UNIVERSIDADE FEDERAL DE PERNAMBUCO
CENTRO DE TECNOLOGIA E GEOCIÊNCIAS
DEPARTAMENTO DE ENGENHARIA ELÉTRICA
CURSO DE ENGENHARIA DE CONTROLE E AUTOMAÇÃO

PAULO RICARDO MONETA NUNES

UTILIZAÇÃO DE BLOCKCHAIN EM REDES DE COMUNICAÇÃO INDUSTRIAL

Dezembro/2021

PAULO RICARDO MONETA NUNES

UTILIZAÇÃO DE BLOCKCHAIN EM REDES DE COMUNICAÇÃO INDUSTRIAL

Trabalho de Conclusão do Curso de Engenharia de Controle e Automação da Universidade Federal de Pernambuco, como requisito da disciplina de Trabalho de Conclusão de Curso (EL403)

Orientador: Prof. M.Sc. Geraldo Leite Maia Júnior

Ficha de identificação da obra elaborada pelo autor,
através do programa de geração automática do SIB/UFPE

Nunes, Paulo Ricardo Moneta.
UTILIZAÇÃO DE BLOCKCHAIN EM REDES DE COMUNICAÇÃO INDUSTRIAL / Paulo Ricardo Moneta Nunes. - Recife, 2022.
65 : il.

Orientador(a): Geraldo Leite Maia Júnior
Trabalho de Conclusão de Curso (Graduação) - Universidade Federal de Pernambuco, Centro de Tecnologia e Geociências, Engenharia de Controle e Automação - Bacharelado, 2022.

1. Blockchain. 2. Modbus. 3. Ethereum. 4. Contrato inteligente. 5. TLS. I. Júnior, Geraldo Leite Maia. (Orientação). II. Título.

620 CDD (22.ed.)

PAULO RICARDO MONETA NUNES

UTILIZAÇÃO DE BLOCKCHAIN EM REDES DE COMUNICAÇÃO INDUSTRIAL

Trabalho de Conclusão de Curso do Curso de Engenharia de Controle e Automação da Universidade Federal de Pernambuco, como requisito da disciplina de Trabalho de Conclusão de Curso (EL403).

BANCA EXAMINADORA: - Prof. M.Sc. Geraldo Leite Maia Júnior (Orientador)
Universidade Federal de Pernambuco

- Prof. Dr. Márcio Evaristo da Cruz Brito
Universidade Federal de Pernambuco

AGRADECIMENTOS

À minha família, pelos ensinamentos de vida e pelo apoio incondicional ao longo do curso.

À Universidade Federal de Pernambuco, pelo ambiente de aprendizado e convivência que influenciou bastante a minha vida.

Aos amigos e amigas que tive a oportunidade de conhecer durante este período, e que foram essenciais em momentos de lazer e de esforço em diversos projetos e disciplinas.

Aos professores, responsáveis pelo aprendizado profissional e pessoal transmitido com muita dedicação.

Agradeço ao professor e orientador Geraldo Leite Maia, sem o qual não seria possível a realização deste e de vários outros trabalhos durante o curso.

Por fim, obrigado a todos que direta ou indiretamente fizeram parte da minha formação.

RESUMO

Este trabalho teve o objetivo de implementar uma rede blockchain simples com autenticação TLS em leitura e escrita de dados, para comunicar dispositivos de chão de fábrica com as partes que fazem uso dos dados destes dispositivos. A topologia exibida neste trabalho utiliza o protocolo de comunicação MODBUS para adquirir respostas e enviar requisições em rede MODBUS RTU, que foram colocados em uma rede blockchain privada *Ethereum* através de contratos inteligentes, implementada através da plataforma de desenvolvimento *Hyperledger Besu*. Para esta topologia serão utilizadas três máquinas virtuais (cada uma emulando um minicomputador *Raspberry Pi*) e um software emulador de mestres e escravos (*IOServer*) na rede MODBUS RTU, que fará as requisições e aquisições de dados. Este trabalho esclarece alguns conceitos relacionados às redes descentralizadas e criptografia, bem como a apresentação de uma rede blockchain que pode ser utilizada na indústria em conjunto com outros protocolos de redes industriais, visando garantir a integridade dos dados ao utilizar uma comunicação descentralizada e a segurança da rede, pois são usadas várias camadas de autenticação e criptografia dentro e fora da blockchain que impedem diversos vetores de ataque à rede, muito comuns em redes não distribuídas. Como resultado, foi possível estabelecer uma comunicação Modbus entre mestre e escravo, com dados trafegando em uma rede blockchain.

Palavras-chave: Blockchain, Hyperledger Besu, Modbus, Raspberry Pi, contratos inteligentes, TLS, Ethereum, IOServer.

ABSTRACT

This work aimed to implement a simple blockchain network with TLS authentication in data reading and writing, to communicate factory floor devices with the parties that make use of the data from these devices. The topology shown in this work uses the MODBUS communication protocol to acquire responses and send requests on a MODBUS RTU network, which were placed on a private Ethereum blockchain network through smart contracts, implemented through the Hyperledger Besu development platform. For this topology, three virtual machines will be used (each one emulating a Raspberry Pi minicomputer) and a master and slave emulator software (IOServer) in the MODBUS RTU network, which will make the requests and data acquisitions. This work clarifies some concepts related to decentralized networks and cryptography, as well as the presentation of a blockchain network that can be used in industry in conjunction with other industrial network protocols, in order to ensure data integrity by using a decentralized communication and the security of the network, as multiple layers of authentication and encryption are used inside and outside the blockchain that prevent multiple attack vectors to the network, very common in non-distributed networks. As a result, it was possible to establish a Modbus communication between master and slave, with data traveling on a blockchain network.

Key-words: Blockchain, Hyperledger Besu, Modbus, Raspberry Pi, smart contracts, TLS, Ethereum, IOServer.

LISTA DE FIGURAS

LISTA DE SIGLAS

GB	Gygabyte
HTTP	Hyper Text Transfer Protocol
HTTPS	Hyper Text Transfer Protocol Secure
IOT	Internet of Things
IP	Internet Protocol
MB	Megabyte
NVM	Node Version Manager
P2P	Peer-to-Peer
RAM	Random Access Memory
SHA-256	Secure Hash Algorithm - 256
TCP	Transmission Control Protocol
TLS	Transport Layer Security
URL	Uniform Resource Locator
VM	Virtual Machine

SUMÁRIO

1 Introdução

Com o surgimento da indústria 4.0, existe uma forte tendência de que novos dispositivos sejam utilizados para levar os dados da indústria para a internet. O movimento de levar dados de dispositivos do cotidiano é chamado de internet das coisas (IoT) (WIKIPÉDIA). O número de ataques maliciosos a estes tipos de dispositivos dobrou na primeira metade de 2021, quando comparado com todo o ano de 2020 (THREATPOST), o que traz diversos questionamentos quanto a sua segurança. IoT industrial é uma subcategoria de IoT, onde estão incluídos dispositivos dentro e fora da indústria.

Para resolver o problema da segurança na indústria e garantir a integridade dos dados destes dispositivos, é possível utilizar uma rede blockchain que irá fornecer um serviço confiável de dados imutáveis e rastreáveis apenas por participantes da rede, que estarão distribuídos entre diversos dispositivos e uma comunicação criptografada dentro e fora da rede blockchain. Ao fazer essa distribuição de dados, cada dispositivo poderá interagir com outros para verificar a veracidade destes dados, e é essa característica da rede que a torna mais segura quando comparada com um banco de dados centralizado (REYNA, MARTÍN, *et al.*, 2018).

1.1. Objetivos

A topologia desenvolvida visa demonstrar o funcionamento de uma rede blockchain privada conectada a uma rede Modbus para garantir a segurança e veracidade dos dados colocados na rede e a anonimidade dos equipamentos e usuários que modificam ou usam estes dados. Foram utilizadas bibliografias acadêmicas e profissionais para identificar trabalhos relacionados e verificar a possibilidade de integração de uma rede blockchain com uma rede industrial. Para este trabalho, foram definidos os seguintes objetivos:

- **Descrever as blockchains.**
- **Comparação blockchain *vs*. banco de dados.**
- **Comparar algoritmos de consenso.**
- **Configuração da blockchain.**
- **Configuração dos dispositivos Modbus.**
- **Execução do conjunto blockchain + mestre Modbus + escravo Modbus.**

1.2. Por que usar blockchain?

Além da questão de criptografia da comunicação em uma rede blockchain, outro fator importante é a descentralização dos dados trafegados, que ficam armazenados em diversos dispositivos espalhados pela rede. Abaixo, estão mostradas cinco comparações, que analisam vantagens e desvantagens entre uma blockchain (em termos de descentralização de dados) e um banco de dados centralizado:

- **Confiança**: Uma rede blockchain não precisa de terceiros confiáveis para operar, apenas dos nós que mantêm a validação dos blocos na rede. Cada nó participante toma parte na validação de cada bloco que é gerado na rede. Já um banco de dados centralizado necessita de uma autoridade confiável operando, o que traz vantagem para uma rede blockchain (M. J. M. CHOWDHURY, 2018).
- **Confidencialidade e privacidade**: Ao participar de uma rede blockchain, cada nó sabe qual dado foi transferido entre outros nós, mas a identificação de qualquer nó na rede (chave pública) impede que outro nó saiba quem são as partes envolvidas numa transação. Em uma rede blockchain privada, cada chave pública pode ser associada com o nome de algum dispositivo para fazer transações ou procurar essas transações na blockchain e saber quem a realizou. Em comparação com um banco de dados centralizado, cada dado é encaminhado diretamente para onde deve ir, sem que outros participantes

tenham acesso ao dado, isso faz com que bancos de dados tenham vantagem no quesito privacidade (M. J. M. CHOWDHURY, 2018).

- **Robustez/Tolerância a modificação**: Como todas as transações da rede são distribuídas totalmente entre todos os nós, torna-se muito improvável a alteração ou manipulação destes dados por algum ou alguns agentes maliciosos, contanto que existam muitos nós confiáveis, pois, para que uma modificação seja feita em uma transação já realizada, seria necessário descriptografar todos os blocos sucessivos ao bloco que contém esta transação e aplicar essa modificação na maioria dos nós validadores da rede, já que a rede funciona através de consenso. Diferente de um banco de dados centralizado, onde um dado pode ser alterado em apenas um lugar, deixando-o mais tolerante a modificações por ataques maliciosos. Esta comparação faz com que uma rede blockchain seja mais vantajosa (M. J. M. CHOWDHURY, 2018).
- **Desempenho**: Devido ao tempo de criação e validação de novos blocos na rede, uma blockchain ainda não consegue processar muitas transações em curto intervalo de tempo, mas já existem algumas blockchains como a Solana (SOL), que realiza cerca de 50,000 transações por segundo, número semelhante ao de sistemas financeiros de empresas como *Visa* e *Mastercard* (M. J. M. CHOWDHURY, 2018). Em termos de desempenho, as blockchains atuais já possuem desempenho muito próximo de um banco de dados.
- **Segurança**: a segurança de uma blockchain depende da quantidade de nós na rede. Caso a maioria dos nós validem um bloco na rede, esse bloco é tido como válido. Este é o fator que torna uma blockchain segura, pois para alterar um dado, é necessário que a maioria dos nós da rede sejam invadidos, o que é improvável na prática devido à quantidade de criptografia utilizada por todos os nós. Já um banco de dados centralizado depende de controle de acesso de usuário, que pode ser comprometido caso o administrador deste banco de dados também seja comprometido, trazendo vantagem para a rede blockchain (M. J. M. CHOWDHURY, 2018).

O uso de blockchain geralmente é recomendado quando dados precisam ser trocados entre partes que não possuem confiança entre si. No caso da indústria, algumas vezes é necessário fornecer dados para outras empresas ou indivíduos através da internet.

Com a utilização de uma rede blockchain, dados estarão disponíveis quase que instantaneamente para o destinatário (nas redes que usam consenso de prova de autoridade, por exemplo), possibilitando mais transparência entre as partes envolvidas e maior eficiência no uso destes dados, sem a existência de terceiros intermediários e com alta segurança, pois para que os dados sejam considerados corrompidos, é necessária a existência de muitos nós mal-intencionados.

1.3. Justificativas

Existem duas justificativas para este trabalho:

- **Segurança**: Existe uma grande necessidade de proteger sistemas e dados industriais, que podem ser acessados e explorados através de invasões em redes tradicionais de comunicação ou acessos maliciosos aos equipamentos da fábrica. Um exemplo de invasão é a do *worm* de computador *Stuxnet*, que atacou o sistema SCADA da *Siemens*, fazendo com que as centrífugas de uma usina nuclear girassem 40% mais rápido sem notificar o sistema SCADA. Ele foi descoberto antes que causasse um acidente mais grave na fábrica, por um funcionário de uma empresa de segurança que foi contactada pelo suporte técnico da usina para relatar mal funcionamento dos computadores (WIKIPÉDIA).

- **Descentralização**: Manter dados distribuídos da forma proposta diminui as chances de adulterações ou perdas, visto que somente com autenticação correta e verificada por todos os participantes da rede é possível modificar

estes dados. Com isso, estruturas de dados distribuídas podem garantir maior velocidade e robustez na troca de informações entre uma planta industrial e um sistema de supervisão, por exemplo, bem como a integridade de dados já consolidados.

1.4. Estrutura

Este trabalho está estruturado em 5 capítulos:

- O primeiro capítulo (Introdução) apresenta o problema da segurança em dispositivos IoT e possíveis soluções para o problema.
- O segundo é uma revisão teórica de termos e configurações utilizados neste trabalho.
- O terceiro mostra todo o desenvolvimento da topologia de comunicação utilizada para comunicar uma rede blockchain com uma rede industrial.
- O quarto mostra os resultados adquiridos na execução das máquinas virtuais em conjunto com o emulador Modbus.
- O quinto apresenta a conclusão que se obteve a partir dos resultados e trabalhos futuros relacionados.

2 Revisão bibliográfica

Este trabalho cita diversos termos relacionados à criptografia, assinatura digital e redes blockchain. Para ajudar na compreensão do texto, este capítulo foi criado para explicar melhor os termos mais importantes e porque são utilizados no trabalho.

2.1 Blockchain

A Tradução mais aproximada da palavra blockchain para a língua portuguesa seria cadeia de bloco. Esta palavra se refere ao modo como os dados são transmitidos nesta rede: cada bloco é diretamente ligado ao bloco seguinte. Um bloco é composto de informações criptografadas da geração do bloco (*hash* de assinaturas e dados de funcionamento da blockchain) e lista de transações que ocorreram entre o período de geração do bloco até sua validação. Após as transações serem adicionadas, uma assinatura criptografada (*hash* do bloco) é adicionada ao final do bloco. Esta assinatura contém os dados de número do bloco atual, próximo bloco, data e hora de assinatura e a quantidade de transações presentes no bloco, além da associação com a assinatura do bloco anterior (WIKIPÉDIA).

Em uma rede blockchain com mineração, a criação de um bloco é feita pelo minerador que resolveu primeiro um problema matemático complexo através de cálculos computacionais. Ao encontrar essa solução, ela precisa ser aceita por todos os nós validadores da rede para que o bloco seja assinado. Já em uma rede com prova de autoridade, nós confiáveis são reconhecidos como validadores e assinam blocos em determinados períodos de tempo (WIKIPÉDIA).

A rede blockchain é classificada como uma tecnologia de registro distribuído (*Distributed Ledger Technology*), ou seja, um banco de dados que existe em vários locais e possui múltiplos participantes. A primeira menção ao uso de blockchain foi feita por Stuart Haber e W. Scott Stornetta (HABER, 1991), já uma aplicação real destes blocos criptografados em cadeia foi inicialmente proposta por Satoshi nakamoto (NAKAMOTO, 2008), cuja identidade ainda não foi comprovada, ao propor

em 2009 um sistema de pagamento eletrônico de ponto a ponto (*peer-to-peer*) com transações computacionalmente impraticáveis de reverter chamado de *Bitcoin*, acabando com o problema de gasto duplo (fazendo com que a moeda digital utilizada não possa ser copiada para ser usada em outro lugar).

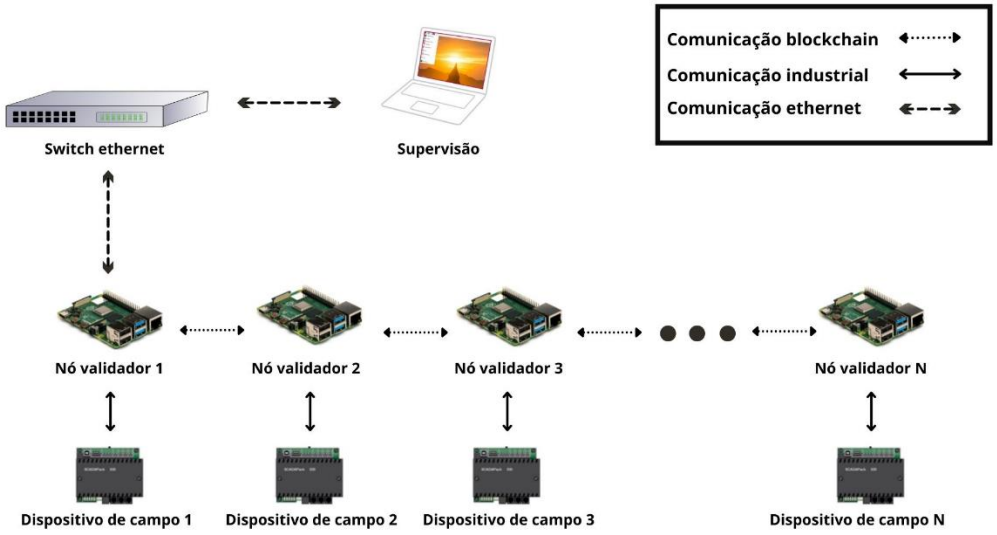
Esta rede é considerada como uma rede *peer-to-peer* (P2P): um conjunto de computadores que compartilham tarefas, trabalhos ou arquivos entre si (*peers*). Nesta rede, cada *peer* é chamado de nó ou ponto, e todas as informações referentes a acréscimo de nós, transações e validação e geração de blocos são compartilhadas entre todos os nós da rede, dependendo de um consenso entre todos os validadores para funcionar. Dentro da rede, cada usuário é associado a uma chave pública e uma chave privada. Sua chave pública (o endereço hexadecimal que os usuários da rede utilizam para enviar ou receber transações) pode ser gerada a partir de uma sequência de 12 até 24 palavras, na rede *Ethereum*, ou a partir de um número de 256 bits, no caso da rede *Bitcoin*. O uso destas chaves públicas garante a anonimidade de cada participante, uma vez que somente a chave pública é utilizada para qualquer interação do usuário com a rede.

As primeiras blockchains criadas eram usadas apenas para transações de moedas digitais anônimas entre usuários e mineração de blocos para validar estas transações. Já a blockchain *Ethereum* foi criada com o intuito de permitir o uso de contratos inteligentes, além de transações entre usuários e da mineração de blocos. Os contratos inteligentes serão melhor explicados na seção de revisão.

Em relação à integração entre IoT e blockchain, existem três tipos de abordagem: IoT-IoT, IoT-Blockchain e abordagem híbrida (REYNA, MARTÍN, *et al.*, 2018). No primeiro tipo (IoT-IoT), a comunicação entre dispositivos IoT é feita apenas entre os próprios dispositivos, com a blockchain sendo utilizada apenas para armazenar dados de saída desses dispositivos. No segundo tipo (IoT-Blockchain), toda a comunicação entre dispositivos IoT e seus dados de saída são colocados em uma blockchain, apresentando maior latência em relação ao primeiro método. Já o terceiro tipo (abordagem híbrida) representa um meio termo entre os dois primeiros, onde apenas seus dados e parte das interações entre os dispositivos IoT serão colocados na blockchain e a outra parte será feita entre os próprios dispositivos.

Uma possível topologia do tipo IoT-IoT (REYNA, MARTÍN, *et al.*, 2018) a ser utilizada na indústria está mostrada na Figura 1. Esta topologia foi escolhida para este trabalho porque a rede industrial que foi integrada à rede blockchain é do tipo Modbus, uma rede que não permite a comunicação entre os dispositivos escravos, pois apenas os dispositivos mestres podem abrir comunicação. Ela mostra vários nós validadores comunicando entre si em uma rede blockchain, onde cada nó está conectado a um dispositivo de campo, enviando ou recebendo requisições de um dispositivo escravo. Apenas as requisições e respostas das requisições são colocadas na blockchain, permitindo uma comunicação segura entre todos os dispositivos. Além disso, essa informação pode ser passada para uma plataforma de supervisão através de rede ethernet, por exemplo.

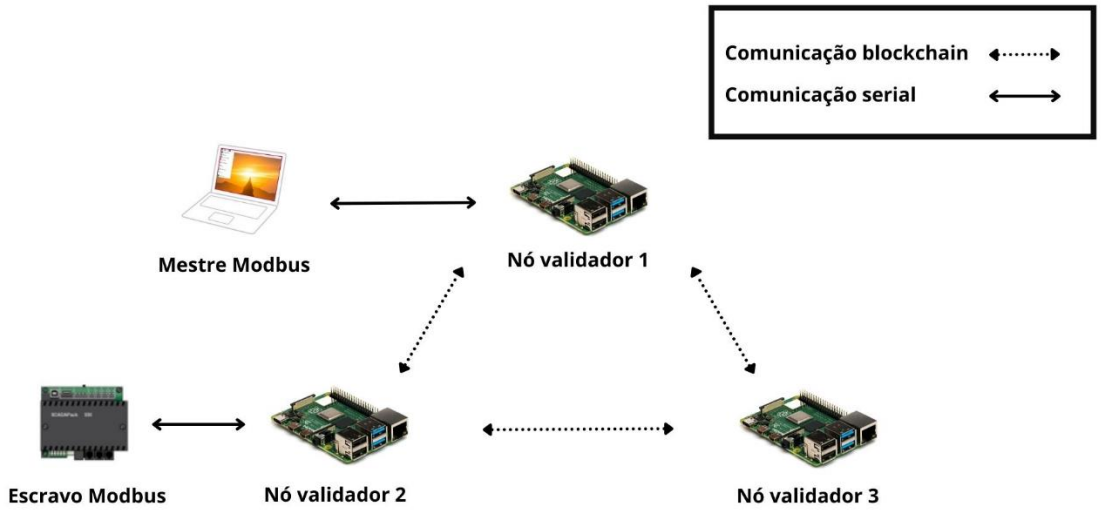
Figura 1 - Possível topologia.



Fonte: Autor.

A topologia utilizada neste trabalho está mostrada na Figura 2. Um mestre Modbus envia suas requisições para a blockchain através do nó 1, enquanto que o nó 2 lê essa requisição e a envia para o escravo Modbus. Por sua vez, o escravo Modbus envia uma resposta de volta para o nó 2 e essa resposta é colocada na blockchain para o nó 1 faça a leitura. Caso seja desejado, é possível compartilhar esse dado com outros dispositivos. Para a criação e comunicação de dispositivos Modbus, foi utilizado o software *IOServer*.

Figura 2 - Topologia do trabalho.



Fonte: Autor.

2.2 Criptografia

É uma área da criptologia que estuda técnicas para se ter uma comunicação segura (criptografada), através do uso de protocolos que modificam a mensagem original, impedindo que terceiros ou estranhos leiam essa mensagem (LEEUWEN, 1990). Ela pode ser dividida em dois tipos: Simétrica e assimétrica (FAULKNER, 2016).

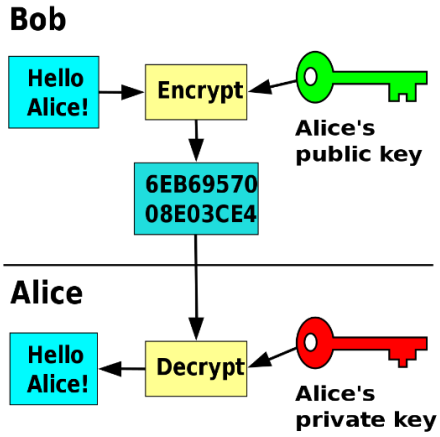
- **Criptografia Simétrica**

Requer que quem enviou a mensagem e quem a recebe conheça a chave utilizada para decodificar a mensagem. Este tipo não é muito seguro, pois ambas as partes precisam manter esta chave em segredo, em um meio que pode ou não ser confiável (FAULKNER, 2016).

- **Criptografia Assimétrica**

É caracterizada pelo uso de pares de chaves (pública e privada), usadas para autenticação e encriptação, onde cada chave privada corresponde a uma chave pública única. A Figura 3 mostra o envio de uma mensagem criptografada por Bob utilizando a chave pública de Alice. Para desencriptar a mensagem, é necessário usar a chave privada que está relacionada com a chave pública de Alice (chave privada de Alice). Não sendo necessário que Alice e Bob compartilhem a mesma chave, mantendo um nível maior de segurança entre as partes e o meio. Com isso, as chaves públicas devem ser de conhecimento de qualquer pessoa ou dispositivo na rede, mas suas chaves privadas não (WIKIPÉDIA).

Figura 3 - Criptografia Assimétrica.

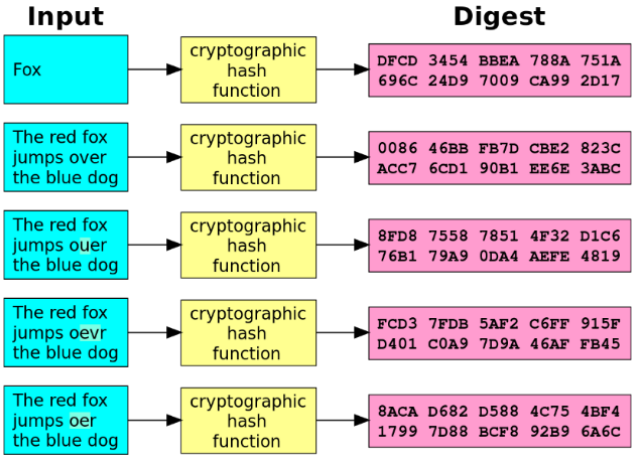


Fonte: Criptografia de chave pública (WIKIPÉDIA).

2.3 Função *hash* criptográfica

É um algoritmo que transforma dados com comprimento variável em dados criptografados de comprimento fixo, onde cada dado de entrada produz um único dado fixo de saída. A Figura 4 mostra um exemplo de função *hash* onde vários dados de entrada (*Input*) produzem *hashes* criptografados com tamanho fixo (*Digest*) (WIKIPÉDIA).

Figura 4 - Função Hash.



Fonte: Função Hash Criptográfica (WIKIPÉDIA).

2.4 Algoritmos de consenso

A segurança de uma rede blockchain não é garantida por autoridades centrais ou terceiros, e sim pelos algoritmos criptográficos utilizados nas gerações de blocos e pelo algoritmo de consenso utilizado para validar os blocos gerados.

Neste trabalho, foi utilizada a plataforma de código aberto de desenvolvimento *Hyperledger Besu* para a blockchain *Ethereum*. Esta plataforma foi escolhida por permitir o uso de algoritmos de consenso do tipo prova de autoridade (*proof of authority*) para validar transações e blocos e por utilizar a blockchain *Ethereum*, que permite o uso de contratos inteligentes. Ela oferece 3 algoritmos de consenso do tipo prova de autoridade: *IBFT 2.0*, *Clique* e *QBFT*. Esses algoritmos fazem com que nós validadores definidos como confiáveis (um nó validador blockchain dentro da indústria, por exemplo) fiquem comunicando entre si para se revezar na assinatura de blocos, definindo-os como válidos. Estes algoritmos também permitem votações entre os nós para adicionar ou remover outros nós e definição de prioridade de assinatura para nós prioritários.

Em sistemas de computação distribuídos, existe uma condição chamada de falta bizantina (*Byzantine Fault*), que é a ocorrência de erros ou de problemas em um ou mais nós validadores da rede. Para a blockchain operar de modo seguro, é necessário que ela seja tolerante a um máximo percentual de nós com falhas ou erros

que sejam minoria em relação a quantidade total de nós. Os algoritmos *IBFT 2.0* e *QBFT* toleram até um terço de nós com falhas, ou seja: se mais de um terço dos nós não estiverem funcionando ou apresentando falhas, a rede para de funcionar. Já o protocolo *Clique* é mais tolerante às falhas bizantinas, pois para funcionar corretamente, este algoritmo precisa que mais da metade dos nós de uma rede sejam válidos: se 50% ou mais dos nós apresentarem erro, a rede para de funcionar. Em compensação, conforme o número de validadores em uma rede aumenta, para os algoritmos *IBFT 2.0* e *QBFT* o tempo de adição de novos blocos na rede aumenta, enquanto que no algoritmo *Clique* a velocidade de criação de blocos praticamente não se altera (HYPERLEDGER FOUNDATION).

2.5 Assinatura digital

É uma forma de autenticação digital que visa substituir uma autenticação física (WIKIPÉDIA). A Assinatura digital garante a autenticidade de um documento ou mensagem digital utilizando uma função *hash*, gerando um *hash* que será criptografado com a chave privada do usuário, transformando-se assim em uma assinatura digital.

2.6 *Transport Layer Security* (protocolo TLS)

O objetivo principal do protocolo TLS 1.3 criado em 2018 é prover um canal seguro de comunicação entre dois pontos (cliente e servidor) (INTERNET ENGINEERING TASK FORCE, 2018). Ele atua na camada de transporte do modelo OSI, recebendo todo o tráfego da camada de sessão, dividindo-o em partes e criptografando essas partes, que são repassadas para a camada de rede. Com isto, o protocolo garante as seguintes propriedades:

- **Autenticação**: o servidor da comunicação estará sempre autenticado numa comunicação TLS, com autenticação opcional no cliente. A autenticação poderá ocorrer via criptografia assimétrica, simétrica de chaves pré-compartilhadas (*PSK*), algoritmo de assinatura digital de curva elíptica (ECDSA) e algoritmo de assinatura digital de curva-Edwards (EdDSA).
- **Confidencialidade**: os dados enviados através do canal após o estabelecimento da comunicação segura só podem ser vistos por quem envia ou recebe.
- **Integridade**: os dados não podem ser modificados sem que haja uma detecção, pois durante o início da comunicação o cliente e o servidor geram números aleatórios de 32 bytes que serão usados para gerar a chave criptográfica da comunicação. Caso um desses números seja modificado, a chave criptográfica gerada não será aceita em um dos lados, acabando com a comunicação.

O protocolo possui dois componentes primários:

- **Protocolo *Handshake* (aperto de mão)**: autentica as partes que estão se comunicando, negociando os parâmetros criptográficos utilizados entre as partes. O aperto de mão foi criado de forma que um agente malicioso não consegue forçar dois pontos a negociar diferentes parâmetros daqueles utilizados numa comunicação TLS normal.
- **Protocolo *Record* (gravação)**: usa os parâmetros estabelecidos pelo protocolo *Handshake* para proteger a comunicação entre as partes, dividindo o tráfego da comunicação em partes menores e criptografadas.

2.7 Contratos inteligentes

São definidos como um programa de computador que pode ser executado dentro da blockchain *Ethereum* (ETHEREUM FOUNDATION). Cada contrato

representa um endereço na blockchain, portanto, pode realizar transações e armazenar a criptomoeda *Ethereum*. Seu diferencial é que ele não é controlado por um usuário, e sim pela rede, que garante sua execução através da validação dos blocos que contêm transações deste contrato.

São programados pelo usuário através de linguagens de programação, tais como *Solidity* e *Vyper*. Contratos compilados fornecem parâmetros que permitem adicioná-los na blockchain (*ABI* e *bytecode*) através de transações, que também precisam ser validadas de acordo com o algoritmo de consenso utilizado. Cada transação do contrato pode executar ou ler uma função definida ao programá-lo sem modificar seu código, sendo necessário realizar transações para a escrita de valores.

Por padrão, a interação com contratos inteligentes é irreversível, pois não pode ser alterado por um usuário e não podem ser apagados.

2.8 *Gas*

É a unidade básica de esforço computacional para transações na blockchain *Ethereum* (ETHEREUM FOUNDATION). Como cada transação exige esforço computacional da blockchain, a taxa *Gas fee* simboliza o esforço necessário para que uma transação tenha sucesso.

3 Desenvolvimento

3.1 Configuração das máquinas virtuais

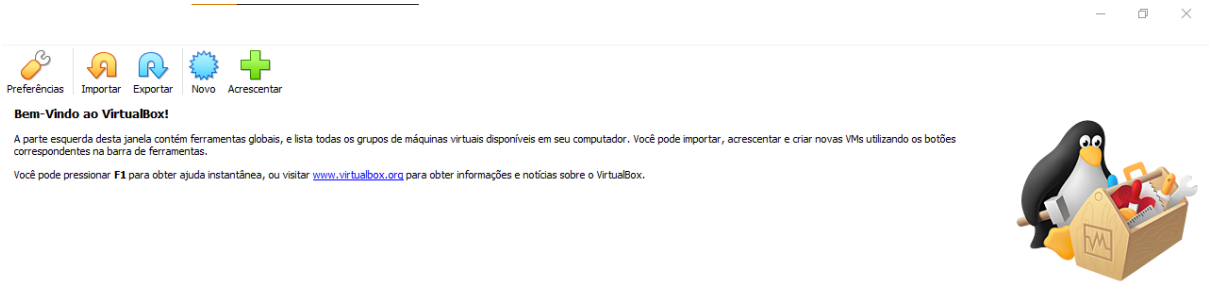
O primeiro passo é construir um conjunto de nós validadores que serão representados pelas máquinas virtuais, onde cada máquina virtual (VM) equivale a um mini computador *Raspberry Pi*, que possui poder de processamento semelhantes ao de computadores simples e modernos nas versões mais atuais.

Na topologia de comunicação apresentada, uma das três VM's receberá uma requisição de informação Modbus (Modbus RTU Mestre) para escrever na blockchain e outra VM irá ler essa requisição armazenada na blockchain para enviá-la a um dispositivo Modbus RTU escravo (equivalente a um dispositivo de campo na indústria) e armazenar a resposta desse dispositivo na blockchain, que ficará disponível para leitura para quaisquer usuários da rede blockchain.

Toda a topologia apresentada foi emulada em um único computador com um sistema operacional de 64 bits e que possui 16 GB de memória RAM. Cada VM foi criada para utilizar um máximo de 4 GB de memória RAM, que é um requisito mínimo para a execução da plataforma *Besu* (ela será executada em cada VM) (HYPERLEDGER FOUNDATION). Para criar uma máquina virtual, é necessário fazer o *download* da imagem de instalação do sistema operacional *Raspberry Pi OS*, para ser instalada em cada VM. As imagens estão disponíveis no site oficial do produto (RASPBERRY PI FOUNDATION). O software de virtualização utilizado para emular as VM's foi o *VirtualBox* (ORACLE). Com o *VirtualBox* instalado e a imagem *Raspberry Pi OS* baixada, é dado início ao processo de instalação da VM:

A Figura 5 mostra a tela inicial do software. Para criar uma instância, é necessário clicar no botão Novo.

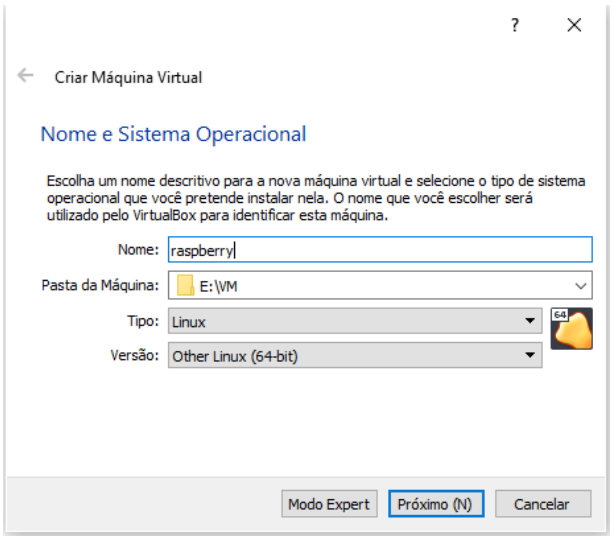
Figura 5 - Tela inicial VirtualBox.



Fonte: Autor.

Logo após, será mostrada a janela da Figura 6, cujos parâmetros da do sistema operacional já estão definidos.

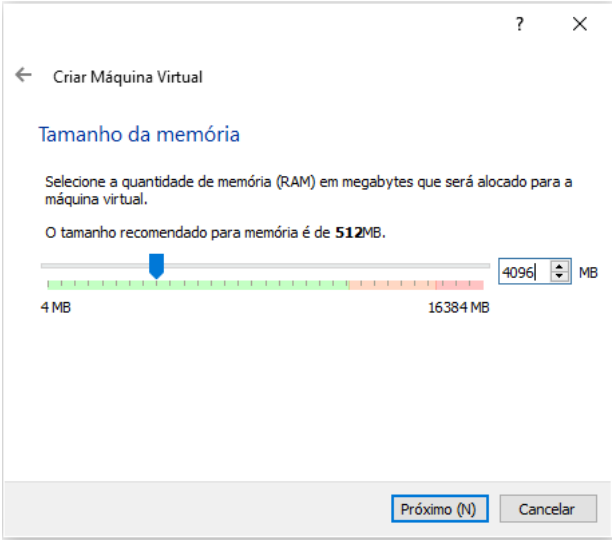
Figura 6 - Janela de criação.



Fonte: Autor.

O próximo passo especifica a quantidade de memória RAM que será utilizada pela VM (4096 MB é o tamanho mínimo recomendado para a plataforma *Besu*), como mostra a Figura 7.

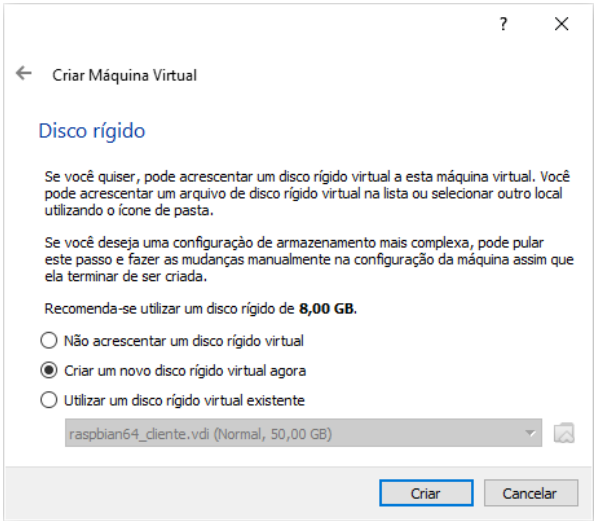
Figura 7 - Memória RAM da VM.



Fonte: Autor.

Por fim, é necessário criar o disco virtual (Figura 8) que será utilizado pela VM como armazenamento. Para este trabalho, cada VM utilizou um espaço de 50 GB (mais que o necessário) de um *SSD* instalado no computador hospedeiro. Após estes passos, a VM pode ser inicializada para que a imagem seja instalada.

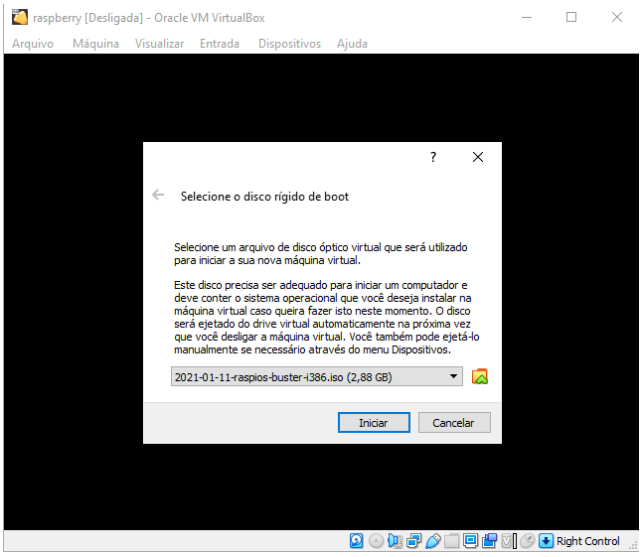
Figura 8 - Criação do disco virtual.



Fonte: Autor.

Ao inicializar a VM, aparecerá a janela da Figura 9, onde é necessário selecionar a imagem baixada do sistema operacional.

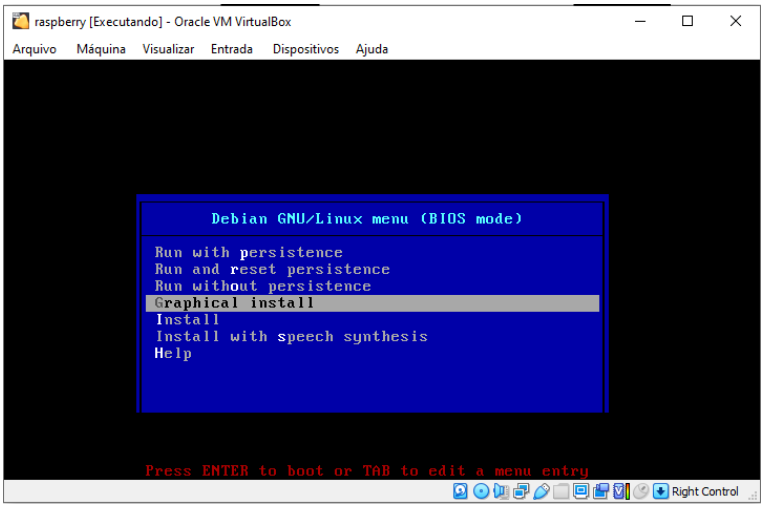
Figura 9 - Inicialização da VM.



Fonte: Autor.

Com a imagem baixada selecionada, a tela da Figura 10 aparecerá para que seja realizada a instalação da imagem, através da opção *Graphical Install*.

Figura 10 - Instalação da imagem.

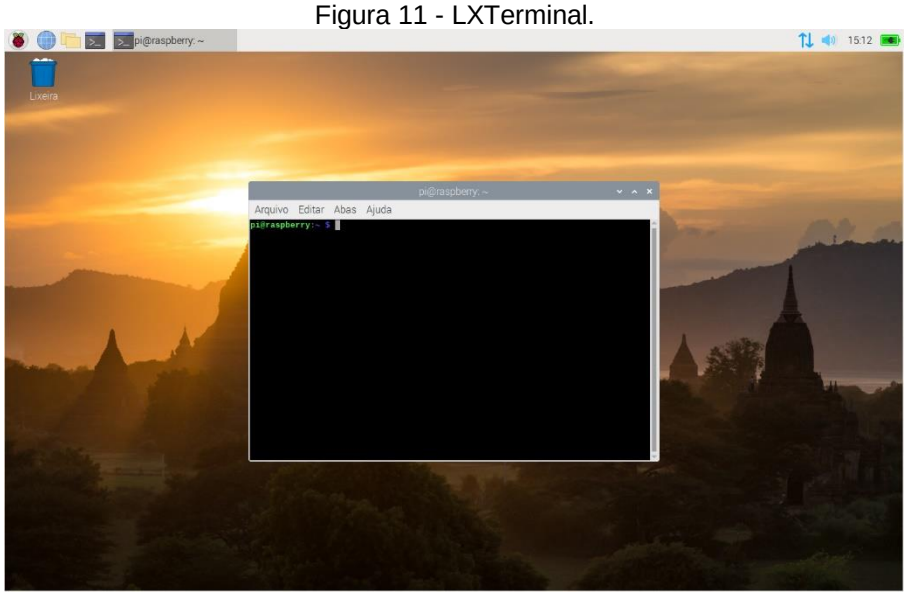


Fonte: Autor.

Os passos da instalação gráfica não serão mostrados aqui, pois a instalação é feita de maneira bastante intuitiva pelo usuário. Após a configuração, é necessário utilizar a placa de rede de cada máquina em modo *Bridge* com a placa de rede do computador hospedeiro, para garantir que cada máquina virtual possua um IP fixo na mesma rede do computador hospedeiro. Isto é necessário para garantir que as VM's possam comunicar entre si. O IP fixo de cada máquina foi definido na página de configuração do roteador utilizado para conectar o computador com a internet. O processo de configuração da máquina virtual deve ser feito para cada uma das três VM's.

Com as máquinas virtuais inicializadas, agora é necessário instalar os softwares que serão utilizados para executar a blockchain e se comunicar com os dispositivos Modbus. Antes de começar a instalar os softwares, é necessário atualizar o sistema operacional e alguns softwares e recursos que já estão previamente instalados na imagem *Raspberry Pi OS*. Para executar cada comando, é necessário utilizar o terminal *LXTerminal*, já instalado no sistema operacional (Figura 11).

Figura 11 - LXTerminal.



Fonte: Autor.

- O primeiro comando utilizado é *sudo apt-get update --allow-releaseinfo-change*, responsável por sincronizar os arquivos indexadores de pacotes de seus

respectivos fornecedores com alterações de mudanças de repositórios (SOFTWARE IN THE PUBLIC INTEREST, INC.).

- O segundo é o comando *sudo apt full-upgrade*, que executará uma atualização completa do sistema com base nos arquivos sincronizados de pacotes.

Os dois próximos comandos são necessários para que seja possível instalar o software NVM.

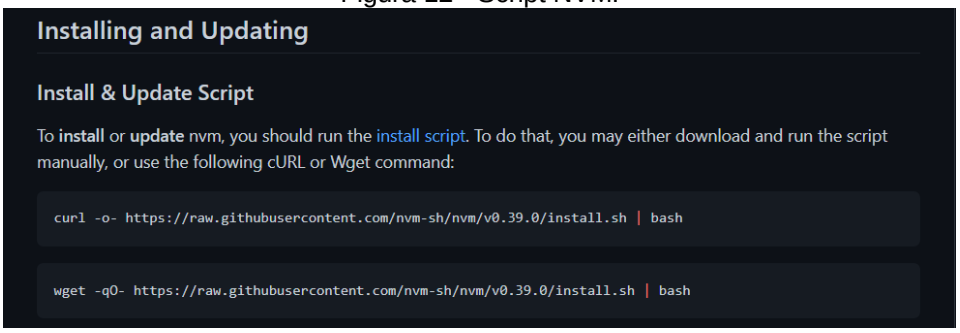
- O terceiro comando é *sudo apt-get install lib32z1*, que instala bibliotecas de 32 *bits* para que o *Node.js* seja compatível com a arquitetura de 64 *bits* utilizada no sistema operacional.
- O quarto comando é *sudo apt-get install g++-multilib* que instala o compilador *GNU C++*, também necessário para a compatibilidade do *Node.js* com a arquitetura.

Com os quatro comandos executados, já é possível instalar os softwares desejados:

- O primeiro software a ser instalado é o da distribuição binária da plataforma *Besu* (HYPERLEDGER FOUNDATION), que deve ser baixado do site do desenvolvedor e instalado na pasta de binários do sistema operacional. Ele é responsável por permitir a execução da blockchain criada pelo usuário em linha de comando (*LXTerminal*), como mostra o comando da Figura 15.
- O segundo a ser instalado é o *Ethsigner*, responsável por assinar as transações na blockchain, através de chaves privadas pré-cadastradas no aplicativo. É necessário baixar a distribuição binária disponível no site do desenvolvedor e instalá-la no mesmo diretório da blockchain (CONSENSYS).
- O terceiro a ser instalado é o *Node Version Manager* (NVM), que é um pré-requisito para a instalação da ferramenta de desenvolvimento baseada em fluxo *Node-RED*. Para instalá-lo é necessário executar um dos dois *scripts* (Figura

12) no terminal do sistema operacional. Estes *scripts* são encontrados em sua página do *Github* (OPENJS FOUNDATION).

Figura 12 - Script NVM.

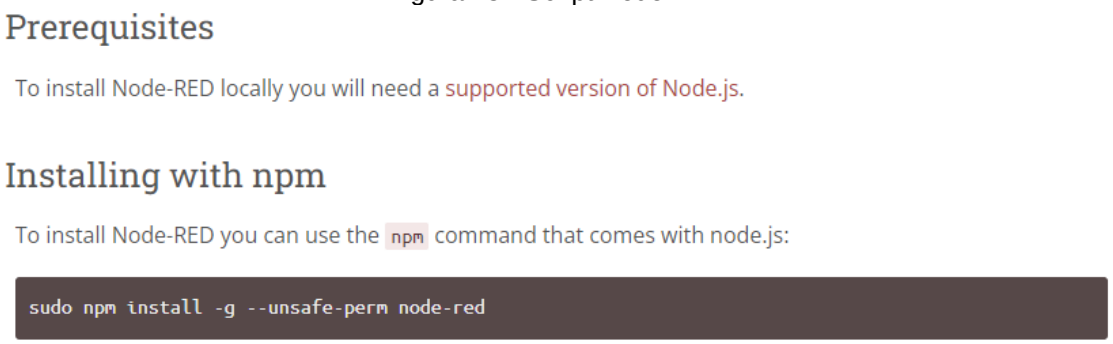


Fonte: Node Version Manager (OPENJS FOUNDATION).

Após a instalação do NVM, é necessário abrir um novo terminal e executar o comando *nvm install node* que instalará a versão mais recente do *Node.js*, um software de código aberto que permite a execução de códigos na linguagem *Javascript* fora de um navegador web.

- O quarto a ser instalado é o *Node-RED*, que servirá como uma ponte entre a blockchain e os dispositivos Modbus. Para instalá-lo, é necessário executar o *script* da Figura 13, que está disponível no site do desenvolvedor (OPENJS FOUNDATION).

Figura 13 - Script Node-RED.



Fonte: Running Node-RED locally (OPENJS FOUNDATION).

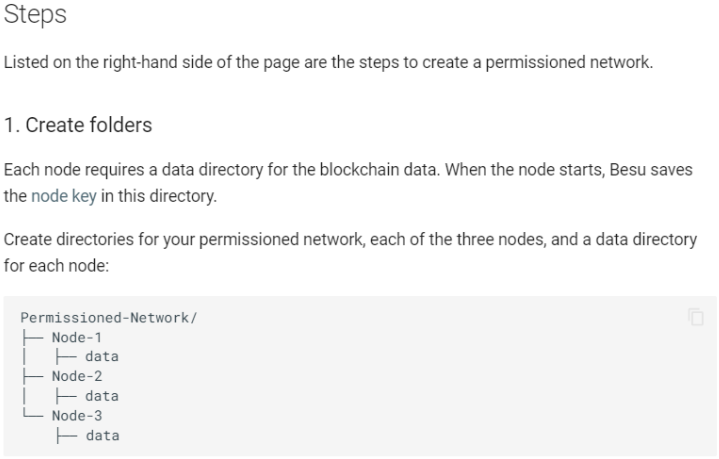
3.2 Programação da blockchain

O modelo utilizado neste trabalho utiliza o exemplo de rede permissionada do desenvolvedor, que utiliza o protocolo *Clique* (HYPERLEDGER FOUNDATION). Devido à sua alta velocidade em comparação com os protocolos mostrados anteriormente, ele foi escolhido como algoritmo de consenso utilizado na blockchain deste trabalho, pois é necessário que haja rapidez no envio das requisições e na leitura dos dados da rede (SZILÁGYI, 2017). Este protocolo requer que exista, analogamente a uma rede com mineração, pelo menos um dos nós validadores como assinante do primeiro bloco da blockchain (bloco gênese). Neste trabalho foram utilizados três assinantes iniciais e a criação de uma rede permissionada é dividida nas seguintes 10 partes:

3.2.1 Parte 1

A primeira parte envolve a criação de pastas e subpastas referentes a cada nó validador, como mostra a Figura 14. Para este trabalho, as pastas foram criadas dentro da VM correspondente: a VM 1 contém apenas as pastas e subpastas do nó 1, a VM 2 contém apenas as pastas e subpastas do nó 2, etc.

Figura 14 – Criação de pastas.



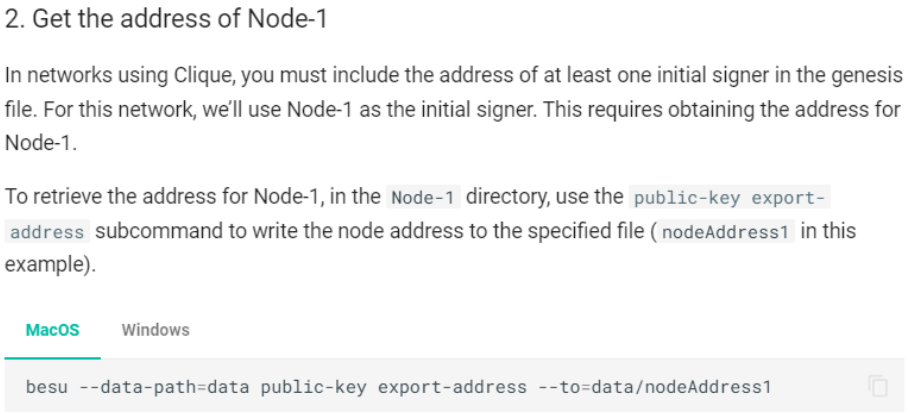
Fonte: Create a permissioned network – Hyperledger Besu (HYPERLEDGER FOUNDATION).

3.2.2 Parte 2

A parte 2 se refere à criação do endereço dos assinantes iniciais, que estão presentes no bloco gênese. O exemplo do desenvolvedor faz o uso de apenas um assinante inicial. O bloco gênese utilizado neste trabalho utiliza três assinantes iniciais (a instância blockchain de cada VM se comporta como um assinante). Caso metade ou mais dos assinantes de blocos apresentem falhas (neste trabalho, dois ou mais blocos), a rede deixará de funcionar.

Para obter o endereço de cada nó, é necessário executar o comando mostrado na Figura 15 no terminal de cada uma das VM's. Este comando irá gerar um endereço aleatório e salvar dentro de um arquivo de nome *nodeAddress1,* mostrado no subcomando *--to=data/nodeAddress1*, que aparece como exemplo para o nó 1. O endereço gerado nesta etapa será utilizado no arquivo do bloco gênese.

Figura 15 – Comando de exportação de endereços.



Fonte: Create a permissioned network – Hyperledger Besu (HYPERLEDGER FOUNDATION).

3.2.3 Parte 3

O arquivo gênese define os parâmetros do bloco gênese utilizado pela blockchain. A Figura 16 mostra o código utilizado para o arquivo gênese do exemplo.

Figura 16 – Código exemplo do arquivo gênese.


```
{
  "config":{
    "chainId":1981,
    "constantinoplefixblock": 0,
    "clique":{
      "blockperiodseconds":15,
      "epochlength":30000
    }
  },
  "coinbase":"0x0000000000000000000000000000000000000000",
  "difficulty":"0x1",

"extraData":"0x0000000000000000000000000000000000000000000000000000000000000000<Node 1
Address>0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000",
  "gasLimit":"0xa00000",
  "mixHash":"0x0000000000000000000000000000000000000000000000000000000000000000",
  "nonce":"0x0",
  "timestamp":"0x5c51a607",
  "alloc": {
      "fe3b557e8fb62b89f4916b721be55ceb828dbd73": {
        "privateKey": "8f2a55949038a9610f50fb23b5883af3b4ecb3c3bb792cbcefbd1542c692be63",
        "comment": "private key and this comment are ignored.  In a real chain, the private key should NOT be stored",
        "balance": "0xad78ebc5ac6200000"
      },
      "627306090abaB3A6e1400e9345bC60c78a8BEf57": {
        "privateKey": "c87509a1c067bbde78beb793e6fa76530b6382a4c0241e5e4a9ec0a0f44dc0d3",
        "comment": "private key and this comment are ignored.  In a real chain, the private key should NOT be stored",
        "balance": "90000000000000000000000"
      },
      "f17f52151EbEF6C7334FAD080c5704D77216b732": {
        "privateKey": "ae6ae8e5ccbfb04590405997ee2d52d2b330726137b875053c36d94e974d162f",
        "comment": "private key and this comment are ignored.  In a real chain, the private key should NOT be stored",
        "balance": "90000000000000000000000"
      }
   },
  "number":"0x0",
  "gasUsed":"0x0",
  "parentHash":"0x0000000000000000000000000000000000000000000000000000000000000000"
}
```


Fonte: Autor.

Os parâmetros utilizados no código estão definidos na página do protocolo *Clique* (SZILÁGYI, 2017) e no site da plataforma *Besu* (HYPERLEDGER FOUNDATION), e são responsáveis por definir o comportamento da rede blockchain, dentre os quais:

- ***ChainId***: identificador da rede no processo de assinatura de transações. Caso o parâmetro *networkId* (identificador da comunicação ponto a ponto entre os nós) não seja especificado, ele será idêntico a este parâmetro.
- ***constantinopleFixBlock***: em redes privadas, este parâmetro fornece a versão do protocolo da rede blockchain.
- ***Clique***: Algoritmo de consenso utilizado. Este parâmetro possui dois subparâmetros (*blockperiodseconds* e *epochlength*).
- ***blockperiodseconds***: intervalo de tempo da criação de blocos.
- ***epochlength***: número de blocos necessários para resetar votos pendentes dos assinantes.

- ***coinbase***: endereço para pagar recompensas de mineração (não utilizado neste trabalho).
- ***difficulty***: quando usado com algoritmos de prova de autoridade, este parâmetro refere-se à qualidade da blockchain (blocos com qualidade 2 possuem prioridade em relação a blocos de qualidade 1, para serem gerados por assinantes específicos).
- ***extraData***: parâmetro referente aos endereços concatenados dos assinantes iniciais. Para este trabalho, foram utilizados três assinantes.
- ***gasLimit***: taxa máxima de *Gas* que cada transação pode utilizar.
- ***mixHash***: parâmetro reservado.
- ***nonce***: parâmetro utilizado para autorizar ou não novos assinantes na rede. Deve ser deixado em 0 no bloco gênese.
- ***timestamp***: estampa de tempo de criação do primeiro bloco, definida em *unix time*.
- ***alloc***: seção que define contas e seus saldos ou contratos inteligentes pré-criados.
- ***number***: número do bloco gênese da rede.
- ***gasUsed***: quantidade de *Gas* utilizado para criação do bloco gênese.
- ***parentHash***: identificador do bloco que veio antes do bloco gênese (valor em 0 para indicar que não houve nenhum).

Para este trabalho, foi utilizado o código de bloco gênese da Figura 17.

Figura 17 - Código do arquivo gênese utilizado.

```
{
  "config":{
    "chainId":1981,
    "constantinoplefixblock": 0,
    "clique":{
      "blockperiodseconds":1,
      "epochlength":30000
    }
  },
  "coinbase":"0x0000000000000000000000000000000000000000",
  "difficulty":"0x1",
"extraData":"0x00000000000000000000000000000000000000000000000000000000000000003832f355d57bceb096a9c0f1999a37e7727fba06397d632b41a499e884cb1690a4828
86e35021b65c136889c1e727af474d0d815e328270a65022b010000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000",
  "gasLimit":"0xa00000",
  "mixHash":"0x0000000000000000000000000000000000000000000000000000000000000000",
  "nonce":"0x0",
  "timestamp":"0x5c51a607",
  "alloc": {
      "fe3b557e8fb62b89f4916b721be55ceb828dbd73": {
        "privateKey": "8f2a55949038a9610f50fb23b5883af3b4ecb3c3bb792cbcefbd1542c692be63",
        "comment": "private key and this comment are ignored.  In a real chain, the private key should NOT be stored",
        "balance": "0xad78ebc5ac6200000"
      },
      "627306090abaB3A6e1400e9345bC60c78a8BEf57": {
        "privateKey": "c87509a1c067bbde78beb793e6fa76530b6382a4c0241e5e4a9ec0a0f44dc0d3",
        "comment": "private key and this comment are ignored.  In a real chain, the private key should NOT be stored",
        "balance": "90000000000000000000000"
      },
      "f17f52151EbEF6C7334FAD080c5704D77216b732": {
        "privateKey": "ae6ae8e5ccbfb04590405997ee2d52d2b330726137b875053c36d94e974d162f",
        "comment": "private key and this comment are ignored.  In a real chain, the private key should NOT be stored",
        "balance": "90000000000000000000000"
      }
   },
  "number":"0x0",
  "gasUsed":"0x0",
  "parentHash":"0x0000000000000000000000000000000000000000000000000000000000000000"
}
```

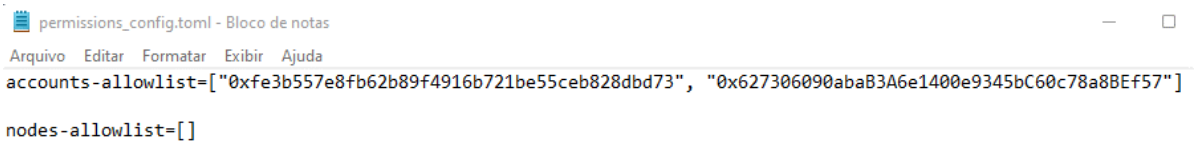
Fonte: Autor.

Como é citado no próprio arquivo gênese, em uma blockchain real não se deve armazenar as chaves privadas no arquivo do bloco gênese, que é utilizado neste código apenas para testes.

3.2.4 Parte 4

Esta etapa define a criação de um arquivo de configuração de permissões de nome *permissions_config.toml*, com o código da Figura 18, que aumenta a segurança da rede blockchain ao definir quais contas poderão realizar transações e quais nós validadores poderão sincronizar com a rede. O arquivo de permissões deve estar presente dentro da pasta de cada nó (dentro de cada VM). Este arquivo será modificado pela rede na etapa de inicialização e é utilizado apenas para armazenar dados, não sendo executado em nenhum terminal.

Figura 18 - Criação do arquivo de permissões.

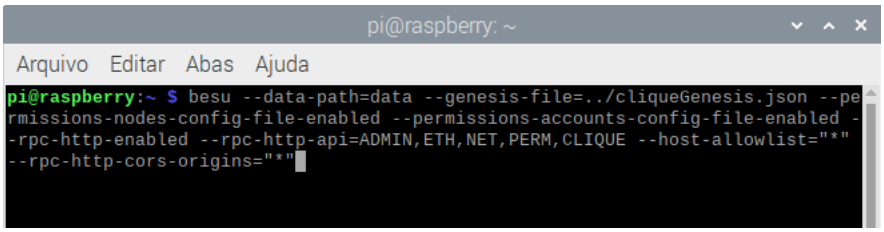


Fonte: Autor.

3.2.5 Parte 5

A Figura 19 mostra a primeira etapa de inicialização da rede, que se refere ao comando de inicialização do nó 1. Ao executar o comando em um terminal, será gerada uma URL identificadora do nó na rede. É necessário copiar essa URL para depois adicioná-la ao arquivo de permissões. Além dos parâmetros especificados é necessário modificar outros como o preço mínimo de *Gas*, que precisa ser 0 e alterar parâmetros de rede referentes aos IP's e portas http de cada nó. A porta http 8545 utilizada nesta inicialização será alterada para 8590, para poder funcionar em conjunto com o aplicativo *Ethsigner*.

Figura 19 - Exemplo de comando de inicialização do nó 1.



Fonte: Autor.

Este comando irá gerar o resultado da Figura 20:

Figura 20 – URL gerada do nó 1.

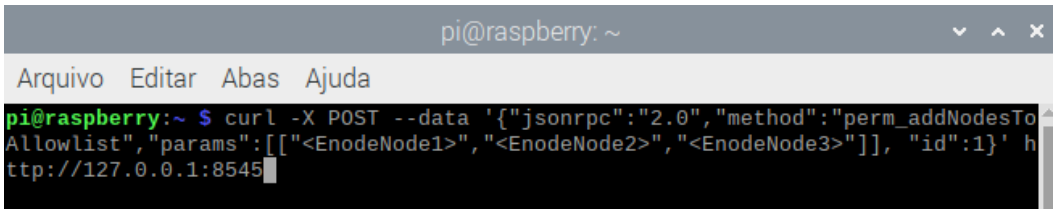
Fonte: Create a permissioned network – Hyperledger Besu (HYPERLEDGER FOUNDATION).

As partes 6 e 7 do tutorial apenas repetem os passos da Parte 5 para os nós 2 e 3.

3.2.6 Parte 8

Utilizando as URL's geradas, é necessário abrir um novo terminal e executar o comando da Figura 21 em cada VM. Este comando modificará o arquivo de permissões criado na Parte 4, e deve retornar sucesso ao modificar o arquivo de permissões de cada nó para que eles possam ser reconhecidos entre si. Ele deverá ser executado em cada um dos nós.

Figura 21 - Modificação do arquivo de permissões.



Fonte: Autor.

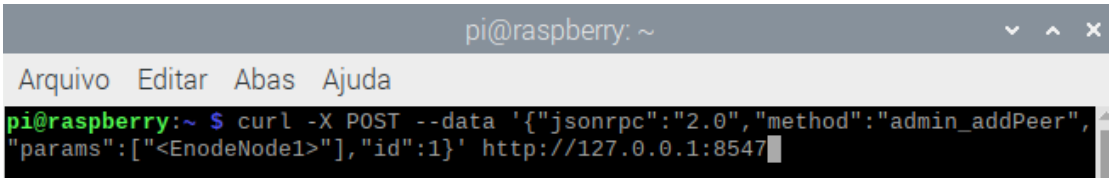
3.2.7 Parte 9

Com a blockchain em execução, agora é necessário sincronizar os nós permitidos nos arquivos de configurações para que eles possam se comunicar e começar a gerar blocos. Esta etapa precisa ser realizada toda vez que a blockchain for inicializada, caso ela pare por qualquer motivo. Os comandos dessa etapa (Figura 22, Figura 23 e Figura 24) foram automatizados no software *Node-RED* e serão mostrados na execução final da blockchain.

Figura 22 - Sincronização do nó 1 como ponto (*peer*) para o nó 2.

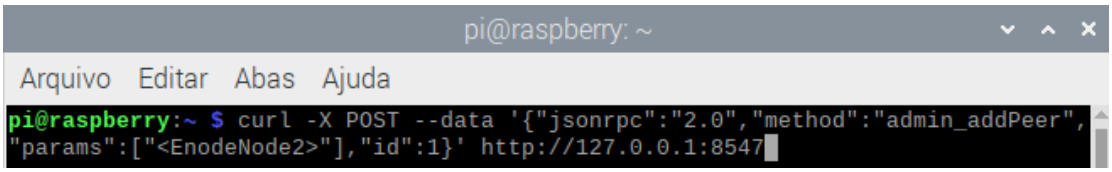
Fonte: Autor.

Figura 23 - Sincronização do nó 1 como ponto (*peer*) para o nó 3.



Fonte: Autor.

Figura 24 - Sincronização do nó 2 como ponto (*peer*) para o nó 3.



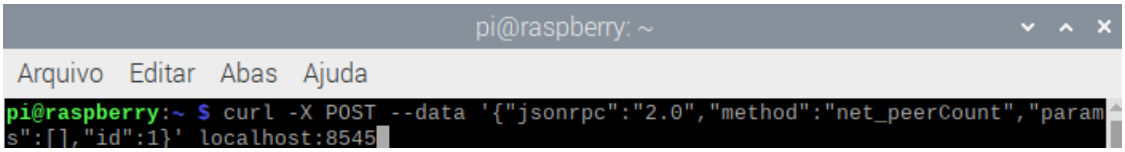
Fonte: Autor.

Cada um desses comandos retornará uma resposta do mesmo tipo da Figura 47.

3.2.8 Parte 10

A última parte permite verificar a funcionalidade da blockchain. O comando da Figura 25 pode ser executado em um terminal e retornará os dados da Figura 26, caso esteja funcionando corretamente.

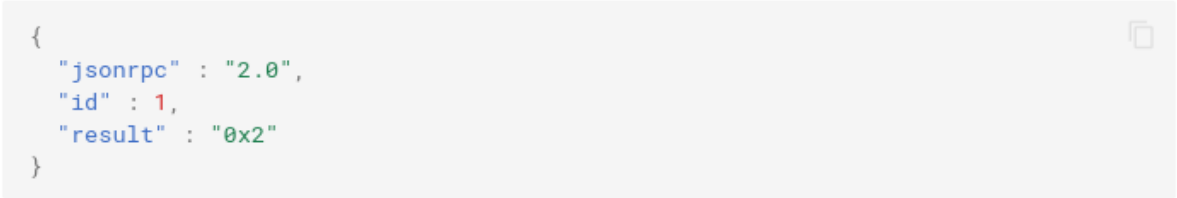
Figura 25 - Funcionamento da blockchain.



Fonte: Autor.

Figura 26 - Resultado positivo de checagem da rede.

The result confirms Node-1 (the node running the JSON-RPC service) has two peers (Node-2 and Node-3):



Fonte: Autor.

3.3 Programação do aplicativo *EthSigner*

Para poder assinar as transações realizadas entre contas e contratos, é necessário utilizar o aplicativo *Ethsigner* em paralelo com a blockchain. Este aplicativo separa o gerenciamento de chaves privadas (feito pelo aplicativo) da validação de transações (feita pela blockchain), utilizando uma chave privada que pode ser armazenada na nuvem ou criptografada localmente. O aplicativo foi instalado na VM correspondente ao nó 1.

O passo a passo utilizado sofreu algumas modificações em relação ao original, que pode ser encontrado no site do desenvolvedor (CONSENSYS).

3.3.1 Parâmetro da plataforma *Besu*

A Figura 27 mostra um exemplo de comando para inicializar a blockchain em um nó, que fornecerá um resultado semelhante ao da Figura 20. A única alteração feita no comando de início de cada nó neste trabalho é a mudança da porta http 8545

para 8590, pois a porta 8545 agora será utilizada por este aplicativo para realizar a comunicação entre a blockchain e o *Node-RED* (fará requisições de leitura e escrita de contrato).

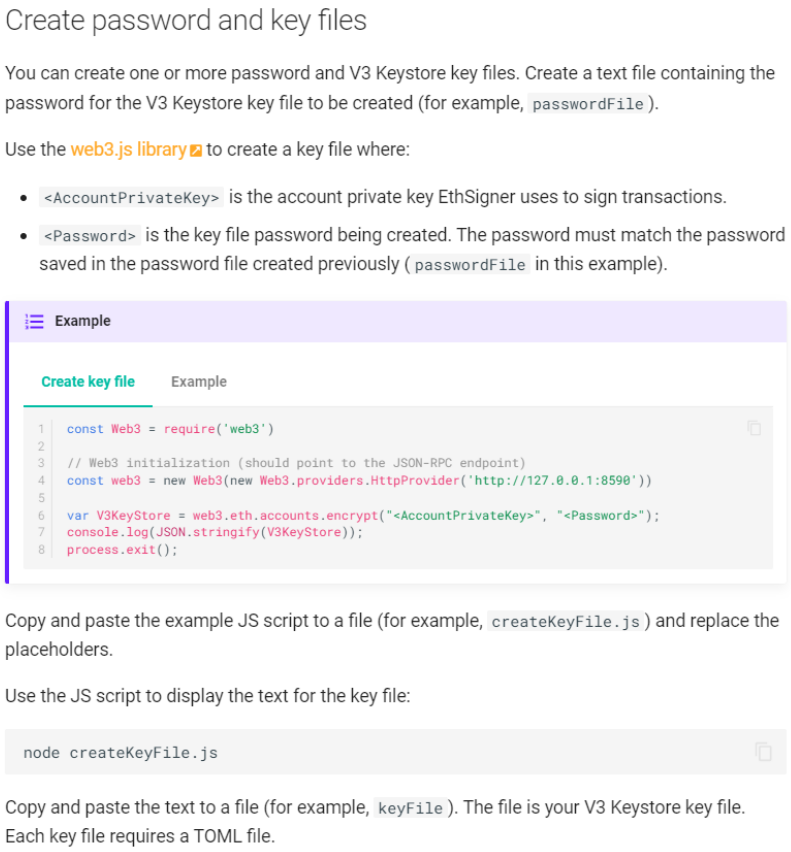
Figura 27 - Comando de inicialização Besu.

Fonte: Autor.

3.3.2 Criar senhas e arquivos de chaves

A Figura 28 mostra um script utilizado para criar um arquivo chave, que especificará quais contas poderão interagir com o aplicativo. Para criar o script, é preciso fornecer os parâmetros de chave privada e senha. Para executá-lo, basta criar um arquivo com extensão *js*, copiar o script no arquivo, substituir os parâmetros desejados e salvá-lo. Depois, abre-se um terminal no diretório do arquivo para executar o comando *node nome_do_arquivo.js*, onde *nome_do_arquivo.js* corresponde ao arquivo criado. Este comando criará um arquivo chave protegido por senha que será usado pelo aplicativo para validar as transações da conta presente no arquivo chave. Para este trabalho foram usadas apenas duas contas com permissão (dois arquivos chaves).

Figura 28 - Criação de senhas e arquivos de chaves.



Create password and key files

You can create one or more password and V3 Keystore key files. Create a text file containing the password for the V3 Keystore key file to be created (for example, `passwordFile`).

Use the web3.js library to create a key file where:

- `<AccountPrivateKey>` is the account private key EthSigner uses to sign transactions.
- `<Password>` is the key file password being created. The password must match the password saved in the password file created previously (`passwordFile` in this example).

Example

Create key file | Example

```
const Web3 = require('web3')

// Web3 initialization (should point to the JSON-RPC endpoint)
const web3 = new Web3(new Web3.providers.HttpProvider('http://127.0.0.1:8590'))

var V3KeyStore = web3.eth.accounts.encrypt("<AccountPrivateKey>", "<Password>");
console.log(JSON.stringify(V3KeyStore));
process.exit();
```

Copy and paste the example JS script to a file (for example, `createKeyFile.js`) and replace the placeholders.

Use the JS script to display the text for the key file:

```
node createKeyFile.js
```

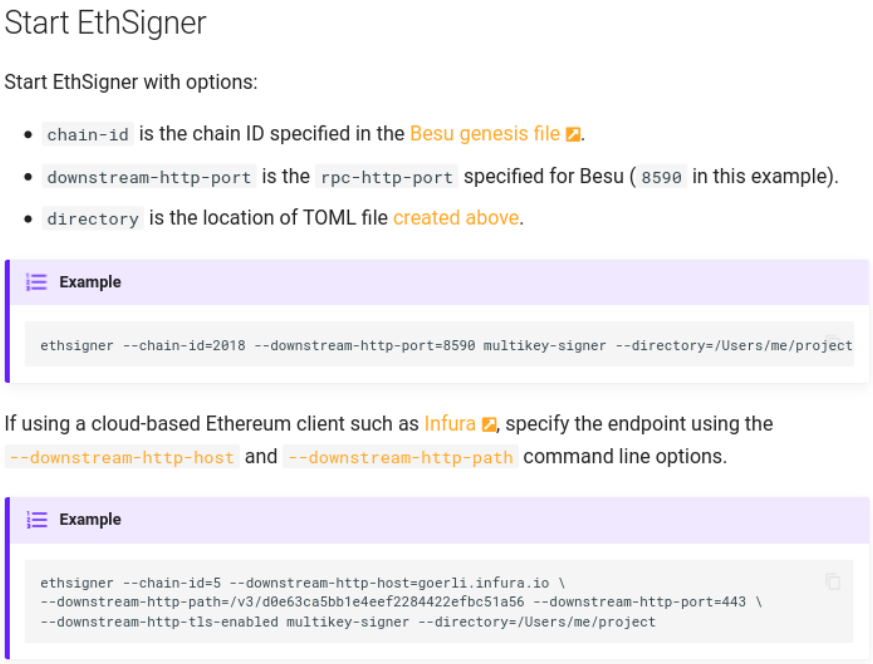
Copy and paste the text to a file (for example, `keyFile`). The file is your V3 Keystore key file. Each key file requires a TOML file.

Fonte: Start EthSigner with multiple signers - EthSigner (CONSENSYS).

3.3.3 Inicializar o *Ethsigner*

Com os arquivos chaves de contas configurados, basta apenas inicializar o *Ethsigner*, através do comando mostrado na Figura 29 com os parâmetros modificados para serem compatíveis com os parâmetros definidos na Programação da blockchain.

Figura 29 - Inicializar o Ethsigner.

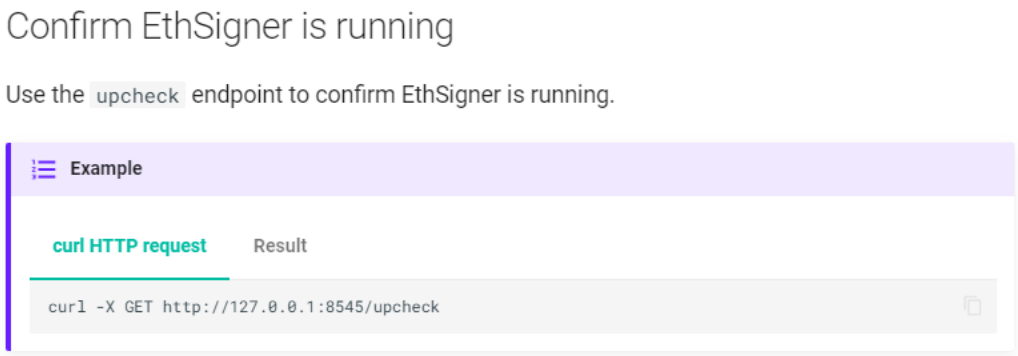


Fonte: Start EthSigner with multiple signers - EthSigner (CONSENSYS).

3.3.4 Checando o funcionamento

Para ter certeza que o aplicativo está funcionando normalmente, é preciso executar o comando da Figura 30, que retornará a frase *I'm up* caso não haja erros.

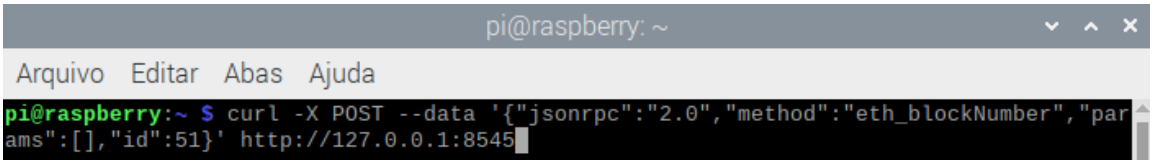
Figura 30 - Verificação de funcionamento Ethsigner.



Fonte: Start EthSigner with multiple signers - EthSigner (CONSENSYS).

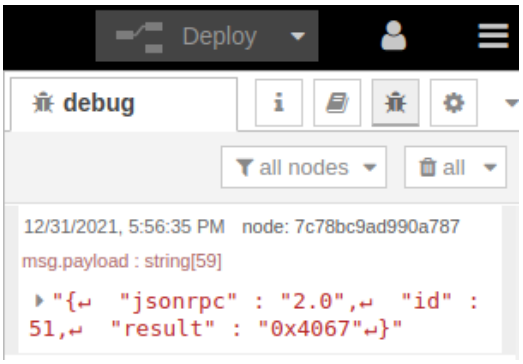
Para confirmar que o aplicativo está passando suas requisições para a blockchain, pode-se utilizar o comando mostrado na Figura 31 em um terminal, que retornará o número hexadecimal do último bloco gerado pela blockchain, mostrado na Figura 32.

Figura 31 - Confirmação de requisições.



Fonte: Autor.

Figura 32 - Número do último bloco produzido.



Fonte: Autor.

3.4 Autenticação via protocolo TLS

Nas configurações feitas até esta etapa, já é possível utilizar a blockchain para criar contratos inteligentes ou ler e escrever valores nestes contratos. Contudo, ainda é possível incrementar a segurança da rede através de protocolo TLS sobre comunicação http, também conhecido como protocolo https. Tanto a plataforma *Besu* quanto o aplicativo *Ethsigner* possuem configurações próprias de comunicação via TLS, bem como arquivos próprios para armazenamento de certificados.

Para realizar a configuração TLS, é necessário a existência de certificados assinados por uma autoridade certificadora reconhecida tanto pelo cliente (na

comunicação *Besu* - *Ethsigner*, *Ethsigner* é o cliente e *Besu* é o servidor. Já na comunicação *Ethsigner* - *Node-RED*, o *Node-RED* será o cliente e *Ethsigner* será servidor) quanto pelo servidor.

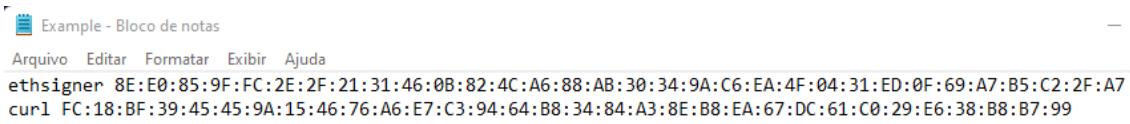
Neste trabalho será utilizado um certificado para a blockchain, um para o *Ethsigner* e outro para o *Node-RED*. Os certificados utilizados foram assinados localmente para fins de testes e não por uma autoridade certificadora, sendo necessário modificar alguns parâmetros da plataforma *Besu* e do *Ethsigner*.

O passo a passo da criação de certificados assinados localmente e a obtenção de seus parâmetros (SHA-256 *fingerprint*, *common name*, *etc.*) não serão mostrados neste trabalho.

3.4.1 Configuração TLS na plataforma *Besu*

A lista de clientes que farão requisições para a blockchain precisa conter o nome comum (*common name*) e a impressão digital SHA-256 (SHA-256 *fingerprint*) dos certificados dos clientes que farão requisições para a blockchain. Neste trabalho, o *Ethsigner* será o único cliente a se comunicar com a blockchain. A Figura 33 mostra um exemplo de lista de clientes conhecidos, composta do nome comum do certificado do cliente seguido de sua impressão digital SHA-256. A configuração TLS da plataforma é mostrada na página do desenvolvedor (HYPERLEDGER FOUNDATION).

Figura 33 - Arquivo de clientes conhecidos (Besu).

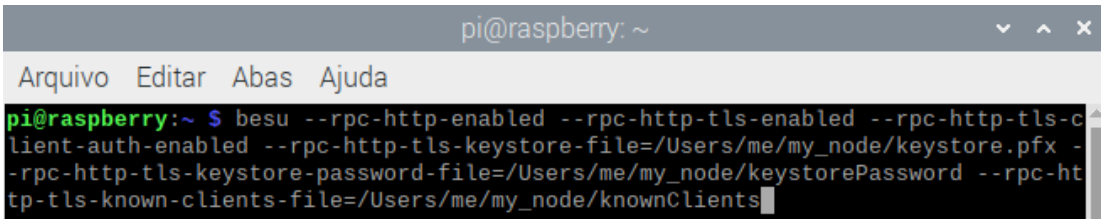


Fonte: Autor.

Após criar o arquivo de clientes conhecidos, é necessário incluí-lo como parâmetro na inicialização da blockchain, assim como o arquivo de armazenamento de certificados, como mostra a Figura 34. Ela também mostra um comando utilizado

para permitir apenas certificados assinados por autoridades confiáveis, que está desabilitado neste trabalho. O resultado deste comando é a inicialização de um nó, que já foi mostrado na Figura 20.

Figura 34 - Parâmetros TLS na blockchain.



Fonte: Autor.

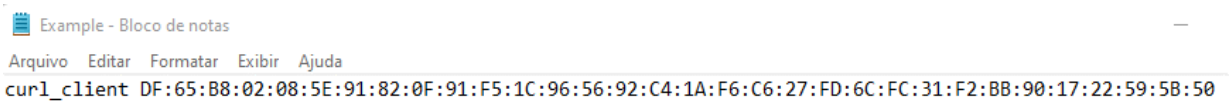
3.4.2 Configuração TLS no aplicativo *Ethsigner*

A configuração TLS deste aplicativo é mostrada no site do desenvolvedor (CONSENSYS).

3.4.2.1 Cliente

Para o *Ethsigner*, apenas o *Node-RED* será visto como cliente, e deverá ter o nome comum de seu certificado e sua impressão digital SHA-256 colocados no arquivo de clientes conhecidos do *Ethsigner*, como mostra o exemplo da Figura 35.

Figura 35 – Arquivo de clientes conhecidos (Ethsigner).

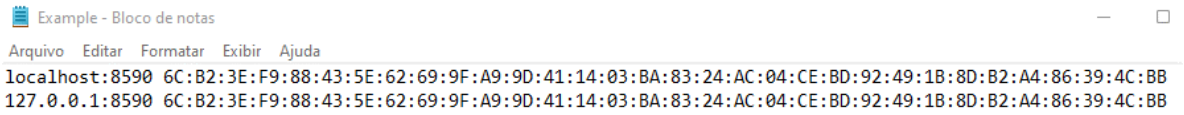


Fonte: Autor.

3.4.2.2 Servidor

Apenas a blockchain (*Besu*) será vista como servidor em relação ao *Ethsigner*, que também deve ter seus dados colocados em um arquivo semelhante ao de clientes conhecidos. Com a diferença de que este arquivo deverá conter o IP e a porta http utilizados pela instância blockchain onde o nó que comunica com o aplicativo está localizado (neste trabalho é o nó 1), substituindo o parâmetro nome comum.

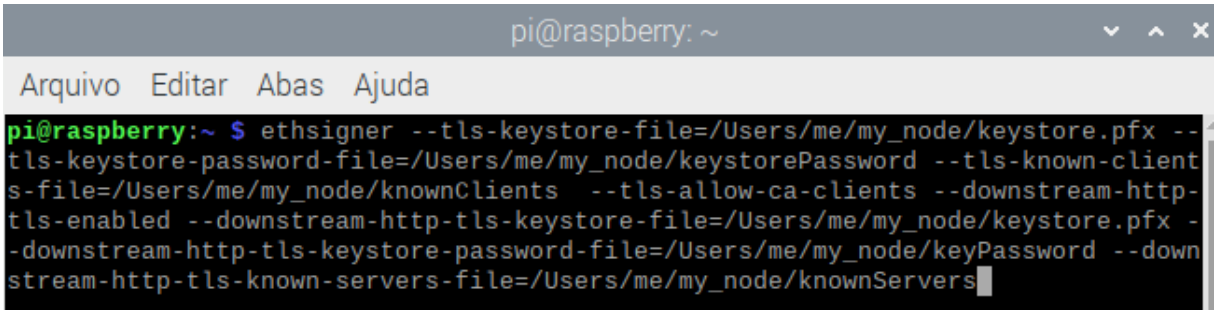
Figura 36 - Arquivo de servidores conhecidos (Ethsigner).



Fonte: Autor.

Para este aplicativo, também é necessário habilitar a função que permite o uso de certificados não reconhecidos por autoridades certificadoras, pois os certificados deste trabalho foram assinados localmente. Após estes passos, a blockchain poderá se comunicar seguramente com o *Ethsigner* e ele também poderá se comunicar de modo seguro com o *Node-RED*. A configuração TLS do *Node-RED* será mostrada na seção da sua configuração.

Figura 37 - Exemplo de inicialização TLS.



Fonte: Autor.

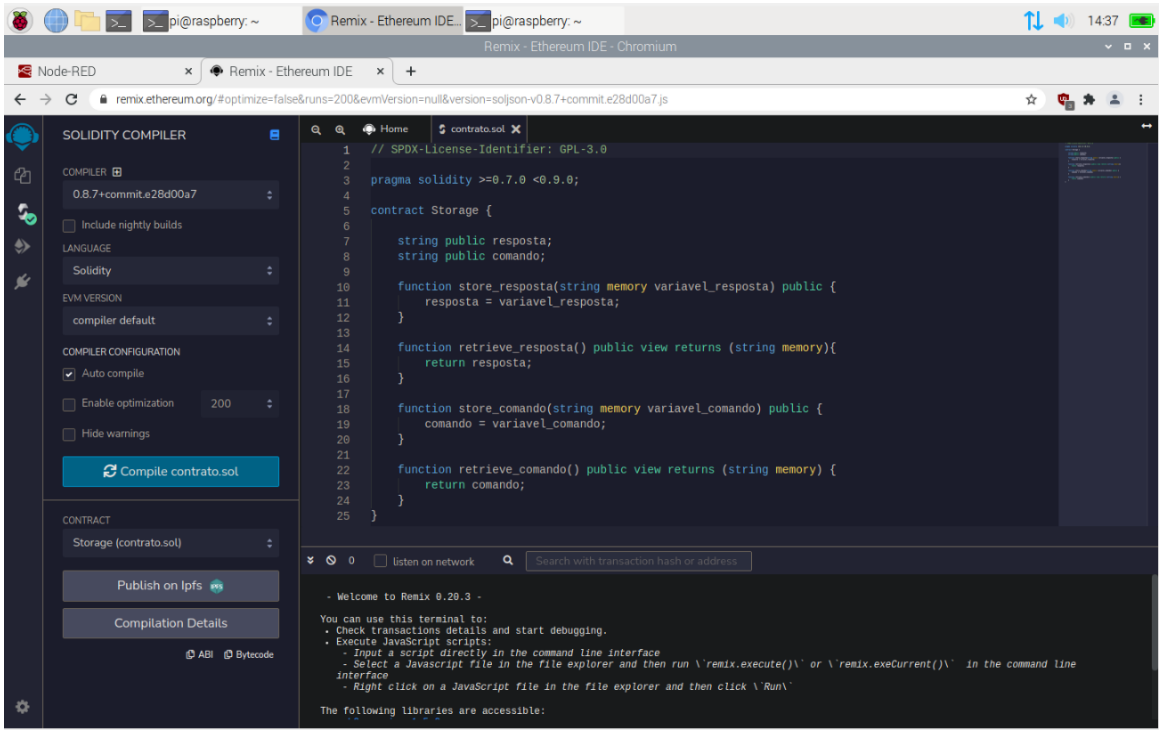
O Resultado deste comando é semelhante ao apresentado no terminal direito da Figura 41.

3.5 Criação do contrato inteligente

O contrato inteligente utilizado pelo *Node-RED* foi programado no site *Remix* (ETHEREUM FOUNDATION). A Figura 38 mostra o código do contrato inteligente já compilado, onde o contrato apenas utiliza quatro funções: duas para leitura e duas para escritas de variáveis do tipo string, referentes às requisições e respostas Modbus. O contrato não possui qualquer restrição de leitura ou escrita de usuários da rede, permitindo que qualquer usuário possa ler ou escrever. Em uma blockchain real, é recomendada a adição de restrições, para limitar o acesso ao contrato de cada usuário na rede.

A parte inferior esquerda da Figura 38 mostra os botões *ABI* e *Bytecode*. Ao clicar em um desses botões, o código correspondente ao botão clicado é copiado para a área de transferência, permitindo que seja colado dentro de uma função no *Node-RED* para ser utilizado como variável.

Figura 38 - Contrato inteligente.



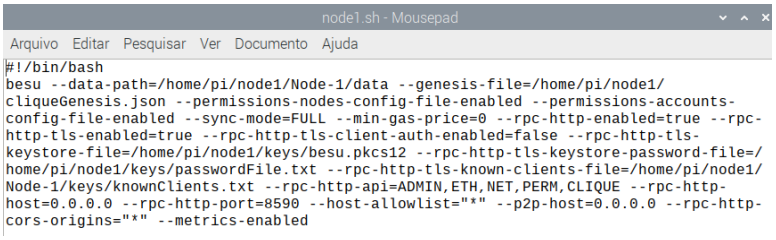
Fonte: Remix - Ethereum IDE (ETHEREUM FOUNDATION).

3.6 Inicialização da rede blockchain

Para realizar configurações na blockchain dentro do *Node-RED* a blockchain precisa estar inicializada. Dentro de cada VM é possível criar um atalho executável que abre um terminal com os comandos de inicialização de cada nó. A

Figura 39 mostra o arquivo texto do atalho de inicialização do nó 1, que ao ser executado abre uma nova instância do terminal e funciona como um nó validador da blockchain. O mesmo atalho deve ser executado nas VM's 2 e 3.

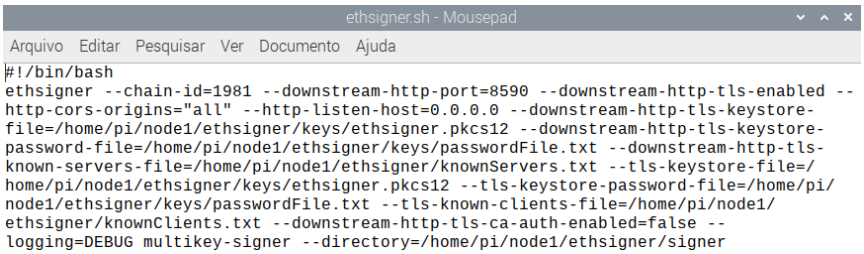
Figura 39 - Atalho de inicialização do node 1.



Fonte: Autor.

Após executar os três atalhos, os três nós estarão ativos, mas ainda não sincronizados. A sincronização será feita na seção de configuração do *Node-RED*, utilizando os comandos da Parte 9 de forma automatizada. No caso do aplicativo *Ethsigner* localizado na VM 1, a Figura 40 mostra seu atalho de inicialização.

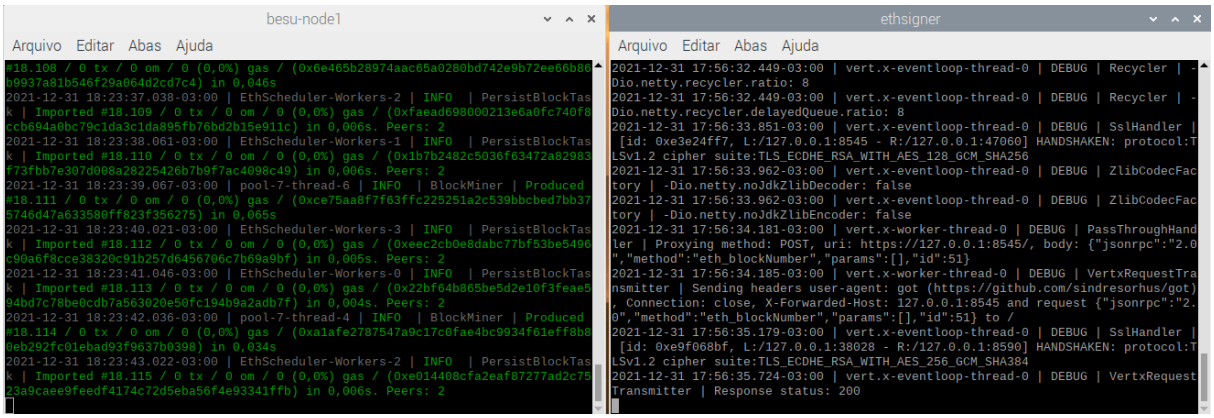
Figura 40 – Atalho de inicialização do Ethsigner.



Fonte: Autor.

A Figura 41 mostra o nó 1 da blockchain e o aplicativo *Ethsigner*, ambos em execução em seus respectivos terminais.

Figura 41 - Nó 1 e Ethsigner em execução.



Fonte: Autor.

3.7 Configuração do *Node-RED*

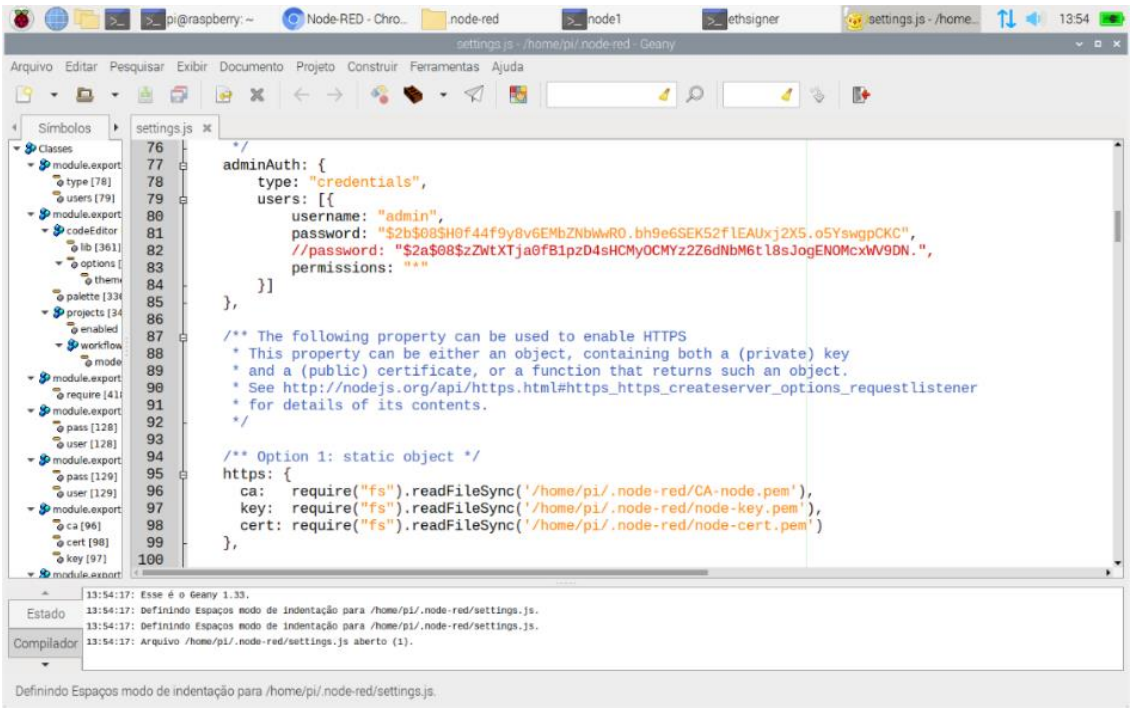
Com o *Node-RED* instalado, é possível habilitar o uso de credenciais para acessar o ambiente de programação, aumentado ainda mais a segurança da rede. Para habilitar o uso de credenciais, é necessário modificar o arquivo de configurações *settings.js*, que está localizado no diretório padrão de instalação do *Node-RED*. A Figura 42 mostra a seção do arquivo com os comentários retirados na parte de credenciais.

O *Node*-red utiliza o *hash* de uma palavra que será utilizada como senha no seu arquivo de configuração, impedindo que alguém que leia o arquivo saiba qual a senha usada. A senha original foi comentada e substituída pelo *hash* da palavra *nodered*. O *hash* é gerado através do comando *node-red admin hash-pw*, que pode ser executado em um terminal da VM.

A Figura 42 também mostra outra configuração que permite incrementar a segurança do *Node-RED*. O uso de comunicação https entre quem irá programar o *Node-RED* e o *Node-RED* também pode ser feito no arquivo de configurações, definindo o caminho do certificado usado, da chave privada do certificado e do

certificado de autoridade responsável por assinar o certificado usado. Neste trabalho, foi usado um certificado de autoridade local para testes que não é reconhecido oficialmente.

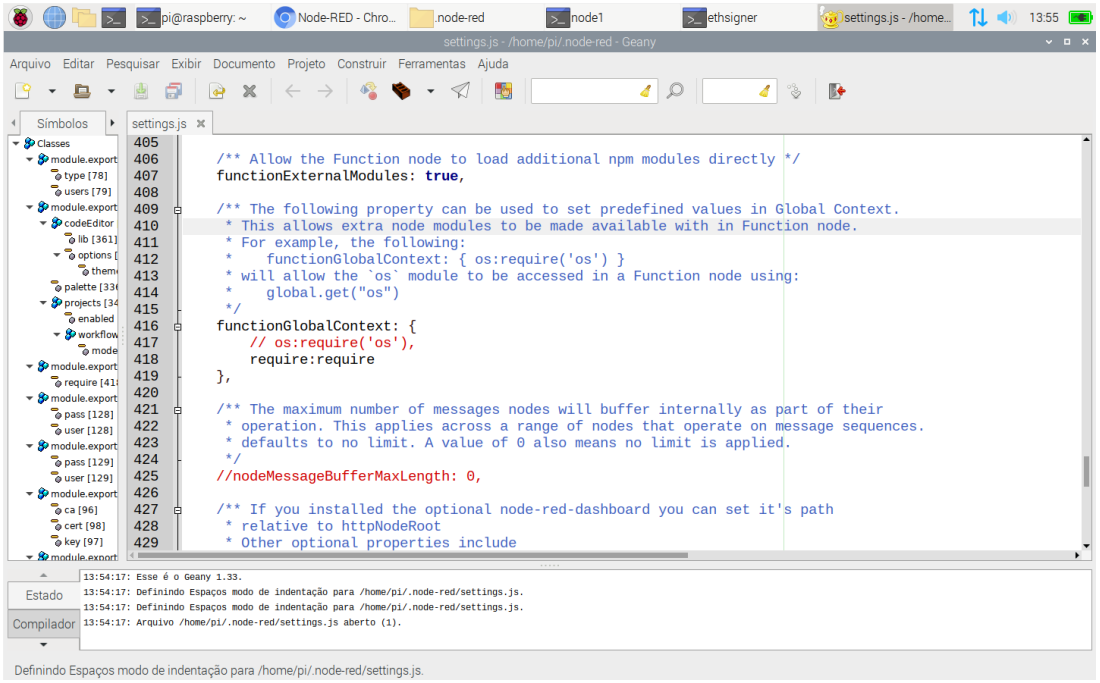
Figura 42 - Arquivo settings.js.



Fonte: Autor.

Ainda no arquivo de configuração, também é necessário permitir o uso de pacotes *javascript* utilizados na manipulação dos contratos inteligentes dentro do *Node-RED*. A Figura 43 mostra a habilitação do pacote *require*.

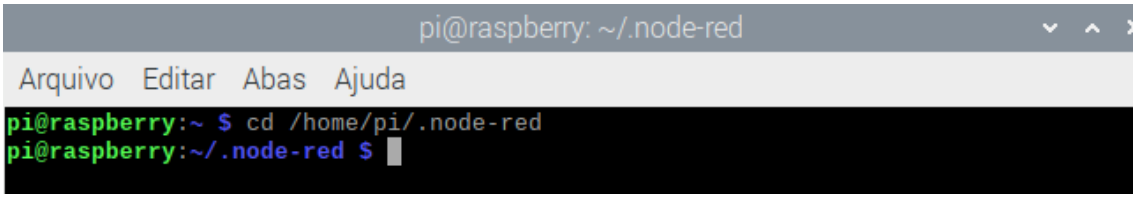
Figura 43 - Habilitação de pacotes externos.



Fonte: Autor.

Para instalar os pacotes *javascript*, é necessário abrir um terminal no diretório de instalação do *Node-RED*, como mostra a Figura 44. Dentro do terminal são instalados os pacotes *require*, *fs*, *web3*, *web3-ssl-ext* e *xhr2*. O comando de instalação é do tipo *npm install* (nome do pacote). Exemplo de comando: *npm install require*. O pacote *require* é usado para permitir que uma função do *Node-RED* possa utilizar um pacote *javascript externo*. Os demais pacotes são usados para referências de variáveis, manipulação de contratos inteligentes e comunicação segura via TLS.

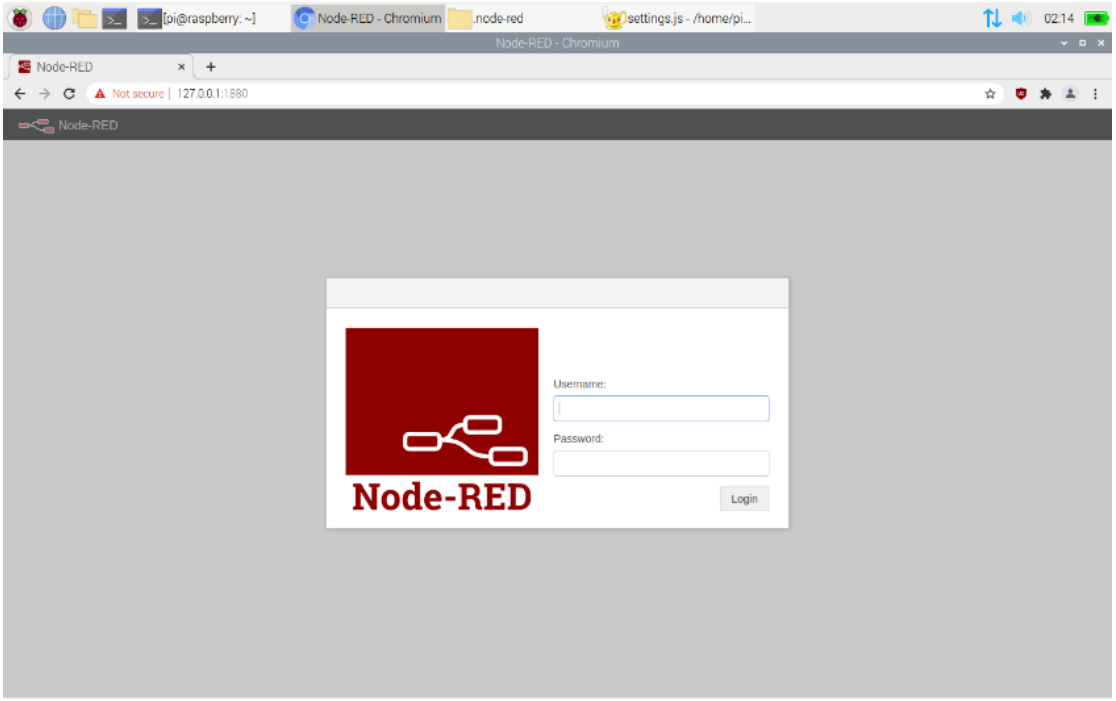
Figura 44 - Terminal no diretório Node-RED.



Fonte: Autor.

Após a configuração e inicialização do *Node-RED*, é possível acessar o ambiente de programação do *Node-RED* na máquina onde foi instalado através do endereço IP local, seguido pela porta TCP 1880 padrão do *Node-RED*, com as credenciais definidas no arquivo de configuração. A Figura 45 mostra a tela inicial acessada através do IP local (127.0.0.1 ou *localhost*).

Figura 45 - Tela inicial Node-RED.



Fonte: Autor.

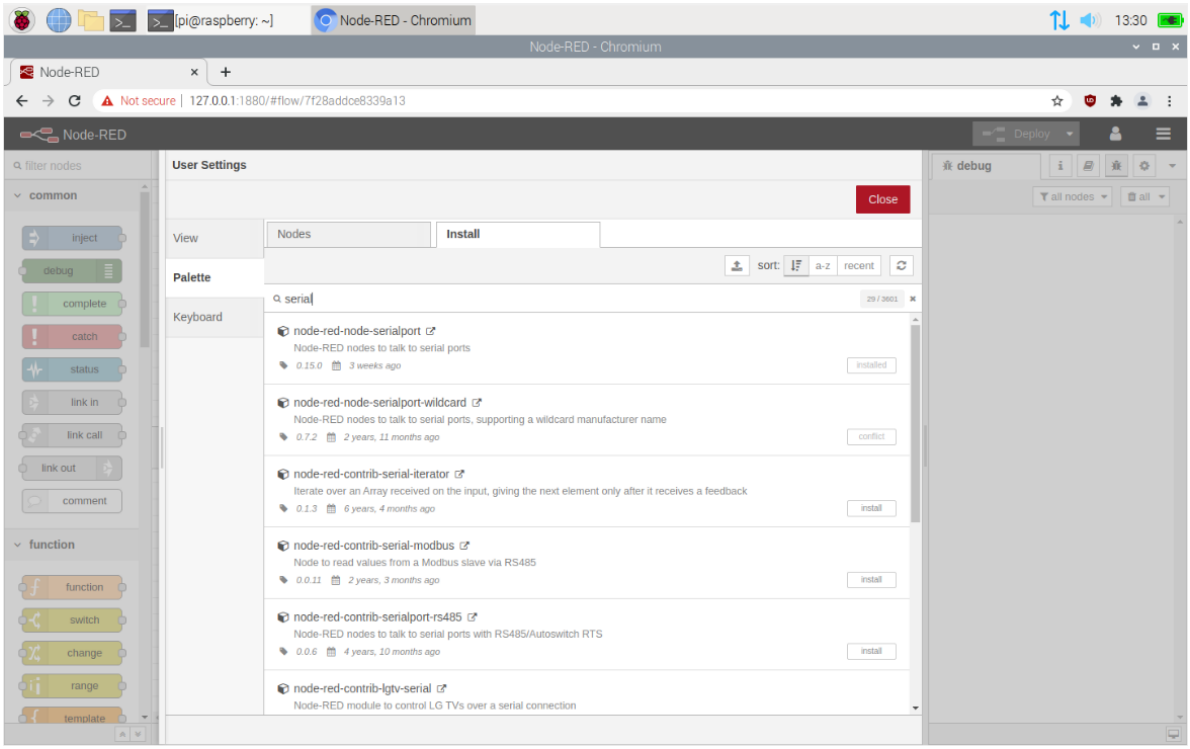
3.7.1 Configuração do nó 1

De acordo com a Figura 2, o nó 1 da blockchain receberá requisições do mestre Modbus via comunicação serial. As portas seriais foram criadas a partir de software emulador e de duas portas seriais físicas, que funcionam através de conversores *USB*. Para permitir o uso na máquina virtual, é necessário habilitar o uso de portas seriais nas suas configurações.

A Figura 46 mostra que ao buscar pela palavra serial no instalador de *nodes* do *Node-RED*, é possível instalar pacotes que permitem interações do *Node-RED* com

portas seriais. Neste trabalho foi utilizado o *node node-red-node-serialport* para comunicar com portas seriais.

Figura 46 - Pacote de portas seriais.



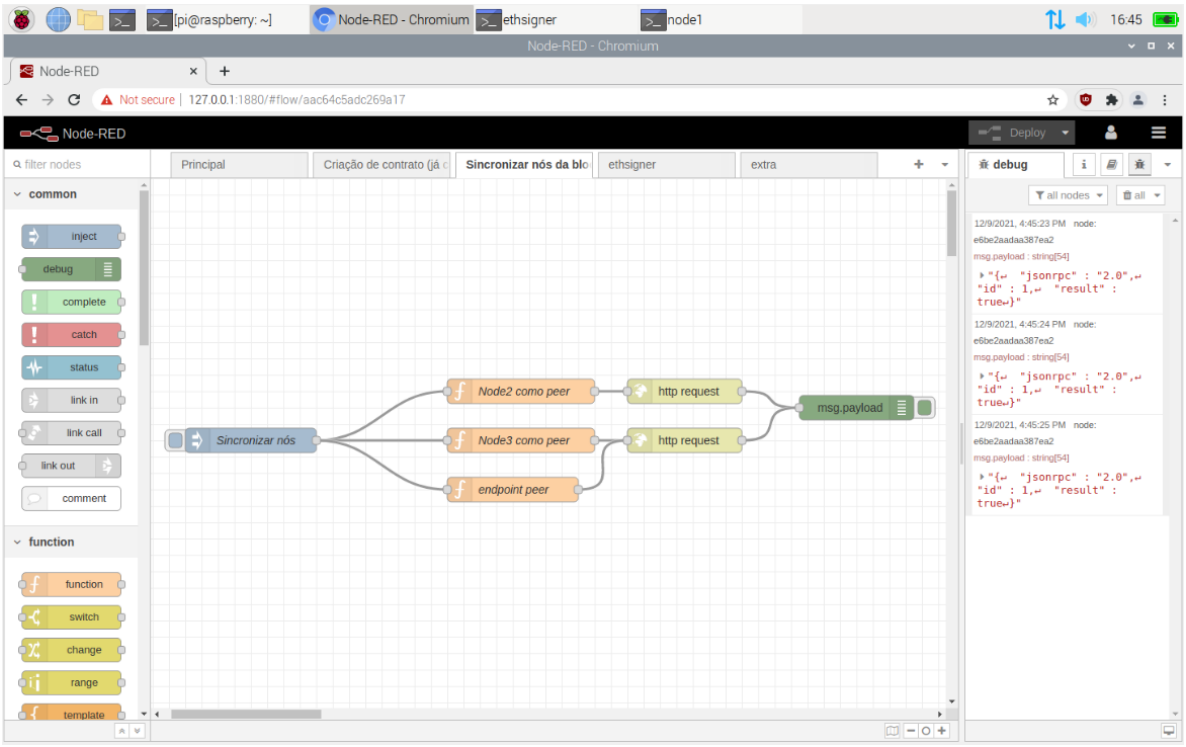
Fonte: Autor.

3.7.1.1 Sincronização dos nós

Para realizar a sincronização dos três nós, os comandos descritos na Parte 9 foram automatizados em uma sequência no *Node-RED*, mostrada na Figura 47. A requisição do *node2 como peer* é feita através do método *admin_addPeer* da plataforma *Besu*, onde se fornece a URL identificadora do nó 1 para enviar via requisição http com protocolo TLS para o endereço IP do nó 2. Isso é feito para o *node node3 como peer*, que faz a requisição para o IP do nó 3. O *node endpoint peer* representa o último ponto do canal de comunicação, que é sincronizado a partir da URL identificadora do nó anterior (nó 2) via requisição https ao nó final (nó 3).

Ao pressionar o botão do *node sincronizar nós*, os comandos serão enviados simultaneamente para os nós 2 e 3, que fornecerão as respostas *true* mostradas no texto em vermelho da lateral direita da Figura 47, mostrando que a sincronização foi um sucesso.

Figura 47 - Sincronização dos nós.



Fonte: Autor.

3.7.1.2 Adição do contrato à blockchain

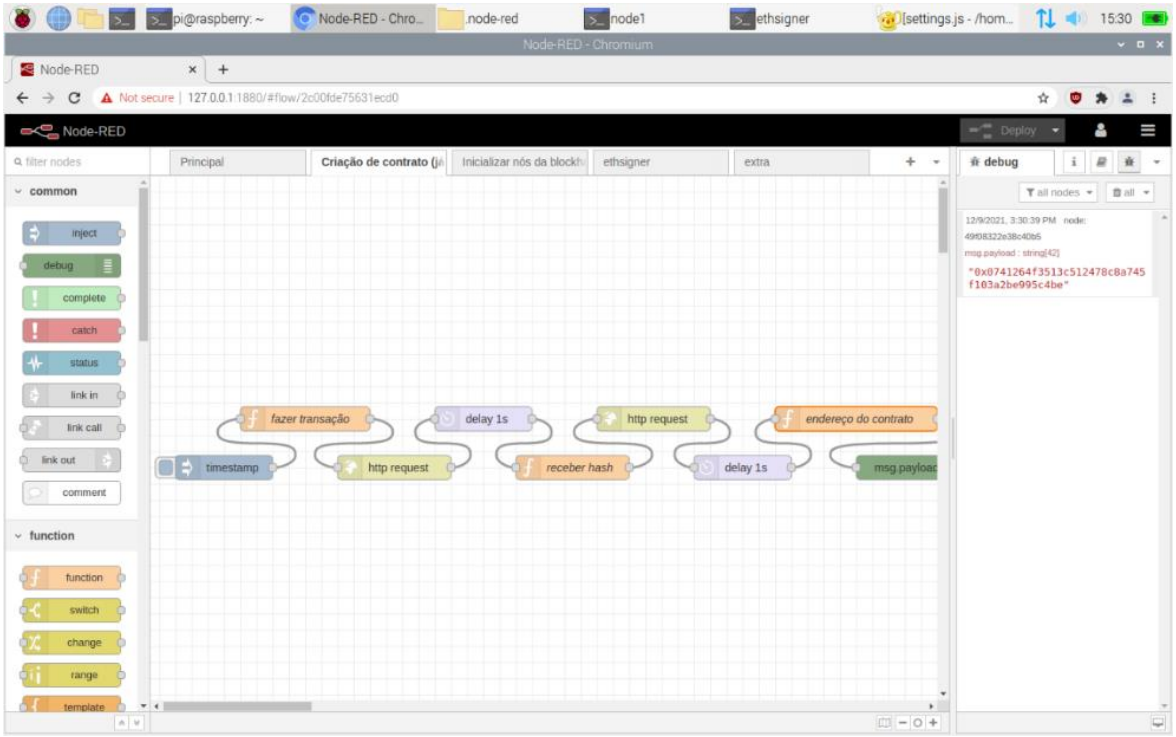
Para adicionar o contrato inteligente da Figura 38 na blockchain, é preciso enviar uma transação a partir de uma das contas do arquivo gênese através do método *eth_sendTransaction* utilizando o *Bytecode* do contrato inteligente como parâmetro da transação, dentro do *node fazer transação*. O método *eth_sendTransaction* da plataforma *Besu* permite a criação de contratos inteligentes na rede quando é especificado um endereço de envio e não se especifica um endereço de recebimento.

Dentro do ambiente de programação do *Node-RED* foi criada uma página exclusiva para a adição de um contrato na blockchain. A Figura 48 mostra a sequência de *nodes* necessária para enviar uma transação de criação de contrato, que envolve o uso de requisições http e manipulação de objetos em *javascript*. Todas as requisições http do *Node-RED* utilizam o protocolo TLS, que utiliza os mesmos certificados e chaves presentes no arquivo de configuração do *Node-RED*. As requisições http da Figura 48 são feitas para o aplicativo *Ethsigner*, que passa suas requisições para a blockchain.

Os atrasos de tempo de um segundo presentes na sequência servem para esperar a criação do bloco onde a transação foi realizada.

A lateral direita da Figura 48 mostra um exemplo de endereço de contrato criado, no texto vermelho da janela *debug* e que foi recebido da última requisição http, exibido através do *node msg.payload*.

Figura 48 - Adição de contrato na rede.



Fonte: Autor.

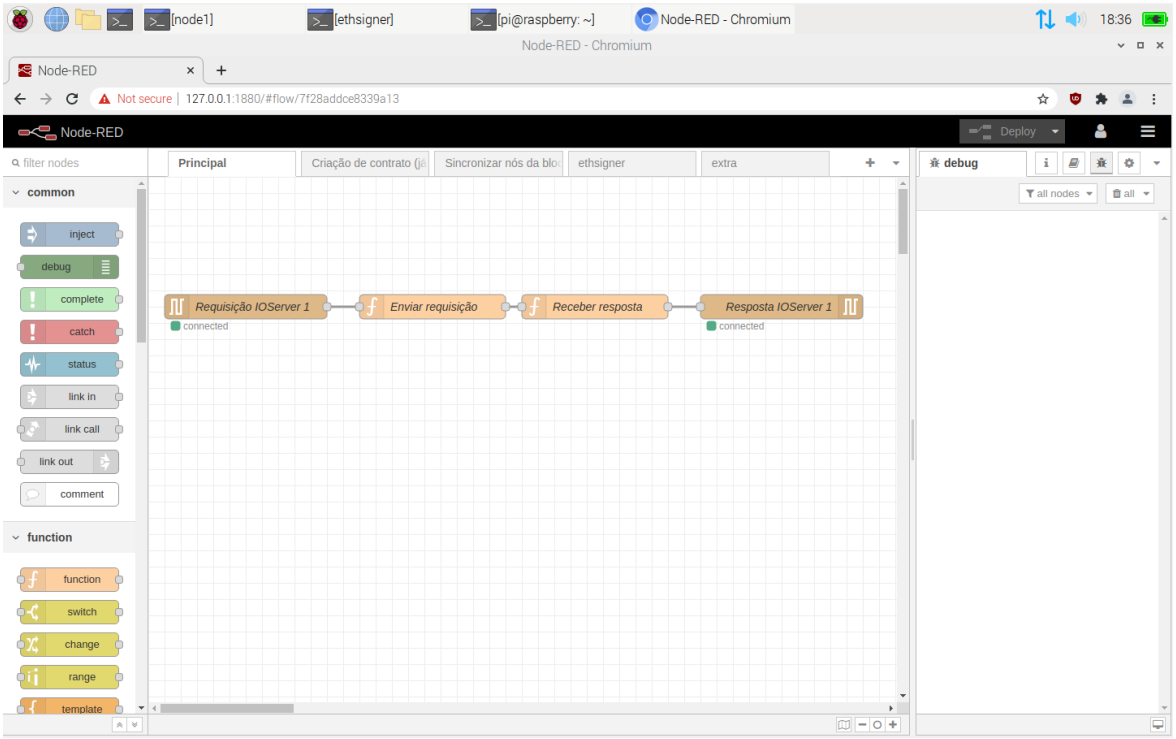
3.7.1.3 Comunicação serial e escrita no contrato

A comunicação serial do *Node-RED* com o *IOServer* e a escrita da requisição do *IOServer* na blockchain são feitas em uma única sequência no *Node-RED*. O *node Requisição IOServer 1* do pacote mostrado na Figura 49 corresponde ao recebimento de comunicação serial Modbus proveniente do cliente Modbus no *IOServer*.

Para modificar o contrato, o *node* de nome *enviar requisição* (segundo *node* na Figura 49) utiliza os pacotes *javascript* definidos na Configuração do *Node-RED*, a mensagem serial, e o parâmetro *ABI* do contrato inteligente, modificando apenas a função *store_comando* do contrato. O nó 2 irá ler esta função e escreverá a resposta no contrato.

Após o envio da requisição, o *node* de nome *receber resposta* (terceiro *node* na Figura 49) irá ler o valor da resposta no contrato, através da função *retrieve_resposta*. Ao ler a resposta, a mensagem Modbus é repassada como resposta serial para o *IOServer*.

Figura 49 - Comunicação Mestre-Blockchain-Mestre.



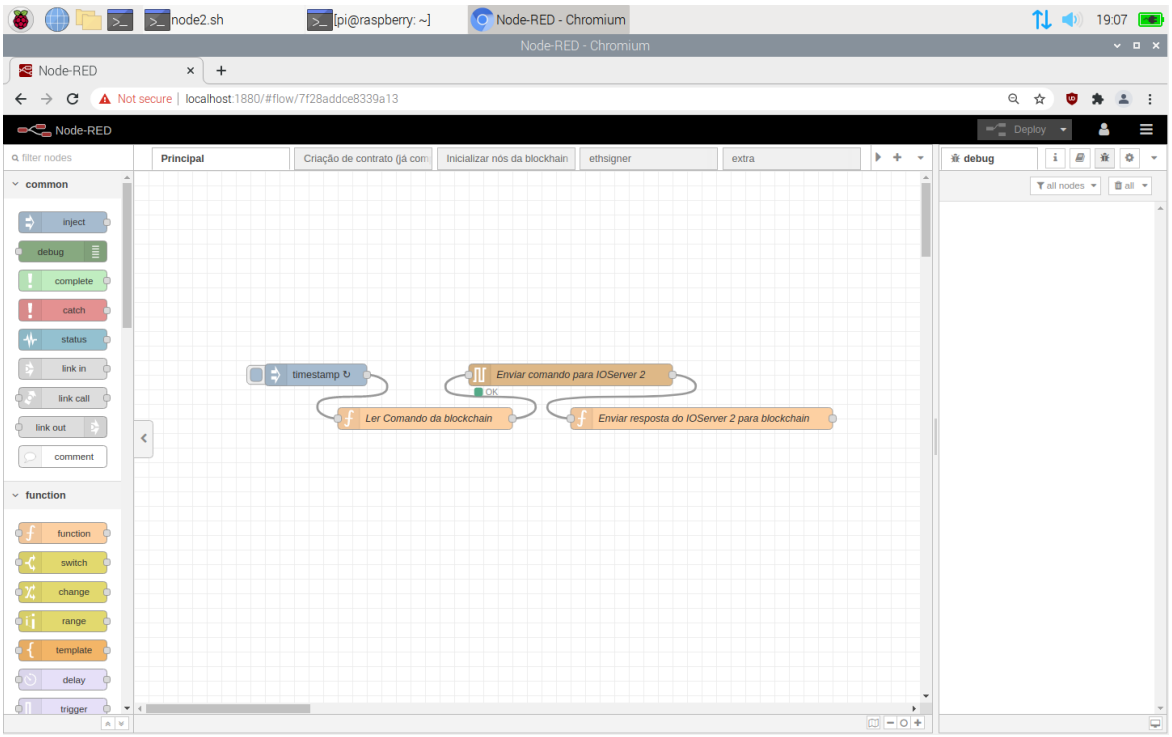
Fonte: Autor.

3.7.2 Configuração do nó 2

De acordo com a Figura 2, o nó 2 irá ler uma requisição na blockchain e repassará a requisição para o servidor Modbus no *IOServer*. O servidor Modbus enviará uma resposta para o nó 2, referente à requisição do cliente Modbus. A Figura 50 mostra a sequência utilizada para esta comunicação.

O *node timestamp* foi configurado para ser executado automaticamente a cada 1 segundo, ativando o *node Ler Comando da blockchain*, que utilizará os mesmos componentes do nó 1 na Comunicação serial e escrita no contrato para ler a mensagem, utilizando a função *retrieve_comando*. Após ler a mensagem, ela é repassada pelo *node enviar comando para IOServer 2* para o escravo Modbus, o *node* esperará pela resposta do servidor e a escreverá no contrato inteligente, através do *node enviar resposta do IOServer 2 para a blockchain*, através da função *store_resposta*.

Figura 50 - Comunicação Blockchain-Escravo-Blockchain.



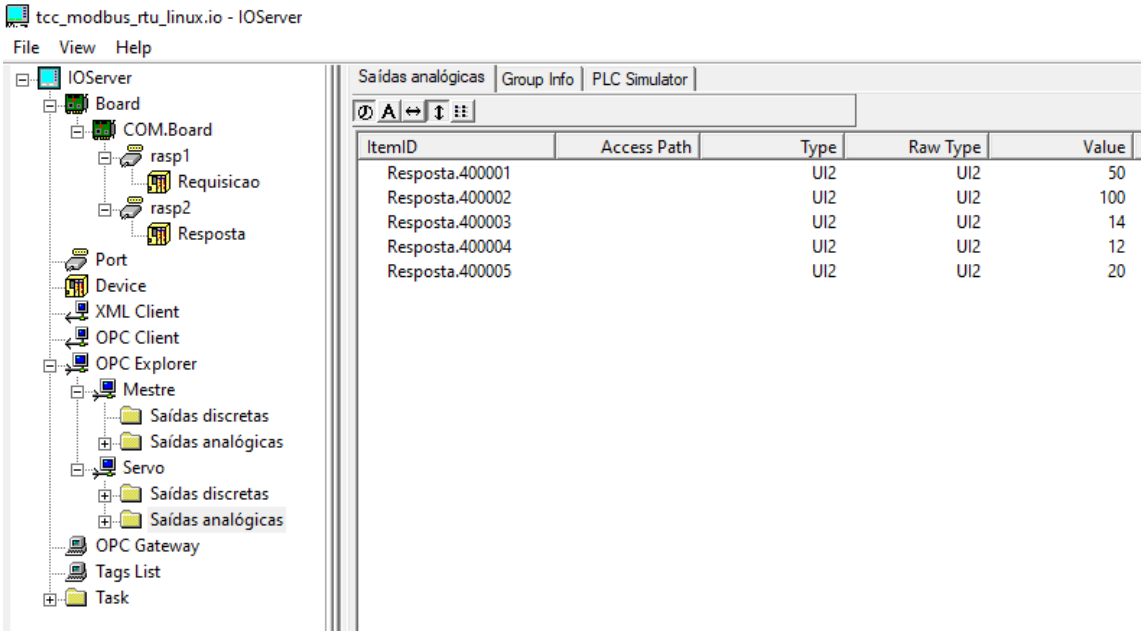
Fonte: Autor.

4 Resultados obtidos

Agora que todos os programas estão configurados, basta apenas inicializar o aplicativo *IOServer* no computador hospedeiro. A Figura 51 mostra o aplicativo com dois dispositivos adicionados, onde *rasp1* representa o mestre Modbus e *rasp2* o escravo Modbus.

Os valores mostrados representam os 5 primeiros endereços Modbus para saídas analógicas, que foram modificados com valores aleatórios que serão requisitados pelo cliente Modbus. Esses dados serão requisitados pelo nó 2, que escreverá uma mensagem Modbus referente a esses valores.

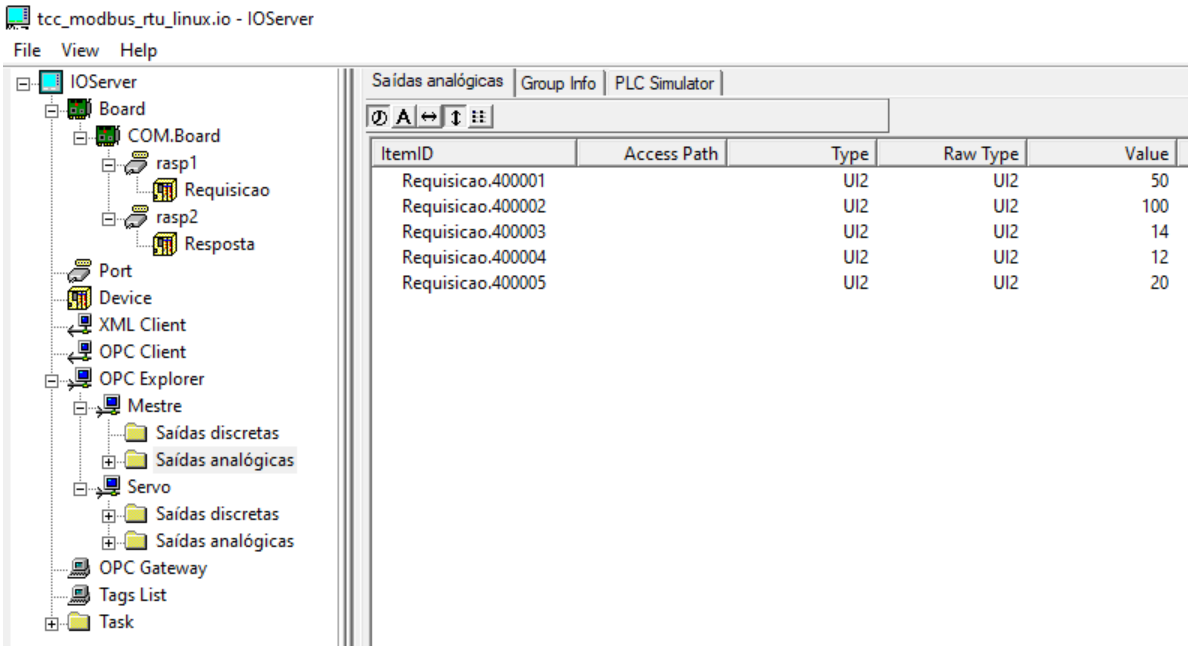
Figura 51 - Aplicativo IOServer (Servo).



Fonte: Autor.

A Figura 52 mostra os dados de saída analógicas requisitados pelo cliente Modbus. Os dados foram lidos da blockchain e repassados para o cliente Modbus através da configuração *Node-RED* do nó 1, mostrada anteriormente.

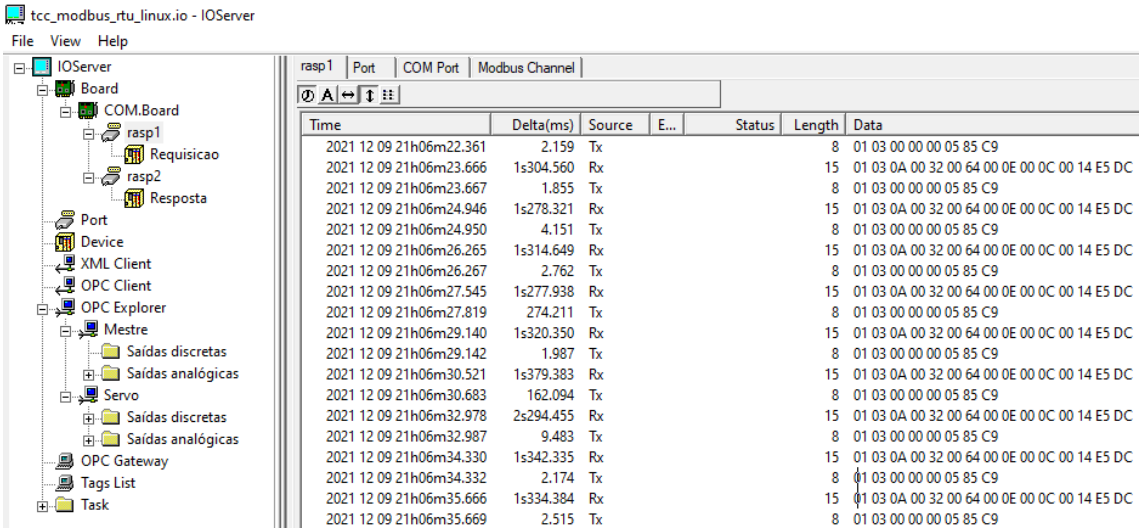
Figura 52 - Aplicativo IOServer (Mestre).



Fonte: Autor.

A comunicação serial entre cliente e *Node-RED* 1 está mostrada na Figura 53. O nó 1 demora entre 1 e 3 segundos para fazer uma requisição e receber a resposta da mesma requisição, cujas mensagens tramitam entre duas redes diferentes.

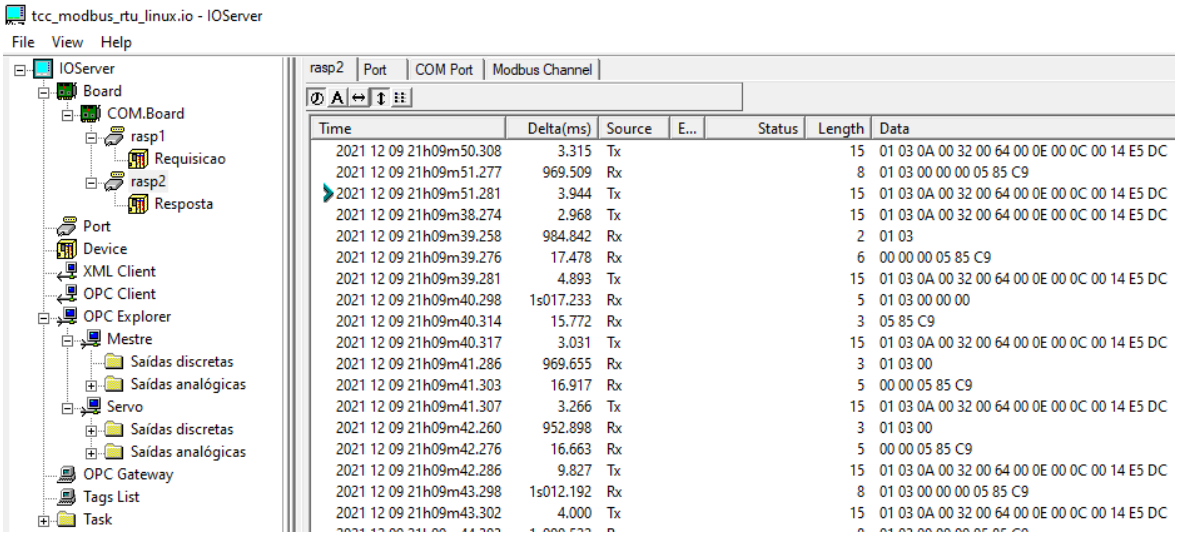
Figura 53 - Funcionamento da porta serial do cliente Modbus.



Fonte: Autor.

A Figura 54 mostra a comunicação serial entre escravo e blockchain, que é feita de maneira cíclica (a cada um segundo), como programado no *Node-RED* do nó 2.

Figura 54 - Funcionamento da porta serial do servidor Modbus.



Fonte: Autor.

5 Conclusão

A rede blockchain apresentou o resultado esperado da criação de um bloco com transações contendo uma comunicação Modbus entre um dispositivo mestre e um escravo a cada um segundo, aproximadamente, e um baixo uso de esforço computacional da rede para cada transação (aproximadamente 0,3% do limite padrão, definido no bloco gênese da rede).

O software *IOServer* realizou com sucesso a leitura dos dados da blockchain pelo mestre, mas ao tentar realizar uma modificação em um endereço do escravo, os comandos de leitura e controle entraram em conflito, devido ao modo como o *IOServer* realiza leituras.

5.1 Trabalhos futuros

Como trabalhos futuros, são considerados o desenvolvimento desta mesma topologia em dispositivos físicos (minicomputadores, sistema de supervisão e dispositivos de chão de fábrica), que produziriam resultados mais condizentes com um funcionamento real na indústria e a utilização de outras redes industriais em conjunto com uma rede blockchain.

Referências bibliográficas

CONSENSYS. Configure TLS - EthSigner. Disponível em: <https://docs.ethsigner.consensys.net/en/stable/HowTo/Configure-TLS/>. Acesso em: 8 Dezembro 2021.

CONSENSYS. Install binary distribution - EthSigner. Disponível em: <https://docs.ethsigner.consensys.net/en/stable/HowTo/Get-Started/Install-Binaries/>. Acesso em: 8 Dezembro 2021.

CONSENSYS. Start EthSigner with multiple signers - EthSigner. Disponível em: <https://docs.ethsigner.consensys.net/en/stable/Tutorials/Multifile/>. Acesso em: 8 Dezembro 2021.

ETHEREUM FOUNDATION. Gas and fees | ethereum.org. Disponível em: <https://ethereum.org/en/developers/docs/gas/>. Acesso em: 7 Dezembro 2021.

ETHEREUM FOUNDATION. INTRODUCTION TO SMART CONTRACTS. Disponível em: <https://ethereum.org/en/developers/docs/smart-contracts/>. Acesso em: 6 Dezembro 2021.

ETHEREUM FOUNDATION. Remix - Ethereum IDE. Disponível em: <https://remix.ethereum.org/>. Acesso em: 9 Dezembro 2021.

FAULKNER, John. **Getting Started with Cryptography in.NET**.

HABER, S., S. W. S. **How to time-stamp a digital document**.

HYPERLEDGER FOUNDATION. Client and server TLS - Hyperledger Besu. Disponível em: <https://besu.hyperledger.org/en/stable/HowTo/Configure/TLS/Configure-TLS/>. Acesso em: 8 Dezembro 2021.

HYPERLEDGER FOUNDATION. Comparing PoA consensus protocols - Hyperledger Besu. Disponível em: <https://besu.hyperledger.org/en/stable/Concepts/Consensus-Protocols/Comparing-PoA/>. Acesso em: 6 Dezembro 2021.

HYPERLEDGER FOUNDATION. Create a permissioned network - Hyperledger Besu. Disponível em: <https://besu.hyperledger.org/en/stable/Tutorials/Permissioning/Create-Permissioned-Network/>. Acesso em: 8 Dezembro 2021.

HYPERLEDGER FOUNDATION. Genesis file items - Hyperledger Besu. Disponível em: <https://besu.hyperledger.org/en/stable/Reference/Config-Items/>. Acesso em: 8 Dezembro 2021.

HYPERLEDGER FOUNDATION. Hyperledger Besu system requirements - Hyperledger Besu. Disponível em: <https://besu.hyperledger.org/en/stable/HowTo/Get-Started/System-Requirements/System-Requirements-Private/>.

HYPERLEDGER FOUNDATION. Install binary distribution - Hyperledger Besu. Disponível em: <https://besu.hyperledger.org/en/stable/HowTo/Get-Started/Installation-Options/Install-Binaries/>. Acesso em: 7 Dezembro 2021.

INTERNET ENGINEERING TASK FORCE. rfc8446, 2018. Disponível em: <https://datatracker.ietf.org/doc/html/rfc8446>. Acesso em: 6 Dezembro 2021.

LEEUWEN, Jan V. **Handbook of Theoretical Computer Science**.

M. J. M. CHOWDHURY, A. C. M. A. K. J. H. A. P. S. Blockchain versus Database: A Critical Analysis. **17th IEEE International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)**, 1-3 Agosto 2018.

NAKAMOTO, Satoshi. Bitcoin: A Peer-to-Peer Electronic Cash System, 2008. Disponível em: <https://bitcoin.org/bitcoin.pdf>.

OPENJS FOUNDATION. GitHub - nvm-sh/nvm: Node Version Manager - POSIX-compliant bash script to manage multiple active node.js versions. Disponível em: <https://github.com/nvm-sh/nvm>. Acesso em: 8 Dezembro 2021.

OPENJS FOUNDATION. Running Node-RED locally. Disponível em: <https://nodered.org/docs/getting-started/local>. Acesso em: 8 Dezembro 2021.

ORACLE. Oracle VM VirtualBox. Disponível em: <https://www.virtualbox.org/>. Acesso em: 7 Dezembro 2021.

RASPBERRY PI FOUNDATION. Operating system images - Raspberry Pi. Disponível em: <https://www.raspberrypi.com/software/operating-systems/>. Acesso em: 7 Dezembro 2021.

REYNA, Ana et al. On blockchain and its integration with IoT. Challenges and opportunities. **Future Generation Computer Systems**, Novembro 2018. 173-190.

SOFTWARE IN THE PUBLIC INTEREST, INC. apt-get(8). Disponível em: <https://manpages.debian.org/unstable/apt/apt-get.8.en.html>. Acesso em: 7 Dezembro 2021.

SZILÁGYI, Péter. EIP-225: Clique proof-of-authority consensus protocol, 6 Março 2017. Disponível em: <https://eips.ethereum.org/EIPS/eip-225>.

THREATPOST. IoT Attacks Skyrocket, Doubling in 6 Months. Disponível em: <https://threatpost.com/iot-attacks-doubling/169224/>. Acesso em: 30 Dezembro 2021.

WIKIPÉDIA. Assinatura digital. Disponível em: <https://pt.wikipedia.org/wiki/Assinatura_digital>. Acesso em: 6 Dezembro 2021.

WIKIPÉDIA. Blockchain. Disponível em: <https://pt.wikipedia.org/wiki/Blockchain>. Acesso em: 6 Dezembro 2021.

WIKIPÉDIA. Criptografia de chave pública. Disponível em: <view-source:https://pt.wikipedia.org/wiki/Criptografia_de_chave_p%C3%BAblica>. Acesso em: 7 Dezembro 2021.

WIKIPÉDIA. Função hash criptográfica. Disponível em: <https://pt.wikipedia.org/wiki/Fun%C3%A7%C3%A3o_hash_criptogr%C3%A1fica>. Acesso em: 6 Dezembro 2021.

WIKIPÉDIA. Internet das coisas. Disponível em: <https://pt.wikipedia.org/wiki/Internet_das_coisas>. Acesso em: 30 Dezembro 2021.

WIKIPÉDIA. Stuxnet. Disponível em: <https://pt.wikipedia.org/wiki/Stuxnet>.