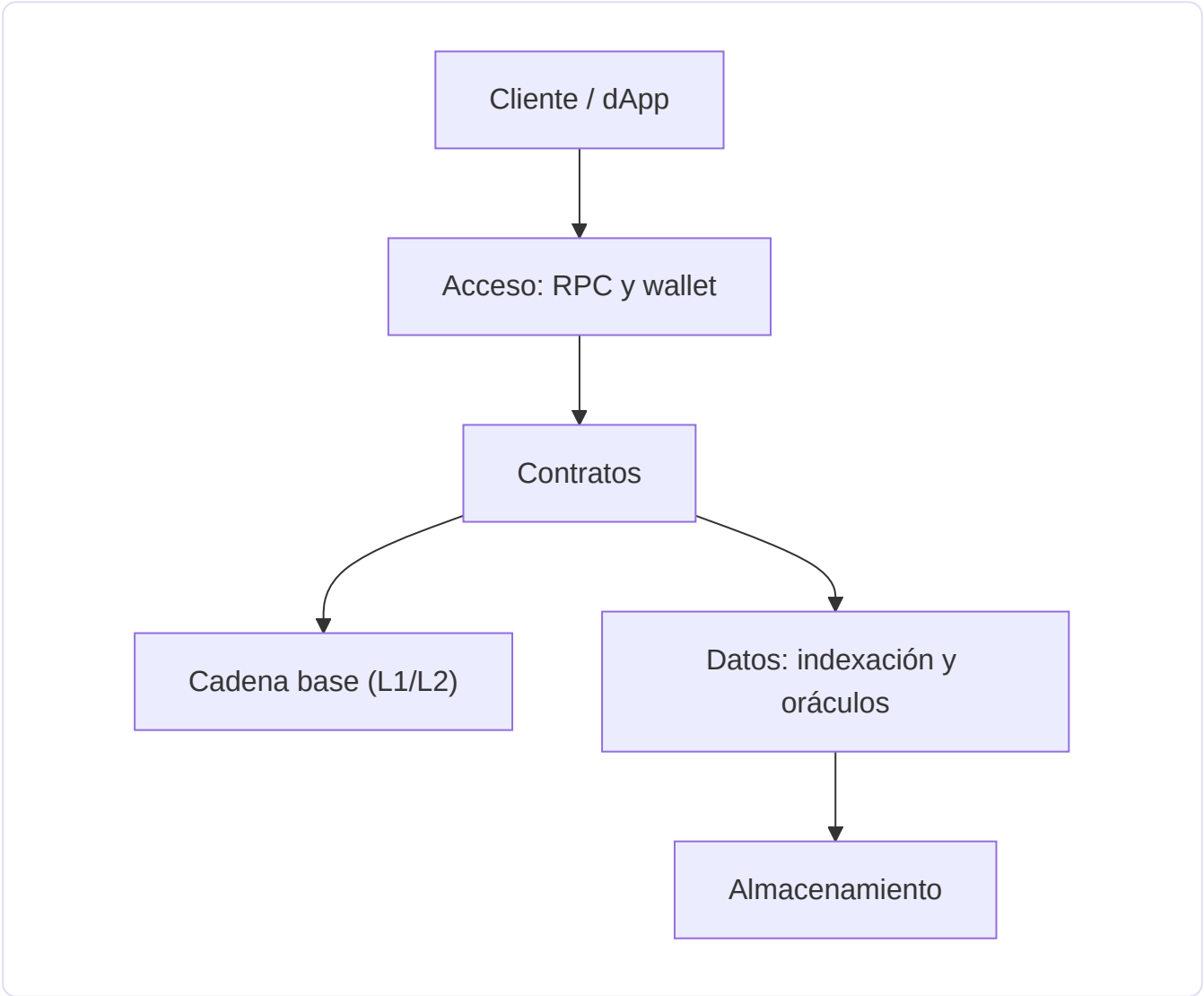
Mapa de tecnologías

⬅️ Volver al programa · 📚 Currículo · 🏭 Stack tecnológico

Mapa de tecnologías por capa: para cada necesidad, la tecnología principal del programa, alternativas para comparar, cuándo elegir cada una y su madurez. El análisis profundo capa por capa está en Industria · Stack tecnológico.

Las herramientas cambian; los conceptos, invariantes y modelos de amenaza son el centro del aprendizaje. Consulta siempre documentación oficial antes de desplegar.

Capas del stack



Cada capa se elige por separado: puedes cambiar el cliente sin tocar el contrato, o el indexador sin tocar la cadena. El aprendizaje se organiza por estas capas.

Tecnologías por capa

Necesidad	Principal	Alternativas	Cuándo elegir la alternativa	Madurez
Contratos EVM	Solidity + Foundry	Vyper, Hardhat	Vyper por simplicidad/seguridad; Hardhat si el equipo es JS	Alta
Cliente web	TypeScript + viem	ethers.js, web3.js	ethers si ya hay base heredada	Alta
Nodo local	Anvil	Hardhat Network, Geth dev	Hardhat Network si usas Hardhat; Geth para realismo	Alta
Análisis estático	Slither	Mythril, Semgrep	Mythril para ejecución simbólica; Semgrep reglas propias	Media-Alta
Fuzzing	Foundry fuzz, Echidna	Medusa	Echidna/Medusa para invariantes complejas	Media-Alta
Multisig	Safe	multisig nativo de la red	Nativo si la red lo ofrece de forma robusta	Alta
Indexación	eventos + indexador propio	The Graph, Ponder	The Graph para escala; propio para aprender	Media-Alta
Almacenamiento	IPFS con pinning	Arweave, almacenamiento tradicional	Arweave para permanencia; tradicional si no hay descentralización	Media
Oráculos	Chainlink (concepto/adaptador)	Pyth, API3, RedStone	Pyth para baja latencia; API3 para datos de primera parte	Alta
L2	Optimistic y ZK rollups	canales de estado, validiums	Canales para pagos repetidos; validium por costo de datos	Media-Alta
Empresa	Hyperledger Fabric	Besu, R3 Corda	Besu para EVM permisionada; Corda para acuerdos bilaterales	Alta

Lenguajes de contratos

Lenguaje	Plataforma	Modelo	Cuándo elegirlo	Madurez
Solidity	EVM	Orientado a contratos	Estándar del ecosistema, máxima tooling y talento	Alta
Vyper	EVM	Pythónico, minimalista	Se prioriza legibilidad y superficie reducida	Media-Alta
Rust (Anchor)	Solana	Cuentas y programas	Alto rendimiento en Solana	Alta
Cairo	Starknet	Nativo ZK (STARK)	Se necesita validez ZK y escala	Media
Move	Aptos, Sui	Recursos con seguridad de tipos	El activo debe ser un recurso no duplicable	Media

El programa enseña **Solidity** por su ecosistema, y presenta los demás como marcos mentales para entender otros modelos de ejecución.

Por qué el repositorio elige este stack

Elección	Motivo
Foundry	Pruebas en Solidity (menos cambio de contexto), fuzz e invariantes nativos, compilación y ejecución rápidas, `anvil` / `cast` integrados
viem	Cliente TypeScript tipado, moderno y ligero; API explícita que enseña bien qué ocurre en cada llamada
pnpm	Workspaces eficientes para un monorepo, instalaciones rápidas y deterministas, ahorro de disco con enlaces

Estas elecciones optimizan para **aprender el modelo mental con el menor ruido de entorno**: un solo stack estable en todo el programa reduce la carga cognitiva y mantiene las demostraciones reproducibles.

Herramientas de desarrollo auxiliares

Herramienta	Uso	Cuándo
`cast`	Consultas y transacciones desde la terminal	Depurar e interactuar con el contrato
`anvil`	Nodo EVM local instantáneo	Desarrollo y pruebas locales
`forge fmt`	Formato de código Solidity	Antes de cada commit
OpenZeppelin	Contratos y librerías auditadas	Estándares (ERC-20/721), control de acceso
Safe	Custodia multisig	Operación de fondos y roles críticos

Criterios de selección

Al comparar tecnologías, el programa evalúa siempre los mismos ejes en lugar de seguir tendencias:

- **Madurez:** ¿está probada en producción y mantenida activamente?
- **Ecosistema:** ¿hay tooling, documentación y talento disponibles?
- **Superficie de ataque:** ¿cuánta complejidad y confianza añade?
- **Costo:** gas, infraestructura y esfuerzo de operación.
- **Reversibilidad:** ¿qué tan caro es cambiar de decisión después?

Cómo evoluciona el stack

Registra cada elección tecnológica en un ADR (registro de decisión de arquitectura) con sus alternativas y el motivo. Así el stack cambia de forma trazable y no por moda; ver la metodología en Industria · Stack tecnológico.

Recursos relacionados

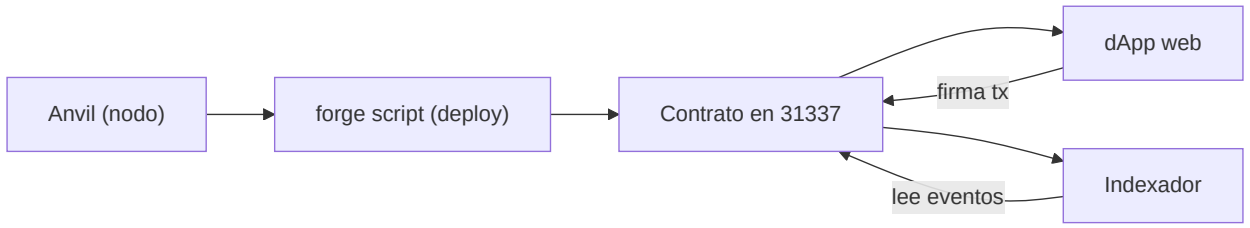
- Industria · Stack tecnológico — análisis por capas.
- Recursos oficiales — documentación de referencia.
- Mejores prácticas — cómo usar bien estas herramientas.

Despliegue integral local

> ⬅️ Volver al programa · 📚 Currículo · 🖥️ dApps

Guía para levantar el sistema completo en local: nodo Anvil, contrato desplegado con Foundry, interfaz web e indexador de eventos. Cada fase indica **qué hace**, el **comando**, la **salida esperada** y su **verificación**. El desarrollo de la dApp se estudia en 07 · dApps.

Flujo local



Puertos y servicios

Servicio	Puerto por defecto	Rol
Anvil (RPC)	8545	Nodo EVM local
dApp web (Vite)	5173	Interfaz de usuario
Indexador	—	Proceso de fondo, escribe estado en disco

Fase 1 · Dependencias

Qué hace: prepara las herramientas necesarias.

Requisitos: Node.js LTS, pnpm y Foundry. Docker es necesario **solo** para Bitcoin regtest.

Verificación: `node -v`, `pnpm -v` y `forge --version` responden sin error.

Fase 2 · Nodo local (Terminal A)

Qué hace: arranca un nodo EVM local con chain ID 31337.

```
anvil --chain-id 31337
```

Salida esperada: una lista de cuentas de prueba con sus claves privadas y el RPC escuchando en `http://127.0.0.1:8545`.

Verificación: la terminal muestra "Listening on 127.0.0.1:8545".

Fase 3 · Contrato (Terminal B)

Qué hace: instala dependencias, configura el entorno y despliega el contrato.

```
cd projects/community-funding
forge install foundry-rs/forge-std --no-commit
cp ../../.env.example .env
# Usa una clave de prueba mostrada por Anvil.
set -a; source .env; set +a
forge script script/Deploy.s.sol:DeployCommunityFunding \
  --rpc-url "$RPC_URL" --broadcast
```

Salida esperada: `forge` reporta la transacción de despliegue y la dirección del contrato.

Verificación: registra la **dirección desplegada, chain ID y bloque**. No copies una clave fuera del entorno local.

Fase 4 · Interfaz

Qué hace: configura y sirve la dApp web apuntando al contrato recién desplegado.

```
cp apps/community-funding-web/.env.example apps/community-funding-web/.env
# Configura la dirección anterior.
pnpm --filter @blockchain-course/community-funding-web dev
```

Salida esperada: Vite sirve la interfaz en `http://localhost:5173`.

Verificación: conecta una wallet configurada para Anvil, crea una campaña mediante `cast` o un script, y contribuye desde la interfaz.

Fase 5 · Indexador

Qué hace: lee los eventos del contrato y deriva un estado consultable.

```
cp apps/event-indexer/.env.example apps/event-indexer/.env
set -a; source apps/event-indexer/.env; set +a
pnpm --filter @blockchain-course/event-indexer start
```

Salida esperada: el proceso reporta bloques procesados y eventos capturados.

Verificación: inspecciona `event-indexer-state.json`, produce un evento nuevo y repite. Para producción debes agregar reorg handling, base transaccional y observabilidad.

Fase 6 · Cierre

Qué hace: detiene los servicios y limpia secretos locales.

Detén servicios, elimina claves locales exportadas y conserva solo direcciones, txids locales y bitácora **sin secretos**.

Verificación: no queda ninguna clave privada en `.env` ni en el historial de la terminal.

Troubleshooting

Síntoma	Causa probable	Solución
`connection refused` en 8545	Anvil no está corriendo	Arranca la Fase 2 en otra terminal
`forge script` falla al firmar	`RPC_URL` /clave no cargadas	Vuelve a `set -a; source .env; set +a`
La dApp no muestra estado	Dirección del contrato mal configurada	Corrige el `.env` de la interfaz
Wallet rechaza la red	Chain ID distinto de 31337	Reconfigura la red de la wallet a 31337
El indexador no ve eventos	Empezó después de emitirlos	Reinícialo desde el bloque de despliegue
Puerto 5173 ocupado	Otra instancia de Vite activa	Ciérrala o deja que Vite use otro puerto

Recursos relacionados

- 07 · dApps
- Modelo de amenazas del proyecto
- Operación e incidentes

Operación y respuesta ante incidentes

⬅️ Volver al programa · 📚 Currículo · 🛡️ Mejores prácticas

Runbook profesional de respuesta a incidentes para protocolos on-chain. En este dominio los incidentes se miden en minutos y los fondos son programáticamente extraíbles: la diferencia entre una pérdida parcial y una total suele ser la preparación previa.

Clasificación de severidad

Severidad	Definición	Ejemplos	Primera acción
SEV-1	Exploit activo con fondos en riesgo inmediato	Drenado en curso, clave de administrador comprometida en uso	Contener ya: pausar, sin esperar consenso completo
SEV-2	Vulnerabilidad crítica confirmada, aún sin explotar	Reporte de whitehat o auditor sobre contrato con fondos	War room privado; preparar mitigación antes de cualquier divulgación
SEV-3	Anomalía operativa con impacto acotado	Oráculo con precio desviado u obsoleto, liquidaciones anómalas	Verificar con segunda fuente; activar protecciones (pausar mercado afectado)
SEV-4	Degradación de infraestructura sin riesgo de fondos	RPC caído, indexador atrasado, frontend con errores	Canal operativo normal; comunicar si afecta usuarios

Ante la duda entre dos niveles, clasifica en el más alto: bajar la severidad después es barato, subirla tarde es carísimo.

Preparación previa (antes del despliegue)

La respuesta se decide antes del incidente. Checklist mínima:

- Inventario de contratos, roles, claves y dependencias.
- Multisig y timelock probados.
- Monitores para eventos, balances, pausas e invariantes, con alertas que despierten a alguien (no solo un dashboard).
- Contactos, severidades y canal de divulgación definidos; contactos de custodios, exchanges y SEAL 911 a mano.
- Copia reproducible del bytecode y parámetros.
- Simulación de actualización, pausa y recuperación.
- War room preconfigurado (canal privado con las personas correctas ya dentro).

- Ensayos trimestrales: simular un incidente completo, con reloj, incluyendo la comunicación.

Roles del incidente

Rol	Responsabilidad	Regla clave
Comandante del incidente (incident commander)	Coordina, decide, mantiene la línea de tiempo	No teclea: dirige. Una sola voz de mando
Responsable técnico	Diagnóstico, contención y remediación en código	Todo cambio se prueba en fork antes de mainnet
Comunicación	Comunicados, redes, respuesta a usuarios y prensa	Solo publica hechos confirmados por el comandante
Legal / cumplimiento	Obligaciones regulatorias, contacto con autoridades	Involucrado desde SEV-2 hacia arriba, no al final

En equipos pequeños una persona cubre varios roles, pero los roles se nombran explícitamente al abrir el incidente.

Runbook por fases

Tiempos objetivo para SEV-1; en severidades menores se relajan, no se eliminan.

1. Detectar (objetivo: minutos)

- La detección debe venir de alertas automáticas (eventos, balances, invariantes), no de Twitter.
- Conservar evidencia desde el primer momento: hashes de transacciones, estado, logs, capturas del mempool.
- Registrar la hora exacta de cada observación: la línea de tiempo del post-mortem se construye ahora.

2. Confirmar y clasificar (≤ 15 min)

- Verificar que la anomalía es real con una segunda fuente antes de escalar (otro RPC, otro oráculo, un fork local).
- Clasificar impacto: fondos, usuarios y redes afectadas; asignar severidad y abrir el war room.
- Nombrar explícitamente al comandante del incidente y a los demás roles.

3. Contener (≤ 30-60 min)

- Usar solo facultades previamente documentadas: pausar contratos, deshabilitar el frontend para frenar depósitos nuevos, revocar o rotar claves comprometidas.

- Contactar custodios y exchanges para congelar rutas de salida de fondos; contactar SEAL 911 si se necesita refuerzo.
- Evaluar efectos secundarios de cada acción de contención (una pausa también puede bloquear retiros legítimos o liquidaciones necesarias).

4. Comunicar (primer comunicado ≤ 2 h)

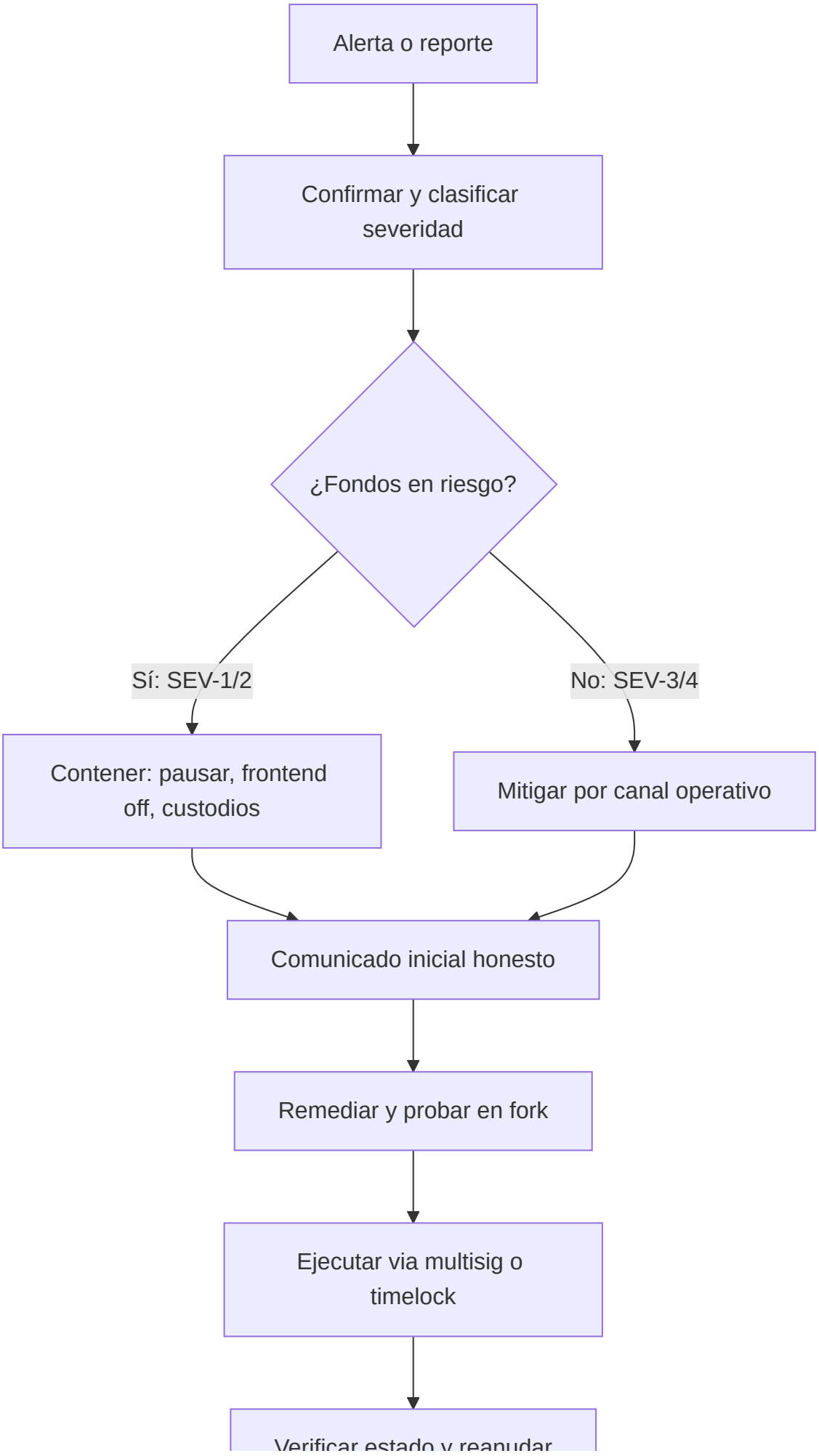
- Hechos confirmados, no especulación; una sola voz autorizada.
- Plantilla inicial honesta: "Estamos investigando un incidente que afecta [componente]. Hemos pausado [funciones]. No interactúes con el protocolo hasta nuevo aviso. Próxima actualización en [plazo]."
- Prometer un plazo de actualización y cumplirlo vale más que prometer soluciones. El silencio prolongado se interpreta como fuga del equipo (exit scam) y agrava el daño.

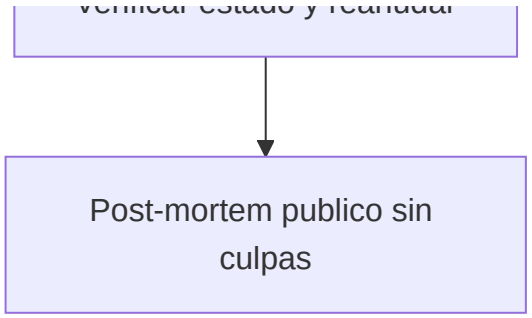
5. Remediar

- Corregir y probar en fork/local; revisión de al menos dos personas antes de tocar mainnet.
- Ejecutar mediante multisig/timelock según urgencia y facultades.
- Verificar estado y compensaciones antes de reanudar; la reanudación es gradual (guarded relaunch), no un interruptor.

6. Post-mortem (≤ 2 semanas)

- Retrospectiva pública y sin culpas (blameless): causas raíz, línea de tiempo, montos, controles nuevos con responsables y fechas.
- Publicarla aunque duela: los mejores equipos del ecosistema se reconocen por sus post-mortems, no por su ausencia de incidentes.


Alerta o reporte
Confirmar y clasificar severidad
¿Fondos en riesgo?
Sí: SEV-1/2
No: SEV-3/4
Contener: pausar, frontend off, custodios
Mitigar por canal operativo
Comunicado inicial honesto
Remediar y probar en fork
Ejecutar via multisig o timelock
Verificar estado y reanudar



Ejemplos reales de buena gestión

Citados como referencia de proceso, no como garantía de resultado:

- **Euler Finance (2023)**: tras un exploit de ~197 M USD, el equipo combinó negociación pública on-chain, presión coordinada y colaboración con investigadores; el atacante devolvió prácticamente todos los fondos en semanas. Lección: mantener canales de negociación abiertos y comunicar con disciplina puede recuperar lo que la técnica ya no puede.
- **Curve (2023, vulnerabilidad de Vyper)**: la coordinación entre equipos, whitehats y MEV searchers permitió rescatar una parte significativa de los fondos en riesgo. Lección: las relaciones con el ecosistema de seguridad se construyen antes del incidente.
- Como contraejemplo estudiado en el módulo de seguridad: los puentes explotados en 2021-2022 (Ronin, Wormhole, Nomad) muestran el costo de detectar tarde y de claves concentradas.

Ejercicio

Simula un oráculo obsoleto y luego una clave de operador comprometida. Para cada caso define alertas, autoridad de pausa, efectos secundarios, comunicación y condición de reanudación. Ejecuta el ejercicio con reloj y registra la línea de tiempo como lo harías en un incidente real; la bitácora cuenta como evidencia de evaluación.

Referencias

- SEAL 911 — línea de emergencia de la Security Alliance para incidentes en curso.
- Trail of Bits — publicaciones y guías y su repositorio secure-contracts.
- Immunefi — programas de recompensas y estándares de divulgación responsable.
- rekt.news — post-mortems de incidentes reales (útil para ensayos; verifica detalles técnicos en fuentes primarias).
- Más contexto de prevención en mejores-practicas.md y en el modelo de amenazas del proyecto.

Modelo de amenazas · Community Funding

⬅️ Volver al programa · 📚 Currículo · 🔐 Seguridad

Modelo de amenazas del proyecto integrador **Community Funding** (financiamiento colectivo on-chain). Estructura los activos, actores, superficie de ataque, amenazas con impacto y mitigación, el flujo de confianza y las invariantes de seguridad. Metodología alineada con 09 · Seguridad; el proyecto se describe en Capstone.

Activos

Activo	Por qué importa
Fondos aportados	Valor real bajo custodia del contrato
Derecho a reembolso	El contribuyente recupera si la campaña falla
Derecho de retiro del creador	El creador cobra solo si se cumplen las condiciones
Integridad de eventos	El indexador y la UI dependen de logs correctos
Disponibilidad del servicio	Sin acceso, los usuarios no operan
Confianza del usuario	Un incidente la destruye y no se recupera fácil

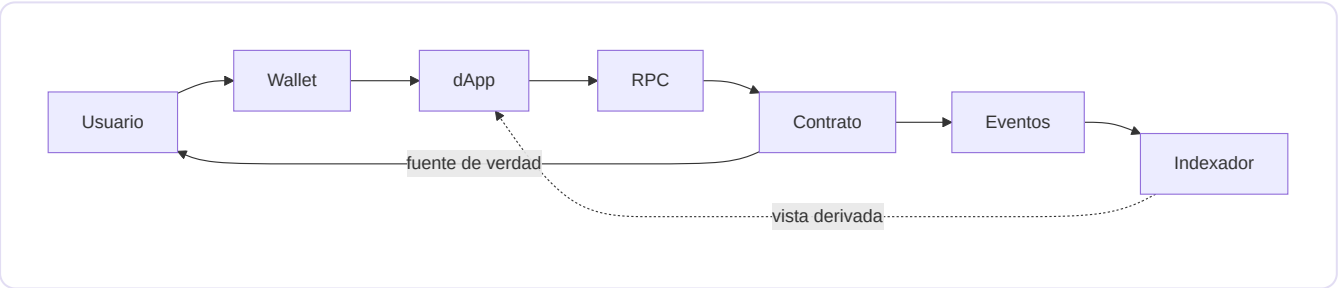
Actores

Actor	Motivación	Confianza asumida
Contribuyente	Financiar una campaña y poder reembolsarse	Ninguna especial
Creador	Recaudar y retirar fondos	Limitada; el contrato lo restringe
Operador de interfaz	Servir la dApp	La UI puede mentir; no es la verdad
Atacante externo	Robar o bloquear fondos	Hostil
Insider / operador	Acceso privilegiado a claves o infraestructura	Mínimo privilegio
Minero / secuenciador	Ordenar transacciones	Puede reordenar y censurar

Superficie de ataque por componente

Componente	Superficie	Riesgo principal
Contrato	Funciones públicas, control de acceso, manejo de valor	Reentrancia, doble retiro, escalada de privilegios
dApp	Frontend, dependencias, dominio	Phishing, código malicioso, suplantación
RPC	Endpoint de lectura/envío	Censura, respuestas engañosas, indisponibilidad
Claves	Custodia del creador/operador	Robo, pérdida, firma no autorizada
Oráculo / datos externos	Feeds y datos off-chain	Dato obsoleto o manipulado
Indexador	Derivación de estado desde logs	Pérdida o reordenamiento por reorganizaciones

Flujo de confianza



Solo el **contrato** determina el estado. La wallet firma, la UI presenta, el RPC transporta y el indexador deriva: todos pueden equivocarse o mentir y deben verificarse contra la cadena.

Amenazas (impacto · probabilidad · mitigación)

Clasificación por categoría inspirada en STRIDE.

Categoría	Amenaza	Impacto	Prob.	Mitigación	Riesgo residual
Elevación	Reentrancia en retiro/reembolso	Alto	Baja	CEI + reentrancy guard	Llamadas cruzadas futuras
Manipulación	Doble retiro	Alto	Baja	Flag y estado antes de interacción	Error en upgrade futuro
Manipulación	Aporte fuera de plazo	Medio	Media	Validación temporal	Manipulación acotada de timestamp
Suplantación	Phishing de contrato/red	Alto	Media	UI muestra chain y dirección	Usuario ignora la advertencia
Denegación	RPC censura o engaña	Medio	Media	Redundancia y verificación on-chain	Disponibilidad
Repudio	Indexador pierde/reordena logs	Medio	Media	Checkpoint, orden y replay	Reorganizaciones profundas
Divulgación	Pérdida o robo de claves	Alto	Baja	Wallet/multisig según riesgo	Recuperación y gobernanza
Fuera de alcance	Creador incumple promesa off-chain	Medio	Media	Reglas y comunicación	El contrato no valida el mundo real

Invariantes de seguridad

Propiedades que **nunca** deben romperse. Se codifican como pruebas de invariante en Foundry y guían tanto el diseño como la auditoría.

- Cada aporte se reclama o reembolsa **como máximo una vez**.
- El creador **no retira** antes del plazo y de alcanzar la meta.
- Una campaña fallida **no puede** ser reclamada por el creador.
- La contabilidad por contribuyente no aumenta sin recibir valor.
- La suma de saldos por contribuyente nunca excede el balance del contrato.

Cómo se prueba este modelo

Amenaza	Prueba que la cubre
Reentrancia	Test con contrato atacante que reingresa en el retiro
Doble retiro / doble reembolso	Test de invariante sobre el estado de cada aporte
Aporte fuera de plazo	Test que avanza el tiempo con `vm.warp`
Contabilidad	Invariante de suma de saldos vs. balance

El modelo de amenazas no es un documento estático: cada amenaza mitigada debe tener una prueba que falle si la mitigación se rompe en un cambio futuro.

Recursos relacionados

- 09 · Seguridad y auditoría
- Capstone · Mejores prácticas
- Operación e incidentes

Recursos oficiales y fuentes primarias

⬅️ Volver al programa · 📚 Currículo · 📖 Bibliografía

Catálogo curado de recursos por categoría. Revisado: 2026-07-28. Las URL y el estado (activo/gratuito) pueden cambiar: verifica antes de citar.

Documentación oficial

Recurso	URL	Para qué sirve	Costo
Bitcoin Developer Reference	https://developer.bitcoin.org/reference/	Referencia técnica del protocolo Bitcoin	Gratuito
Bitcoin Core	https://bitcoincore.org/	Cliente de referencia, notas de versión	Gratuito
Ethereum Developer Docs	https://ethereum.org/developers/docs/	Punto de entrada oficial al desarrollo en Ethereum	Gratuito
Seguridad de smart contracts (ethereum.org)	https://ethereum.org/developers/docs/smart-contracts/security/	Guía oficial de seguridad de contratos	Gratuito
Solidity	https://docs.soliditylang.org/	Referencia del lenguaje y registro de bugs del compilador	Gratuito
Especificaciones EIP/ERC	https://eips.ethereum.org/	Texto normativo de estándares (ERC-20, EIP-1559, EIP-4844…)	Gratuito
Cosmos SDK	https://docs.cosmos.network/	Desarrollo de cadenas de aplicación	Gratuito
Solana Developers	https://solana.com/docs	Documentación oficial de Solana	Gratuito
Hyperledger Fabric	https://hyperledger-fabric.readthedocs.io/	Redes permisionadas empresariales	Gratuito

Libros de referencia

Los libros del programa, con edición y capítulos recomendados por módulo, están catalogados en bibliografia.md. Regla general: prefiere la edición más reciente (el ecosistema deja obsoletos los detalles en 2-3 años) y contrasta cualquier afirmación técnica con la especificación vigente.

Exploradores y datos

Recurso	URL	Para qué sirve	Costo
Etherscan	https://etherscan.io/	Explorar transacciones, contratos verificados y eventos en Ethereum	Gratuito (API con plan pago)
mempool.space	https://mempool.space/	Mempool, tarifas y bloques de Bitcoin en vivo	Gratuito
L2BEAT	https://l2beat.com/	Estado, riesgos y supuestos de confianza de cada L2 (no solo TVL)	Gratuito
DefiLlama	https://defillama.com/	TVL, volúmenes y métricas DeFi entre cadenas	Gratuito
Dune	https://dune.com/	Consultas SQL sobre datos on-chain, dashboards comunitarios	Gratuito (planes pagos para uso intensivo)

Testnets y faucets

Recurso	URL	Para qué sirve	Costo
Sepolia (documentación)	https://ethereum.org/developers/docs/networks/	Testnet recomendada para probar contratos y dApps	Gratuito
Faucets de Sepolia	https://faucetlink.to/sepolia	Directorio de faucets activos (rotan con frecuencia; verifica cuál funciona)	Gratuito

Nunca uses claves de testnet en mainnet ni viceversa; trata las claves de prueba como desechables.

Herramientas

Recurso	URL	Para qué sirve	Costo
Foundry Book	https://book.getfoundry.sh/	Herramienta principal del programa: forge, cast, anvil	Gratuito
Remix	https://remix.ethereum.org/	IDE en el navegador; útil para explorar y prototipar, no para proyectos serios	Gratuito
viem	https://viem.sh/	Cliente TypeScript para interactuar con EVM desde dApps	Gratuito
Tenderly	https://tenderly.co/	Simulación, depuración y monitoreo de transacciones	Plan gratuito limitado; pago para equipos

Seguridad

Recurso	URL	Para qué sirve	Costo
OWASP Smart Contract Top 10	https://owasp.org/www-project-smart-contract-top-10/	Clases de vulnerabilidad más frecuentes	Gratuito
SWC Registry	https://swcregistry.io/	Catálogo histórico de debilidades de contratos (útil, aunque ya no se actualiza activamente)	Gratuito
secure-contracts (Trail of Bits)	https://secure-contracts.com/	Guías de desarrollo seguro, Slither y Echidna	Gratuito
Registro de bugs de Solidity	https://docs.soliditylang.org/en/latest/bugs.html	Bugs conocidos por versión del compilador	Gratuito
Immunefi	https://immunefi.com/	Recompensas por vulnerabilidades y divulgación responsable	Gratuito para investigadores
Security Alliance (SEAL)	https://securityalliance.org/	Respuesta a incidentes (SEAL 911) y ejercicios; ver operacion-incidentes.md	Gratuito

Comunidades y noticias serias

Recurso	URL	Para qué sirve	Costo
Ethereum Magicians	https://ethereum-magicians.org/	Discusión de EIPs y estándares	Gratuito
ethresear.ch	https://ethresear.ch/	Investigación de protocolo (consenso, DA, MEV)	Gratuito
Boletines tipo "Week in Ethereum"	—	El boletín original cesó en 2024-2025 y surgieron sucesores; verifica cuál sigue activo antes de suscribirte	Gratuito
EIPs en GitHub	https://github.com/ethereum/EIPs	Seguir propuestas en tiempo real	Gratuito

Regulación

Recurso	URL	Para qué sirve	Costo
MiCA (texto oficial, EUR-Lex)	https://eur-lex.europa.eu/legal-content/ES/TXT/?uri=CELEX%3A32023R1114	Reglamento europeo de criptoactivos, en aplicación plena desde diciembre de 2024	Gratuito
CMF Chile	https://www.cmfchile.cl/	Regulador financiero chileno; Ley Fintech y registro de proveedores	Gratuito

Contexto local ampliado en chile-regulacion-tributacion.md.

Criterio de uso de fuentes

- **Documentación no es marketing**: los sitios de proyectos mezclan ambas; las afirmaciones sobre comportamiento, versiones o seguridad deben verificarse en especificaciones, código fuente y documentación oficial.
- **Cuidado con "influencers" financieros**: quien recomienda comprar un activo suele tener posición o patrocinio en él. Este programa enseña tecnología, no decisiones de inversión.
- **Verifica fechas**: un artículo de 2021 sobre tarifas, L2 o staking está casi con certeza desactualizado. Toda fuente externa debe registrar fecha de consulta.
- **Cifras volátiles** (TVL, tarifas, rankings de L2): no las cites de memoria ni de artículos; consúltalas en vivo en L2BEAT, DefiLlama o el explorador correspondiente.
- Los blogs ayudan a comprender, pero no sustituyen a la fuente primaria cuando la afirmación importa.

Diseño pedagógico

⬅️ Volver al programa · 📚 Currículo · 📝 Evaluación

Este documento describe **cómo está diseñado el aprendizaje** del programa: el ciclo de trabajo de cada módulo, la taxonomía de objetivos, la construcción de una lección, el manejo de la carga cognitiva y la estrategia de evaluación. No es un temario; es la ingeniería instruccional que sostiene los 19 módulos.

Principios

- La aprobación no depende solo de compilar. El estudiante justifica decisiones, identifica riesgos y reproduce resultados.
- Cada concepto se estudia por su modelo mental, no por su sintaxis.
- Los casos reales anclan la teoría; las analogías se usan y luego se rompen.
- La evidencia es reproducible: comando, resultado esperado y resultado obtenido.

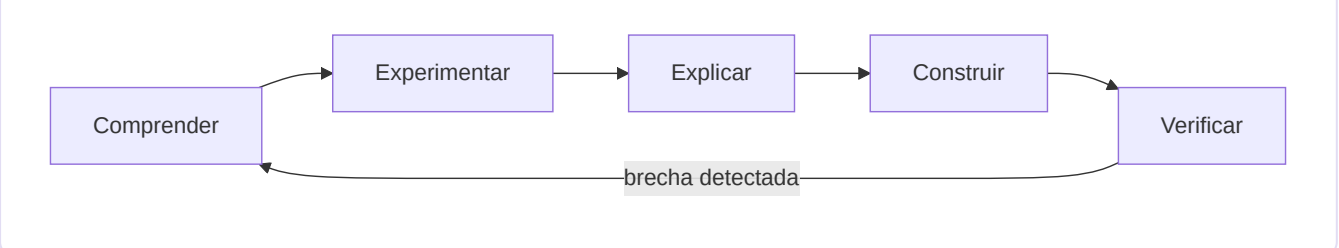
Competencias finales

Al completar el programa, el estudiante puede:

1. decidir justificadamente si un problema necesita blockchain;
2. explicar criptografía, consenso, Bitcoin y EVM con modelos propios;
3. implementar, probar y conectar contratos;
4. descubrir fallos técnicos y económicos;
5. diseñar operación, gobernanza y respuesta ante incidentes;
6. comparar ecosistemas sin depender de marketing;
7. presentar un producto con costos, riesgos y límites explícitos.

El ciclo de aprendizaje

Cada módulo recorre cinco fases. No se avanza a "construir" sin haber "experimentado", ni se cierra sin "verificar". El ciclo es iterativo: verificar puede devolver a comprender.



Fase	Pregunta que responde	Actividad típica
Comprender	¿Qué problema resuelve y por qué?	Modelo mental, contraejemplo
Experimentar	¿Qué ocurre si lo ejecuto?	Laboratorio guiado, demo reproducible
Explicar	¿Puedo enseñarlo con mis palabras?	Nota técnica, diagrama propio
Construir	¿Puedo aplicarlo a un caso nuevo?	Reto independiente, ADR
Verificar	¿Cómo sé que funciona y dónde falla?	Pruebas, rúbrica, criterio de dominio

Taxonomía de objetivos

Cada nivel cognitivo se mapea a un verbo observable, un tipo de actividad y una evidencia calificable. Un módulo maduro cubre desde recordar hasta crear.

Nivel	Verbo observable	Tipo de actividad	Evidencia
Recordar	identificar	cuestionario diagnóstico	test corto
Entender	explicar	nota técnica, diagrama	documento
Aplicar	ejecutar	laboratorio	comandos y salida
Analizar	comparar	matriz de decisión	ADR o tabla
Evaluar	auditar	revisión, CTF	informe de hallazgos
Crear	diseñar	reto, proyecto final	artefacto verificable

Construcción de una lección

Cada módulo usa la plantilla de `curriculum/MODULE_TEMPLATE.md`. Cada sección tiene un propósito instruccional explícito.

Sección	Propósito
Diagnóstico breve	Activar conocimiento previo y detectar brechas
Explicación y demostración	Presentar el modelo mental con un ejemplo real
Práctica guiada	Reducir carga cognitiva con andamiaje
Reto independiente	Transferir a un contexto nuevo sin ayuda
Reflexión y bitácora	Consolidar y hacer visible el razonamiento
Prueba o rúbrica	Verificar el dominio con evidencia objetiva
Criterio de dominio	Definir el umbral para avanzar

Andamiaje y carga cognitiva

El programa gestiona la **carga cognitiva** para que el esfuerzo se invierta en aprender, no en pelear con el entorno.

- **Andamiaje decreciente:** la práctica guiada da estructura; el reto la retira.
- **Un concepto nuevo por vez:** las herramientas se introducen cuando el concepto ya se entiende, no antes.
- **Entorno estable:** el mismo stack (Foundry, viem, Anvil) en todo el programa evita recargar memoria de trabajo con configuraciones cambiantes.
- **Fragmentación:** se construyen "trozos" reutilizables (hash, cuenta, invariante) que luego se combinan en sistemas.

Contenido en capas: servir al principiante y al experto a la vez

Un curso que va de cero a producción tiene un problema estructural: el mismo texto lo lee alguien que nunca escribió un contrato y alguien que lleva años en el sector. Bajar el nivel aburre al segundo; subirlo expulsa al primero.

La solución en este programa **no** es escribir dos cursos, sino escribir cada bloque denso en **cuatro capas visibles**, para que cada lector sepa dónde entrar y qué puede saltarse sin perder el hilo:

Capa	Qué contiene	Para quién
1 · **La idea en llano**	El concepto en lenguaje corriente, con una analogía y sin jerga sin glosar	Quien llega sin base. Nadie debería tener que buscar un término fuera para seguir leyendo
2 · **El cálculo trabajado**	El ejemplo completo, con números reales y cada paso a la vista	Quien está aprendiendo haciendo. Aquí es donde se resuelve el atasco
3 · 🎓 **Si ya dominas esto**	Bordes, excepciones y detalle fino, en un bloque **plegable**	Quien ya conoce el tema. Plegado, no intimida a quien no lo necesita
4 · 💡 **En una frase**	La idea que hay que retener si solo se retiene una	Todos. Fija el aprendizaje y sirve de repaso

Reglas de aplicación:

- **La capa 3 va siempre plegada** (`<details>`). Que exista no debe aumentar la carga cognitiva de quien no la abre.
- **Ningún término de la capa 1 se usa sin glosar.** Si aparece "vbyte" o "calldata", se explica en la misma frase, no en un enlace.
- **La capa 2 usa números concretos, no símbolos.** "209 vB × 12 sat/vB = 2 508 sat" enseña más que "tamaño × tasa = comisión".

- **La capa 4 nunca introduce información nueva.** Si algo solo aparece ahí, está en el sitio equivocado.
- **Un bloque puede omitir la capa 3** si el tema no tiene profundidad extra honesta. Inventar detalle para rellenar la plantilla es peor que no tenerla.

Ejemplos aplicados: la comisión de Bitcoin en el módulo 04, el desglose de gas en el módulo 05, la lectura de trazas en el módulo 06 y los decimales en el módulo 07.

Evaluación formativa y sumativa

Ambas conviven; ver Evaluación para rúbricas y umbrales.

Tipo	Cuándo	Función	Ejemplos
Formativa	Durante el módulo	Corregir el rumbo, dar retroalimentación	Diagnóstico, ticket de salida, revisión en pareja
Sumativa	Al cerrar el módulo o etapa	Certificar dominio	Reto verificable, CTF, proyecto final

La retroalimentación formativa es frecuente y de bajo costo; la sumativa es escasa y exige evidencia reproducible.

Casos reales y límites de la analogía

- **Casos reales:** cada módulo ancla la teoría en incidentes o protocolos concretos (reentrancia del DAO, congestión de gas, fallos de oráculo). Un modelo sin caso queda como abstracción inerte.
- **Límites de la analogía:** toda analogía se presenta y luego **se rompe** de forma explícita ("hash ≠ cifrado", "dirección ≠ persona"). Nombrar dónde falla la analogía previene modelos mentales erróneos que cuestan caro en producción.

Recursos relacionados

- Planes de clase — sesiones listas para instructores.
- Evaluación — rúbricas y criterios de dominio.
- Glosario — vocabulario común del programa.

Evaluación y rúbrica

⬅️ Volver al programa · 📚 Currículo · ✅ Checkpoints

Este documento define cómo se mide el aprendizaje en el programa: los pesos de cada componente, la rúbrica por dimensión, la evidencia que se registra y las condiciones de aprobación.

Componentes y pesos

Componente	Peso	Qué mide
Cuestionarios de módulo	20 %	Comprensión conceptual verificada por checkpoint
Laboratorios y bitácora	30 %	Práctica guiada, reproducibilidad y reflexión escrita
Proyectos de nivel	30 %	Integración de habilidades al cierre de cada nivel
Proyecto final (capstone)	20 %	Sistema completo con modelo de amenazas y demo (capstone)

Rúbrica por entrega

Cada entrega (laboratorio, proyecto de nivel o capstone) se evalúa sobre 100 puntos:

Dimensión	Puntos	Evidencia
Comprensión	20	Explica decisiones, límites y alternativas
Corrección	20	Cumple requisitos e invariantes
Pruebas	20	Casos felices, fallos, fuzz/invariantes según nivel
Seguridad	20	Modelo de amenazas y mitigaciones
Calidad	10	Código legible, reproducible y documentado
Ética y operación	10	Privacidad, costos, gobernanza e incidentes

Dos reglas no negociables:

- Una vulnerabilidad crítica no reconocida limita la calificación total a 60.
- Una clave real publicada implica detener y rotar credenciales antes de continuar.

Niveles de desempeño por dimensión

Comprensión

Nivel	Descripción
Insuficiente	Repite definiciones sin poder justificar decisiones propias
Suficiente	Explica qué hizo y por qué, con imprecisiones menores
Bueno	Justifica decisiones frente a alternativas y reconoce límites
Sobresaliente	Anticipa casos borde y conecta el diseño con compromisos del ecosistema real

Corrección

Nivel	Descripción
Insuficiente	No compila o incumple requisitos centrales
Suficiente	Cumple el criterio de aceptación del módulo en el camino feliz
Bueno	Cumple requisitos e invariantes también en entradas adversas
Sobresaliente	Documenta invariantes explícitos y demuestra que se preservan

Pruebas

Nivel	Descripción
Insuficiente	Sin pruebas o solo pruebas triviales que no fallan nunca
Suficiente	Casos felices y de fallo básicos que pasan con `forge test` o `pnpm lab:*`
Bueno	Cobertura de reverts, límites y al menos un fuzz test donde aplica
Sobresaliente	Invariantes y fuzzing dirigidos por el modelo de amenazas

Seguridad

Nivel	Descripción
Insuficiente	Vulnerabilidades conocidas sin mención (reentrancia, control de acceso)
Suficiente	Identifica los riesgos principales de su entrega
Bueno	Modelo de amenazas escrito con mitigaciones implementadas
Sobresaliente	Analiza vectores económicos (oráculos, MEV) además de los de código

Calidad, ética y operación

Nivel	Descripción
Insuficiente	Código irreproducible o sin documentación mínima
Suficiente	README con pasos que funcionan desde cero
Bueno	Código legible, sin secretos, con costos y supuestos documentados
Sobresaliente	Considera operación real: pausas, monitoreo y plan de incidentes (runbook)

Evidencia y registro de progreso

El avance se registra en el archivo de progreso del estudiante (plantilla en `student/progress.example.json`). Por cada módulo se guarda:

- Fecha de inicio y de cierre del módulo.
- Resultado del cuestionario del checkpoint (assessments/checkpoints.md).
- Comandos de verificación ejecutados y su salida resumida (`pnpm lab:*` , `forge test`).
- Enlace o ruta a la bitácora de laboratorio (qué intentaste, qué falló, qué aprendiste).
- Nota de la entrega según la rúbrica anterior.

La bitácora no es burocracia: escribir por qué algo falló es la evidencia más fuerte de comprensión y pesa dentro del 30 % de laboratorios.

Criterios de aprobación

- **Por módulo**: nota ≥ 80/100 en la entrega del módulo y checkpoint aprobado. Es el mismo umbral que exige el certificado del programa.
- **Retos verificables**: cada módulo define un criterio de aceptación explícito (una salida esperada, una prueba que debe pasar, un invariante que debe sostenerse). El reto está aprobado cuando el criterio se cumple de forma reproducible, no cuando "parece funcionar".
- **Por nivel**: todos los módulos del nivel aprobados más el proyecto de nivel con nota ≥ 80.
- **Programa completo**: todos los niveles más el capstone con nota ≥ 80.

Un módulo con nota entre 60 y 79 puede reentregarse una vez corrigiendo las observaciones; se registra la nota de la reentrega.

Autoevaluación y evaluación con instructor

Modalidad	Cómo funciona	Límite
Autoevaluación	Aplicas la rúbrica tú mismo usando los criterios de aceptación y las carpetas `solutions/` como pauta de revisión	Honesta solo si primero resolviste sin mirar la solución
Con instructor	El instructor revisa la entrega, la bitácora y hace preguntas de defensa oral	Es la modalidad válida para certificación formal

En autoevaluación, el criterio operativo es: si no puedes explicar una línea de tu propia entrega, la dimensión Comprensión no supera "insuficiente".

Integridad académica

- **Colaborar sí**: discutir enfoques, depurar en pareja y revisar código ajeno es parte del oficio y se fomenta.
- **Copiar soluciones no**: las carpetas `solutions/` existen como criterio de revisión posterior, no como punto de partida. Entregar una solución copiada anula la entrega.
- **Uso de IA**: puedes usar asistentes para explorar y depurar, pero debes poder defender cada decisión de tu entrega sin el asistente. La defensa oral existe precisamente para eso.
- **Citas**: si adaptas código de terceros (OpenZeppelin, ejemplos de documentación), decláralo en el README de la entrega.

La sanción por plagio es la nota mínima en la entrega y, en reincidencia, en el componente completo.

Ruta rápida

> ⬅️ Volver al programa · 📚 Currículo · 🗺️ Hoja de ruta completa

Ruta intensiva para quien ya tiene experiencia sólida de programación (idealmente backend) y puede dedicar **15-20 horas semanales durante 8 semanas**. Es una compresión honesta: cubre los módulos 00-18 sacrificando profundidad de práctica, no temas de seguridad.

Advertencia clara: la ruta recomendada del programa es la completa de 26 semanas descrita en ROADMAP.md. Elige esta ruta solo si de verdad tienes la experiencia previa y las horas; si no, la compresión produce lagunas que se pagan caras en los módulos de seguridad y en el capstone.

Plan de 8 semanas

Semana	Módulos	Foco	Entregable de la semana
1	00-03	Orientación, criptografía, sistemas distribuidos y consenso	Cadena de hashes y árbol de Merkle funcionando; explicar PoW vs PoS por escrito
2	04-05	Bitcoin (UTXO, Script) y Ethereum/EVM	Selección de UTXO resuelta; transacción en regtest; selectores ABI calculados a mano
3	06	Solidity y Foundry a fondo	Vault con pruebas unitarias y de fuzzing en verde
4	07-08	dApps y tokens	dApp mínima (frontend + contrato) y un ERC-20 con pruebas de casos de fallo
5	09-10	Seguridad, oráculos e indexación	Explotar y corregir una reentrancia; consumir un oráculo con protección TWAP
6	11-12	DAO/gobernanza y escalabilidad	Propuesta con timelock ejecutada en local; desplegar el vault en una testnet L2 (Sepolia o L2 de prueba)
7	13-15	Interoperabilidad, privacidad/ZK y arquitectura avanzada	Verificar una prueba de Merkle cross-chain; explicar SNARK vs STARK; simulación de tokenomics
8	16-18 + capstone	Infraestructura, empresa y cierre	Capstone reducido: contrato + pruebas + modelo de amenazas + demo grabada

Hitos de verificación semanales

Al cerrar cada semana, estos comandos deben pasar. Si algo falla, no avances: la deuda se acumula.

Semana	Verificación
1	`pnpm lab:hash` · `pnpm lab:merkle` · `pnpm lab:pow`
2	`pnpm lab:utxo` · `pnpm lab:abi`
3	`forge test` en `labs/06-solidity-vault` (unitarias y fuzz en verde)
4	`forge test` del token; la dApp corre en local de punta a punta
5	Prueba que demuestra el exploit y prueba que demuestra el arreglo, ambas en verde
6	Propuesta de gobernanza ejecutada tras el timelock; despliegue verificado en testnet
7	`pnpm lab:tokenomics` · prueba de Merkle verificada
8	Suite completa del capstone en verde más los criterios de evaluación

Qué puede saltear un dev backend (y qué arriesga)

Módulo	¿Saltear?	Riesgo si lo haces
00 Orientación	Lectura rápida (1-2 h)	Bajo: es contexto y setup
02 Sistemas distribuidos	Parcial si ya operaste sistemas distribuidos reales	Medio: el modelo de fallas bizantinas no es el de tus microservicios
07 dApps	Parcial si dominas frontend web3	Medio: firma de transacciones y manejo de errores RPC tienen trampas propias
16 Infraestructura de nodos	Parcial si eres SRE/DevOps	Medio: los cuellos de botella (IOPS, checkpoint sync) son específicos
17-18 Empresa	Solo si tu meta es exclusivamente DeFi público	Medio: pierdes el contexto regulatorio y de consorcios
01, 05, 06, 09	**Nunca**	Criptografía, EVM, Solidity/pruebas y seguridad son el núcleo; saltearlos invalida la ruta

Si solo tienes X semanas

Tiempo	Subconjunto realista	Qué obtienes	Qué NO obtienes
2 semanas	00-01, 05-06 (hasta el vault con pruebas)	Leer y escribir un contrato simple con pruebas	Ningún criterio de seguridad ni despliegue: no publiques nada con fondos
4 semanas	Semanas 1-5 del plan (00-10)	Base técnica más el mínimo de seguridad y oráculos	Escalabilidad, gobernanza, ZK y capstone
8 semanas	Plan completo de esta página	Panorama completo con práctica comprimida	La profundidad de práctica de la ruta de 26 semanas

Variante sprint de 30 días

La versión anterior de esta ruta era un sprint de 30 días a dedicación casi completa. Sigue siendo viable solo con jornada completa disponible:

- Días 1-4: módulos 00-03 y laboratorios criptográficos.
- Días 5-9: módulos 04-06; EVM, Solidity y Foundry.
- Días 10-13: módulo 07 y una dApp mínima.
- Días 14-18: tokens, seguridad y fuzzing.
- Días 19-22: oráculos, indexación y gobernanza.
- Días 23-26: L2, interoperabilidad y privacidad.
- Días 27-30: capstone, informe de amenazas y demo.

Reglas de la ruta rápida

- No omitas seguridad por acelerar. Si una prueba falla o no puedes explicar un concepto, vuelve al módulo correspondiente.
- Los checkpoints de assessments/checkpoints.md aplican igual que en la ruta completa: nota ≥ 80 por módulo.
- Si al final de la semana 4 vas retrasado más de una semana, cambia a la ruta completa: es la señal de que la compresión no está funcionando para ti.
- Otras rutas por perfil están en learning-paths.

Chile · Regulación, tributación y prevención

> Navegación: Inicio · Currículo · Módulo 17 · Blockchain en la empresa

Revisado: **2026-07-29**.

> **Descargo.** Material **educativo**, no asesoría legal, tributaria, contable ni de inversión. Las obligaciones dependen de la actividad, el instrumento, la residencia y los hechos concretos. Las cifras, criterios y normas cambian: **consúltalo en vivo** con las fuentes oficiales y valida con profesionales antes de decidir. Nunca presentes esta página como interpretación definitiva.

Qué debe aprender el estudiante

- "Criptoactivo", token y servicio **no** reciben automáticamente el mismo tratamiento.
- Construir software no equivale necesariamente a custodiar, intermediar o asesorar.
- La actividad profesional puede activar registro, autorización, deberes de información, normas de conducta y gestión de riesgos.
- Las ganancias por venta de criptomonedas pueden constituir renta afecta a impuestos.
- La privacidad técnica **no** elimina los deberes de prevención de lavado ni la trazabilidad.

Marco regulatorio · CMF y Ley Fintech

La Ley **21.521 (Ley Fintech)** define un conjunto de servicios financieros sujetos a supervisión de la **Comisión para el Mercado Financiero (CMF)**.

Organismo / norma	Rol	Referencia oficial
Ley 21.521 (Fintech)	Define servicios financieros regulados y su registro	Biblioteca del Congreso Nacional
CMF · Implementación Fintech	Publica normativa e instrucciones de implementación	Información de implementación
CMF · Registro de Prestadores	Registro de Prestadores de Servicios Financieros	Registro de prestadores

Actividades definidas por la ley que pueden exigir registro o autorización:

Actividad	Ejemplo
Plataformas de financiamiento colectivo	Crowdfunding de inversión
Intermediación de instrumentos financieros	Poner en contacto oferta y demanda
Asesoría de inversión	Recomendación personalizada
Custodia de instrumentos financieros	Controlar claves o activos de terceros

La clasificación depende de las funciones reales, no del nombre del producto. En 2026 la CMF publicó un criterio indicando que la sola provisión de software **no custodial** para interactuar directamente con blockchain —sin recibir ni controlar claves— no configura por sí sola custodia. Ese criterio **no** debe extrapolarse a modelos distintos sin análisis propio.

Tributación · SII

El **Servicio de Impuestos Internos (SII)** ha señalado que el mayor valor obtenido en la venta de criptomonedas es **renta afecta** a impuestos, con tratamiento distinto según se trate de persona o empresa.

Aspecto	Tratamiento según el SII	Referencia
Mayor valor en venta de cripto	Renta afecta a impuestos	Nota SII 2025
Persona vs. empresa	Distinto tratamiento según el contribuyente	Instrucciones del SII
Proveedores de servicios de criptoactivos	Obligaciones de información establecidas	Normativa del SII

Obligación práctica: conserva historial, costos, comisiones, fechas y equivalencias con trazabilidad suficiente para determinar el mayor valor. Verifica montos, tramos y plazos vigentes directamente con el SII: **consúltalo en vivo**.

Prevención · UAF

La **Unidad de Análisis Financiero (UAF)** previene el uso del sistema financiero y otros sectores para el lavado de activos y el financiamiento del terrorismo. Su guía pública incluye señales relacionadas con activos virtuales.

Deber	Qué implica	Referencia
Determinar si eres sujeto obligado	Depende de la actividad concreta	UAF · Quiénes somos
Debida diligencia	Conocer al cliente cuando corresponda	Guías de la UAF
Reporte de operaciones sospechosas	Cuando la actividad lo exija	Normativa UAF

El cumplimiento no es una casilla que se marca al final: se diseña desde el inicio.

Taller · separar funciones

Para una plataforma hipotética, separa estas funciones y analiza cada una por separado:

1. software no custodial;
2. custodia de claves;
3. intercambio o intermediación;
4. recomendación o asesoría;
5. emisión y comercialización;
6. financiamiento colectivo.

Por cada función registra: datos tratados, control de claves, flujo de dinero, contraparte, organismo competente y las preguntas que **requieren asesoría profesional**.

Regla de mantenimiento

Revisa **semestralmente** las fuentes oficiales, porque los criterios y cifras cambian:

- CMF (normativa Fintech y registro),
- SII (tributación de criptoactivos),
- UAF (prevención de lavado),
- Biblioteca del Congreso Nacional y Diario Oficial (texto vigente de la ley).

Registra los cambios en el historial de Git. Esta página es un punto de partida educativo, **no** una interpretación oficial ni definitiva.

Evaluación y proyecto final

Checkpoints

> Navegación: Inicio · Currículo · Evaluación · Banco de preguntas

Puntos de control por etapa del programa. Cada checkpoint define **qué debes poder hacer**, **qué evidencia lo demuestra** y una **pregunta de autoevaluación** para verificar que comprendes, no solo que ejecutaste. Para aprobar cada checkpoint se requiere el 80% de la rúbrica correspondiente y **ninguna práctica crítica de seguridad omitida**.

Cómo usar esta guía

Avanza de una etapa a la siguiente solo cuando puedas producir la evidencia sin ayuda y responder la pregunta de autoevaluación con un argumento, no con una definición memorizada.

Etapa 00 · Orientación

Qué debes poder hacer	Evidencia	Autoevaluación
Explicar cuándo blockchain aporta valor y cuándo no	Un caso propio con la alternativa descartada	¿Qué problema resuelve mejor una base de datos firmada?
Preparar el entorno de trabajo	Repositorio clonado, herramientas instaladas	¿Puedes reproducir el entorno en otra máquina?

Etapas 01–03 · Fundamentos

Qué debes poder hacer	Evidencia	Autoevaluación
Distinguir hash, firma y Merkle sin confundirlos	Explicación de qué garantiza cada uno	¿Qué información **no** aporta una prueba Merkle?
Demostrar manipulación en la mini cadena	Bloque alterado y verificación que falla	¿Por qué cambiar un bloque invalida los siguientes?
Comparar mecanismos de consenso con criterios claros	Tabla con cuatro criterios (coste Sybil, finalidad, etc.)	¿Cómo afecta una partición a seguridad y vivacidad?

Etapas 04–07 · Desarrollo

Qué debes poder hacer	Evidencia	Autoevaluación
Seguir una transacción de punta a punta	Traza desde firma hasta confirmación	¿Por qué una wallet no "contiene" monedas?
Implementar el Vault y demostrar sus invariantes	Suite verde con fuzzing	¿Por qué CEI no basta sin una guarda de reentrancia?
Conectar una interfaz que simula antes de firmar	dApp que muestra red, valor y efecto	¿Qué error se evita simulando antes de pedir la firma?

Etapas 08–11 · Profesional

Qué debes poder hacer	Evidencia	Autoevaluación
Reproducir y corregir vulnerabilidades	PoC que falla antes del fix y prueba de regresión	¿Distingues causa raíz de síntoma?
Diseñar un oráculo con freshness y fallback	Contrato que rechaza datos obsoletos	¿Cómo se manipula un precio spot con liquidez puntual?
Entregar una política multisig + timelock	Gobernador con quorum y retardo de ejecución	¿Cómo sale un usuario si la gobernanza es capturada?

Etapas 12–15 · Avanzado

Qué debes poder hacer	Evidencia	Autoevaluación
Defender un ADR L1/L2/appchain	Documento con trade-offs y alternativa descartada	¿Qué supuesto de confianza añade tu elección?
Modelar amenazas de un puente	Threat model con actores y superficies	¿Dónde está el punto único de falla del puente?
Razonar sobre privacidad y ZK	Explicación de qué se revela y qué se prueba	¿Qué datos quedan correlacionables pese a la privacidad técnica?

Etapas 16–18 · Operación y empresa

Qué debes poder hacer	Evidencia	Autoevaluación
Planificar infraestructura y operación	Plan de nodos, monitoreo y respaldo	¿Cómo respondes a un incidente sin improvisar?
Presentar tokenomics y caso de negocio	Análisis de valor y costo total	¿Justifica el proyecto su costo frente a una alternativa sin blockchain?
Preparar la defensa del capstone	Demo reproducible y rúbrica respondida	¿Reconoces los límites de tu sistema en lugar de ocultarlos?

Banco de preguntas

> Navegación: Inicio · Currículo · Checkpoints · Evaluación

Banco de preguntas de **razonamiento**, no de memoria: cada una busca que argumentes, no que recites una definición. Están organizadas por etapa del programa. Cada pregunta se marca como de respuesta **objetiva** (O, tiene una respuesta correcta demostrable) o **abierta** (A, admite varias respuestas defendibles según supuestos).

Cómo usarlas

- **Autoevaluación:** responde por escrito con supuesto, argumento, contraejemplo y evidencia. Si no puedes dar un contraejemplo, probablemente no entiendes el límite del concepto.
- **Quiz entre pares:** una persona pregunta y la otra defiende; luego se intercambian. Las preguntas abiertas se evalúan por la calidad del razonamiento, no por coincidir con una clave.
- Una respuesta objetiva mal fundamentada sigue siendo incorrecta: exige la demostración, no solo el resultado.

Fundamentos (etapas 01–03)

1. (A) ¿Qué cambia en la confianza al pasar de una base de datos firmada a una blockchain?
2. (A) ¿Por qué una firma válida no demuestra que el firmante comprendió lo que firmaba?
3. (O) ¿Qué información **no** aporta una prueba Merkle sobre un elemento?
4. (A) ¿Cómo afecta una partición de red a las propiedades de seguridad y vivacidad?
5. (O) ¿Qué costo económico impide crear identidades Sybil en cada mecanismo de consenso?
6. (O) Dado un árbol Merkle de 8 hojas, ¿cuántos hashes necesita una prueba de inclusión?
7. (A) ¿En qué escenario un timestamp de bloque es una fuente de aleatoriedad insegura?
8. (A) Si dos nodos honestos ven cadenas distintas, ¿qué regla decide cuál prevalece y por qué?

Bitcoin y EVM (etapas 04–05)

1. (O) Demuestra la conservación de valor en una transacción UTXO con inputs y outputs concretos.
2. (A) ¿Por qué se dice que una wallet no "contiene" bitcoins?
3. (O) Distingue el nonce de una cuenta Ethereum del nonce de la prueba de trabajo.
4. (A) ¿Cuándo conviene emitir un evento y cuándo escribir en storage?
5. (O) ¿Por qué una llamada `view` invocada desde otro contrato puede consumir gas?
6. (O) ¿Por qué una recompensa coinbase no es gastable hasta 100 confirmaciones?
7. (A) ¿Qué implica para la privacidad reutilizar una misma dirección en Bitcoin?
8. (O) ¿Cuál es la diferencia de coste de gas entre `SSTORE` de un slot nuevo y uno ya usado, y por qué?

Desarrollo y seguridad (etapas 06–09)

1. (A) Escribe tres invariantes de un contrato **antes** de implementarlo.
2. (O) ¿Por qué CEI no resuelve por sí solo toda forma de reentrancia?
3. (O) ¿Cómo se manipula un precio spot usando liquidez temporal en un solo bloque?
4. (O) ¿Qué campos impiden reutilizar una firma en otra cadena o contrato?
5. (A) ¿Qué riesgo añade un patrón proxy aunque uses una librería conocida?
6. (O) Ante `call` que devuelve `false`, ¿qué pasa si no verificas el retorno?
7. (A) ¿Cómo diseñarías una prueba de invariante para "la contabilidad iguala los fondos"?
8. (O) ¿Por qué un `require` después de una transferencia externa puede ser demasiado tarde?
9. (A) ¿Qué distingue una corrección mínima de una reescritura al arreglar una vulnerabilidad?
10. (O) En un ataque de reentrancia clásico, ¿qué línea concreta habilita el drenaje?

Tokens, oráculos y gobernanza (etapas 08, 10, 11)

1. (A) ¿Qué derecho real representa el token de tu proyecto? ¿Existe ese derecho sin el token?
2. (O) ¿Por qué un oráculo debe rechazar datos más viejos que cierto `maxAge`?
3. (A) ¿Cómo mitigas la manipulación de precio: TWAP, múltiples fuentes, circuit breaker? Justifica.
4. (O) ¿Qué garantiza un timelock entre la aprobación y la ejecución de una propuesta?
5. (A) ¿Cómo se captura una gobernanza con voto ponderado y cómo lo defiendes?
6. (A) ¿Cómo sale un usuario si la gobernanza queda capturada?
7. (O) ¿Por qué `mint` sin un `cap` verificable rompe la escasez declarada del token?
8. (A) ¿Cuándo un proyecto **no** necesita un token propio?

Avanzado (etapas 12–15)

1. (A) ¿Qué supuesto de confianza añade un rollup optimista frente a la L1?
2. (A) ¿Dónde está el punto único de falla de un puente y cómo lo mitigas?
3. (O) ¿Qué distingue una prueba de validez de una prueba de fraude en cuanto a finalidad?
4. (A) ¿Qué datos quedan correlacionables pese a usar una técnica de privacidad?
5. (A) ¿Cuándo justifica una appchain su coste frente a desplegar en una L2 existente?
6. (A) ¿Qué comprometes al elegir disponibilidad de datos fuera de la L1?

Producto y operación (etapas 16–18)

1. (A) ¿Quién puede censurar, pausar, actualizar o retirar fondos en tu sistema?
2. (A) ¿Cuál es el costo total de operar el sistema frente a una alternativa sin blockchain?
3. (A) ¿Qué datos personales quedarían almacenados o correlacionables, y cómo lo evitas?
4. (A) ¿Cómo respondes a un incidente (clave comprometida, oráculo caído) sin improvisar?
5. (A) ¿Qué nodos e infraestructura exige tu proyecto para estar disponible y monitoreado?
6. (A) ¿Por qué el proyecto justifica su costo de mantenimiento en el tiempo?
7. (A) ¿Qué obligaciones regulatorias podrían aplicar según la actividad real del proyecto?
8. (A) Si tuvieras más tiempo, ¿qué harías distinto y por qué?

Plantilla · Informe de auditoría

Navegación: Inicio · Currículo · Módulo 09 · Seguridad · Retos de seguridad

Plantilla profesional para redactar el informe de auditoría de tu proyecto o capstone. Copia este archivo, reemplaza los marcadores `<…>` y elimina las notas en cursiva. Un buen informe **reduce el riesgo dentro de un alcance declarado**; no promete ausencia total de vulnerabilidades.

Portada

Campo	Valor
Proyecto	`<nombre del protocolo>`
Alcance	`<contratos y archivos revisados>`
Fuera de alcance	`<lo explícitamente no revisado>`
Commit auditado	`<hash de git>`
Red / compilador	`<testnet, solc 0.8.x>`
Auditor(es)	`<nombre>`
Fechas	`<inicio – fin>`
Versión del informe	`<1.0>`

Resumen ejecutivo

Tres a cinco frases legibles por alguien no técnico: qué se revisó, el resultado general, cuántos hallazgos por severidad y el mensaje principal.

Severidad	Cantidad	Corregidos	Aceptados
Crítica	0	0	0
Alta	0	0	0
Media	0	0	0
Baja	0	0	0
Informativa	0	0	0

Metodología

Describe cómo se realizó la revisión.

- Revisión manual línea por línea de `<archivos>`.
- Análisis estático con `<Slither / otra>`.
- Pruebas dirigidas y **fuzzing/invariantes** con Foundry sobre las propiedades críticas.
- Revisión de supuestos de confianza, privilegios administrativos y modos de falla.

Arquitectura y confianza

Activos, actores, privilegios, dependencias externas (oráculos, tokens) y un diagrama de componentes con límites de confianza.

Invariantes verificadas

#	Invariante	Método de verificación	Estado
1	`<propiedad crítica>`	`<prueba de invariante / fuzz>`	`<verificada / no cubierta>`

Hallazgos

Tabla resumen; luego una ficha por hallazgo.

ID	Título	Severidad	Estado
H-01	`<título>`	`<Alta>`	`<Corregido>`

Formato de un hallazgo

ID: `H-01` · **Severidad:** `<Alta>` · **Estado:** `<Abierto / Corregido / Aceptado>`

- **Ubicación:** `<archivo:línea>`
- **Descripción:** *qué es y por qué es un problema.*
- **Impacto:** *qué puede lograr un atacante (fondos, control, DoS).*
- **Probabilidad:** *qué tan fácil es explotarlo y bajo qué condiciones.*
- **Prueba de concepto:** *prueba mínima que demuestra el impacto (referencia al test; no incluyas exploits listos para atacar sistemas de terceros).*
- **Recomendación:** *corrección mínima que ataca la causa raíz.*
- **Verificación:** *cómo se confirmó que la corrección cierra la falla (prueba de regresión).*

Matriz de severidad

La severidad combina **impacto** (qué se pierde) y **probabilidad** (qué tan fácil ocurre):

Impacto ↓ / Probabilidad →	**Baja**	**Media**	**Alta**
Alto (pérdida de fondos, control total)	Media	Alta	Crítica
Medio (bloqueo temporal, pérdida parcial)	Baja	Media	Alta
Bajo (molestia, gas, cosmético)	Informativa	Baja	Media

Riesgos no técnicos

Gobernanza, dependencia de oráculos, concentración de poder, operación y respuesta ante incidentes: aspectos que ninguna corrección de código resuelve por sí sola.

Anexos

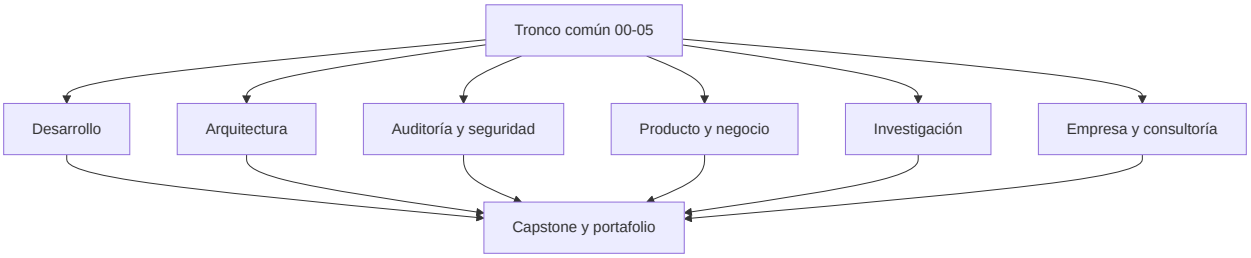
- **A. Herramientas y versiones** usadas (compilador, análisis estático, Foundry).
- **B. Cobertura de pruebas** y salida relevante de la suite.
- **C. Registro de correcciones**: commits que atienden cada hallazgo.

Descargo

La revisión reduce el riesgo dentro del alcance declarado y en el commit auditado; **no garantiza la ausencia de vulnerabilidades**. Cambios posteriores al commit no están cubiertos. Este informe es material educativo del programa y no sustituye una auditoría profesional independiente.

Rutas por perfil profesional

Todas las rutas parten del tronco común 00–05 (orientación, criptografía, sistemas distribuidos, consenso, Bitcoin y EVM). A partir de ahí, cada perfil prioriza módulos, laboratorios y un entregable de portafolio distinto. Elige la ruta que mejor describa el rol al que apuntas; puedes cambiar de ruta sin perder avance, porque el registro de progreso es el mismo.



Resumen de las seis rutas

Perfil	Módulos prioritarios	Laboratorios clave	Entregable de portafolio	Salida laboral típica
Desarrollo	06–10, 12	21–30, 31–35	dApp integral probada con Foundry	Smart contract / full-stack Web3 developer
Arquitectura	02–05, 10–16	01–10, 41–46	ADR de plataforma con trade-offs	Arquitecto de soluciones blockchain
Auditoría y seguridad	05–09, 11, 15	31–40, 41–45	Informe de seguridad estilo auditoría	Auditor de contratos / security researcher
Producto y negocio	00, 04, 07–08, 11–15, 17–18	01–05, 11–15	Validación de caso y tokenomics	Product manager / analista Web3
Investigación	01–03, 12–15	01–10, 41–48	Réplica comentada de un paper	Investigador / protocol engineer junior
Empresa y consultoría	00, 02, 10–13, 15–18	01–10, 46–50	Diseño de red permisionada con ADR	Consultor / líder técnico enterprise

Desarrollo

- **A quién le sirve:** programadores que quieren construir contratos y aplicaciones descentralizadas de nivel profesional.

- **Secuencia recomendada:** 00–05 completos → énfasis fuerte en 06-solidity-foundry, 07-dapps, 08-tokens, 09-seguridad y 10-oraculos-indexacion → 12-escalabilidad para desplegar en L2.
- **Puede aligerar:** 14 (ZK) y 17–18 (empresa) a lectura de síntesis.
- **Laboratorios clave:** prácticas 21–30 (EVM y desarrollo) al completo y 31–35 del bloque profesional; consulta el catálogo.
- **Entregable de portafolio:** una dApp integral — contratos con pruebas unitarias, fuzzing e invariantes, interfaz y despliegue reproducible en testnet.
- **Salida laboral:** smart contract developer, full-stack Web3 developer, integrador de protocolos DeFi.

Arquitectura

- **A quién le sirve:** ingenieros con experiencia que decidirán qué cadena, capa y modelo de confianza usa un sistema.
- **Secuencia recomendada:** profundiza 02–03 (distribuidos y consenso) y 05; después 10-oraculos-indexacion, 12-escalabilidad, 13-interoperabilidad, 15-arquitectura-avanzada y 16-infraestructura-nodos.
- **Puede aligerar:** 08 (tokens) y hacer 06–07 solo hasta poder leer contratos.
- **Laboratorios clave:** prácticas 01–10 (modelado de sistemas) y 41–46 (rollups, puentes, MEV).
- **Entregable de portafolio:** un ADR de plataforma que compare al menos dos alternativas (L1 vs. L2, pública vs. permisionada) con trade-offs medibles.
- **Salida laboral:** arquitecto de soluciones blockchain, staff engineer de infraestructura Web3.

Auditoría y seguridad

- **A quién le sirve:** quienes quieren encontrar y explicar vulnerabilidades antes de que cuesten dinero.
- **Secuencia recomendada:** 05 y 06 a fondo (no se audita lo que no se sabe escribir), énfasis máximo en 09-seguridad; complementa con 07–08 (superficies de ataque de dApps y tokens), 11-dao-gobernanza (ataques de gobernanza) y 15.
- **Puede aligerar:** 13 y 17–18.
- **Laboratorios clave:** prácticas 31–40 (seguridad profesional) completas y 41–45 (front-running, storage collision y afines).
- **Entregable de portafolio:** un informe de seguridad con formato de auditoría real — hallazgos clasificados por severidad, prueba de concepto y recomendación; usa la plantilla de `assessments/audit-report-template.md` y los checkpoints.
- **Salida laboral:** auditor de smart contracts, security researcher, participante de concursos de auditoría (Code4rena, Sherlock).

Producto y negocio

- **A quién le sirve:** perfiles de producto, negocio o emprendimiento que deben decidir si blockchain aporta valor y cómo se monetiza.
- **Secuencia recomendada:** 00 y 04 con calma; 07–08 para entender qué se puede construir; 11-dao-gobernanza a 15-arquitectura-avanzada en modo conceptual; y con énfasis fuerte 17-blockchain-en-la-empresa y 18-implementacion-empresarial, junto con toda la sección de industria (modelos de negocio, equipos y ciclo de vida).
- **Puede aligerar:** 06 y 09 a nivel de vocabulario.
- **Laboratorios clave:** prácticas 01–05 (fundamentos con criterio) y 11–15 (economía de Bitcoin y consenso).
- **Entregable de portafolio:** una validación de caso de uso con ADR "¿por qué blockchain?" y un análisis de tokenomics (derechos, emisión, demanda y gobernanza).
- **Salida laboral:** product manager Web3, analista de negocio blockchain, fundador o responsable de innovación.

Investigación

- **A quién le sirve:** perfiles académicos o muy técnicos interesados en los fundamentos y en las fronteras del área.
- **Secuencia recomendada:** máxima profundidad en 01–03 (criptografía, distribuidos, consenso); después 12-escalabilidad, 13-interoperabilidad, 14-privacidad-zk y 15, apoyándose en la bibliografía.
- **Puede aligerar:** 07–08 y 17–18.
- **Laboratorios clave:** prácticas 01–10 (implementar primitivas desde cero) y 41–48 (rollups, ZK, MEV).
- **Entregable de portafolio:** la réplica comentada de un paper relevante (por ejemplo, un mecanismo de consenso o una construcción ZK) con implementación mínima y análisis crítico.
- **Salida laboral:** investigador en criptografía aplicada, protocol engineer junior, estudiante de posgrado con base sólida.

Empresa y consultoría

- **A quién le sirve:** consultores e ingenieros que integran blockchain en organizaciones existentes, con requisitos de cumplimiento y privacidad.
- **Secuencia recomendada:** 00 y 02 para el modelo mental; 10–13 para datos, escalado e interoperabilidad; 15–16 para arquitectura e infraestructura de nodos; y como núcleo 17-blockchain-en-la-empresa y 18-implementacion-empresarial, complementados con la sección de industria completa (cómo se construye una blockchain, stack, roles y ciclo de vida de un proyecto).
- **Puede aligerar:** 06 (escritura de contratos) y 14.

- **Laboratorios clave:** prácticas 01–10 y 46–50 (arquitectura avanzada y cierre de capstone).
- **Entregable de portafolio:** el diseño de una red permisionada o híbrida con ADR, modelo de gobernanza, plan de operación y análisis de cumplimiento estilo módulo 18.
- **Salida laboral:** consultor blockchain, líder técnico de proyectos enterprise, arquitecto de integraciones.

Nivelación

- Sin programación previa: completa ejercicios de terminal y JavaScript básico antes del módulo 05.
- Con experiencia general: realiza `assessments/diagnostic.json`; si alcanzas 80 %, usa la ruta rápida descrita en la documentación del programa.
- Con experiencia EVM: comienza por seguridad, pero entrega igualmente el ADR "¿por qué blockchain?".

Navegación

- Inicio del programa
- Currículo completo (19 módulos)
- Catálogo de laboratorios
- Evaluación · Checkpoints
- Sección de industria · Roadmap

Proyecto final (capstone)

Objetivo y qué demuestra

Construye un protocolo pequeño que resuelva un problema real y permita demostrar dominio integral del programa: diseño con criterio, contratos probados con rigor, una interfaz utilizable, datos accesibles y una defensa técnica honesta. El capstone no busca originalidad absoluta, sino evidencia de que sabes tomar decisiones, justificarlas y verificar que el sistema hace lo que afirmas.

Un capstone aprobado demuestra que puedes: elegir blockchain solo cuando corresponde, escribir contratos que resisten fuzzing e invariantes, razonar sobre amenazas antes de que ocurran y comunicar todo eso a un evaluador escéptico.

Requisitos mínimos

1. **Protocolo con contratos probados con Foundry**, incluyendo pruebas unitarias, de integración, fuzzing e invariantes sobre las propiedades críticas del sistema. Contratos documentados con NatSpec.
2. **dApp o interfaz** accesible que permita ejercitar los flujos principales (no hace falta diseño pulido; sí estados de transacción claros y manejo de errores).
3. **Indexador o estrategia de datos** explícita: subgraph, indexador propio, eventos + consultas RPC o equivalente, con justificación de la elección.
4. **Documento de arquitectura** estilo módulo 18-implementacion-empresarial: contexto, ADR "¿por qué blockchain y qué alternativa se descartó?", diagrama de componentes y límites de confianza.
5. **Threat model**: actores, superficies de ataque, privilegios administrativos declarados, plan de incidentes. Puedes apoyarte en `docs/threat-model-project.md`.
6. **Despliegue reproducible** local y en testnet mediante scripts (quien clona el repositorio debe poder levantar todo con instrucciones de un solo documento).
7. **Estimación de gas y operación**: costos aproximados de las funciones principales y qué implica operar el sistema.
8. **Plan de infraestructura** estilo módulo 16-infraestructura-nodos: nodos requeridos, disponibilidad, monitoreo y respaldo, y cómo se opera el sistema en el tiempo.
9. **Caso de negocio** estilo módulo 17-blockchain-en-la-empresa: problema, usuarios, propuesta de valor y viabilidad; por qué el proyecto justifica su costo y mantenimiento.

Ideas de proyecto con alcance acotado

Además del protocolo de financiamiento comunitario que sirve de hilo conductor del repositorio, estas tres ideas tienen un alcance realista para un capstone individual:

1. Registro de certificados verificables

- Una institución emite credenciales ancladas on-chain (hash + Merkle root), con revocación y verificación pública.
- **Incluye:** emisión por lotes, revocación, verificación sin wallet desde la interfaz.
- **Queda fuera:** identidad soberana completa, estándares de credenciales verificables W3C, integración con sistemas reales.
- **Ejercita:** Merkle proofs, control de acceso, diseño de datos off-chain/on-chain.

2. Mercado de escrow con árbitro opcional

- Compraventa entre pares con depósito en garantía, disputa con timelock y árbitro designado.
- **Incluye:** máquina de estados completa del trato, disputa, expiración y reembolso.
- **Queda fuera:** reputación, catálogo de productos, pagos con múltiples tokens.
- **Ejercita:** máquinas de estados, invariantes de conservación de fondos, análisis de incentivos.

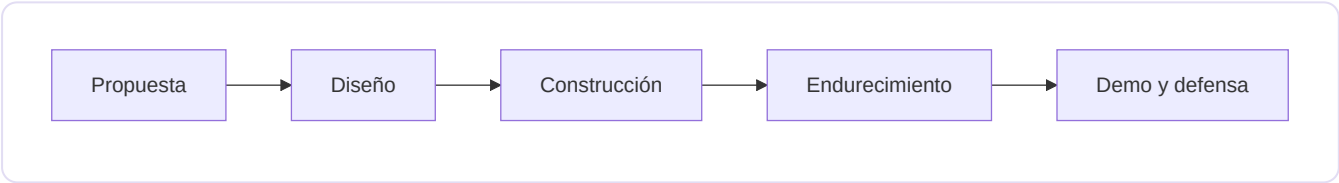
3. Tesorería con gobernanza mínima

- Bóveda multifirma con propuestas, votación ponderada y timelock de ejecución.
- **Incluye:** ciclo propuesta → votación → cola → ejecución, con quórum y cancelación.
- **Queda fuera:** token de gobernanza propio, delegación líquida, gobernanza cross-chain.
- **Ejercita:** patrones del módulo 11-dao-gobernanza y ataques de gobernanza en el threat model.

Cualquier otra idea es válida si cabe en las fases siguientes y el instructor aprueba la propuesta. Regla práctica de alcance: si no puedes enumerar las invariantes críticas en cinco líneas, el proyecto es demasiado grande.

Fases y entregables

Fase	Entregable	Criterio de salida
1. Propuesta	Problema, usuarios, criterios de éxito y ADR inicial	El instructor confirma que el alcance es realista
2. Diseño	Documento de arquitectura, threat model v1, esquema de datos	Límites de confianza y privilegios declarados
3. Construcción	Contratos + pruebas, interfaz, indexación, scripts de despliegue	Suite verde con fuzzing e invariantes
4. Endurecimiento	Análisis estático, revisión del threat model, informe de auditoría propio	Hallazgos corregidos o justificados por escrito
5. Demo y defensa	Demo funcional en testnet + defensa de 10 minutos	Responde la rúbrica sin depender de la suerte



Rúbrica de defensa técnica

Qué se pregunta	Qué se espera
¿Por qué blockchain y no una base de datos?	ADR con alternativa concreta descartada y trade-offs honestos
¿Cuáles son las invariantes críticas y cómo las pruebas?	Invariantes formuladas como propiedades y verificadas con Foundry
¿Qué puede hacer el administrador y qué pasa si su clave se compromete?	Privilegios enumerados, mitigaciones (multisig, timelock) y plan de incidentes
¿Cómo falla el sistema? (oráculo caído, reorg, front-running)	Modos de falla identificados en el threat model con respuesta definida
¿Cuánto cuesta usarlo y operarlo?	Estimación de gas por función y análisis de operación
Muestra el peor bug que encontraste y cómo lo detectaste	Evidencia de proceso: prueba que falló, causa raíz, corrección

Preparación de la demo y defensa

- Ensaya la demo de punta a punta en un entorno limpio; una demo que solo funciona en tu máquina no cuenta.
- Prepara datos de ejemplo que muestren también un camino de error (transacción revertida, disputa, revocación).

- Ten a mano la salida de la suite de pruebas y del análisis estático: te las van a pedir.
- La defensa dura 10 minutos: dos de contexto, cinco de demo, tres de preguntas de la rúbrica.

Criterios de excelencia

- Invariantes no triviales encontradas por fuzzing antes que por lectura.
- Threat model que condicionó decisiones de diseño (y lo documenta).
- Interfaz que comunica estados intermedios de transacción y errores con lenguaje humano.
- Despliegue reproducible en un comando por entorno.
- Defensa que reconoce límites del sistema en lugar de ocultarlos.

Qué NO hace falta

- **No mainnet**: local y testnet bastan; usar fondos reales es causa de reprobación.
- **No auditoría pagada**: el informe de auditoría lo elaboras tú con las técnicas del módulo 09-seguridad.
- **No token propio**: solo incluye un token si el problema lo exige y el análisis de tokenomics lo respalda.
- No frontend de producción ni marca comercial: la evaluación es técnica.

Puertas de calidad

No se aprueba si usa fondos reales, expone secretos, oculta privilegios administrativos o carece de pruebas para invariantes críticas.

Cómo presentarlo en el portafolio

- README propio del proyecto con problema, demo (capturas o video corto), decisiones clave y cómo reproducirlo.
- Enlaza el documento de arquitectura, el threat model y el informe de auditoría: son lo que diferencia tu repositorio de un tutorial.
- Describe qué harías distinto con más tiempo; la autocrítica fundada es señal de seniority.

Navegación

- Inicio del programa · Currículo
- Catálogo de laboratorios · Evaluación
- Checkpoints