Universidad del Valle de Guatemala

Facultad de Ingeniería

Departamento de Computación

Algoritmos y estructuras de datos

Pablo Godoy

Fase 1: Intérprete de Bitcoin Scripts

Juan Pablo Flores -25454

Esteban Sanchez -25213

Javier Sánchez -25341

Índice:

Introducción

Este proyecto consiste en diseñar e implementar un intérprete de Bitcoin Script, un lenguaje stack-based utilizado para validar transacciones mediante la ejecución secuencial de instrucciones sobre una pila. El sistema evalúa la concatenación de los scripts de desbloqueo y bloqueo (scriptSig + scriptPubKey) y determina la validez del programa cuando la ejecución finaliza sin errores y deja un valor verdadero en la cima de la pila. Con un enfoque académico, el proyecto prioriza la correcta selección y justificación de estructuras del Java Collections Framework (JCF), considerando su complejidad en tiempo y espacio, así como el diseño del sistema, el uso de pruebas unitarias y la aplicación de buenas prácticas de ingeniería de software.

Objetivos

Objetivo General

Diseñar e implementar un prototipo de una versión simplificada de Bitcoin Script, basado en una pila y utilizando estructuras del Java Collections Framework, aplicando principios de diseño y buenas prácticas de ingeniería de software.

Objetivos Específicos

1. Entender cómo funciona Bitcoin Script, especialmente su modelo basado en pila y la forma en que se validan transacciones al ejecutar scriptSig junto con scriptPubKey.

2. Diseñar la estructura del sistema usando diagramas UML (clases y secuencia), mostrando cómo interactúan componentes como Parser, Interpreter, Token y Stack.

3. Desarrollar un prototipo funcional que ejecute instrucciones básicas y permita probar un caso tipo P2PKH con criptografía simulada.

Bitcoin Script

Fundamentos de Bitcoin Script

Bitcoin Script es un lenguaje de *scripting* utilizado como mecanismo de bloqueo para las salidas de las transacciones dentro del protocolo de Bitcoin. En este esquema, cada salida incluye un script de bloqueo (scriptPubKey) que debe ser satisfecho mediante un script de desbloqueo (scriptSig o *witness*) cuando dicha salida se utiliza como entrada en una transacción posterior. Si la ejecución completa del script es válida, la salida se desbloquea y puede ser gastada. Este mecanismo permite la creación de distintos tipos de condiciones de bloqueo mediante combinaciones de *OPcodes*, como rompecabezas matemáticos, rompecabezas hash y esquemas basados en colisiones hash (Walker, 2025).

Entre sus características principales, Bitcoin Script es un lenguaje stack-based que prioriza la seguridad y la previsibilidad sobre la complejidad computacional, diferenciándose de los lenguajes de propósito general. Su naturaleza no Turing completa elimina la posibilidad de bucles infinitos y reduce el riesgo de ataques de denegación de servicio. Esto con el fin de garantizar que cada ejecución finalice de manera acotada. A su vez, la ejecución determinista de los scripts asegura que todos los nodos honestos lleguen a la misma conclusión respecto a la validez de una transacción. En consecuencia, Bitcoin Script está orientado exclusivamente a la validación y no al cómputo general, lo que refuerza su papel como un mecanismo seguro dentro del sistema de transacciones de Bitcoin (Manya, 2025).

Modelo de ejecución y representación de datos

Un Script en Bitcoin Script se evalúan de derecha a izquierda, de instrucción en instrucción. Cada una de las instrucciones interactúa con una pila principal que sigue una política LIFO (Last in, First out). Cada instrucción en este lenguaje se ejecuta consecutivamente, una después de la otra. Cuando una instrucción requiere datos, los extrae de la cima de la pila, y cuando produce un resultado lo pone nuevamente en ella. Aquí no existen variables ni registros. Todo el estado del programa se está conteniendo en una pila (Segura, 2019). Los literales (operaciones empuje que colocan datos en la pila) y llas operaciones lógicas aritméticas o de verificación (operaciones que consumen uno o más elementos desde la cima para producir nuevos resultados) son algunas de las instrucciones que interactuan con la pil principal. Si una instrucción intenta operar sobre una cantidad insuficiente de elementos, se produce un error de ejecución que invalida el script (Walker, 2025).

Bitcoin Script maneja los datos como arreglos de bytes que se pueden interpretar como valores booleanos o enteros según la operación. En este caso, 0 representa falso y cualquier otro número verdadero. La validez de un script se determina evaluando el valor que permanece en la cima de la pila al concluir la ejecución. El script es válido únicamente si ninguna instrucción falla y el resultado final es verdadero. Este modelo de evaluación asegura que el proceso de validación sea predecible, restringido y consistente (Rhodes, 2025).

Instrucciones, control de flujo y manejo de errores

OP_CODES son el conjunto de instrucciones que compone a Bitcoin Script. Representan operaciones atómicas que manipulan la pila o controlas la lógica de validación. Las instrucciones se pueden agrupar en categorías como empuje de datos, operaciones sobre la pila, lógica y comparación, aritmética básica, control de flujo y verificación criptográfica. Cada opcode define de forma estricta cuantos elementos consume de la pila y los resultados que produce. La simplicidad de los opcodes está diseñada para reducir la complejidad del lenguaje y evitar los efectos colaterales que pueda comprometer la seguridad (Maldonado, 2020).

El control de flujo en Bitcoin Script se implementa mediante instrucciones condicionales que permiten ejecutar bloques de código cuando se cumplen ciertas condiciones evaluadas a partir de la misma pila. Además, las instrucciones como OP_VERIFY y OP_RETURN, introducen mecanismos de validación y finalización del script. Cualquier error durante la ejecución provoca la invalidación del script. Este manejo de errores refuerza la seguridad del lenguaje, asegurando que únicamente los scripts con las condiciones establecidas sean válidos (Antonopoulos, 2014).

Operaciones criptográficas y simulación en un entorno didáctico

Bitcoin Script tiene operaciones criptográficas que permiten verificar la autenticidad y la autorización de una transacción, principalmente mediante funciones de *hash* y validación de firmas digitales. Instrucciones como OP_SHA256, OP_HASH160 y OP_CHECKSIG desempeñan un rol importante al garantizar que solo el propietario legítimo pueda desbloquear la salida. Estas operaciones fortalecen la seguridad del sistema al vincular de forma directa la validez de la transacción con pruebas criptográficas verificables por todos los nodos de la red (Segura, 2019).

Las operaciones criptográficas se pueden simular con funciones *mock*, permitiendo concentrar el desarrollo en la correcta evaluación del script, el manejo de la pila y el diseño del intérprete. Las funciones mock permiten probar los vínculos entre código, deshaciendo la implementación real de una función. Todas las funciones simuladas mock tienen una propiedad *.mock* donde se guarda la información sobre cómo se ha llamado a esa función. La propiedad guarda también el valor de *this* correspondiente a cada llamada (JEST Documentation, 2017).

Flujo de validación de transacciones

ScriptSig y scriptPubKey

En el modelo de transacciones de Bitcoin, los fondos no se gestionan mediante cuentas, sino a través de UTXOs (Unspent Transaction Outputs), los cuales representan cantidades específicas de bitcoin disponibles para ser gastadas. Para proteger estos UTXOs, Bitcoin utiliza un sistema de scripts que define las condiciones bajo las cuales una salida puede ser utilizada. El ScriptPubKey, incluido en la salida de una transacción, actúa como un mecanismo de bloqueo al establecer dichas condiciones, mientras que el ScriptSig, incluido en la entrada de una transacción posterior, proporciona los datos necesarios como firmas digitales y claves públicas para satisfacerlas. Esta separación entre condiciones y evidencias criptográficas refuerza la seguridad y verificabilidad del sistema de transacciones (Morel, 2024).

Durante la validación, los nodos verificadores combinan el ScriptSig con el ScriptPubKey y los ejecutan como un único programa siguiendo el modelo basado en pila de Bitcoin Script. En este proceso, el ScriptSig se ejecuta primero para preparar la pila con los datos requeridos, seguido del ScriptPubKey, que evalúa si se cumplen las condiciones de gasto definidas originalmente. Esta ejecución se realiza de manera secuencial y local por cada nodo, eliminando la necesidad de confiar en el emisor de la transacción y garantizando un comportamiento determinista compartido por toda la red (Morel, 2024; Walker, 2025).

El esquema Pay-to-Public-Key-Hash (P2PKH) representa el caso de uso más común de Bitcoin Script y ejemplifica este flujo de validación. En este modelo, el ScriptSig aporta la firma digital y la clave pública del gastador, mientras que el ScriptPubKey verifica que la clave pública corresponda al hash esperado y que la firma sea válida, autorizando el gasto del UTXO únicamente al propietario legítimo de los fondos.

Proceso de validación y criterios de éxito

El criterio de éxito en la validación de una transacción en Bitcoin Script es estricto. Una transacción se considera válida únicamente si la ejecución completa del script concatenado finaliza sin errores, lo que implica que todas las verificaciones se hayan satisfecho correctamente y que no ocurra ninguna condición inválida durante la evaluación. Al concluir la ejecución, la cima de la pila debe contener un valor interpretado como verdadero; de lo contrario, el gasto del UTXO es rechazado de forma inmediata.

Este enfoque garantiza que todos los nodos honestos lleguen a la misma conclusión respecto a la validez de una transacción, evitando ambigüedades en la interpretación de las reglas de gasto. Al basarse exclusivamente en la ejecución determinista de scripts y en verificaciones criptográficas locales, Bitcoin Script asegura un proceso de validación coherente, predecible y resistente a manipulaciones, lo que constituye uno de los pilares de la seguridad del sistema de transacciones de Bitcoin (Walker, 2025).

Selección de estructuras de datos (JCF)

Fundamentos

Una colección es un objeto que agrupa múltiples elementos en una sola unidad, permitiendo almacenarlos y manipularlos de manera estructurada. El Java Collections Framework (JCF) es una arquitectura unificada incluida en el paquete java.util que estandariza la forma en que estas colecciones se representan y gestionan en Java (Oracle, n.d.). Su propósito principal es proporcionar un modelo común para trabajar con grupos de objetos sin depender de los detalles específicos de implementación, favoreciendo así un diseño modular y reutilizable.

El JCF se compone de tres elementos fundamentales: interfaces, implementaciones y algoritmos. Las interfaces definen los tipos abstractos de colecciones y las operaciones disponibles; las implementaciones proporcionan estructuras de datos concretas reutilizables; y los algoritmos permiten realizar operaciones como búsqueda u ordenamiento sobre cualquier colección compatible (Oracle, n.d.). Esta separación entre definición y ejecución permite intercambiar implementaciones sin modificar la lógica general del programa, promoviendo flexibilidad y mantenibilidad.

Estructuras principales y su funcionamiento

Dentro del JCF, las colecciones se organizan en torno a interfaces que modelan distintos comportamientos de almacenamiento. La interfaz List representa colecciones ordenadas que permiten elementos duplicados; Set modela colecciones de elementos únicos; y Map almacena pares clave-valor con claves únicas. Asimismo, existen interfaces como Queue y Deque, diseñadas para gestionar elementos según un orden específico de procesamiento (Oracle, n.d.). Queue sigue

generalmente el principio FIFO (First-In, First-Out), mientras que Deque permite funcionar tanto como cola como pila.

El comportamiento tipo pila (stack) sigue el principio LIFO (Last-In, First-Out), donde el último elemento insertado en la pila es el primero en salir. En el JCF moderno, este comportamiento puede implementarse de forma eficiente mediante distintas estructuras en lugar de depender únicamente de la clase tradicional Stack. Las distintas implementaciones, como ArrayList, LinkedList, HashMap o ArrayDeque, presentan variaciones en acceso, inserción y eliminación, permitiendo seleccionar la estructura más adecuada según los requerimientos del sistema (Oracle, n.d.).

Justificación de JCF implementado

En este proyecto se utilizó el JCF para gestionar las estructuras dinámicas necesarias durante el procesamiento y ejecución del programa. Dado que el sistema funciona con un modelo basado en pila y ejecución secuencial, era importante contar con estructuras que permitieran agregar y eliminar elementos de forma eficiente, lo cual se logró mediante las implementaciones proporcionadas por el JCF.

La estructura central del proyecto es la pila utilizada durante la ejecución del programa. Para modelarla se implementó la clase PilaArrayList<T>, que utiliza internamente un ArrayList<T> para almacenar los elementos. Esta elección permite implementar el comportamiento LIFO requerido por Bitcoin Script. Las operaciones push, pop y peek se realizan sobre el final de la lista lo que permite agregar, eliminar y consultar elementos en tiempo constante O(1). En el caso de push es O(1) amortizado que significa que aunque en ocasiones el arreglo interno de ArrayList deba ampliarse y copiar sus elementos, en promedio el costo por operación sigue siendo constante.

Debido a que todas las operaciones de la pila se realizan en el extremo final de la lista, ArrayList resulta una opción eficiente tanto en tiempo como en uso de memoria. Además, se definió una interfaz propia Collection<T> que establece las operaciones básicas de la pila, permitiendo usarla sin preocuparse por cómo está construida por dentro. Esto hace que el código esté mejor organizado y permite realizar cambios futuros sin modificar la parte principal del programa.

Conclusión

En esta primera fase del proyecto se logró desarrollar un prototipo funcional básico de Bitcoin Script, permitiendo comprender de forma práctica cómo se validan transacciones mediante un modelo basado en pila. A través de la ejecución secuencial de instrucciones, se evidenció cómo el estado de la pila determina si un script es válido o no.

Se investigaron los fundamentos del lenguaje, se modeló el sistema mediante diagramas UML y se seleccionaron estructuras del Java Collections Framework adecuadas para implementar la pila de manera eficiente. Además, se implementaron opcodes básicos y se simuló la verificación criptográfica para probar un caso tipo P2PKH.

La base desarrollada en esta primera fase facilita la ampliación del proyecto en la siguiente etapa donde se incorporarán más instrucciones, manejo completo de errores y mayor cobertura de pruebas.

Referencias

Antonopoulos, A.M. (2014). *Dominando Bitcoin.* O'Reilly Media

Maldonado, J. (2020). *What is an OP_CODE?* https://academy.bit2me.com/en/what-is-an-op-code/#:~:text=Types%20of%20opcodes%20in%20the,of%20them%20are%20mentioned%20here.

Manya. (2025). *Advantages of Bitcoin's Scripting Language.* https://www.nadcab.com/blog/bitcoin-scripting-language

Morel, L. (2024). *Scripts, conditions, signatures…¿How are bitcoins secured?* *https://www.bitstack-app.com/en/learn-bitcoin/how-are-bitcoins-secured?c=EUR*

Oracle. (n.d.). *Collections framework overview*. https://docs.oracle.com/en/java/javase/24/docs/api/java.base/java/util/doc-files/coll-overview.html#:~:text=The%20Java%20platform%20includes%20a,with%20which%20to%20manipulate%20them.

Oracle. (n.d.). *Lesson: Introduction to collections*. https://docs.oracle.com/javase/tutorial/collections/intro/index.html#:~:text=Benefits%20of%20the%20Java%20Collections,pass%20collections%20back%20and%20forth.

Rhodes, D. (2025). *What is Bitcoin Script? Unveiling its Role in Bitcoin.* https://komodoplatform.com/en/academy/bitcoin-script/#:~:text=la%20red%20Bitcoin.-,Notaci%C3%B3n%20polaca%20inversa%20(RPN),una%20ruta%20clara%20y%20predecible.

Segura, J. (2019). *¿Qué es Bitcoin Script?*. *https://academy.bit2me.com/que-es-bitcoin-script/*

Walker, G. (2025). *Script: A mini programming language.* https://learnmeabitcoin.com/technical/script/