Universidade de Brasília – UnB
Faculdade UnB Gama – FGA
Engenharia de Software

Aleatoriedade em Blockchain: Oráculo em rede Solana com testes de aleatoriedade

Autor: Arthur Paiva Tavares
Orientador: Prof. Dr. Tiago Alves da Fonseca

Brasília, DF
2023

Arthur Paiva Tavares

Aleatoriedade em Blockchain: Oráculo em rede Solana com testes de aleatoriedade

Monografia submetida ao curso de graduação em Engenharia de Software da Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em Engenharia de Software.

Universidade de Brasília – UnB

Faculdade UnB Gama – FGA

Orientador: Prof. Dr. Tiago Alves da Fonseca

Brasília, DF

2023

Arthur Paiva Tavares

Aleatoriedade em Blockchain: Oráculo em rede Solana com testes de aleatoriedade/ Arthur Paiva Tavares. – Brasília, DF, 2023-

48 p. : il. (algumas color.) ; 30 cm.

Orientador: Prof. Dr. Tiago Alves da Fonseca

Trabalho de Conclusão de Curso – Universidade de Brasília – UnB
Faculdade UnB Gama – FGA , 2023.

1. Blockchain. 2. Oráculo. I. Prof. Dr. Tiago Alves da Fonseca. II. Universidade de Brasília. III. Faculdade UnB Gama. IV. Aleatoriedade em Blockchain: Oráculo em rede Solana com testes de aleatoriedade

CDU 02:141:005.6

Arthur Paiva Tavares

Aleatoriedade em Blockchain: Oráculo em rede Solana com testes de aleatoriedade

Monografia submetida ao curso de graduação em Engenharia de Software da Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em Engenharia de Software.

Trabalho aprovado. Brasília, DF, 14 de julho de 2023:

Prof. Dr. Tiago Alves da Fonseca
Orientador

Prof. Matheus Bernardini de Souza
Convidado 1

Dr. José Antônio Carrijo Barbosa
Convidado 2

Brasília, DF
2023

Resumo

Tecnologias e plataformas baseadas em redes blockchain possuem aplicações amplas e poderosas graças à imutabilidade e distribuição de base de dados. Por outro lado, existem limitações trazidas pelo próprio conceito construtivo da tecnologia blockchain, uma delas é que deve haver determinismo em toda transação executada pelos pares da rede. Com o determinismo, torna-se inviável a geração de valores aleatórios em contratos inteligentes. A proposta deste trabalho é a criação de um oráculo aleatório para redes blockchain na plataforma Solana, junto com testes estatísticos para validação dos resultados e, dessa forma, para avaliação da confiabilidade do oráculo. O oráculo consiste de um programa a ser executado pelos nós de uma plataforma Solana junto com um servidor responsável pela geração dos valores pseudoaleatórios e envio de transações com os resultados para a rede, com o benefício do histórico de transações do livro-razão da blockchain.

Palavras-chave: vrf. solana. oráculo. blockchain.

Abstract

Technologies and platforms based on blockchain networks have wide and powerful applications thanks to the immutability and distribution of databases. On the other hand, there are limitations imposed by the blockchain technology concepts, one of them is that every transaction performed by the network peers must be deterministic. With the determinism, the generation of random values in smart contracts becomes unfeasible. This work aims to build a random oracle for blockchain networks in Solana platform and evaluate the generated random sequences using statistical tests to assess the operation and reliability of the oracle. The oracle consists of a program to be executed by the nodes of a Solana platform along with a server responsible for generating pseudorandom values and sending transactions conveying the results to the network, with the benefit of the history of blockchain ledger transactions.

Key-words: vrf. solana. oracle. blockchain.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

CPI	Cross Program Invocation
eDSL	Embedded Domain Specific Language
IDL	Interface Description Language
NFT	Non-Fungible Token
PDA	Program Derived Address

Sumário

1 Introdução

1.1 Justificativa

Desde a criação e surgimento do Bitcoin em 2008 (NAKAMOTO, 2008), diversas tecnologias baseadas em blokchain surgiram e novas ferramentas foram criadas. O interesse de corporações nos casos de uso que estavam surgindo começaram já em 2014, ao longo dos anos grande parte das maiores corporações abraçou a ideia de que a tecnologia blockchain é uma inovação séria com capacidade de contribuir com seus modelos de negócios ou operações.

Em setembro de 2021, 65 das 100 maiores empresas de capital aberto já estavam ativamente desenvolvendo soluções com blockchain e outras 16 estavam em fase de estudos, explorando as oportunidades e decidindo quais tecnologias melhor se aplicariam para as iniciativas em blockchain de cada uma (SCHWEIGER, 2021).

Por ser uma tecnologia nova, as diversas plataformas e ferramentas baseadas em blockchain ainda podem ser consideradas imaturas. A plataforma de blockchain Solana foi proposta em novembro de 2017 no documento técnico publicado por Anatoly Yakovenko (YAKOVENKO, 2018). A falta de programadores especializados disponíveis e a necessidade de mudança de comportamento que a nova tecnologia traz cria uma barreira que dificulta a evolução da àrea e das soluções criadas. (BAMBYSHEVA, 2022)

Algumas plataformas blockchain fazem uso de contratos inteligentes na camada de execução: usuários podem implementar contratos inteligentes que farão algum processamento com os dados recebidos afim de chegar a um resultado alterando dados armazenados na rede blockchain ou executando novas instruções nativas da plataforma ou de outros contratos inteligentes. Em alguns casos, as soluções exigem uma fonte de aleatoriedade, no entanto não existe de forma nativa o acesso a um gerador de números aleatórios dentro da plataforma devido à natureza determinística do conceito de redes blockchain.

Sistemas com este tipo de exigência que são executados dentro das plataformas blockchain dependem do comportamento aleatório das sequências geradas comprovadamente equiprovável e igualmente incerta para todos os participantes, para que não seja possível prever o resultado de uma transação que use o contrato inteligente. (FOUNDATION, 2022)

1.2 Objetivos

Desenvolver um oráculo aleatório para a plataforma blockchain Solana e fazer testes estatísticos para validação dos resultados do gerador.

Os objetivos específicos do trabalho são:

- Elicitar requisitos técnicos de um oráculo para a plataforma Solana;
- Desenvolver o contrato inteligente que armazenará os resultados dos valores aleatórios;
- Desenvolver o servidor que será responsável por gerar os valores aleatórios e enviar transações para a rede;
- Implementar testes estatísticos para validação dos resultados a partir das sementes no formato escolhido.

2 Fundamentação Teórica

2.1 Conceitos Blockchain

A tecnologia Blockchain é um avanço recente da computação segura sem autoridade centralizada em um sistema de rede aberta. Do ponto de vista do gerenciamento de dados, um blockchain é um banco de dados distribuído que registra uma lista em evolução de registros de transações, organizando-os em uma cadeia hierárquica de blocos. De uma perspectiva de segurança, a cadeia de blocos é criada e mantida usando uma rede de sobreposição ponto a ponto e protegida por meio da utilização inteligente e descentralizada de criptografia com computação coletiva.

O blockchain garante que, uma vez que um registro de transação é adicionado a um bloco e o bloco foi criado e confirmado com sucesso no blockchain, o registro de transação não pode ser alterado ou comprometido retrospectivamente. A integridade do conteúdo de dados em cada bloco da cadeia é garantida e os blocos, uma vez inseridos na blockchain, não podem ser adulterados sem perda da integridade e consistência do histórico de transações da rede. Assim, um blockchain serve como um livro-razão seguro e distribuído que arquiva todas as transações entre quaisquer duas partes de um sistema de rede aberto de forma eficaz, persistente e verificável.

Para permitir que o blockchain funcione em escala global com garantia de segurança e correção, o livro público compartilhado depende de um algoritmo de consenso eficiente e seguro, que deve ser tolerante a faltas e garantir que todos os nós mantenham simultaneamente uma cadeia idêntica de blocos e que não depende de uma autoridade central para impedir que adversários mal-intencionados interrompam o processo de coordenação para chegar a um consenso. Em suma, toda mensagem transmitida entre os nós deve ser aprovada pela maioria dos participantes da rede por meio de um acordo baseado em consenso (ZHANG; XUE; LIU, 2019).

2.2 A plataforma Solana

Solana é uma plataforma blockchain fundada em 2017 por Yakovenko (2018) e teve sua rede principal lançada em março de 2020. A proposta trouxe uma aplicação do algoritmo de consenso Proof of History que é uma sequência de computação que pode fornecer uma maneira de verificar criptograficamente a passagem de tempo entre dois eventos. O algoritmo usa uma função criptograficamente segura construída de forma que a saída não possa ser prevista a partir da entrada e deve ser completamente executada

para gerar a saída.

Quando comparado a outras plataformas que utilizam outros algoritmos de consenso, a Solana se destaca por permitir muitas transações por segundo com baixo consumo energético por transação Watts (2023). Yakovenko (2018) mostra que é possível chegar à taxa de 710 mil transações por segundo com computadores atuais.

Na Solana, os contratos inteligentes são chamados de programas, que são códigos executáveis que interpretam as instruções enviadas dentro de cada transação na blockchain. Eles podem ser implantados diretamente no núcleo da rede como programas nativos ou publicados por qualquer pessoa como programas On Chain. Os programas são os principais blocos de construção da rede e lidam com tudo, desde o envio de ativos digitais entre carteiras até sistemas de votação e o rastreamento da propriedade de ativos digitais, que podem representar uma variedade de coisas, como moedas criptográficas, ações, bens digitais, etc.

São permitidos que, em tempo de execução, os programas chamem uns aos outros por meio de um mecanismo denominado invocação entre programas, do inglês, cross program invocation ou CPIs. A chamada entre programas é realizada por um programa invocando uma instrução do outro. O programa chamador é interrompido até que o programa chamado termine de processar a instrução.

2.3 Data Feeds e Oráculos

As plataformas Blockchain modernas permitem a criação de contratos inteligentes para que as partes que se comunicam estabeleçam acordos com base em regras predefinidas e sem a necessidade de um terceiro confiável. Várias aplicações possíveis para contratos inteligentes foram exploradas, incluindo assistência médica, comércio, transporte, IoT, gestão de direitos digitais e serviços governamentais. Existem casos de uso em que os contratos inteligentes precisam adquirir dados sobre o estado e eventos do mundo real, de fora do sistema blockchain, que não podem ser alcançados por contratos inteligentes porque o ambiente blockchain é isolado do mundo externo. Para superar essa limitação com contratos inteligentes, houve a necessidade de alimentação de dados para trazer dados externos para dentro do blockchain. As ferramentas usadas para essa alimentação de dados externos são chamados de oráculos (AL-BREIKI et al., 2020).

Além de dados reais do mundo externo, também existem contratos inteligentes e ferramentas baseadas em blockchain que precisam de valores aleatórios a serem usados em nível de execução. Como citado por (FOUNDATION, 2022), alguns casos de uso para valores aleatórios são:

- Definição de atributos de NFTs;

- Distribuição de ativos raros;
- Resultados imprevisíveis de jogos;
- Seleção aleatória de ganhadores.

2.4 Aleatoriedade

Uma sequência de bits aleatórios pode ser interpretada como o resultado dos lançamentos de uma moeda com lados que são rotulados como 0 e 1, com cada lançamento tendo uma probabilidade de exatamente 1/2 de resultar em “0” e 1/2 de resultar “1”. Além disso, os lançamentos são independentes entre si: o resultado de qualquer lançamento anterior da moeda não afeta futuras jogadas de moeda. A moeda não enviesada é, portanto, o gerador de fluxo de bits aleatório perfeito, pois o “0” e o “1” estarão distribuídos uniformemente. Todos os elementos da sequência são gerados independentemente um do outro, e o valor do próximo elemento na sequência não pode ser previsto, independentemente de quantos elementos já foram obtidos.

2.4.1 Números Pseudoaleatórios

Um gerador de números pseudoaleatório (PRNG do inglês Pseudorandom Number Generator) usa uma ou mais entradas e pode gerar de forma determinística uma sequência de números uniformemente distribuídos que, em função do processo de geração, podem ser chamados de "pseudoaleatórios”. As entradas para os PRNGs são chamadas de sementes. Em contextos em que a imprevisibilidade é necessária, a semente deve ser aleatória. Portanto, por padrão, um PRNG deve obter suas sementes das saídas de um gerador de números verdadeiramente aleatórios (TRNG do inglês True Random Number Generator); ou seja, um PRNG requer, também, um TRNG como companheiro para quaisquer aplicações que exigem tal nível de segurança.

2.4.2 Imprevisibilidade

Números aleatórios e pseudoaleatórios gerados para aplicativos criptográficos devem ser equiprováveis e imprevisíveis. No caso de geração mediante processos pseudoaleatórios, também não deve ser viável determinar a semente geradora a partir do conhecimento de quaisquer sequências geradas. No caso do uso de endereços públicos e privados da Solana como semente, deve ser inviável prever os endereços usados a partir dos resultados obtidos.

2.5 Testes estatísticos

Vários testes estatísticos podem ser aplicados a uma sequência para tentar comparar e avaliar o seu comportamento aleatório. A aleatoriedade é uma propriedade probabilística, ou seja, as propriedades de uma sequência aleatória podem ser caracterizadas e descritas em termos de probabilidade. O resultado provável de testes estatísticos, quando aplicados a uma sequência verdadeiramente aleatória, são conhecidos a priori e podem ser descritos de forma probabilística.

Existem inúmeros testes estatísticos possíveis, cada um avaliando a presença ou ausência de um padrão que, se detectado, indicaria que a sequência não tem comportamento aleatório. Como existe uma grande variedade de testes para avaliar se uma sequência tem comportamento aleatório ou não, nenhum conjunto finito específico de testes é considerado "completo".

2.5.1 Suítes de Teste

2.5.1.1 NIST Test Suite

A Suíte de testes do Instituto Nacional de Padrões e Tecnologia (NIST do inglês National Institute of Standards and Technology) (BARKER, 2010) é um pacote estatístico composto por 15 testes que foram desenvolvidos para testar a aleatoriedade de sequências binárias (arbitrariamente longas) geradas por geradores de números aleatórios baseados em hardware ou software. Esses testes têm como objetivo verificar a aleatoriade de uma sequência. Alguns testes são decompostos em uma variedade de subtestes.

Dentre os testes do NIST, os testes executados com sucesso foram:

- O teste de frequência (monobit): verifica uma distribuição uniforme de bits contando o número de uns e zeros na sequência.
- O teste de frequência de bloco: verifica padrões na sequência dividindo-a em blocos de um determinado comprimento e contando o número de vezes que cada bloco aparece.
- O teste de somas cumulativas (Cusum): verifica padrões na sequência calculando as somas cumulativas da sequência e comparando-as com uma sequência aleatória do mesmo comprimento.
- O teste de sequências: verifica sequências de bits consecutivos do mesmo valor (por exemplo, 111 ou 000) na sequência.
- O teste da maior sequência de uns: verifica a maior sequência consecutiva de uns na sequência.

- O teste de rank de matriz binária: verifica a dependência linear entre os bits na sequência construindo uma matriz.
- O teste de transformada discreta de Fourier (DFT): verifica padrões periódicos na sequência realizando uma DFT nela.
- O teste serial: verifica padrões na sequência comparando cada bit com o bit anterior.
- O teste de complexidade linear: verifica padrões lineares na sequência construindo um registrador de deslocamento de realimentação linear e calculando sua complexidade.

2.5.1.2 Dieharder

O Dieharder é uma ferramenta de teste de uniformidade para sequências aleatórias que é usada para avaliar a qualidade dos geradores de números aleatórios. Ele executa vários testes estatísticos em uma sequência de números gerados pelo gerador de números aleatórios em questão, a fim de determinar se a sequência é realmente aleatória. A versão em que os testes e estudos neste projeto utiliza é a 3.31.1, disponível e detalhada em (BROWN; BAUER, 2023).

A ferramenta é capaz de executar uma ampla gama de testes estatísticos para avaliar a qualidade dos geradores de números aleatórios. Dentre os testes disponíveis, os testes executados foram:

- Teste de Aniversário: Verifica a probabilidade de encontrar dois valores iguais em uma sequência de números gerados.
- Teste de Permutações: Analisa permutações em blocos de números para verificar a independência e uniformidade dos valores gerados.
- Teste de Rank 32x32: Avalia os ranks (ordens) de matrizes 32x32 de valores para detectar correlações e padrões.
- Teste de Rank 6x8: Verifica os ranks (ordens) de matrizes 6x8 de valores para identificar independência e uniformidade.
- Teste de Sequência de Bits: Analisa a sequência de bits gerada para verificar uniformidade e independência estatística.
- Teste de Operações Sequenciais: Avalia operações sequenciais em blocos de valores gerados para identificar não uniformidades.
- Teste de DNA: Verifica a qualidade da sequência gerada como se fosse uma sequência de DNA, buscando uniformidade e independência.

- Teste de Contagem de Uns: Conta o número de uns (1) em sequências de valores gerados para verificar uniformidade.
- Teste de Estacionamento: Avalia a distribuição de valores gerados para simular carros em um estacionamento, buscando uniformidade.
- Teste de Compressão: Analisa a qualidade da sequência gerada quando submetida a uma função de compressão "squeeze".
- Teste de Somas: Verifica a distribuição de somas de valores gerados para identificar padrões.
- Teste de Corridas: Analisa as corridas (runs) de valores gerados para verificar a uniformidade.
- Teste de Craps: Simula o jogo de dados "Craps" para avaliar a qualidade dos números gerados.
- Teste GCD: Verifica a qualidade dos números gerados usando o algoritmo de "Greatest Common Divisor".
- Teste Monobit: Verifica o número de uns (1) em sequências de bits para identificar desvios significativos.
- Teste Serial: Verifica a correlação entre valores sucessivos em sequências de bits.
- Teste de Distribuição de Bytes: Verifica a distribuição de bytes em sequências de valores gerados.
- Teste de Transformada Discreta do Cosseno: Avalia a qualidade dos valores gerados usando a Transformada Discreta do Cosseno.
- Teste de Preenchimento de Árvore: Analisa o preenchimento de uma árvore binária com os valores gerados.

3 Desenvolvimento

3.1 Arquitetura

Os sistemas com oráculos para blockchain são usados para levar dados externos para dentro da blockchain. Como os contratos não podem acessar dados externos por conta própria, eles dependem de oráculos para fornecer os dados desejados. Neste caso, os dados desejados são valores aleatórios a serem gerados a partir de alguma semente previamente registrada na blockchain.

Existem algumas maneiras em que sistemas com oráculos podem ser implementados, mas geralmente envolvem:

- Execução do contrato para solicitação de dados;
- O servidor externo busca ou gera os dados;
- O servidor externo envia os dados para a rede blockchain.

Na Solana, dados são armazenados em contas derivadas de programas (PDAs, do inglês Program Derived Adress), que são endereços gerados automaticamente pelos programas. Essas contas podem ser usadas tanto para armazenar simples valores numéricos quanto estruturas de dados mais complexas.

No contexto de números pseudoaleatórios, o usuário solicitará um número criando uma nova PDA, que terá um endereço próprio na rede assim como as informações a serem usadas pelo oráculo. O oráculo, por sua vez, usará o endereço e as informações da PDA para gerar o resultado e enviá-lo para a rede, atualizando os dados da PDA e revelando publicamente o número gerado.

O desenvolvimento de programas para Solana pode ser feito usando linguagem C ou linguagem Rust, que é a mais comum.

Para o auxílio ao desenvolvimento de programas para a Solana e também integração de sistemas externos, o framework Anchor (ANCHOR, 2022) é uma boa solução. O framework provê ferramentas como:

- Linguagem de Domínio Específico Embutida (eDSL, do inglês Embedded Domain-Specific Language) para a Rust;
- Especificação de Linguagem de Definição de Interface (IDL do inglês Interface Definition Language);

- Pacote da linguagem TypeSript para gerar clientes a partir de IDL;
- Pacote da Interface de Linha de Comando.

Desta forma, usaremos o o eDSL do Anchor para o desenvolvimento do programa, geração do IDL e geração do cliente em TypeScript. O cliente em TypeScript será o usado pelo oráculo para leitura e envio das informações da rede blockchain.

Para a execução de códigos Javascript ou Typescript fora de navegador web, existe o software Node.js (FOUNDATION, 2023) que possui arquitetura assíncrona e orientada por eventos.

3.1.1 Estrutura

Os variados casos de uso de um oráculo aleatório podem precisar de diferentes formatos e intervalos de resultados. Por conveniência ao usuário ou sistema externo que fará uso dos resultados, É permitida a escolha de intervalo de números inteiros no qual o resultado do gerador pseudoaleatório deve estar.

O oráculo precisará verificar periodicamente por novas PDAs criadas pelo programa criado. Após gerar o resultado de uma PDA e enviar o resultado para a rede, esta mesma PDA precisa ser marcada como já respondida para não ser processada novamente pelo oráculo.

A partir disto, as informações a serem armazenadas na rede Solana devem ser:

- Valor Mínimo - Inteiro sem sinal
- Valor Máximo - Inteiro sem sinal
- Resultado Obtido - Inteiro sem sinal (Maior ou igual ao valor mínimo e menor ou igual ao valor máximo)
- Processamento Ocorrido - Booleano
- Commits - Inteiro sem sinal (Número de solicitações feitas com a mesma PDA)
- User - Chave pública do usuário criador da PDA

3.2 Programa

Além de definir a estrutura dos dados a serem armazenados pelas PDAs, o programa também deve definir as instruções que poderão ser executadas através dele. De acordo com as informações definidas acima, é necessário definir uma estrutura com os

quatro campos que serão armazenados a cada solicitação de um valor pseudoaleatório, por conveniência chamaremos a estrutura de RandomValue.

Diferente de chaves públicas normais, PDAs só podem ser alteradas através da execução das instruções do contrato que originou a criação da PDA. As duas instruções necessárias para seguir o fluxo base do funcionamento do oráculo são a de criar a PDA, que chamaremos de Commit e a de revelar o número pseudoaleatório resultante, que chamaremos de Reveal.

3.2.1 Commit

Com o auxílio do framework Anchor, é possível definir constraints para auxiliar a segurança da PDA a ser criada e dos dados que serão armazenados juntos à PDA. Para os campos recebidos na instrução, são necessários ao menos três contas:

- RandomValue - O endereço público da PDA que está sendo criada na instrução de Commit
- SystemProgram - O endereço do programa nativo da Solana
- User - O endereço público da carteira que está executando a instrução

O endereço que precisa de mais verificações é o RandomValue, que está sendo criado na própria instrução, a criação deve falhar se o endereço já tiver sido criado.

O espaço reservado para todas as PDAs criadas com a estrutura RandomValue deve ser o mesmo:

- Valor mínimo - 4 bytes
- Valor máximo - 4 bytes
- Resultado - 4 bytes
- Quantidade de commits - 4 bytes
- Endereço do criador do commit - 32 bytes
- Resultado processado - 1 byte

Além dos 49 bytes que devem ser reservados para o armazenamento destes dados, 8 bytes devem ser reservados para o framework Anchor conseguir gerenciar os diferentes tipos de estruturas que um contrato pode definir, totalizando assim 57 bytes reservados e armazenados na rede a cada criação de PDA através do programa.

3.2.2 Recommit

A transação de Recommit deve ser executada pelo usuário que precisa gerar um novo resultado pseudoaleatório sem precisar pagar a taxa de Rent novamente, refazendo a solicitação com uma PDA criada previamente em uma transação de Commit. As contas utilizadas para a transação devem ser:

- RandomValue - O endereço público da PDA que está sendo reutilizada
- SystemProgram - O endereço do programa nativo da Solana
- User - O endereço público da carteira que está executando a instrução

Existem duas verificações obrigatórias, uma é de que o Recommit só pode ser executado se o resultado do Commit ou Recommit anterior já tiver sido revelado pelo servidor anteriormente, outra é de que o User deve ser o mesmo que criou a PDA.

3.2.3 Reveal

Para a instrução de revelar o resultado na PDA, além do inteiro sem sinal com o resultado gerado precisamos também de três endereços:

- RandomValue - O endereço público da PDA que está sendo atualizada com o resultado gerado
- Revealer - O endereço público da carteira com permissão de executar a transação
- SystemProgram - O endereço do programa nativo da Solana

A principal verificação a ser feita pela instrução é de que a carteira que está executando a transação seja a definida como a do oráculo, de forma que apenas o oráculo tenha capacidade de atualizar a PDA com o resultado pseudoaleatório.

3.2.4 Custos de Transação

As taxas de transação na Solana são as tarifas cobradas pelos validadores de rede para incluir transações em blocos. Essas taxas são pagas em SOL, a criptomoeda nativa da rede Solana. Elas servem para compensar os validadores pelo uso dos recursos da rede e incentivá-los a manter a rede segura e estável. O valor das taxas de transação varia dependendo do número de transações na rede e pode ser ajustado dinamicamente para garantir a escalabilidade.

Além das taxas nativas de transação, algumas instruções podem gerar uma taxa que cobra um custo de armazenamento para os programas que executam e precisam

guardar algum dado na rede. Esse custo é chamado de "aluguel" (rent, em inglês) e também é cobrado em SOL. Quanto maior a estrutura de dados reservada por uma PDA, maior o valor em SOL a ser pago em rent. É possível que em alguns contratos não seja necessário o armazenamento permanente de dados, neste caso é permitido fazer a limpeza dos dados que estão armazenados para liberar espaço e receber de volta o valor em SOL que foi pago anteriormente.

Ambas as taxas de transação e de rent descritas acima são cobradas ao executar transações com instruções de um contrato que já está implantado na rede. Antes disso, é necessário fazer a própria implantação do contrato desenvolvido, primeiro o código-fonte deve ser compilado para arquivo binário executável, depois de compilado ele pode ser enviado para a rede através de transações de implantação. Estas transações incluem custos de implantação que são tanto para pagar pelos recursos de computação para a execução do contrato quanto custo de rent para o código do próprio contrato.

Quando compilado usando o Anchor, o programa desenvolvido gera um arquivo IDL, que pode ser usado para facilitar a comunicação de outros sistemas graças à descrição detalhada das estruturas e métodos definidos no programa.

Como referência ao custo de transação em reais, será utilizada a cotação do dia 21/06/2023 às 08:50, quando o preço de 1 SOL em reais é (R$ 80.53) na rede principal da Solana (mainnet-beta).

Em rede de testes pública, a implantação de contrato custou 3.4305352 SOL, o equivalente a R$ 276.26 na rede principal.

Transação final de Implantação em rede de testes

A transação de Commit nas mesmas condições custou 0.00131848 SOL ou aproximadamente R$ 0.105

Transação de Commit em rede de testes

A transação de Recommit nas mesmas condições custou 0.000005 SOL ou aproximadamente R$ 0.0004. Resultando em uma economia de pouco mais do que dez centavos por transação quando comparado a criação de um novo Commit.

Transação de Recommit em rede de testes

A transação de Reveal custou 0.000005 SOL, mil transações de Reveal custariam aproximadamente R$ 0.40

Transação de Reveal em rede de testes

3.3 Servidor Oráculo

Seguindo o funcionamento definido para o sistema, o servidor do oráculo deve funcionar em um ciclo de ler PDAs não reveladas e, para cada PDA, gerar o resultado pseudoaleatório e atualizar os dados armazenados na PDA com o resultado gerado.

3.3.1 Leitura de PDAs não reveladas

Os nós Solana disponibilizam uma API (Interface de Programação de Aplicação) para acessar suas funcionalidades. APIs são mecanismos que permitem que dois componentes de software se comuniquem usando um conjunto de definições e protocolos. Mais especificamente, os nós disponibilizam APIs com protocolo RPC (Chamadas de Procedimento Remoto). O cliente conclui uma função (ou um procedimento) no servidor e o servidor envia a saída de volta ao cliente.

Dentre os métodos disponibilizados, o que mais condiz com a necessidade de ler PDAs é o método getProgramAccounts, que quando executado retorna todas as PDAs do programa cujo endereço público tenha sido enviado como parâmetro.

Os dados de uma conta ficam armazenados como vetores de bytes e seguindo a estrutura RandomValue definida anteriormente, que, quando armazenada na rede, ocupa 57 bytes, tem-se que o último byte é reservado ao booleano que nos diz se a PDA já foi revelada.

Além do parâmetro de endereço público do contrato, o método também pode receber um objeto de configuração com alguns campos, dentre eles há o campo filters, que nos permite enviar uma sequência de bytes e um deslocamento a partir do primeiro byte para filtragem. Para filtrar apenas PDAs não reveladas, é necessário então enviar como parâmetro um deslocamento com valor 56 e uma sequência de bytes apenas com um byte 0.

Como resposta, o método retorna um conjunto de objetos em notação JSON (Notação de Objetos do JavaScript) com a chave pública do endereço da PDA e os dados armazenados nela. Com o IDL fornecido pelo framework Anchor no momento da compilação, é possível transformar os dados de cada PDA em um objeto da linguagem JavaScript com campos seguindo as nomenclaturas do contrato desenvolvido. O oráculo pode então usar os dados recebidos para gerar o resultado que será revelado para a rede.

A leitura das PDAs não reveladas deve então ser feita periodicamente, afim de gerar o resultado de uma nova PDA o mais rápido possível depois que a PDA é criada. Caso o intervalo entre as leituras seja menor do que dois minutos, que é o tempo máximo que uma transação na Solana pode demorar para ser processada e aceita, é necessário um armazenar em cachê as transações que foram enviadas nos últimos dois minutos, para que

não sejam enviadas mais do que uma transação para revelar o resultado de uma mesma PDA.

3.3.2 Geração do Resultado

Para a geração de um valor pseudoaleatório é necessária uma semente verdadeiramente aleatória. Do ponto de vista do usuário, a informação que difere entre uma PDA e outra é o próprio endereço público da PDA. Apesar de ser computacionalmente impossível gerar um endereço público específico, é possível que um usuário gere vários conjuntos de chaves privadas e públicas até encontrar uma chave com alguma característica.

Com o intuito de ser transparente com o algoritmo usado para a geração dos resultados, se faz necessário o uso de uma outra semente que o usuário final não tenha conhecimento para ser usada em conjunto com o endereço público da PDA.

A chave privada da conta que possui permissão de revelar os resultados deve sempre ser conhecida somente pelo oráculo e nenhum usuário pode ter acesso à ela. Desta forma, para a geração dos resultados pseudoaleatórios, o servidor possui acesso a:

- Conta Reveladora: Chave privada desconhecida pelo usuário, conhecida apenas pelo servidor oráculo durante todo o ciclo.
- Endereço da PDA: Chave pública conhecida pelo usuário antes mesmo da transação de Commit.
- Mínimo e Máximo: Intervalo onde o resultado deverá estar, escolhido pelo usuário ao criar a transação de Commit.
- Commits: O número de commits que já foram efetuados pela mesma PDA.

A chave privada da conta reveladora, a chave pública da PDA e o número de commits da PDA devem ser usados como sementes do PRNG para gerar o resultado, que, por sua vez, deve ser mapeado para o intervalo de mínimo e máximo.

São muitos os algoritmos e combinações de algoritmos que podem ser usados para a geração de um valor pseudoaleatório a partir de alguma semente. Neste caso, os testes serão feitos com uma combinação dos algoritmos SHA-256 e Mersenne Twister. O uso do SHA-256 permite usar as três sementes disponíveis como uma só semente para o Mersenne Twister e o formato das sementes e resultados são compatíveis com as funções das bibliotecas de código que são integradas com a plataforma Solana.

O SHA-256 (PENARD; WERKHOVEN, 2008) é um algoritmo que recebe uma entrada de tamanho arbitrário e transforma em uma saída de tamanho fixo, chamada de hash ou de resumo criptográfico. Ele processa os dados de entrada através de uma série de

operações matemáticas e produz uma saída de 256 bits, o que significa que, juntando as duas chaves que temos como semente em uma só entrada para o SHA-256, uma pequena alteração em qualquer uma das chaves resultaria em uma saída completamente diferente. Além disso, é altamente improvável que dois conjuntos diferentes de dados produzam a mesma hash.

O Mersenne Twister (MATSUMOTO; NISHIMURA, 1998) é um algoritmo PRNG que foi projetado para ser um gerador de alta qualidade e com boa distribuição. Ele recebe um valor inteiro positivo como semente e retorna uma sequência de números pseudoaleatórios. A combinação dos dois algoritmos pode ser feita usando os primeiros 32 ou 64 bits de saída do SHA-256 como semente de entrada para o Mersenne Twister.

Tomando como exemplo de entradas as chaves:

Conta Reveladora: `489csUPUbk6Gvq51YQh3K1S9jhV9aHSW1kxec79WxzgoD25ja3cSNc3GzFqYEN88umDM2SsbDS7FjBLMeLgAccSC`

Chave da PDA: `Akw3KRafUZ2jTQym8KkwNg7bv2vmkgokKsKKoxVZ5GpD`

Número de commits: 0

E usando a concatenação de ambas como entrada em codificação base58 para o algoritmo SHA-256, temos como resultado a hash:

`473899c847ce2d33c1ada956711096f34efc1f74fea09e7dc5cbae607196ca36`

No servidor em Node.Js, convertendo os 48 primeiros bits em uma variável de inteiro sem sinal, o resultado é 24692174755938.

Usando a variável como entrada para a implementação do Mersenne-Twister em javascript da biblioteca mersenne-twister, cujo código-fonte está disponível no repositório (GUMENYUK, 2016), tem-se como retorno o número 0.22893909248523414, que por sua vez deve ser mapeado para um inteiro dentro do intervalo mínimo e máximo definido pelo usuário.

Com o inteiro resultante final e acesso à chave privada da conta que possui permissão de executar a transação de Reveal na rede, o código em Node.Js precisa usar a IDL gerada no momento da compilação do contrato para criar a instrução de Reveal com o endereço público da PDA a ser revelada e o inteiro que será revelado como resultado, a instrução deve ser adicionada em um objeto de transação, que por sua vez deve ser assinado com a conta reveladora e enviada via RPC para a rede Solana.

3.4 Integração entre Programas

A geração dos números com a interação direta entre um usuário ou servidor externo e o uso dos números gerados em aplicações fora da rede poderia levar a brechas de

segurança no sistema, A utilização deles dependeria de uma série de verificações feitas pelo criador da PDA apesar de os números estarem sendo armazenados dentro da rede para cada PDA criada. Por isso, o ideal é a criação de um terceiro programa que interaja com o programa do oráculo e execute dentro da rede as operações pertinentes a ele com os dados obtidos.

Num exemplo de um jogo integrado à blockchain, o usuário final executaria transações diretamente no programa do jogo, que, por sua vez, envia transações para o programa do oráculo através de CPIs. O programa do jogo então deve armazenar em uma PDA própria o endereço da PDA criada pelo programa do oráculo, assim como as informações referentes a aquela operação que o usuário final executou. Dessa forma, quando o número pseudoaleatório for gerado pelo servidor, o usuário final pode executar outra transação no programa do jogo, que faz a leitura do resultado contido na PDA do programa do oráculo e usa o número gerado no restante do processamento da transação. A interação entre um usuário ou sistema externo, um contrato terceiro e o sistema do oráculo está diagramada na (Figura 1).

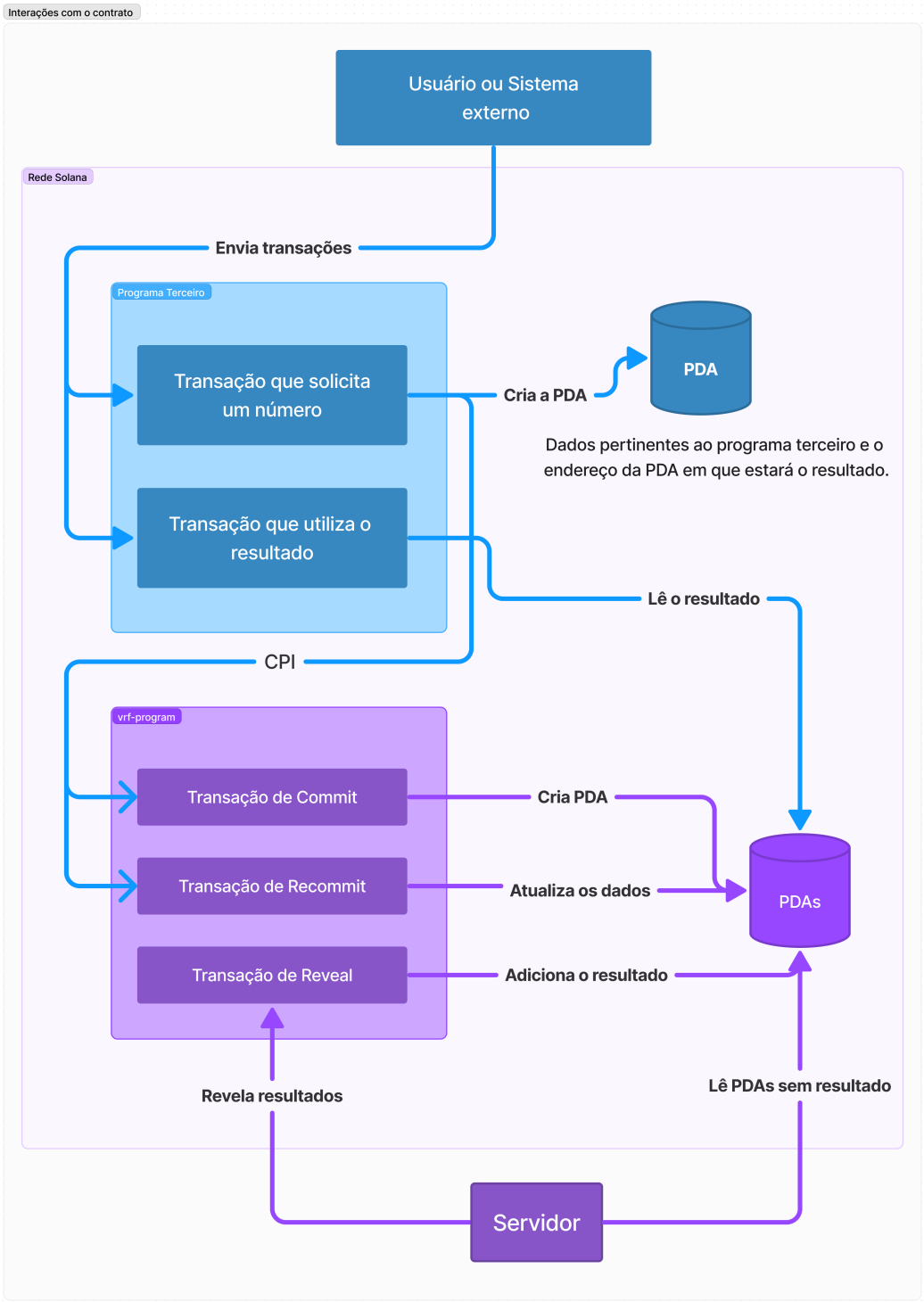

Figura 1 – Interações com contrato terceiro

4 Testes Estatísticos

Um gerador pseudoaleatório é uma ferramenta computacional que gera números que são aproximadamente independentes entre si. Esses números são frequentemente usados em simulações, criptografia e outros campos onde a aleatoriedade é importante para a segurança ou usabilidade do sistema. No entanto, ao contrário dos números verdadeiramente aleatórios, gerados a partir de fontes físicas específicas, os números gerados por geradores pseudoaleatórios são determinísticos.

Por essa razão, é importante testar rigorosamente a qualidade dos números gerados por um gerador pseudoaleatório antes de usá-los em aplicações críticas. Ferramentas de testes estatísticos fornecem uma maneira de verificar se um gerador pseudoaleatório está gerando números que tenham comportamento aleatório o suficiente para atender o nível de segurança que a aplicação exige

No algoritmo de geração pseudoaleatória do oráculo, são usadas duas sementes, uma permanente durante o funcionamento do servidor e outra que pode ser parcialmente escolhida pelo usuário. Pensando nas duas formas de solicitar números, serão feitos testes com:

- Chave privada fixa com endereços de PDA quaisquer em primeiro Commit
- Chave privada fixa com um mesmo endereço de PDA em Recommits

Para cada uma das formas de gerar sementes, foram gerados 1000000000 bits com o gerador pseudoaleatório a partir de chaves geradas com a função disponibilizada pela biblioteca crypto no ambiente NodeJs getRandomValues(), que usa fontes de entropia como dados do dispositivo ou do sistema operacional.

4.1 Testes com NIST Test Suite

O NIST Test Suite é um conjunto de testes estatísticos desenvolvidos para avaliar a qualidade de geradores aleatórios. Ele é amplamente utilizado em diversas aplicações e inclui uma variedade de testes estatísticos projetados para detectar diferentes tipos de não-aleatoriedade. Os resultados dos testes são avaliados com base em um valor P, que indica a probabilidade de que os números gerados sejam considerados aleatórios. As métricas consideradas para validação dos resultados assim como o funcionamento de cada teste é descrito detalhadamente em (BARKER, 2010).

Para cada teste realizado com o NIST, os 1000000000 bits são separados em 100000 sequências de 10000 bits, cada grupo é usado nos testes a fim de levar a proporção de sequências que passam pelos testes, se por exemplo 99000 sequências passarem, a proporção é de 99000/100000.

4.1.1 Chaves privadas e endereços de PDA quaisquer

Os testes cujos resultados estão detalhados na Tabela (1) foram feitos usando a ferramenta NIST com valores obtidos através do algoritmo gerador usando chaves privadas e públicas como semente, ambas geradas com fontes de entropia a partir da função disponibilizada pela biblioteca crypto do NodeJs. Cada número foi gerado com uma mesma chave privada, diferentes PDAs e número de commit igual a 0.

Tabela 1 – Resultados de testes com NIST em diferentes PDAs

Teste	Proporção
Frequência	98916/100000
Frequência de bloco	99040/100000
Somas cumulativas	98944/100000
Sequências	99000/100000
Maior sequência de uns	99017/100000
Rank de matriz binária	99246/100000
Transformada discreta de Fourier	98908/100000
Serial	98955/100000
Complexidade Linear	96404/100000

4.1.2 Chave privada fixa com endereços de PDA quaisquer

Os testes cujos resultados estão detalhados na Tabela (2) foram feitos usando uma mesma chave privada e diferentes chaves públicas como semente, ambas geradas com fontes de entropia a partir da função disponibilizada pela biblioteca crypto do NodeJs. Cada número foi gerado com uma mesma chave privada, uma mesma PDA e diferentes números de commit.

4.1.3 Análise dos Resultados

Para os dois testes com as diferentes formas de geração de sementes, a proporção mínima para que os 1000000000 bits passem por cada teste é de aproximadamente 98905 para as amostras de 100000 sequências.

Tendo como referência os resultados dos dois grupos de bits e os 11 testes executados em cada um, as Tabelas (1) e (2), apresentam resultados com valores próximos, em que dos 11 testes, os três grupos de bits passaram com sucesso por 10 e falharam ape-

Tabela 2 – Resultados de testes com NIST em diferentes Commits

Teste	Proporção
Frequência	98928/100000
Frequência de bloco	99019/100000
Somas cumulativas	99005/100000
Sequências	98982/100000
Maior sequência de uns	98979/100000
Rank	99187/100000
Transformada discreta de Fourier	98914/100000
Serial	99004/100000
Complexidade Linear	96515/100000

nas no teste de Complexidade Linear, que avalia se a distribuição de saída dos números gerados é ou não linear.

Nos dois grupos, os testes que passaram e falharam foram os mesmos, com proporções que tiveram pouca divergência entre um grupo e outro, o que mostra que, com os bits gerados para estes testes, a forma de escolha das sementes não afetou negativamente nem positivamente a aleatoriedade dos resultados obtidos. Como princípio, sementes de um gerador pseudoaleatório não podem afetar o comportamento de uma sequência gerada. Mas, é inquestionável que uma semente que tenha sido acessada indevidamente pode comprometer a segurança do sistema como um todo.

4.2 Testes com a ferramenta Dieharder

O Dieharder é uma ferramenta de testes estatísticos desenvolvidos para avaliar a qualidade de geradores aleatórios. Ele inclui uma variedade de testes estatísticos projetados para detectar diferentes tipos de não-aleatoriedade.

Para os testes relatados abaixo, foi utilizada a versão 3.31.1 do Dieharder, disponível em (BROWN; BAUER, 2023).

O Dieharder realiza vários testes estatísticos para avaliar a qualidade dos números gerados pelo gerador pseudoaleatório. Estes testes são realizados com amostras de números gerados, e a amostra é diferente a cada execução do Dieharder. Isso também pode levar a resultados diferentes a cada vez que o Dieharder é executado. Para uma melhor documentação dos resultados e de quantas vezes cada teste é considerado como sucesso (PASSED), fraco (WEAK) ou falho (FAILED), serão executados mais do que um teste de cada um dos que estão descritos como boa confiabilidade em (BROWN; BAUER, 2023).

4.2.1 Chave privada fixa com endereços de PDA quaisquer em primeiro Commit

Os testes cujos resultados estão detalhados na Tabela (3) foram feitos usando a ferramenta Dieharder com valores obtidos através do algoritmo gerador usando como semente chaves privadas e públicas como semente, ambas geradas com fontes de entropia a partir da função disponibilizada pela biblioteca crypto do NodeJs. Cada número foi gerado com uma mesma chave privada, diferentes PDAs e número de commit igual a 0.

Tabela 3 – Resultados de testes com dieharder com diferentes PDAs

Teste	Resultado
Aniversário	PASSED
Permutações	PASSED
Rank 32x32	PASSED
Rank 6x8	PASSED
Sequência de bits	PASSED
Operações sequenciais	2/2 PASSED
DNA	PASSED
Contagem de Uns	PASSED
Estacionamento	PASSED
Compressão	WEAK
Somas	PASSED
Corridas	2/2 PASSED
Craps	2/2 PASSED
GCD	2/2 FAILED
Monobit	PASSED
Serial	1/16 WEAK 15/16 PASSED
Distribuição de bytes	FAILED
Transformada discreta	PASSED
Preenchimento de árvore	PASSED

4.2.2 Chave privada fixa com um mesmo endereço de PDA em Recommits

Na Tabela (4) cada número usado nos testes foi gerado com uma mesma chave privada, uma mesma PDA e diferentes números de commit.

4.2.3 Análise dos Resultados

Para os dois testes com as diferentes formas de geração de sementes, houve alguns testes que passaram, alguns com resultados fracos, e alguns que não passaram. O teste que mais apresentou falhas foi o RGB Lagged Sum, que verifica se os números gerados apresentam algum padrão ou tendência, mesmo que com algum atraso entre um padrão e outro na sequência de bits.

Tabela 4 – Resultados de testes com dieharder com PDA fixa e Recommits

Teste	Avaliação
Aniversário	PASSED
Permutações	PASSED
Rank 32x32	PASSED
Rank 6x8	PASSED
Sequência de bits	PASSED
Operações sequenciais	2/2 PASSED
DNA	PASSED
Contagem de Uns	2/2 PASSED
Estacionamento	PASSED
Compressão	PASSED
Somas	PASSED
Corridas	2/2 PASSED
Craps	2/2 PASSED
GCD	2/2 FAILED
Monobit	PASSED
Serial	1/16 WEAK 15/16 PASSED
Distribuição de bytes	FAILED
Transformada discreta	PASSED
Preenchimento de árvore	2/2 PASSED

Diferente dos resultados obtidos com os testes testes realizados usando o NIST, alguns testes do dieharder geraram resultados diferentes dentre os grupos de bits usados como entrada. Um motivo que pode ter gerado tal inconsistência é de que o número de bits gerados e utilizados como entrada para os testes não tenha sido suficiente para garantir consistência dos resultados, já que quanto maior a quantidade de bits, mais eficientes e precisas são as avaliações realizadas pelo dieharder.

De acordo com a aplicação do algoritmo gerador e do uso do sistema com oráculo, pode ser necessária uma avaliação específica sobre a suficiência da força da aleatoriedade, e, em casos de uso onde sejam necessários números com aleatoriedade que passem por todos os testes do dieharder, é indispensável que sejam refeitos os testes com uma maior amostra de bits ou que seja feito o uso de um algoritmo gerador diferente do utilizado nos testes acima.

5 Considerações Finais

5.1 Conclusão

O desenvolvimento do contrato inteligente, do servidor de geração de números aleatórios, o fluxo do sistema com a solicitação de um número e o envio do resultado através de rede Solana funcionaram como o esperado, com baixo custo para os usuários que fazem a solicitação dos números pseudoaleatorios. A disponibilização dos códigos fonte de ambos os sistemas está descrita no Apêndice A.

A interação com o sistema desenvolvido pode ser feita tanto por um sistema externo que envie instruções para a Solana diretamente para o contrato do oráculo quanto por um outro contrato que faça chamadas entre programas e solicite um valor pseudoaleatório para um objetivo próprio.

Embora a qualidade dos números gerados pode não ter sido suficiente para que os testes fossem realizados de forma significativa e eficiente, a suficiência dos resultados do gerador pode depender do caso de uso a que o sistema será aplicado. É importante lembrar que os sistemas de oráculo são frequentemente usados em aplicações que exigem aleatoriedade para garantir a integridade e a segurança das operações realizadas na plataforma. Assim, é fundamental avaliar a aleatoriedade antes de utilizar o sistema de oráculo.

Em alguns casos, como jogos ou geração de atributos de tokens, os resultados que foram obtidos no Capítulo 4 podem ser suficientes enquanto em outros, como sorteios de prêmios, pode ser necessário realizar ajustes no servidor, principalmente na escolha das sementes e do algoritmo gerador de números pseudoaleatorios.

5.2 Melhorias Futuras

Com base nos resultados e experiências obtidos durante o desenvolvimento do projeto, surgiram algumas visões de pontos que podem ser melhorados futuramente:

- Desenvolver testes unitários para o código fonte do contrato e do servidor;
- Executar testes com diferentes algoritmos e mais amostras para o Dieharder;
- Criar uma plataforma integrada ao servidor oráculo que permita a verificação externa de resultados para PDAs que já foram criadas previamente.

Referências

AL-BREIKI, H. et al. Trustworthy blockchain oracles: review, comparison, and open research challenges. *IEEE Access*, IEEE, v. 8, p. 85675–85685, 2020. Citado na página 20.

ANCHOR. *Anchor*. 2022. Disponível em: <https://github.com/coral-xyz/anchor>. Acesso em: 20 dez. 2022. Citado na página 25.

BAMBYSHEVA, N. *Web3 Growth Stymied By Scarcity Of Programmers*. 2022. Disponível em: <https://www.forbes.com/sites/ninabambysheva/2022/08/29/web3-growth-stymied-by-scarcity-of-programmers/>. Acesso em: 25 dez. 2022. Citado na página 17.

BARKER, E. *A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications*. ITL Bulletin, National Institute of Standards and Technology, Gaithersburg, MD, 2010. Disponível em: <https://nvlpubs.nist.gov/nistpubs/legacy/sp/nistspecialpublication800-22r1a.pdf>. Citado 2 vezes nas páginas 22 e 35.

BROWN, D. E. R. G.; BAUER, D. *Dieharder: A Random Number Test Suite*. 2023. Disponível em: <https://webhome.phy.duke.edu/~rgb/General/dieharder.php>. Citado 2 vezes nas páginas 23 e 37.

FOUNDATION, C. *CHAINLINK VRF*. 2022. Disponível em: <https://blog.chain.link/verifiable-random-function-vrf/>. Acesso em: 20 dez. 2022. Citado 2 vezes nas páginas 17 e 20.

FOUNDATION, O. *Node Js*. 2023. Disponível em: <https://nodejs.org/pt-br/>. Acesso em: 02 jan. 2023. Citado na página 26.

GUMENYUK, E. *Mersenne-Twister*. 2016. Disponível em: <https://github.com/boo1ean/mersenne-twister>. Citado na página 32.

MATSUMOTO, M.; NISHIMURA, T. Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, ACM New York, NY, USA, v. 8, n. 1, p. 3–30, 1998. Citado na página 32.

NAKAMOTO, S. Bitcoin: A peer-to-peer electronic cash system. *Decentralized Business Review*, p. 21260, 2008. Citado na página 17.

PENARD, W.; WERKHOVEN, T. van. On the secure hash algorithm family. *Cryptography in context*, Wiley Newyork, p. 1–18, 2008. Citado na página 31.

SCHWEIGER, L. *BLOCKDATA*. 2021. Disponível em: <https://www.blockdata.tech/blog/general/81-of-the-top-100-public-companies-are-using-blockchain-technology>. Acesso em: 24 dez. 2022. Citado na página 17.

THE MIT License. 2023. Disponível em: <https://opensource.org/license/mit/>. Acesso em: 28 jul. 2023. Citado na página 47.

WATTS, A. *Layer-1 Performance comparision*. 2023. Disponível em: <https://coincodex.com/article/14198/layer-1-performance-comparing-6-leading-blockchains/>. Acesso em: 19 jun. 2023. Citado na página 20.

YAKOVENKO, A. *Solana: A new architecture for a high performance blockchain v0.8.13*. [S.l.], 2018. Citado 3 vezes nas páginas 17, 19 e 20.

ZHANG, R.; XUE, R.; LIU, L. Security and privacy on blockchain. *ACM Computing Surveys (CSUR)*, ACM New York, NY, USA, v. 52, n. 3, p. 1–34, 2019. Citado na página 19.

Apêndices

APÊNDICE A – Códigos Fonte

Repositórios

O sistema ficou dividido em dois repositórios, um com o código fonte do programa que é executado nos nós de rede Solana e outro com o código fonte do gerador de números pseudoaleatórios que é executado externamente.

Os códigos fonte do gerador externo e do programa interno à blockchain estão disponibilizados de forma pública na plataforma Github em:

- vrf-server: https://github.com/ArthurPaivaT/vrf-server
- vrf-program: https://github.com/ArthurPaivaT/vrf-program

Licenciamento

O licenciamento de ambos os códigos fonte é o MIT. O licenciamento MIT diz que a permissão é concedida gratuitamente, a qualquer pessoa que obtenha uma cópia do software e arquivos de documentação, incluindo, sem limitação, os direitos de usar, copiar, mesclar, publicar, distribuir, sublicenciar e/ou vender cópias do software. O licenciamento também diz que o software é fornecido sem garantia de qualquer tipo, expressa ou implícita, incluindo, sem limitação, as garantias de comerciabilidade, adequação a um determinado fim e não violação. Em nenhum caso os autores ou detentores dos direitos autorais serão responsáveis por qualquer reinvidicação, danos ou outra responsabilidade The... (2023).

vrf-server

O repositório vrf-server, com o código fonte para o gerador pseudoaleatório que se comunica com a rede Solana, está estruturado em:

- src/: Pasta com o código fonte principal do projeto com a lógica de implementação do servidor;
- README.md: Documento de texto com formatação Markdown, que serve como introdução e guia para executar o projeto;
- tsconfig.json: Configuração do compilador typescript;
- package.json: Configuração das dependências de bibliotecas do javascript e de comandos de execução do servidor;

- package-lock.json: Arquivo gerado automaticamente pelo npm, que registra as versões e hashes exatas de cada pacote instalado;
- .gitignore: Usado pelo Git para definir arquivos ou pastas que devem ser ignorados e não incluídos no controle de versão.

vrf-program

O repositório vrf-program, com o código fonte para o programa a ser executado em plataforma Solana, está estruturado em:

- migrations/: Pasta dedicada ao framework anchor com os scripts de implantação de programa;
- programs/: Pasta com o código principal dos programas;
- .gitignore: Usado pelo Git para definir arquivos ou pastas que devem ser ignorados e não incluídos no controle de versão;
- Anchor.toml: Configuração do framework anchor com as informações dos scripts e integração com a rede;
- Cargo.lock: Arquivo gerado automaticamente pelo ambiente rust, que registra as versões e hashes exatas de cada pacote instalado;
- README.md: Documento de texto com formatação Markdown, que serve como introdução e guia para implantar os contratos.