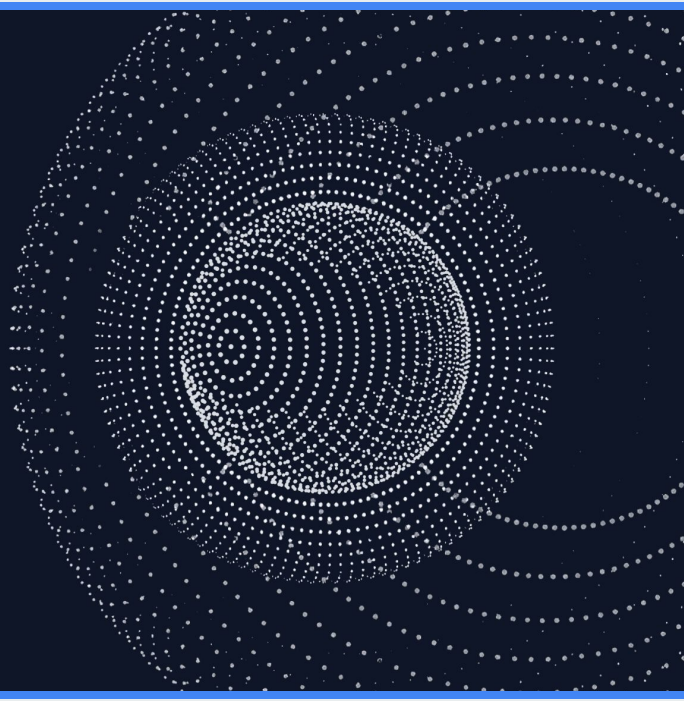
Fairgate

Implementing a RISC-V VCPU on Bitcoin



Martin Jonas | BitVMX Principal Engineer

SEPT 2025

Preguntas claves sobre la CPU de BitVMX

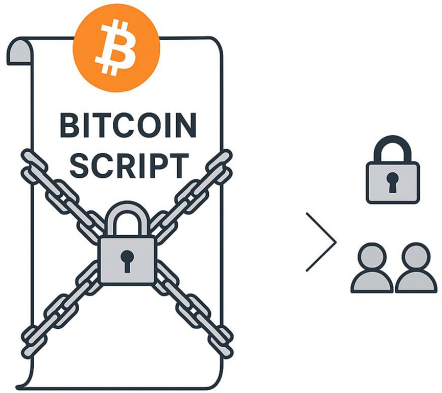
¿Para qué necesito un programa?

Bitcoin permite transferencias entre usuarios

BitVMX permite definir un programa para controlar el destino de los fondos

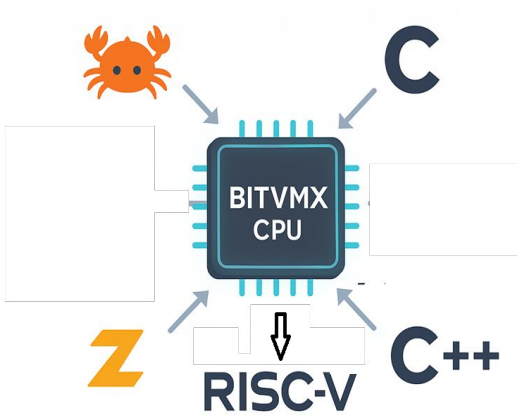
¿Por qué no escribirlo en Bitcoin Script?

- Se limitó por razones de seguridad
- Permite realizar algunas cosas simples:
  - HashLocks
  - Threshold signatures
- Permite hacer algunas cosas más complejas, pero muy caras (mul 5k)



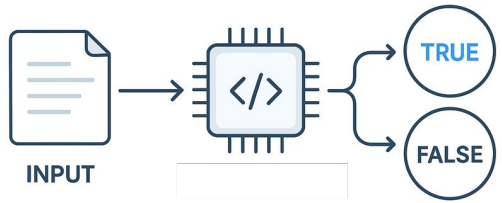
¿Cómo se escribe un programa?

- La CPU de **BitVMX** usa una arquitectura **RISC-V**, que es:
  - Una arquitectura **abierta** (como si fuera AMD o Intel)
  - Modular, permite construir un set inicial de instrucciones y luego extenderlo (ejemplo: primero enteros, luego multiplicación “M”).
  - Compatible con varios lenguajes: Rust, C, C++, Zig, entre otros.
- Esto facilita el desarrollo de programas y la capacidad de debuggear.



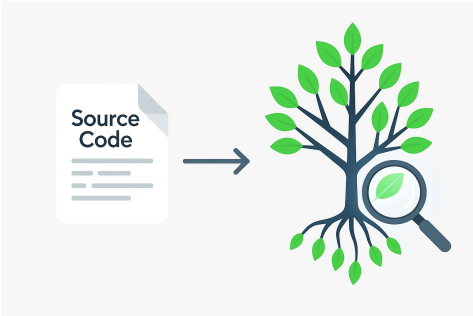
¿Qué tipo de programas podes escribir?

- Programas que reciban un **input** y que devuelvan **true/false**
- Determinísticos, siempre producen el mismo resultado
- Sin acceso al file system ni a otro componentes externos (algunos podrían simularse)
- Sin efectos colaterales: no interactúan con el mundo externo mientras corren



¿Cómo acordamos qué programa ejecutar?

- Partimos del mismo **source code**, que cada parte puede auditar.
- Cada uno compila por su cuenta (con la misma **toolchain**).
- Luego creamos un **grafo de transacciones** firmado por ambos, donde queda plasmado qué programa estamos ejecutando.
- ¿Dónde se guarda esto?
    - En un **Taproot/Taptree**: tipo de transacción en Bitcoin que permite hacer una Merkle proof (probar que una hoja pertenece a un árbol sin mostrar todo el árbol).
    - **Las hojas contienen instrucciones RISC-V (escritas en Bitcoin Script)**

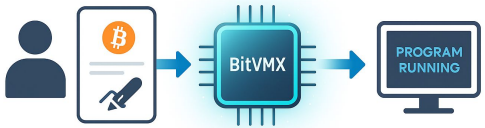


¿Ejecución off-chain y verificación on-chain?

- El programa completo se ejecuta en nuestras máquinas, **no en Bitcoin**.
- ¿Por qué?
  - Porque un programa puede tener **millones de instrucciones,** lo cual sería imposible llevarlo a Bitcoin
- Flujo:
  - Se ejecuta off-chain (cada parte por su lado).
  - Solo si **no hay acuerdo**, se resuelve **on-chain**.
  - Partimos de un estado inicial, cada instrucción modifica el estado de la CPU en un paso.
  - Se busca el **primer paso en que no estamos de acuerdo**.
  - Eso es lo que se challengea en Bitcoin: **una sola instrucción**.
  - Esto reduce el costo de ejecutar **500 millones** de instrucciones a **1 instrucción, + una búsqueda binaria** para ubicar dónde divergen las ejecuciones.

¿Cómo se le pasa un input a un programa?

- Una parte pone el input en una transacción de Bitcoin y la firma.
- La otra parte observa la transacción, toma el input y lo ejecuta off-chain en la CPU.
- Si la discrepancia está en la memoria donde está el input, la contraparte puede mostrar que **se firmaron dos cosas distintas** y así ganar.
- Bitcoin Script no permite leer datos de transacciones anteriores, entonces el input se codifica usando **OTS** (One-Time Signatures).
- Esto permite referenciar valores de otras transacciones, aunque tiene un **costo alto**.

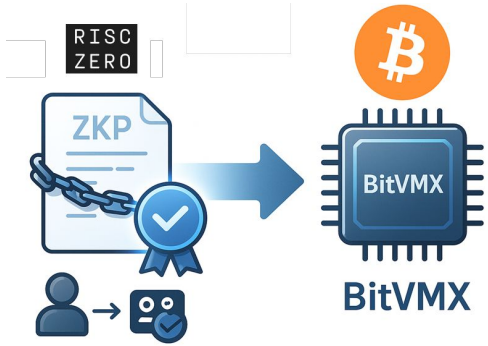


¿Qué pasa si el input es muy grande?

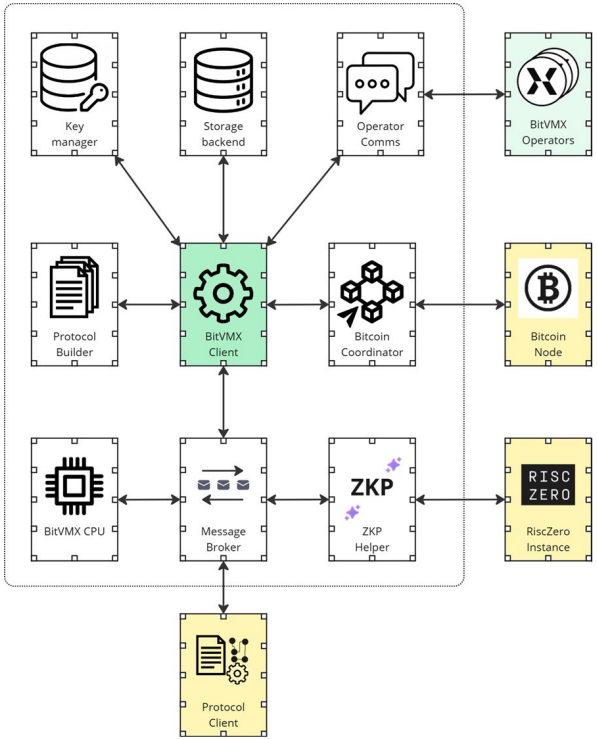
- **¿Cómo lo solucionamos? Usando Zero Knowledge Proofs (ZKP).**
- Las ZKP permiten que una parte pruebe a la otra que tiene el input correcto **sin revelar el input**
- Sirve para:
  - Verificar que alguien tiene información sin revelar (preimagen de un hash)
  - Probar algo costoso de demostrar, pero barato de verificar (una tx no existe en la blockchain)
- Reduce los inputs a 300 bytes.

¿Cómo lo aplicamos?

- Usamos **RiscZero**, una tecnología ZKP que también usa RISC-V (aunque no tiene relación directa con el RISC-V de BitVMX).
- Se puede escribir en **Rust** un programa verificador
- Luego escribimos otro programa, ahora sí para nuestra CPU, que verifica que la prueba anterior es correcta



Arquitectura de BitVMX



¿Cómo construimos usando BitVMX ?

- Definir un programa verificador
  - Escribir el programa que represente la lógica a validar.
  - Dependiendo del tamaño: para la CPU o usando ZKP
- Construir protocolos que conectan con la disputa
  - Como se manejan múltiples disputas entre varias partes
  - Lógica específica de la aplicación
- Usar la API de BitVMX y conectarlo con tu aplicación
  - Integrar el verificador y el protocolo en la aplicación real
  - BitVMX provee una API para interactuar con la CPU, manejar inputs y coordinar la disputa, generar pruebas

Gracias!

https://github.com/FairgateLabs

BitVMX

https://bitvmx.org