

Seminario Socrático #15 — temas para preparar

miércoles 28 de octubre de 2026 · resumen de los temas del temario · BitDevs Córdoba

Este resumen es para llegar al seminario sabiendo de qué se va a hablar. Cada tema abre en lenguaje natural —para cualquiera— y sigue con el detalle técnico en el recuadro. Cada uno lleva su fuente primaria, por si querés ir al original.

Explicaciones preparadas por Maxwell Labib (agente automatizado) y revisadas antes de publicar.

1. BIP54, limpieza de consenso: el soft fork que sigue en pie

BIP54 Consensus Cleanup: the soft fork that is still standing — [fuente](#)

BIP54, Consensus Cleanup, es un soft fork que junta varios arreglos a las reglas de consenso de Bitcoin. Cubre el ataque de timewarp, los bloques armados para tardar minutos u horas en validarse, una debilidad del árbol de Merkle con transacciones de 64 bytes y el chequeo de transacciones duplicadas que hoy hace BIP-30. Lo escribieron Antoine Poinot y Matt Corallo, y en el repositorio de BIPs figura como Complete.

Por qué ahora: hay un PR abierto en Bitcoin Core que lo implementa solo para regtest y deja la activación en mainnet para otra discusión. El 7 de septiembre entró a Core el lado del minero, una de las reglas de timewarp aplicada a las plantillas de bloque, con milestone 32.0. En Bitcoin Inquisition está activo desde el 12 de febrero de 2026. En septiembre, Pieter Wuille mostró en Delving que las dos reglas de timewarp alcanzan para acotar cuántos bloques puede tener la cadena.

Por qué a Córdoba: es el primer tema del temario propuesto para el seminario #15. El BIP no trae mecanismo ni fecha de activación, y la pregunta queda para la sala: cómo y cuándo se activa un soft fork que ya está escrito.

Timewarp. En el primer bloque de cada período de ajuste (altura N con $N \% 2016 = 0$), el timestamp tiene que ser al menos el del bloque anterior menos 7200 segundos. En el último ($N \% 2016 = 2015$), tiene que ser al menos el del bloque $N-2015$. Según el BIP, esa segunda regla cierra una falla por la que un período podía durar un tiempo negativo, y el PR #35949 de Core la llama regla Murch-Zawy. Sin estos arreglos, el BIP calcula que un atacante con mayoría de hashrate llega a la dificultad mínima en 38 días. Wuille mostró que con las dos reglas la cantidad de bloques queda acotada: $n \leq t/596,4285 + 121746,97$, con t en segundos desde el génesis y trabajo total de hasta 2^{128} . A largo plazo eso da un bloque cada 9m56s. Las fórmulas están probadas en Lean 4.

Sigops del script viejo. En cada transacción que no sea coinbase se cuentan las operaciones de firma del scriptSig de cada input, del scriptPubKey que gasta y del redeemScript P2SH. CHECKSIG y CHECKSIGVERIFY valen 1. CHECKMULTISIG precedido de OP_1 a OP_16 vale ese número, y si no, 20. Si el total pasa de 2500, la transacción es inválida.

Coinbase. El nLockTime tiene que ser la altura del bloque menos 1, y el nSequence no puede ser 0xffffffff. Así cada coinbase es única y BIP-30 deja de hacer falta. Según el BIP, ese chequeo volvería a ser necesario desde el bloque 1.983.702 y estorba a diseños como Utreexo.

64 bytes. Es inválida toda transacción que mida exactamente 64 bytes sin witness. Una así se puede leer como un nodo interno del árbol de Merkle y engañar a un verificador SPV.

Para discutir: el BIP no tiene sección de activación, y el PR #35793 dice «Mainnet activation, if any, is to be considered separately».

2. Escalar la red Lightning con reconciliación de conjuntos aplicada

Scaling the Lightning Network with Practical Set Reconciliation — [fuente](#)

Los nodos Lightning conocen la topología de la red (los canales, sus actualizaciones y los nodos) por gossip. Un paper publicado el 16 de agosto de 2026 midió ese gossip, que funciona por inundación, y concluye que es fundamentalmente ineficiente. Los autores proponen sincronizar esa información con reconciliación de conjuntos: dos nodos calculan qué le falta a cada uno y se mandan solo la diferencia, en vez de mandar todo. Es la misma familia de técnicas que Erelay propone para anunciar transacciones en la capa base. En una simulación sobre Core Lightning con capturas reales de gossip, la sincronización baja de horas a unos pocos minutos. Para BitDevs Córdoba, lo interesante es la discusión que abre. La especificación de Lightning, BOLT 7, ya tiene mecanismos para no pedir de más: consultas por rango de bloques con checksums y un filtro por fecha. El paper dice que igual el resultado es ineficiente. ¿Falla el diseño o falla cómo se usa? Y si reconciliar rinde tanto, ¿por qué Erelay sigue en estado de borrador en la capa base?

El problema. Según el paper, el gossip de Lightning reenvía por inundación los anuncios de canal, las actualizaciones de canal y los anuncios de nodo. Sus mediciones lo muestran fundamentalmente ineficiente. La propuesta. Compartir esa información con protocolos de reconciliación de conjuntos. Los autores evalúan los enfoques existentes en condiciones de red realistas y proponen ADAPTIVEIBLT, un protocolo adaptativo basado en IBLT (Invertible Bloom Lookup Table) con una mejora de decodificación parcial. La evaluación. Simulan la reconciliación en Core Lightning con snapshots reales de gossip. La sincronización pasa de horas a pocos minutos. El contraste con Erelay. BIP 330, en estado Draft, especifica la reconciliación de anuncios de transacciones entre dos nodos. Usa sketches PinSketch, basados en BCH, que implementa la librería minisketch. Erelay combina inundación, donde hace falta velocidad, con reconciliación para el resto. El paper usa IBLT, que es otra construcción. Lo que ya tiene la spec. BOLT 7 define cuatro mensajes. `query_channel_range` consulta canales en un rango de bloques. `reply_channel_range` puede traer timestamps y checksums CRC32C de cada `channel_update`, para que el par no pida actualizaciones redundantes. `query_short_channel_ids` pide los anuncios de canales puntuales. `gossip_timestamp_filter` restringe el gossip futuro a un rango. Además, un nodo no debe enviar gossip que no generó él mismo hasta recibir `gossip_timestamp_filter`. Para la sala. Si la spec ya permite evitar lo redundante, ¿de dónde sale la ineficiencia medida? ¿Por qué IBLT y no PinSketch, que ya está implementado en la capa base? ¿Qué tendría que cambiar en BOLT 7 para adoptarlo?

3. Shielded Bitcoin: transferencias privadas en la capa base de Bitcoin

Shielded Bitcoin: Private Transfers on the Bitcoin L1 — [fuente](#)

En Bitcoin el historial de transacciones es público. Shielded Bitcoin es una propuesta de Clara Shikhelman, Mikhail Komarov y Aleksei Moskvín para mover bitcoin sin que se vean los montos ni quién recibe, sobre la capa base tal como está hoy. El hilo que la presenta en Delving Bitcoin, del 24 de septiembre, abre con «No soft forks. No operators. No bridges.». Es un metaprotocolo: no tiene cadena propia ni reglas de consenso nuevas.

La propuesta dice que no necesita un cambio de consenso ni un operador de confianza. Eso tiene un precio: las transferencias ocupan más que una transacción común, hay que confiar en una ceremonia de configuración, y la entrada y la salida se ven en la cadena. Los depósitos y retiros se van a detallar en un segundo paper, que todavía no salió.

Para BitDevs Córdoba es un buen tema socrático porque la discusión sigue abierta. En el mismo hilo, ZmnSCPxj, AdamISZ y una de las autoras debaten cómo se sale del sistema. La pregunta para la sala: qué privacidad se puede conseguir sin tocar el consenso, y qué se paga a cambio.

El valor vive en notas cifradas. Cada una guarda un monto y la forma de llegar a su dueño. Para pagarle a Bob, Alice cifra una nota para él y publica tres cosas como datos de una transacción de Bitcoin: las notas nuevas, un nullifier por cada nota que gasta y una prueba de conocimiento cero. La prueba acredita que las notas existen, que Alice puede gastarlas y que los montos cuadran, sin decir cuáles son ni cuánto valen.

Las reglas no las aplica el consenso. Las aplican indexers que leen la cadena, verifican las pruebas y controlan que ningún nullifier se repita, y cualquiera puede repetir esa verificación. Un indexer deshonesto puede demorar, omitir o servir datos viejos, pero no puede gastar. La wallet de Bob prueba abrir cada nota nueva con su clave de visualización: si abre, es suya.

Quedan ocultos el monto, qué notas se gastaron y quién recibe. En la cadena se ve que hubo una transferencia, cuándo, cuántas notas tiene y qué comisión pagó.

Límites que declara el propio post: con pocos usuarios, o con wallets de patrones propios, se pueden inferir relaciones. La entrada y la salida son visibles. Y la prueba de referencia es Groth16, cuya ceremonia exige al menos un participante honesto. Para entrar y salir se usan PIPEs, basados en witness encryption.

Para discutir: ZmnSCPxj y AdamISZ preguntan cómo se prueba un intento de salida sin una prueba tipo SPV con desafío. ZmnSCPxj agrega que, sin filtros tipo BIP-157, recuperar una wallet obliga a bajar todos los bloques o a confiar en un indexer. Y dorugoqu advierte que Bitcoin no rechaza las transferencias mal formadas, así que un cambio de circuito puede hacer que wallets viejas y nuevas acepten notas distintas.

4. BIP138: esquema compacto de cifrado para los datos de una wallet que no son la semilla

BIP138: Compact Encryption Scheme for Non-seed Wallet Data — [fuente](#)

BIP138 propone un formato estándar para cifrar el respaldo de lo que describe una wallet: los descriptors de salida (BIP380), las wallet policies (BIP388) y metadatos como etiquetas. Lo cifrado no puede llevar claves privadas.

Según el BIP, perder el descriptor es casi tan grave como perder la semilla: en un multisig la semilla firma, pero sin el descriptor no se sabe dónde están las monedas. Un descriptor es demasiado largo para grabarlo en papel o en acero. Guardarlo en claro en la nube expone las claves públicas y la estructura de la wallet.

Con BIP138, cualquiera que tenga una de las claves del descriptor puede descifrar el respaldo sin otro secreto. Por eso se lo puede dejar en un medio que no es de confianza.

El texto entró al repositorio de BIPs el 21/09/2026, con un PR abierto desde septiembre de 2025, y sigue en borrador, versión 0.1.3. Liana usa una versión temprana, incompatible con la final.

Para la sala: dónde y con quién se guarda una copia que no da acceso a los fondos pero dice cómo está armada la wallet, y qué pasa si un cosignatario ya le dio su xpub a un servidor.

Secreto. Se toman las claves públicas raíz x-only de las expresiones elegibles, ordenadas lexicográficamente: $s = \text{tagged_hash}(\text{"BIP138_DECRYPTION_SECRET"}, p1|...|pn)$. Por participante, $si = \text{tagged_hash}(\text{"BIP138_INDIVIDUAL_SECRET"}, pi)$, y el archivo guarda $ci = s \text{ XOR } si$. Quien tiene una pi calcula su si , la combina con cada ci y descifra solo si su clave participaba.

Elegibilidad. Fuera de `musig()` solo cuentan las `xpub` con derivación posterior (`xpub/0/*`, `xpub/*`, `multipath`). Una `xpub` pelada o una clave literal quedaría a la vista en la cadena al gastar, y cualquiera recalcularía el secreto. Se excluye la clave NUMS de BIP341. Con el conjunto vacío, el codificador debe negarse.

Para discutir: una `xpub` que se mandó a un servidor la conoce un tercero. El BIP recomienda armar el multisig con cuentas cuya `xpub` nunca salió, como las de BIP48 o BIP87.

Cifrado. ChaCha20-Poly1305 (RFC 8439), el de BIP324: nonce aleatorio de 96 bits (se rechaza el todo ceros), tag de 128 bits, AAD vacío.

Formato. MAGIC "BIP138" de 6 bytes, versión 0x01, rutas de derivación opcionales, los ci , el tipo de cifrado y el payload. Se recomiendan ci señuelo para ocultar cuántos participantes hay y relleno dentro del cifrado para ocultar el tamaño. No lleva fingerprints, para no delatar participantes. El payload va en tipo-longitud-contenido: los tipos desconocidos debajo de 0x80 se saltean y desde 0x80 se rechazan, lo que deja lugar para extenderlo. Para texto, Base64, como las PSBT.

Implementaciones: una en Rust del autor y una prueba de concepto de Sjors sobre Bitcoin Core.

5. BIP461: firmas ECDSA deterministas

BIP461: Deterministic ECDSA Signatures — [fuente](#)

BIP461 es un BIP de especificación, en borrador, de la capa de aplicaciones. No toca el consenso: define una forma de firmar con ECDSA en la que la firma depende solo de la clave privada y del mensaje. Se mergeó en el repositorio de BIPs el 16 de septiembre de 2026.

Por qué importa: en ECDSA el que firma elige un número secreto por firma, el nonce, y esa libertad se puede abusar para filtrar la clave. Con un algoritmo determinista y escrito como estándar, esa libertad desaparece: se firma el mismo mensaje con dos firmadores independientes y se comparan las firmas. Si difieren, uno de los dos no cumple. De paso, las firmas quedan en 70 bytes como máximo, contra 72 de una firma ECDSA cualquiera.

En BitDevs Córdoba está en el temario propuesto para el seminario #15, junto con BIP138. La pregunta para la sala: qué se gana con poner por escrito, como estándar, una práctica de firma, y qué pasa con el firmador que no la adopta, cuyas firmas con r alto se ven en la cadena.

El nonce se deriva al estilo de RFC 6979: un generador con HMAC-SHA256 que toma la clave secreta, el hash del mensaje y datos extra, vacíos en el primer intento. Con el nonce k se calcula $R = k \cdot G$, $r = x(R) \bmod n$ y $s = k^{-1}(d \cdot r + m) \bmod n$.

Después viene el low- R grinding: si r no es bajo, se repite con un contador i en los datos extra, codificado en 4 bytes little-endian seguidos de 28 bytes en cero, hasta que $r < 2^{255}$, o sea que el byte alto de r sea menor que 0x80. El contador llega hasta $2^{32} - 1$; si no aparece un r bajo, la firma falla. Al final se normaliza s : si supera $(n - 1)/2$, se reemplaza por $n - s$. La firma se autoverifica antes de devolverse.

Por qué r y s bajos: en DER y con el byte de sighash, una firma con r o s altos ocupa bytes de más. Con los dos bajos queda en 70 bytes DER, 71 con el sighash. El BIP estima un ahorro de unos 0,5 vbytes por input legacy, 0,125 por input SegWit v0 y 2,5 por input P2SH 2-de-3, a cambio de una operación de firma extra y una resta. Las firmas resultantes son ECDSA válidas comunes y verifican con las reglas de consenso actuales.

RFC 6979 se eligió porque admite datos extra y ya es el default en libsecp256k1. Se descartaron nonces aleatorios en cada intento (no son reproducibles, y eso rompe la comparación entre firmadores), buscar un r todavía más chico (unos 512 intentos en promedio) y el atajo del endomorfismo λ (beneficio despreciable).

Para discutir: el BIP pide adopción uniforme porque un firmador que no hace grinding se distingue por sus firmas con r alto. Y está en borrador: al mergearse faltaban la implementación de referencia y los vectores de prueba, que aborda el PR #2328.