UNIVERSIDADE FEDERAL DO AMAZONAS
FACULDADE DE TECNOLOGIA
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

MARIO RUBEN LIMA DE OLIVEIRA

ANÁLISE COMPARATIVA DE DESEMPENHO DE *SMART CONTRACTS* EM ARQUITETURA *ETHEREUM* E *BITCOIN*

MANAUS
2025

UNIVERSIDADE FEDERAL DO AMAZONAS
FACULDADE DE TECNOLOGIA
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

MARIO RUBEN LIMA DE OLIVEIRA

ANÁLISE COMPARATIVA DE DESEMPENHO DE SMART CONTRACTS EM ARQUITETURA ETHEREUM E BITCOIN

Dissertação apresentada ao Curso de Mestrado em Engenharia Elétrica, área de concentração em Controle e Automação de Sistemas na linha de pesquisa em Sistemas Inteligentes e Microeletrônica do Programa de Pós- Graduação em Engenharia Elétrica da Universidade Federal do Amazonas.

Orientador: Prof. Dr. Carlos Augusto de Moraes Cruz
Coorientador: Prof. Dr. André Luiz Duarte Cavalcante

MANAUS
2025

Ficha Catalográfica

Elaborada automaticamente de acordo com os dados fornecidos pelo(a) autor(a).

O48a Oliveira, Mario Ruben Lima de
Análise comparativa de desempenho de smart contracts em arquitetura ethereum e bitcoin / Mario Ruben Lima de Oliveira. - 2025.
76 f. ; 31 cm.

Orientador(a): Carlos Augusto de Moraes Cruz.
Coorientador(a): André Luiz Duarte Cavalcante.
Dissertação (mestrado) - Universidade Federal do Amazonas, Programa de Pós-Graduação em Engenharia Elétrica, Manaus, 2025.

1. Blockchain. 2. Bitcoin. 3. Ethereum. 4. Contratos inteligentes. 5. Comparação de desempenho. I. Cruz, Carlos Augusto de Moraes. II. Cavalcante, André Luiz Duarte. III. Universidade Federal do Amazonas. Programa de Pós-Graduação em Engenharia Elétrica. IV. Título

---

Ministério da Educação
Universidade Federal do Amazonas
Coordenação do Programa de Pós-Graduação em Engenharia Elétrica

FOLHA DE APROVAÇÃO

Poder Executivo Ministério da Educação
Universidade Federal do Amazonas
Faculdade de Tecnologia
Programa de Pós-graduação em Engenharia Elétrica

Pós-Graduação em Engenharia Elétrica. Av. General Rodrigo Octávio Jordão Ramos, nº 3.000 - Campus Universitário, Setor Norte - Coroado, Pavilhão do CETELI. Fone/Fax (92) 99271-8954 Ramal:2607. E-mail: ppgee@ufam.edu.br

MARIO RUBEN LIMA DE OLIVEIRA

ANÁLISE COMPARATIVA DE DESEMPENHO DE SMART CONTRACTS EM ARQUITETURA ETHEREUM E BITCOIN

Dissertação apresentada ao Programa de Pós-Graduação em Engenharia Elétrica da Universidade Federal do Amazonas, como
requisito parcial para obtenção do título de Mestre
em Engenharia Elétrica na área de concentração
Controle e Automação de Sistemas.

Aprovado em 23 de setembro de 2025.

BANCA EXAMINADORA
Prof. Dr. André Luiz Duarte Cavalcante - Presidente
Prof. Dr. Rainer Xavier de Amorim - Membro Titular 1 - Externo
Prof. Dr. Anderson Vinícius Corrêa de Oliveira - Membro Titular 2 - Externo

Manaus, 18 de setembro de 2025.

Documento assinado eletronicamente por **André Luiz Duarte Cavalcante**, **Professor do Magistério Superior**, em 29/09/2025, às 16:17, conforme horário oficial de Manaus, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015 .

Documento assinado eletronicamente por **Rainer Xavier de Amorim**, **Professor do Magistério Superior**, em 02/10/2025, às 09:48, conforme horário oficial de Manaus, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015 .

Documento assinado eletronicamente por **Anderson Vinícius Corrêa Oliveira**, **Usuário Externo**, em 02/10/2025, às 10:44, conforme horário oficial de Manaus, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

A autenticidade deste documento pode ser conferida no site https://sei.ufam.edu.br/sei/controlador_externo.php?acao=documento_conferir&id_orgao_acesso_externo=0, informando o código verificador **2802437** e o código CRC **5191E876**.

Av. General Rodrigo Octávio Jordão Ramos, nº 3.000 - Bairro Coroado Campus Universitário, Setor Norte - Telefone: 99271-8954
CEP 69080-900 Manaus/AM - Pavilhão do CETELI. E-mail: ppgee@ufam.edu.br

Referência: Processo nº 23105.041454/2025-24 SEI nº 2802437

AGRADECIMENTOS

A Deus, fonte de toda sabedoria, saúde e discernimento, sem Sua presença constante, força espiritual nos momentos de incerteza, este percurso acadêmico não teria sido possível.

Aos meus pais, Mario Sérgio e Alessandra Anne, agradeço profundamente pelo amor incondicional, pelos valores transmitidos e pelo apoio contínuo em todas as fases da minha vida. Aos meus irmãos, Erik Gustavo e Alan Daniel, deixo meu reconhecimento pelo companheirismo e incentivo constantes.

À minha namorada Daniela, manifesto meu carinho e gratidão pelo amor, paciência e estímulo dedicados ao longo desta jornada. Seu apoio foi fundamental para que eu mantivesse o foco e a motivação necessários para a realização desta pesquisa.

Ao Professor Dr. Carlos Cruz, meu orientador, expresso meu sincero agradecimento pelos valiosos ensinamentos sobre blockchain, pelas conversas produtivas e inspiradoras que tantas vezes ocorreram durante um café, e pelo incentivo permanente à concretização deste trabalho. Sua orientação foi essencial para o êxito desta dissertação.

Expresso minha gratidão ao amigo Derik Adan, cuja amizade iniciada na Engenharia Eletrônica levo para a vida. Sua sabedoria, amizade e conselhos, especialmente quanto à melhor colocação de escrita e formatação, foram de grande importância para a comunicação clara e a fluidez deste trabalho. Registro também meus agradecimentos ao colega Wilson, pelo suporte técnico prestado e pelas contribuições relevantes na etapa de codificação deste estudo.

Às colegas Talita Pinheiro e Jessica Mariela, que atuam na mesma linha temática, reconheço com apreço as valiosas trocas de conhecimento e a colaboração ao longo de todo este processo.

RESUMO

O presente trabalho realiza uma análise comparativa de desempenho de contratos inteligentes implementados em duas arquiteturas distintas de blockchain: a Ethereum Virtual Machine (via Binance Smart Chain) e a Bitcoin Virtual Machine (via Bitcoin Satoshi Vision). O estudo teve como objetivo avaliar o impacto das diferenças arquiteturais no custo de transações, no tempo de execução e na ocupação de rede, a fim de identificar vantagens e limitações práticas de cada plataforma.

Para alcançar esse propósito, foram desenvolvidos dois contratos inteligentes de referência: um contrato de mensagem, representativo de operações simples de armazenamento, e um contrato de contador, concebido para simular cenários de maior complexidade lógica e de estresse computacional. Ambos foram implementados em linguagens específicas das plataformas (Solidity para BSC e sCrypt para BSV) e executados em ambientes de teste controlados. A metodologia incluiu a configuração de infraestrutura experimental, a realização de testes de desempenho, a medição de custos, bem como a execução de funções de ataque para avaliar a escalabilidade sob alta carga.

Palavras-chaves: Blockchain; Bitcoin; Ethereum, Contratos inteligentes, Comparação de desempenho.

ABSTRACT

This study presents a comparative performance analysis of smart contracts implemented in two distinct blockchain architectures: the Ethereum Virtual Machine (via Binance Smart Chain) and the Bitcoin Virtual Machine (via Bitcoin Satoshi Vision). The objective was to evaluate the impact of architectural differences on transaction costs, execution time, and network usage, in order to identify the practical advantages and limitations of each platform.

To achieve this goal, two reference smart contracts were developed: a message contract, representing simple storage operations, and a counter contract, designed to simulate scenarios of greater logical complexity and computational stress. Both contracts were implemented in the specific languages of each platform (Solidity for BSC and sCrypt for BSV) and executed in controlled test environments. The methodology included the configuration of an experimental infrastructure, performance testing, cost measurement, as well as the execution of attack functions to assess scalability under high load.

Keywords: Blockchain; Bitcoin; Ethereum; Smart Contracts; Performance Comparison.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SIGLAS

BSC	*Binance Smart Chain*
BSV	*Bitcoin Satoshi Vision*
BVM	*Bitcoin Virtual Machine*
DEFI	*Decentralized Finance*
ETH	*Ethereum*
EVM	*Ethereum Virtual Machine*
NFT	*Non Fungible Token*
POH	*Proof of History*
POS	*Proof Of Stake*
POSA	*Proof Of Stake Authority*
POW	*Proof of Work*
SDK	*Software Development Kit*
SHA	*Security Hash Augorithm*
TPS	*Transactions Per Second*
UTXO	*Unspect Transaction Output*

SUMÁRIO

INTRODUÇÃO

Com o advento da tecnologia blockchain, a implementação de contratos inteligentes tem se destacado como uma das inovações mais transformadoras no cenário digital, particularmente nas áreas de finanças descentralizadas (DeFi) e gestão de ativos digitais. As redes *Ethereum* e *Bitcoin*, as duas mais populares e amplamente utilizadas, oferecem plataformas que suportam esses contratos inteligentes, mas com arquiteturas e características diferentes. Enquanto a *Ethereum*, com sua *Ethereum Virtual Machine* (EVM), foi projetada especificamente para suportar contratos inteligentes complexos e dApps (aplicações descentralizadas), a rede *bitcoin*, com sua *Bitcoin Virtual Machine* (BVM), oferece suporte que prioriza transações simples e seguras. Essa diferença estrutural levanta um problema de pesquisa central: como essas arquiteturas impactam o desempenho de contratos inteligentes em termos de custo, tempo ' execução e ocupação na rede.

De acordo com Buterin (2013) em seu artigo sobre a *Ethereum*, a arquitetura da EVM foi criada para permitir a execução de contratos inteligentes de forma mais flexível e com capacidade para lidar com uma ampla gama de aplicações descentralizadas. Já no caso do *Bitcoin*, Nakamoto (2008) introduziu o conceito de blockchain com foco na transferência de valor digital, restringindo a complexidade dos contratos a *scripts* simples que garantem a segurança das transações na rede. Essa distinção entre as duas plataformas gera diferentes impactos em relação ao desempenho dos contratos inteligentes, especialmente quando se observa o custo das transações, a estrutura dos contratos e a velocidade de execução.

Esse problema é relevante, pois a eficiência no processamento de contratos inteligentes afeta diretamente a escalabilidade das soluções em blockchain, bem como sua viabilidade em aplicações práticas, especialmente no contexto de finanças descentralizadas e automação de processos digitais. O estudo de Bartoletti e Pompianu (2017), ao avaliar as plataformas de contratos inteligentes, aponta que as diferenças nas estruturas de blockchain, como o modelo de consenso e o uso de *gas*, podem afetar diretamente o desempenho de operações. No caso da *Ethereum*, o uso de *gas*, como detalhado por Wood (2014), oferece flexibilidade para calcular o custo de execução de contratos, mas também pode tornar as transações mais caras dependendo da complexidade do código. Em contrapartida, a rede *Bitcoin*, com seu foco em segurança e simplicidade, tende a ser mais eficiente em termos de custos, no entanto, menos flexível em relação à complexidade dos contratos inteligentes que pode suportar.

Diante desse cenário, o presente trabalho visa analisar o desempenho de contratos inteligentes nas duas plataformas, *Ethereum* (via *Binance Smart Chain*) e *Bitcoin* (via *Bitcoin Satoshi Vision*), comparando o custo das transações, a estrutura dos contratos e a velocidade de execução. Para isso, serão utilizados dois contratos inteligentes com funcionalidades semelhantes, implantados em ambas as redes, de modo a avaliar como essas variáveis afetam o desempenho geral.

Essa avaliação é fundamental para desenvolvedores e pesquisadores que buscam melhorar a eficiência e a escalabilidade de soluções descentralizadas, de maneira a oferecer uma visão crítica sobre as vantagens e limitações de cada rede. Ao entender os pontos fortes e fracos da *Ethereum* e do *Bitcoin* no contexto de contratos inteligentes, espera-se contribuir para o aprimoramento das soluções em blockchain de alta performance.

OBJETIVO GERAL

Analisar o desempenho de contratos inteligentes nas redes Ethereum e Bitcoin, comparando o custo das transações, a estrutura dos contratos e a velocidade de execução, a fim de fornecer insights sobre a eficiência e a escalabilidade de soluções descentralizadas em diferentes plataformas blockchain.

OBJETIVOS ESPECÍFICOS

- Realizar experimentos práticos, através de testes de contratos inteligentes nas redes *Ethereum* e *Bitcoin*.
- Comparar o custo de transações em ambas as redes, em diferentes tipos de contratos, mesmo sobre as particularidades do uso de *gas* na rede *Ethereum* e o custo das transações na rede *Bitcoin*.
- Executar os contratos com diferentes cargas e registrar os dados de desempenho.
- Medir o tempo necessário para a execução de transações dos contratos inteligentes em ambas as redes para comparar a eficiência de cada plataforma em termos de tempo de processamento.
- Realizar testes de Ataque (stress) em contratos equivalentes

ORGANIZAÇÃO DO TRABALHO

Este trabalho está organizado em seis capítulos, estruturados de modo a oferecer uma abordagem progressiva sobre o tema da análise de desempenho de contratos inteligentes nas redes blockchain BSC (*Binance Smart Chain*) e BSV (*Bitcoin Satoshi Vision*).

No capítulo 1 são discutidos os conceitos fundamentais relacionados à tecnologia blockchain, como funções hash, assinaturas digitais, carteiras digitais e mecanismos de consenso. Também são apresentados os modelos de transação *account-based* e UTXO, os princípios de contratos inteligentes e as arquiteturas Ethereum Virtual Machine (EVM) e Bitcoin Virtual Machine (BVM). O capítulo aborda ainda aspectos de escalabilidade, soluções de segunda camada, *sharding* e a análise comparativa de *turing-completude*. Em seguida, são descritos os trabalhos relacionados, incluindo estudos fundamentais sobre blockchain, pesquisas recentes em contratos inteligentes e comparações específicas entre as arquiteturas BSV e BSC.

O capítulo 2 detalha os elementos empregados para a realização dos experimentos. Inicialmente, são apresentados os critérios de seleção dos contratos inteligentes utilizados (mensagem e contador). Em seguida, descrevem-se o ambiente de desenvolvimento, as tecnologias e ferramentas empregadas, tais como Solidity, sCrypt, Hardhat, React.js, além das plataformas BSCscan Testnet e WhatsOnChain. O capítulo inclui também a utilização de carteiras digitais, chaves privadas e *faucets* como parte do processo experimental.

No capítulo 3 são descritos a configuração prática do ambiente, tanto na rede BSV quanto na BSC, bem como a interface web desenvolvida para interação com os contratos. Posteriormente, descreve-se a metodologia de testes adotada, contemplando a execução dos contratos de mensagem e contador, bem como os experimentos de medição de tempo de execução, custo por transação e complexidade de código. Este capítulo inclui ainda a análise de ocupação em rede, testes de estresse e experimentos com funções de ataque (*attack 10x, 50x e 100x*), possibilitando uma avaliação mais abrangente do desempenho das arquiteturas analisadas.

No capítulo 4 reúne os principais resultados obtidos, discutindo as implicações da análise comparativa entre as arquiteturas BSC e BSV no contexto de desempenho de contratos inteligentes.

No Capítulo 5 são sugeridas direções de continuidade para pesquisas na área, incluindo possíveis melhorias metodológicas e novas abordagens de experimentação.

Por fim, o Capítulo 6 apresenta todas as obras consultadas para embasar teoricamente e metodologicamente o estudo, seguindo as normas da ABNT.

1 FUNDAMENTAÇÃO TEÓRICA

O presente capítulo tem como objetivo apresentar os fundamentos conceituais necessários para a compreensão do estudo, bem como discutir os elementos técnicos que sustentam a análise comparativa entre as arquiteturas de contratos inteligentes em diferentes plataformas de blockchain. Para tanto, serão abordados conceitos estruturais, mecanismos de funcionamento e particularidades tecnológicas que permitem contextualizar os experimentos desenvolvidos.

1.1 BLOCKCHAIN: CONCEITOS RELACIONADOS

A tecnologia blockchain se destaca como uma das maiores inovações do século XXI, pois revolucionou a forma como registramos e compartilhamos informações. Em essência, a blockchain funciona como um grande livro razão digital, descentralizado e distribuído, onde os dados são organizados em blocos conectados por meio de criptografia. Cada bloco reúne um conjunto de transações, uma marca temporal (*timestamp*) e o *hash* do bloco anterior, que formam uma cadeia cronológica quase impossível de ser alterada retroativamente. Essa arquitetura garante não só a integridade dos dados, mas também uma transparência inédita nas informações acessíveis aos participantes da rede. Como ressaltam Narayanan et al. (2016), a blockchain muda completamente o paradigma da confiança em sistemas digitais, ao diminuir ou até eliminar a dependência de intermediários centralizados.

A descentralização é um dos grandes diferenciais da blockchain. Em vez de confiar em uma única autoridade para validar transações ou manter registros, essa tecnologia distribui essas funções entre diversos computadores (nós) espalhados pela rede. Isso torna o sistema muito mais robusto, resistente a falhas pontuais e a ataques maliciosos. Tapscott (2016) destaca que essa descentralização abre portas para novas formas de governança, muito mais transparentes e colaborativas, com impacto direto em áreas como bancos, cartórios, registros públicos e até mesmo eleições.

Outro ponto central da blockchain é a imutabilidade dos dados. Uma vez que uma informação é registrada, não é possível alterá-la facilmente sem modificar toda a cadeia subsequente, o que só pode ser feito se a maioria da rede concordar. Esse princípio é assegurado por funções *hash* criptográficas e um mecanismo de consenso robusto, como *Proof*

of Work (PoW). Crosby et al. (2016) apontam que essa característica torna a blockchain especialmente segura para aplicações como transações financeiras, registros médicos, rastreamento de produtos e contratos jurídicos.

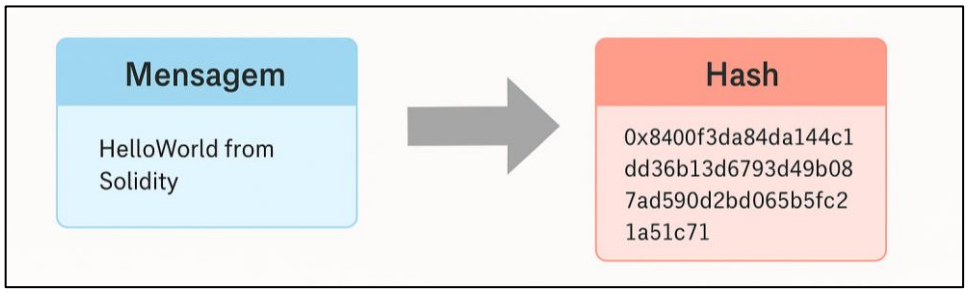
Por fim, o potencial transformador da blockchain está em sua capacidade de servir como uma infraestrutura digital universal, sobre a qual podem ser criadas soluções para os mais diversos setores. Desde seu uso como livro razão financeiro e de dados até sua integração com Internet das Coisas (IoT), inteligência artificial e *big data*, a blockchain está pavimentando o caminho para um novo modelo de confiança, descentralização e automação. Conforme observa Antonopoulos (2017), estamos apenas começando a presenciar uma revolução que promete remodelar sistemas econômicos, jurídicos e sociais.

1.1.1 Função *hash*

A função *hash* criptográfica desempenha um papel fundamental na garantia da integridade, segurança e imutabilidade dos dados transacionados em rede. Trata-se de uma operação algorítmica que transforma uma entrada de tamanho arbitrário (como um texto, uma transação ou um bloco) em uma saída de comprimento fixo, comumente expressa em forma hexadecimal. Essa saída é chamada de *hash*, e deve ser única para cada entrada distinta, ainda que pequenas alterações na entrada gerem mudanças drásticas no valor final.

Conforme mostra a Figura 1, mesmo uma entrada simples, como a string "*HelloWorld from Solidity*", pode ser condensada em um hash de 256 bits, o que evidencia o comportamento determinístico e sensível da função.

Figura 1 - Exemplo de geração de *hash*



Fonte: própria.

O principal papel da função *hash* na estrutura da blockchain é encadear os blocos de forma segura. Cada bloco armazena em seu cabeçalho o código *hash* do bloco anterior,

formando uma cadeia sequencial e cronológica. Assim, qualquer modificação em um bloco exigiria a alteração de todos os códigos *hash* subsequentes, o que tornaria o ataque computacionalmente inviável, especialmente em redes que utilizam mecanismos de consenso como o PoW. Essa propriedade garante a imutabilidade dos dados e a confiança descentralizada, pilares centrais da proposta das tecnologias blockchain.

Além da função estrutural nos blocos, os códigos *hash* também são amplamente utilizados na identificação de transações (TxID) e na assinatura digital de contratos inteligentes. Ao utilizar as redes BSV e BSC, foram aplicadas funções *hash* distintas para geração e manipulação de chaves privadas e públicas. No caso da arquitetura BVM (BSV), a geração de chaves foi baseada na função SHA-256 (*Secure Hash Algorithm*), amplamente utilizada no ecossistema *Bitcoin*. Por outro lado, na arquitetura EVM (BSC), a função *hash* predominante foi a Keccak-256, variação da família SHA-3, adotada pela *Ethereum* para o cálculo de endereços e verificação de dados.

1.1.2 Assinatura digital

As assinaturas digitais constituem um mecanismo criptográfico essencial à segurança das transações em redes blockchain. Fundamentadas na criptografia assimétrica, utilizam um par de chaves: uma privada, mantida em segredo pelo usuário, e uma pública, amplamente divulgada para autenticar a origem e garantir a integridade de mensagens ou transações. Segundo Schneier (1996), esse modelo impede a falsificação de assinaturas e assegura a não-repudiação, ou seja, a impossibilidade de o signatário negar a autoria da transação.

Em blockchains públicas, como o *Bitcoin*, a assinatura digital é aplicada a cada transação para garantir que apenas o proprietário legítimo da chave privada possa transferir os ativos associados a determinado endereço. O processo de verificação da assinatura é realizado por qualquer nó participante da rede através da chave pública correspondente, que garante assim, a transparência e a confiança no ambiente descentralizado. Essa operação não exige a revelação da identidade do usuário, o que proporciona um equilíbrio entre segurança e um anonimato relativo.

No campo dos contratos inteligentes, a assinatura digital é fundamental para validar instruções que resultam na execução automática de acordos codificados. Gennaro et al. (2016) observam que, em aplicações complexas, como finanças descentralizadas (DeFi), jogos baseados em blockchain e sistemas de governança autônoma, as assinaturas digitais funcionam

como elementos de orquestração das interações contratuais, o que proporciona que múltiplas partes assinem e autentiquem suas ações sem necessidade de um ente centralizador.

Importante salientar que a eficácia das assinaturas digitais está condicionada à robustez dos algoritmos, como RSA, ECDSA ou EdDSA, bem como à adequada gestão de chaves pelos usuários. Falhas na preservação da chave privada, como sua exposição a *malware* ou armazenamento inseguro, podem comprometer a segurança do sistema como um todo. Por esse motivo, carteiras digitais (*wallets*) empregam medidas adicionais, como autenticação multifatorial e carteiras frias (*cold wallets*), para mitigar tais riscos.

Com efeito, a assinatura digital não apenas assegura a legitimidade das transações e execuções contratuais em ambientes distribuídos, como também viabiliza o modelo *trustless* que caracteriza as redes blockchain. Sua aplicação transcende o campo técnico, tendo implicações diretas na governança digital, na certificação de documentos e em sistemas eleitorais baseados em blockchain, consolidando-se como um instrumento vital para a era da descentralização.

1.1.3 Carteiras digitais (*wallets*)

As carteiras digitais (ou *wallets*) constituem um dos elementos centrais na arquitetura de sistemas baseados em blockchain. Elas funcionam como interfaces digitais que permitem aos usuários armazenar, enviar e receber ativos criptográficos de forma segura, prática e descentralizada. Em essência, uma carteira não armazena fisicamente as criptomoedas, mas sim gerencia as chaves criptográficas que permitem a interação com os registros públicos da blockchain. Através dessa tecnologia, o usuário é capaz de assinar transações digitalmente e verificar saldos em tempo real, por meio de uma interface geralmente amigável e intuitiva (ANTONOPOULOS; WOOD, 2018).

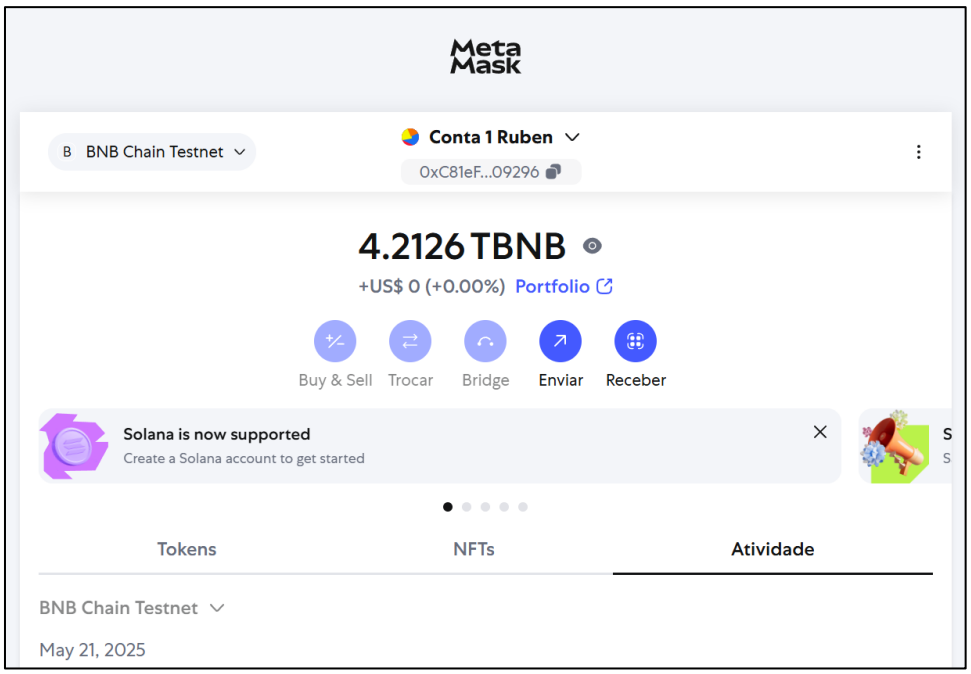
Dessa forma, as carteiras digitais exercem papel fundamental tanto na administração de ativos quanto na validação de transações, sendo indispensáveis para qualquer operação descentralizada com criptomoedas (NAKAMOTO, 2008).

Existem diferentes tipos de carteiras digitais, que variam de acordo com o nível de segurança e acessibilidade. As carteiras quentes são carteiras conectadas à internet, mais indicadas para uso diário e transações rápidas, embora apresentem maior vulnerabilidade a ataques. Já as carteiras frias operam de forma *offline*, garantindo maior proteção ao serem mantidas fora do alcance de redes conectadas que é uma abordagem ideal para armazenamento de longo prazo (MOUGAYAR, 2016). Entre essas, destacam-se as carteiras de hardware, que

armazenam chaves privadas em dispositivos físicos, como pen drives criptografados (ex.: Ledger e Trezor), e as *paper wallets*, que consistem em chaves geradas e impressas em papel, com o intuito de evitar a exposição digital.

No mercado atual, diversas soluções de carteira se destacam. A *MetaMask*, por exemplo, é amplamente utilizada na rede *Ethereum* e compatível com a *BNB Smart Chain* (BSC), oferecendo integração com navegadores e aplicações *Web3*. Essa carteira permite o gerenciamento de múltiplos endereços e *tokens*, e é uma das mais populares entre desenvolvedores e usuários finais (BINANCE, 2025). A Figura 2 demonstra a interface de visualização da carteira *Metamask* com a configuração na rede *Binance Smart Chain* e o saldo disponível para utilização nos testes realizados.

Figura 2 - Interface de carteira Web Metamask.



Fonte: própria.

Para usuários da rede *Bitcoin Satoshi Vision* (BSV), a carteira *Exodus* oferece suporte compatível e interface simplificada, sendo uma opção viável para interação com contratos inteligentes em BSV. É importante destacar que o funcionamento das carteiras digitais está baseado em dois elementos-chave: a chave pública, que funciona como um endereço visível para envio de criptomoedas, e a chave privada, que é o código secreto utilizado para autorizar transações e acessar fundos. Embora carteiras como a *MetaMask* armazenem essas chaves localmente e criptografadas no navegador, há constantes discussões sobre a segurança desses

dados, pois o acesso às chaves privadas deve ser exclusivo do usuário (CHRISTIDIS; DEVETSIKIOTIS, 2016).

1.1.4 Mecanismos de consenso

Os mecanismos de consenso constituem a espinha dorsal das redes blockchain, possibilitando que múltiplos nós distribuídos e independentes concordem sobre o estado único e atualizado sem a necessidade de autoridade centralizada. Essa funcionalidade é vital para a confiabilidade, segurança e descentralização do sistema, garantindo que todas as transações validadas estejam registradas de forma imutável e sincronizada entre os participantes.

O consenso é um problema clássico em sistemas distribuídos e sua resolução em blockchains incorpora características específicas, tais como resistência a falhas bizantinas e a presença de agentes maliciosos. De acordo com Cachin et al. (2011), mecanismos eficientes precisam balancear a tolerância a falhas, a escalabilidade e a descentralização, além de otimizar o consumo energético, o que torna o design do protocolo de consenso uma tarefa complexa e multidimensional.

O *Proof of Work*, ou Prova de Trabalho, é um mecanismo de consenso utilizado por diversas redes blockchain, como o *Bitcoin*, para validar transações e garantir a segurança da rede. Esse modelo exige que os participantes, chamados de mineradores, realizem cálculos matemáticos complexos a fim de encontrar um código *hash* que satisfaça determinados critérios definidos pelo protocolo. O processo é competitivo: o primeiro minerador a encontrar a solução válida propaga o bloco para a rede, que é então verificado e adicionado à cadeia de blocos. Esse esforço computacional dificulta ataques e fraudes, tornando a rede resistente à manipulação e ao gasto duplo.

O *Proof of Stake*, ou Prova de Participação, é um mecanismo de consenso alternativo ao *Proof of Work* (PoW), desenvolvido para validar transações e manter a segurança das redes blockchain de forma mais eficiente e sustentável. Em vez de depender de poder computacional para resolver problemas matemáticos, como no PoW, o PoS seleciona validadores com base na quantidade de criptomoedas que possuem e estão dispostos a travar (ou *stakear*) como garantia de comportamento honesto. Quanto maior o valor em *stake*, maior a probabilidade de o participante ser escolhido para validar o próximo bloco.

Embora o *Proof of Stake* (*PoS*) faça a alegação de ser uma alternativa mais eficiente e sustentável ao *Proof of Work (PoW*), essa generalização tem sido contestada por evidências recentes. O *PoS* elimina a necessidade de trabalho computacional intenso ao permitir que a

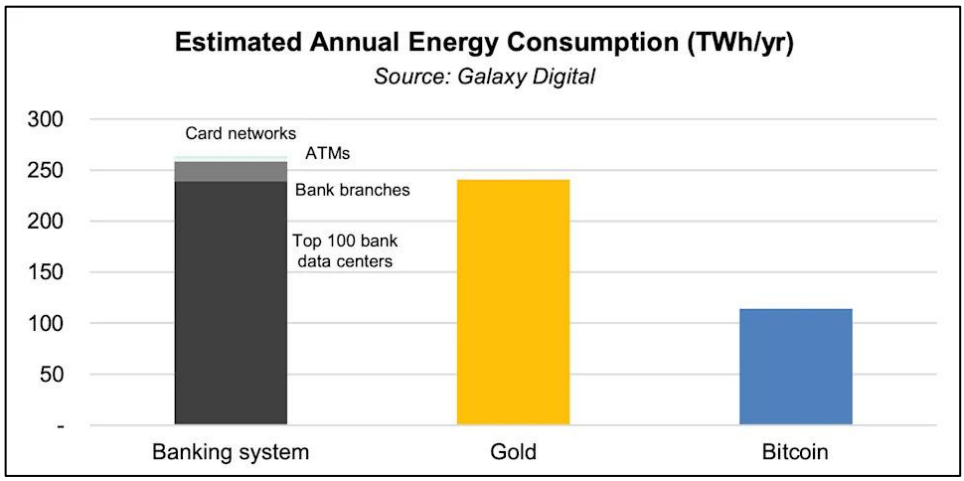
criação de novos blocos seja proporcional à quantidade de *tokens* travados em *stake*, o que reduz o consumo energético direto (Ibañez & Freier, 2023). No entanto, essa redução energética por si só não implica necessariamente maior sustentabilidade ambiental.

Estudos comparativos indicam que o sistema bancário tradicional, o setor de mineração de ouro e até mesmo *data centers* centralizados consomem mais energia do que a rede *Bitcoin* baseada em *PoW*. Segundo análise divulgada pela Nasdaq (2023), o *Bitcoin* consome menos da metade da energia utilizada pelos sistemas bancários e pela indústria do ouro, o que questiona sua repulsão energética diante de outros sistemas legados.

A empresa *Galaxy Digital* realizou uma análise comparativa do consumo energético da rede *Bitcoin* em relação ao sistema bancário tradicional e à indústria do ouro, tendo em vista que a maior criptomoeda do mundo é frequentemente comparada a esses dois setores. De acordo com o relatório, o sistema bancário consome aproximadamente 263,72 TWh por ano, enquanto a indústria do ouro consome cerca de 240,61 TWh por ano. Em contraste, a rede *Bitcoin* apresenta um consumo significativamente menor, estimado em 113,89 TWh anuais.

A Figura 3 ilustra esse comparativo em forma de gráfico, evidenciando a disparidade no consumo energético entre os três setores analisados.

Figura 3 - Análise da Galaxy Digital sobre o consumo de energia do Bitcoin, em comparação com o sistema bancário e a indústria do ouro.



Fonte: Nasdaq (2023).

Esses autores destacam que o *Bitcoin* pode atuar como uma carga flexível na malha energética, absorvendo excedentes de produção renovável e incentivando a expansão dessas fontes. Ao operar como consumidor de último recurso, utilizando energia que seria desperdiçada, a mineração de *Bitcoin* pode contribuir para a descarbonização da matriz

energética, algo que os sistemas *PoS*, por sua natureza passiva e dependente de infraestrutura tradicional, não oferecem diretamente.

Buterin (2014) foi um dos precursores no desenvolvimento do PoS para *Ethereum*, enfatizando a capacidade do modelo em preservar segurança e descentralização ao mesmo tempo que oferece maior escalabilidade. Contudo, o PoS apresenta desafios próprios, como a potencial concentração de poder em grandes detentores de *tokens*, o que pode comprometer a equidade do sistema, conforme analisado por Saleh (2021).

O *Proof of Staked Authority* (PoSA) é um mecanismo de consenso híbrido que combina elementos do Proof of Stake (PoS) e da Proof of Authority (PoA), com o objetivo de oferecer maior eficiência, segurança e escalabilidade às redes blockchain. No modelo PoSA, os validadores são escolhidos com base em um *stake* de criptomoedas (como no PoS), mas também precisam passar por um processo de seleção e aprovação por parte da rede (como no PoA). Isso significa que apenas entidades confiáveis e previamente autorizadas podem validar blocos, desde que mantenham uma certa quantidade de *tokens* em *stake* como garantia de comportamento honesto.

Essa abordagem reduz o risco de centralização extrema (presente no PoA puro) e ao mesmo tempo, mantém a alta performance e baixo consumo energético característicos dos mecanismos baseados em participação. A *B*SC é um dos exemplos mais notórios da aplicação do PoSA, combinando governança delegada e *staking* para validar transações com taxas baixas e alta velocidade (BINANCE, 2025; LI et al., 2020).

Além dos modelos tradicionais, outras abordagens têm ganhado destaque, por otimizarem o desempenho e mitigarem as limitações inerentes ao PoW e PoS. Protocolos como *Delegated Proof of Stake* (DPoS), *Practical Byzantine Fault Tolerance* (PBFT) e variantes híbridas buscam equilibrar descentralização, segurança e eficiência operacional.

A pesquisa contemporânea sugere que a combinação adaptativa desses mecanismos, utilizando modelos híbridos, pode ser a chave para futuras evoluções em blockchains, ajustando dinamicamente os parâmetros do consenso para diferentes contextos e demandas de rede (Li et al., 2020).

1.1.5 *Accounted Based versus* UTXO

O modelo *account-based*, exemplificado pela blockchain *Ethereum*, estrutura-se em torno de contas que armazenam diretamente os saldos e o estado das entidades participantes da rede. Cada usuário é identificado por uma conta única que mantém o registro do saldo atual, do

nonce (contador que previne repetição de transações) e do estado dos contratos inteligentes associados. As transações nesse modelo atuam como instruções para atualizar o estado das contas envolvidas, promovendo mudanças explícitas e diretas no saldo e nos dados armazenados.

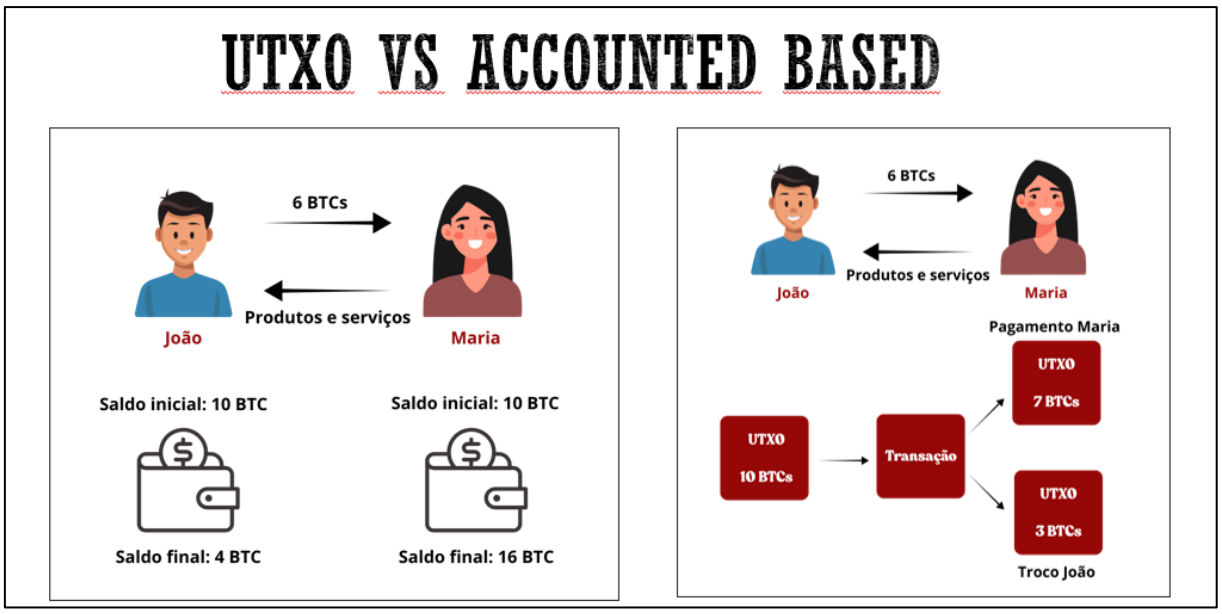
Buterin (2013) destaca que esse paradigma proporciona uma abordagem mais intuitiva e flexível para a execução de contratos inteligentes, pois o sistema pode modificar e consultar o estado global de maneira direta e contínua. Tal característica é especialmente benéfica para aplicações que demandam operações complexas, como *dApps* (aplicações descentralizadas), que requerem rastreamento de múltiplos estados interdependentes.

Contudo, Antonopoulos e Wood (2018) ressaltam que a simplicidade aparente do modelo *account-based* traz consigo desafios intrínsecos relacionados à concorrência e sincronização do estado. Como múltiplas transações podem modificar simultaneamente o estado de uma mesma conta, são necessários mecanismos rigorosos para evitar inconsistências, o que demanda soluções robustas para escalabilidade e segurança da rede.

Conforme Cruz (2024) em sua comparação direta entre UTXO e *Accounted-based*, o modelo UTXO (*Unspent Transaction Output*) funciona como se fosse o troco de uma compra: cada transação consome saídas anteriores e gera novas saídas que ainda não foram gastas. Ele é usado por blockchains como o *Bitcoin* e ajuda a manter um alto nível de segurança e rastreabilidade. Já o modelo *Account-Based*, utilizado pela *Ethereum*, é mais parecido com o sistema bancário tradicional, onde cada conta tem um saldo que é atualizado conforme as transações são feitas. Esse modelo é mais direto e simples de programar, especialmente quando se trata de criar contratos inteligentes. Em resumo, o UTXO oferece mais controle e segurança técnica, enquanto o modelo por contas facilita o desenvolvimento e a gestão dos saldos.

A Figura Figura 4 – Exemplificação de UTXO Vs Accounted Based ilustra um exemplo de comparação. No modelo *Account-Based*, representado à esquerda, a transferência de 6 BTCs de João para Maria é registrada de forma semelhante a uma conta bancária tradicional: o saldo de João é simplesmente reduzido de 10 para 4 BTCs, enquanto o de Maria aumenta de 10 para 16 BTCs. Já no modelo UTXO, representado à direita, a transação é estruturada em saídas de transações não gastas. Nesse caso, João utiliza um UTXO de 10 BTCs para pagar 6 BTCs a Maria, mas como a transação deve consumir o valor inteiro, são criadas duas novas saídas: um UTXO de 7 BTCs para Maria (pagamento) e outro de 3 BTCs como “troco” para João.

Figura 4 – Exemplificação de UTXO Vs Accounted Based



Fonte: própria.

Assim, apesar da vantagem em termos de legibilidade e gestão do estado, o modelo *account-based* exige uma arquitetura que suporte sincronização eficiente e controle preciso da ordem de processamento das transações, garantindo a integridade dos dados.

O modelo UTXO, originado com a concepção do Bitcoin por Nakamoto (2008), difere fundamentalmente do *account-based* ao registrar transações em termos de entradas e saídas não gastas. Cada transação consome um conjunto de saídas não gastas anteriores e gera novas saídas que podem ser gastas futuramente, de modo que o saldo de um usuário é determinado pela soma dessas saídas não gastas em seu controle.

Esse modelo promove uma abordagem fragmentada para o controle de fundos, favorecendo a privacidade e a paralelização da verificação das transações, pois diferentes UTXO podem ser processados de forma independente. Bonneau et al. (2015) enfatizam que o UTXO oferece benefícios significativos no que tange à segurança, dado que cada saída pode ser verificada isoladamente. Isso aumenta a auditabilidade e dificulta a ocorrência de fraudes.

Por outro lado, a estrutura do UTXO impõe limitações para a implementação de contratos inteligentes complexos. Como não há um estado global diretamente acessível, as condições contratuais precisam ser expressas por meio da composição e validação das transações UTXO. Isso exige abordagens alternativas e mais complexas para rastrear e gerenciar a lógica contratual.

1.1.6 Smart Contracts

Contratos inteligentes (ou *smart contracts*) são protocolos computacionais que executam, verificam e aplicam termos de um acordo de forma automática, sem necessidade de intermediários humanos. Essas estruturas são programadas para responder a condições específicas e, uma vez implantadas em uma blockchain, funcionam de maneira imutável, transparente e descentralizada. O conceito foi originalmente introduzido por Nick Szabo, ainda na década de 1990, como uma proposta para traduzir cláusulas contratuais em código executável. Foi viabilizado com o surgimento da tecnologia blockchain. (SZABO, 1996).

Em termos técnicos, um contrato inteligente opera por meio de instruções lógicas escritas em linguagens como *Solidity*, no caso da *Ethereum*. Esses códigos são compilados em *bytecode*, armazenados na blockchain e processados por máquinas virtuais como a *Ethereum Virtual Machine* (EVM), que asseguram a execução uniforme em todos os nós da rede (WOOD, 2014).

As aplicações de contratos inteligentes abrangem diversas áreas. No setor financeiro, são utilizados para criar ativos digitais, executar transações automáticas de empréstimos, seguros e derivativos. No campo da logística, viabilizam o rastreamento de mercadorias com registro confiável de cada etapa do transporte. Na administração pública, surgem como solução para licitações transparentes, votações seguras e gestão de registros públicos. Já no contexto jurídico, oferecem ferramentas para conformidade automatizada de cláusulas contratuais e gestão de obrigações civis (CHRISTIDIS; DEVETSIKIOTIS, 2016).

A natureza programável dos contratos inteligentes também permite a criação de sistemas autônomos e descentralizados, como as DAOs (*Organizações Autônomas Descentralizadas*), onde regras de governança são codificadas diretamente na blockchain. Tais modelos possibilitam tomadas de decisão coletivas sem liderança centralizada, representando uma nova abordagem para estruturas organizacionais (MENDLING et al., 2018).

No entanto, apesar de seu potencial disruptivo, os contratos inteligentes enfrentam desafios significativos. A rigidez da execução automática, a dificuldade de interpretação jurídica de cláusulas codificadas e os riscos de vulnerabilidades no código são pontos críticos. Por isso, práticas como auditorias independentes e testes rigorosos tornam-se indispensáveis para mitigar falhas que possam comprometer ativos ou processos dependentes desses contratos (CHRISTIDIS; DEVETSIKIOTIS, 2016).

1.1.7 Arquitetura Ethereum Virtual Machine (EVM)

A *Ethereum Virtual Machine* (EVM) constitui o núcleo de execução da blockchain *Ethereum*, sendo projetada para interpretar e processar contratos inteligentes de forma descentralizada e segura. Diferentemente das máquinas tradicionais, a EVM é baseada em uma pilha e opera dentro de um ambiente completamente isolado. Isso garante que os códigos executados não interfiram com o sistema host nem com outros contratos. Esse isolamento é essencial para assegurar a confiabilidade e a previsibilidade das execuções em todos os nós da rede.

Cada contrato inteligente na *Ethereum* é transformado em *bytecode*, que é executado pela EVM de maneira idêntica em todos os participantes da rede, o que promove o consenso distribuído. Esse *bytecode* é produzido a partir de linguagens de programação de alto nível, como *Solidity* e *Vyper*. A primeira, tornou-se padrão devido à sua acessibilidade e semelhança com linguagens como *JavaScript* e C++. A compilação do código em instruções específicas da EVM permite que desenvolvedores escrevam lógicas contratuais com alta expressividade, ao mesmo tempo em que mantêm controle sobre custos e riscos.

A EVM utiliza um sistema de contagem chamado gas, que mensura a quantidade de esforço computacional necessário para executar cada instrução de um contrato. Esse sistema impede abusos de recursos, como laços infinitos, e ainda representa uma ferramenta de incentivo econômico, pois o pagamento pelo gas deve ser feito em *Ether*. Como resultado, esse mecanismo promove um equilíbrio entre poder computacional, segurança e eficiência dentro da rede *Ethereum*.

Pesquisadores têm se dedicado a formalizar o comportamento da EVM, propondo modelos matemáticos que descrevem com precisão o funcionamento de cada instrução e seus impactos no estado global da rede. Um exemplo significativo é o KEVM, uma implementação formal da EVM baseada no *K Framework*, que permite a verificação automática de propriedades de segurança dos contratos inteligentes. Esses avanços são fundamentais para aumentar a confiabilidade em sistemas descentralizados que operam sem a supervisão de autoridades centrais.

Com a transição para o *Ethereum* 2.0, surgem novas propostas de arquitetura para a EVM, focando na escalabilidade e na redução do consumo energético. O futuro prevê versões otimizadas que possam operar em sistemas com *sharding* (técnica que fragmenta a rede em múltiplas partes menores – *shards* – cada uma capaz de processar transações e contratos inteligentes de forma paralela, que permite aumentar significativamente a capacidade do

blockchain) ou até mesmo interagir com outras blockchains por meio de mecanismos de interoperabilidade. Assim, a EVM se estabelece não apenas como um componente técnico, mas como um motor de inovação e adaptação contínua no ecossistema blockchain.

1.1.8 Arquitetura Bitcoin Virtual Machine (BVM)

A *Bitcoin Virtual Machine* (BVM) é o ambiente de execução responsável por interpretar e validar os scripts presentes nas transações da rede *Bitcoin*, inclusive em sua vertente *Bitcoin Satoshi Vision* (BSV). Ao contrário da *Ethereum Virtual Machine* (EVM), a BVM adota um modelo intencionalmente mais restrito, não Turing-completo, o que significa que não é capaz de executar laços infinitos nem armazenar estados persistentes de forma autônoma. Essa limitação é, na verdade, uma escolha de design voltada à segurança e à previsibilidade da execução das transações.

O *Bitcoin Script*, linguagem utilizada pela BVM, é baseado em uma estrutura de comandos empilhados (*stack-based*), que são lidos sequencialmente para verificar se as condições de uma transação foram atendidas. Trata-se de uma linguagem de propósito específico, que foca na validação de transações de forma eficiente e auditável. Isso torna a rede *Bitcoin* especialmente robusta em termos de segurança e resistência a ataques, já que seu ambiente de execução é muito menos suscetível a falhas lógicas e vulnerabilidades complexas.

No caso específico do Bitcoin Satoshi Vision (BSV), a estrutura original proposta por Satoshi Nakamoto foi restaurada e expandida para permitir contratos inteligentes simples, mas poderosos. O BSV busca alcançar escalabilidade massiva, permitindo blocos muito maiores e taxas mínimas de transação, o que o torna adequado para aplicações comerciais que demandam volume elevado de operações. Craig Wright, uma das principais figuras por trás do BSV, defende que a visão original de Satoshi estava focada em um sistema de registro público de dados e contratos, e não apenas em uma moeda digital.

Embora o BSV mantenha a estrutura da BVM e do Bitcoin Script, ele também introduziu extensões e ferramentas auxiliares, como o *sCrypt*, que permite o desenvolvimento de contratos inteligentes mais expressivos sem comprometer a segurança do modelo original. O uso dessas ferramentas expande a capacidade de construção de aplicações descentralizadas (*dApps*). Dessa forma, o BSV tenta equilibrar entre a segurança da simplicidade e a funcionalidade de contratos automatizados.

No panorama geral, a *Bitcoin Virtual Machine*, especialmente no contexto do BSV, representa uma abordagem conservadora ao uso de contratos inteligentes. Enquanto

blockchains como *Ethereum* priorizam flexibilidade e inovação técnica, o BSV aposta na estabilidade, previsibilidade e escalabilidade como fatores estratégicos para adoção em larga escala. Isso reflete diferentes filosofias de projeto dentro do ecossistema blockchain: uma voltada à experimentação e outra à confiabilidade e desempenho sob condições de uso real.

1.1.9 Escalabilidade e Soluções de Segunda Camada

O aumento da popularidade das blockchains públicas, sobretudo *Bitcoin* e *Ethereum*, expôs limitações significativas em sua capacidade de processar grandes volumes de transações simultâneas, gerando problemas como congestionamento e aumento das taxas de operação. Para resolver essas limitações sem comprometer a segurança e descentralização, surgiram as soluções de segunda camada.

A *Lightning Network*, proposta para *Bitcoin*, é um exemplo de rede de canais *off-chain* que permite a execução de transações instantâneas e com baixíssimo custo. Ao criar canais diretos entre usuários, as transações podem ocorrer fora da cadeia principal, de forma a reduzir a carga e aumentar a velocidade (POON; DRYJA, 2016). Somente as movimentações finais são registradas na blockchain, o que mantém a segurança e validade do sistema.

Na rede *Ethereum*, soluções chamadas de *Rollups* funcionam de forma similar, agrupando múltiplas transações em lotes que são processados fora da cadeia principal, enquanto dados comprovantes são inseridos na camada base para garantir integridade (BUTERIN; STARK, 2020). Essa estratégia promove um aumento considerável na capacidade da rede, com redução dos custos para os usuários.

A Tabela 1 apresenta uma comparação entre diferentes blockchains quanto à métrica de escalabilidade mais utilizada: o número de transações por segundo (Transactions Per Second – TPS)

Tabela 1 - Comparação de TPS entre Blockchains.

Comparação de TPS entre Blockchains		
Blockchain	TPS (médio estimado)	TPS (máximo teórico)
Bitcoin	7	10
Bitcoin Satoshi Vision (BSV)	100	300
Binance Smart Chain (BSC)	5000	50000

Fonte: Adaptado de (Nakamoto,Whatsonchain e BNBchain)

Essas abordagens são essenciais para a adoção massiva da tecnologia blockchain, principalmente em aplicações comerciais e financeiras que demandam alta velocidade e baixo custo. Ao equilibrar desempenho e segurança, as soluções de segunda camada contribuem para superar os principais entraves técnicos das blockchains atuais.

Contudo, desafios técnicos e de adoção permanecem, principalmente relacionados à interoperabilidade entre as camadas, usabilidade para usuários finais e padrões regulatórios, o que estimula contínuas pesquisas e inovações no campo.

1.1.10 Sharding

Sharding é um conceito extraído do campo de bancos de dados distribuídos e aplicado ao universo blockchain como uma solução para o problema da escalabilidade. Ao invés de todos os nós processarem todas as transações, a rede é dividida em múltiplas partições chamadas *shards*, cada uma responsável por um subconjunto dos dados e operações (ZAMANI et al., 2018). Isso permite que o processamento ocorra em paralelo, aumentando significativamente a capacidade e o desempenho da rede.

No *Ethereum* 2.0, o *sharding* é parte integral da estratégia de atualização da rede para *Proof of Stake* (PoS). Vitalik Buterin (2020) destaca que essa arquitetura fragmentada permitirá que múltiplos *shards* executem contratos inteligentes simultaneamente, aumentando o *throughput* e reduzindo os custos de transação. O modelo contempla ainda mecanismos de coordenação e comunicação entre *shards* para garantir a coesão e segurança global da rede.

Além do aumento do desempenho, o *sharding* pode contribuir para a descentralização, pois torna possível que nós individuais participem validando apenas uma fração da blockchain, reduzindo os requisitos computacionais e de armazenamento necessários (ZAMANI et al., 2018).

Contudo, o *sharding* implica desafios técnicos complexos, principalmente relacionados à segurança da comunicação entre *shards*, prevenção de ataques específicos e sincronização dos dados. Tais aspectos continuam sendo objeto de pesquisa ativa para assegurar a robustez e confiabilidade das soluções adotadas.

Assim, o *sharding* emerge como um dos pilares para a evolução das blockchains públicas, prometendo superar limitações atuais e possibilitar o uso em larga escala.

1.1.11 Análise comparativa de turing-completude entre btc e eth

A noção de Turing-completude desempenha papel importante na avaliação da expressividade computacional de linguagens utilizadas em blockchains. De acordo com Sipser (2012), um sistema é considerado Turing-completo se pode expressar qualquer função computável, ou seja, se for capaz de executar algoritmos arbitrariamente complexos utilizando apenas operações básicas de leitura, escrita, controle condicional e repetição.

Redes como *Ethereum* e suas variantes compatíveis, como a *Binance Smart Chain* (BSC), implementam máquinas virtuais formalmente Turing-completas, como é o caso da *Ethereum Virtual Machine* (EVM). Conforme demonstrado por Wood (2014), a EVM permite o uso de estruturas como loops, recursão e operações aritméticas complexas, provendo, assim, um ambiente totalmente expressivo para computação descentralizada. No entanto, essa completude é limitada na prática pela introdução do mecanismo de gas, que atua como limitador de recursos computacionais disponíveis por transação. De acordo com Buterin (2014), essa limitação foi implementada justamente para evitar que contratos entrem em ciclos infinitos ou consumam recursos excessivos, garantindo a sustentabilidade da rede.

Por outro lado, a blockchain do *Bitcoin Satoshi Vision* (BSV) adota um modelo baseado em UTXO e uma linguagem de script não Turing-completa em sua forma original, herança direta do *Bitcoin* de Satoshi Nakamoto. Entretanto, conforme argumentado por Sincek (2022), a introdução de linguagens de alto nível como a *sCrypt* permite a simulação de Turing-completude através da composição de transações sucessivas, onde o estado do contrato é armazenado no script de saída e atualizado em cada nova transação. Essa abordagem, embora válida, difere do modelo de estado contínuo da EVM e depende de um encadeamento externo de execuções para simular comportamentos computacionalmente completos.

Assim, tanto o BSV quanto o *Ethereum*/BSC enfrentam limitações práticas para a execução irrestrita de algoritmos complexos. No BSV, essas limitações são impostas pela arquitetura UTXO e pela ausência de laços nativos. No *Ethereum*/BSC, pela política de cobrança de gas e pela finitude dos blocos. Em ambos os casos, a Turing-completude existe de forma relativa ao modelo de execução, não como uma garantia absoluta.

Em síntese, embora o BSV e a EVM reivindiquem Turing-completude sob certas premissas, apenas a *Ethereum* o faz de forma nativa e direta dentro de cada transação individual.

1.2 TRABALHOS RELACIONADOS

O crescimento da adoção de tecnologias blockchain impulsionou uma série de estudos voltados ao desempenho de *smart contracts* em diferentes plataformas. Embora a *Ethereum* tenha sido a pioneira nesse campo, alternativas como a *Binance Smart Chain* (BSC) e o *Bitcoin Satoshi Vision* (BSV) têm ganhado destaque por suas características técnicas específicas e diferentes abordagens quanto a escalabilidade e ao custo computacional.

A BSC, uma rede compatível com a *Ethereum Virtual Machine* (EVM), foi projetada para oferecer maior escalabilidade e menor custo por transação. Por manter compatibilidade com *Solidity* e as ferramentas do ecossistema *Ethereum*, vários estudos como o de Wu et al. (2021) compararam seu desempenho em relação a rede *Ethereum* original, mostrando tempos de execução menores e maior *throughput*, ainda que com menor grau de descentralização.

Por outro lado, o *BSV* se diferencia das versões anteriores do *Bitcoin* por buscar a restauração do protocolo original descrito por Satoshi Nakamoto e permitir blocos significativamente maiores, com o objetivo de suportar aplicações em larga escala. Estudos como o de Wright et al. (2020) analisam a viabilidade de execução de contratos inteligentes na rede BSV, que destacam sua abordagem baseada em *scripts* mais simples, porém com suporte ampliado por meio de contratos fora da cadeia (*off-chain*) e arquiteturas híbridas.

A literatura existente ainda é incipiente no que diz respeito à comparação direta entre o desempenho de *smart contracts* em BSV e BSC. As análises geralmente se concentram em métricas como tempo de execução, *throughput*, custo por transação, eficiência computacional, e escalabilidade, mas poucas adotam metodologias padronizadas que permitam comparações diretas entre as plataformas *Ethereum* e *Bitcoin*. Essa lacuna justifica a necessidade de trabalhos como este, que buscam avaliar comparativamente essas arquiteturas sob uma abordagem empírica e controlada.

1.2.1 Blockchain: estudos fundamentais

O conceito de blockchain foi introduzido por Satoshi Nakamoto (2008) com a proposta do Bitcoin, uma moeda digital descentralizada que utiliza uma rede *peer-to-peer* para evitar o gasto duplo sem a necessidade de uma autoridade central. A tecnologia *blockchain* é caracterizada por sua estrutura de dados em blocos encadeados, imutabilidade e segurança criptográfica.

Narayanan et al. (2016) aprofundam a compreensão da arquitetura do *Bitcoin*, com detalhes sobre aspectos da estrutura dos blocos, da mineração e do mecanismo de consenso *Proof of Work* (PoW). Esses autores também destacam as limitações da linguagem de script do *Bitcoin* para aplicações mais complexas.

Vitalik Buterin (2014) propôs o *Ethereum* como uma plataforma que estende as capacidades do *Bitcoin*, introduzindo uma máquina virtual descentralizada (*Ethereum Virtual Machine – EVM*) capaz de executar contratos inteligentes. Essa inovação permitiu a criação de aplicações descentralizadas (*dApps*) com funcionalidades mais complexas.

1.2.2 *Smart Contracts*: pesquisas recentes

Os contratos inteligentes, ou *smart contracts*, são programas autoexecutáveis que operam em redes *blockchain*, permitindo a automação de acordos contratuais sem a necessidade de intermediários. Nick Szabo (1997) foi um dos pioneiros na conceituação desses contratos, destacando sua capacidade de reduzir custos de transação e aumentar a segurança.

Christidis e Devetsikiotis (2016) exploram a aplicação de contratos inteligentes no contexto da Internet das Coisas (IoT), e enfatizam sua utilidade em ambientes distribuídos. Dannen (2017) aborda a linguagem de programação *solidity*, utilizada no *Ethereum*, e discute os desafios relacionados à segurança e eficiência dos contratos inteligentes.

Hirai (2017) destaca a importância da verificação formal de contratos inteligentes, de forma a garantir sua correção e prevenir vulnerabilidades que possam comprometer a integridade da rede.

1.2.3 Arquitetura de Smart Contracts na rede Bitcoin Satoshi Vision (BSV)

O *Bitcoin Satoshi Vision* (BSV) é uma bifurcação do *Bitcoin Cash* que busca restaurar o protocolo original do *Bitcoin*, com focos em escalabilidade e estabilidade. A plataforma suporta contratos inteligentes por meio de sua linguagem de script nativa, a qual permite a criação de aplicações descentralizadas sem a necessidade de camadas adicionais.

Segundo Sinkec (2022), há um equívoco comum sobre a capacidade do BSV em executar contratos inteligentes em larga escala. A linguagem de script do BSV é Turing-completa, alcançada fora da camada do script original do *Bitcoin*, por meio de linguagens de mais alto nível, como a *sCrypt*, que permite a implementação de contratos inteligentes

complexos diretamente na camada base da blockchain. No capítulo 3.7 é discutida a Turing-completude em blockchain.

A plataforma *sCrypt* fornece uma linguagem de alto nível para facilitar o desenvolvimento de contratos inteligentes no BSV, tornando o processo mais acessível e menos propenso a erros. Além disso, o BSV possui um limite de tamanho de bloco significativamente maior em comparação com outras blockchains, que contribui para uma maior capacidade de processamento de transações e execução de contratos inteligentes.

1.2.4 Arquitetura de Smart Contracts na rede Binance Smart Chain (BSC)

A *Binance Smart Chain* (BSC) é uma blockchain desenvolvida pela empresa *Binance* com o objetivo de oferecer uma plataforma de alto desempenho para contratos inteligentes e aplicações descentralizadas. Compatível com a *E*VM, a BSC permite que desenvolvedores utilizem ferramentas e linguagens já familiares, como o Solidity, facilitando a migração e o desenvolvimento de projetos.

A BSC utiliza um mecanismo de consenso denominado *Proof of Staked Authority* (PoSA), que combina elementos de *Proof of Stake* (PoS) e *Proof of Authority* (PoA), que resulta em tempos de bloco mais curtos e taxas de transação mais baixas. Isso torna a BSC uma opção atraente para aplicações que exigem alta velocidade e baixo custo operacional.

De acordo com dados da *Binance* (2025), a BSC possui mais de 1 milhão de usuários ativos diários e suporta uma ampla gama de aplicações, incluindo finanças descentralizadas (DeFi), *tokens* não fungíveis (NFTs) e jogos baseados em blockchain. A plataforma também oferece ferramentas como o BscScan, que permite aos usuários explorarem transações, contratos inteligentes e outras informações relevantes da blockchain.

1.2.5 Comparações entre arquiteturas BSV e BSC

Ao comparar as arquiteturas do BSV e da BSC, observa-se que ambas buscam oferecer soluções escaláveis para a execução de contratos inteligentes, mas adotam abordagens distintas.

O BSV prioriza a escalabilidade na camada base, que permite a execução de contratos inteligentes diretamente na blockchain sem a necessidade de camadas adicionais. Essa abordagem visa manter a simplicidade e a eficiência do sistema, alinhando-se à visão original do *Bitcoin*.

Por outro lado, a BSC adota uma arquitetura compatível com a EVM, que facilita a adoção por desenvolvedores familiarizados com o ecossistema *Ethereum*. A utilização do mecanismo de consenso *Proof of Stake authority* permite à BSC alcançar tempos de bloco rápidos e taxas de transação baixas, tornando-a adequada para aplicações que exigem alta performance.

Ambas as plataformas apresentam vantagens e desafios distintos, e a escolha entre elas dependerá das necessidades específicas de cada aplicação, como requisitos de desempenho, custo e compatibilidade com ferramentas existentes.

2 MATERIAIS E MÉTODOS

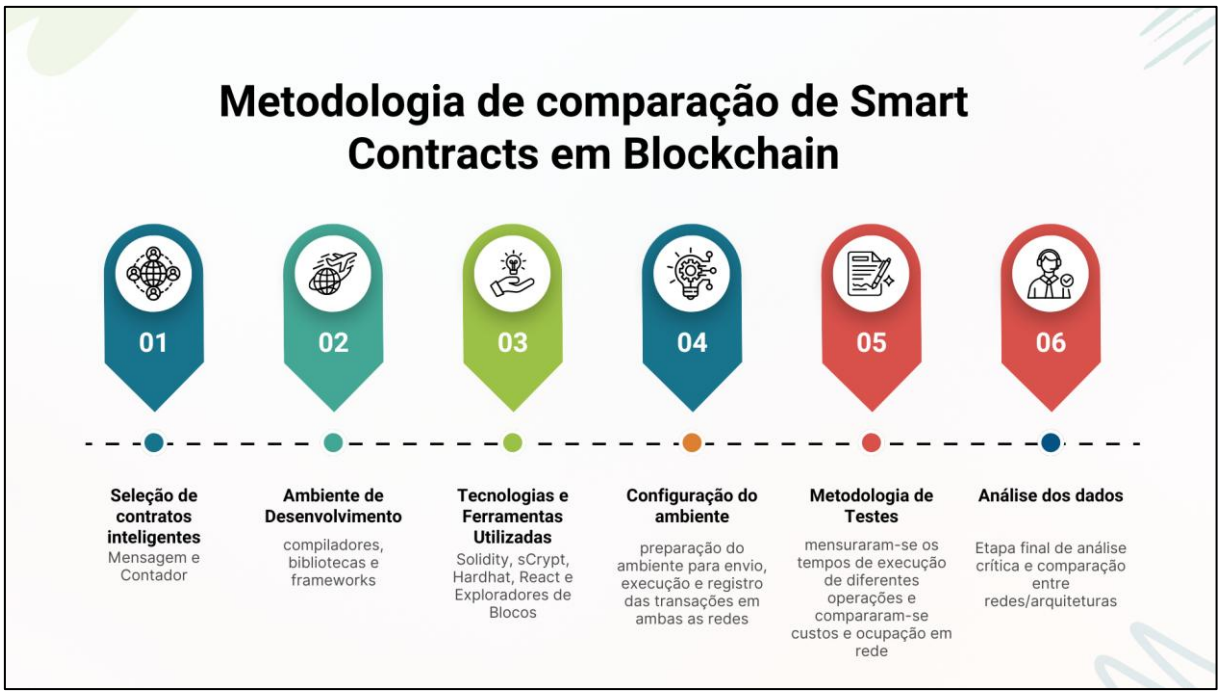
A metodologia proposta foi organizada em sete etapas principais, desenvolvidas de forma a permitir a análise comparativa entre as arquiteturas Ethereum (via Binance Smart Chain) e Bitcoin (via Bitcoin Satoshi Vision). Inicialmente, realizou-se a seleção de contratos inteligentes equivalentes, assegurando que ambos apresentassem as mesmas funcionalidades e estrutura similar para viabilizar a comparação.

Em seguida, definiu-se o ambiente de desenvolvimento, no qual foram especificadas as tecnologias e ferramentas necessárias para programação, compilação e interação com as redes. Após a configuração do ambiente, estruturou-se a metodologia de testes, contemplando a execução dos contratos em diferentes cenários. Essa metodologia incluiu a realização de testes de tempo de execução e comparação de custos, permitindo uma análise crítica do desempenho de cada arquitetura.

Para garantir clareza e alinhamento com os objetivos específicos da pesquisa, as etapas foram sistematizadas conforme descrito a seguir e representadas no fluxograma (

Figura 5)

Figura 5 – Metodologia de comparação de Smart Contracts em Blockchain



Fonte: própria.

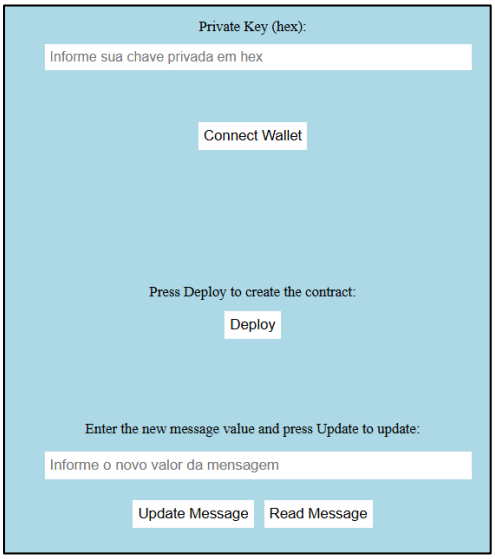
2.1 SELEÇÃO DE CONTRATOS INTELIGENTES

A seleção dos contratos inteligentes utilizados nesta pesquisa teve como objetivo viabilizar testes controlados de desempenho, com operações que possibilitassem medir métricas como tempo de execução, uso de recursos computacionais e escalabilidade em redes blockchain distintas. Foram escolhidos dois tipos de contratos com características complementares: um contrato de mensagem e um contrato de contador.

O primeiro contrato, denominado Contrato de Mensagem (ver interface na Figura 6), simula um armazenamento simples de texto na blockchain. Ele possui uma variável de estado para armazenar uma string e uma função pública que permite a atualização dessa mensagem. A

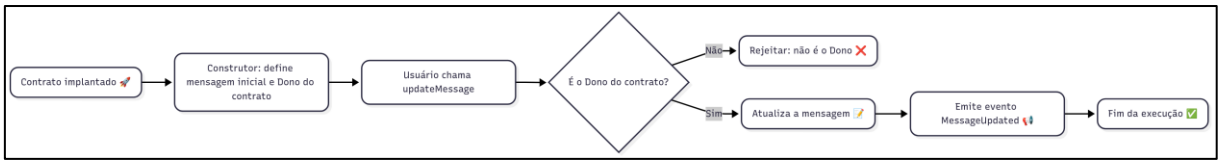
escolha deste contrato se deu pela sua simplicidade funcional e pela capacidade de medir os custos básicos de armazenamento e atualização de dados textuais em diferentes plataformas. Além disso, ele representa um caso comum de uso de contratos inteligentes voltados ao registro de informações, como declarações, descrições ou metadados, sendo particularmente útil para avaliar o desempenho sob operações leves. Para complementar sua descrição e facilitar a compreensão de sua lógica de execução, apresenta-se também o fluxograma do contrato de mensagem (Figura 7), no qual são evidenciadas as etapas de implantação, verificação do proprietário e atualização da variável de estado.

Figura 6 - Interface do contrato de mensagem.



Fonte: própria.

Figura 7 - Fluxo de contrato de mensagem

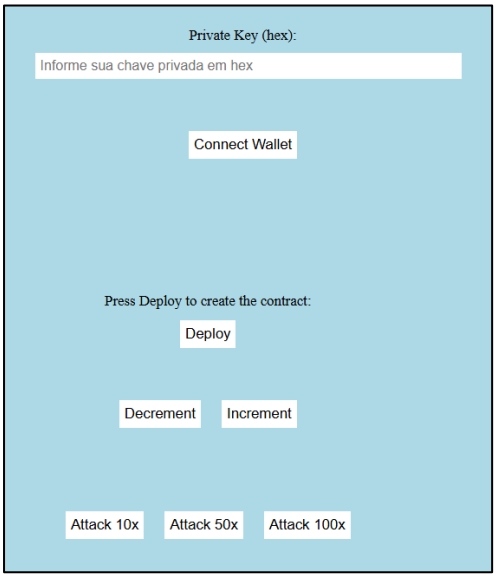


Fonte: própria.

O segundo contrato, denominado Contrato de Contador (ver interface na Figura 8), foi desenvolvido com maior complexidade lógica. Ele permite a manipulação de uma variável numérica inteira por meio de funções que incrementam, decrementam e executam operações repetitivas (denominadas funções de ataque), como attack 10x, attack 50x e attack 100x. Essas funções simulam cenários de uso intensivo da blockchain, sendo úteis para avaliar a escalabilidade da rede, consumo de recursos computacionais e tempo de resposta sob diferentes níveis de carga transacional. Esse tipo de contrato também representa situações reais

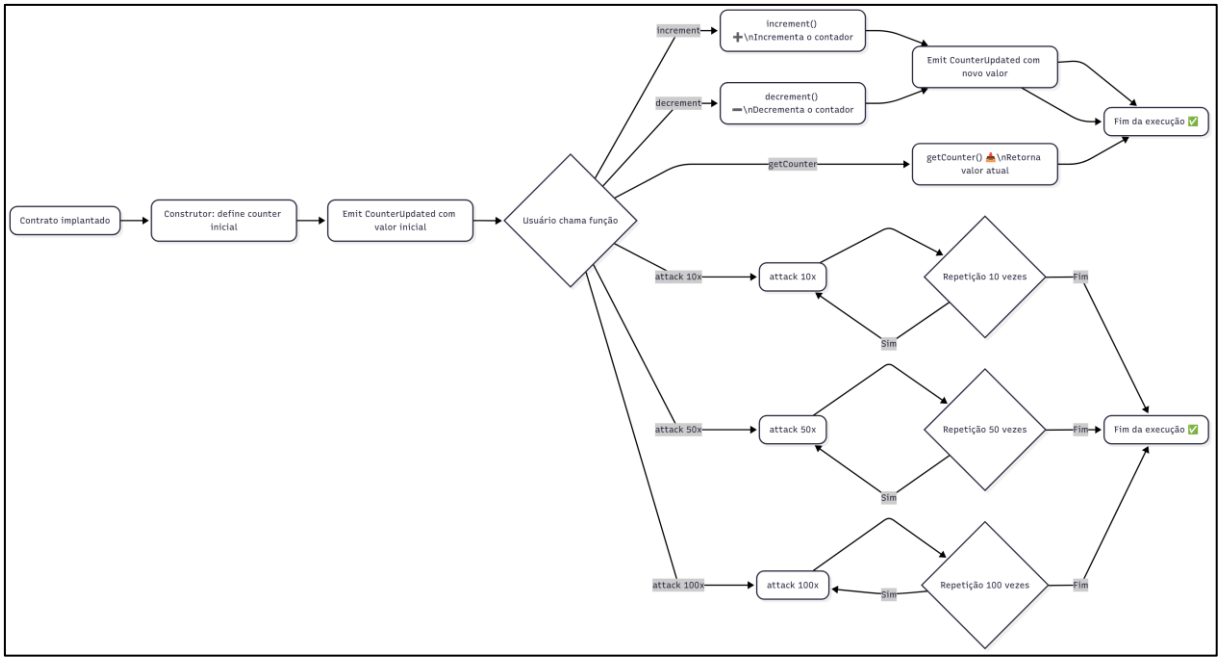
encontradas em jogos, sistemas de reputação, contadores de tokens, entre outros. Para complementar sua descrição e ilustrar sua lógica interna de execução, apresenta-se ainda o fluxograma do contrato de contador (Figura 9), no qual estão representadas as funções básicas de incremento e decremento, bem como as operações repetitivas de ataque que intensificam a carga computacional sobre a rede.

Figura 8 - Interface do contrato contador.



Fonte: própria.

Figura 9 - Fluxo de contrato contador



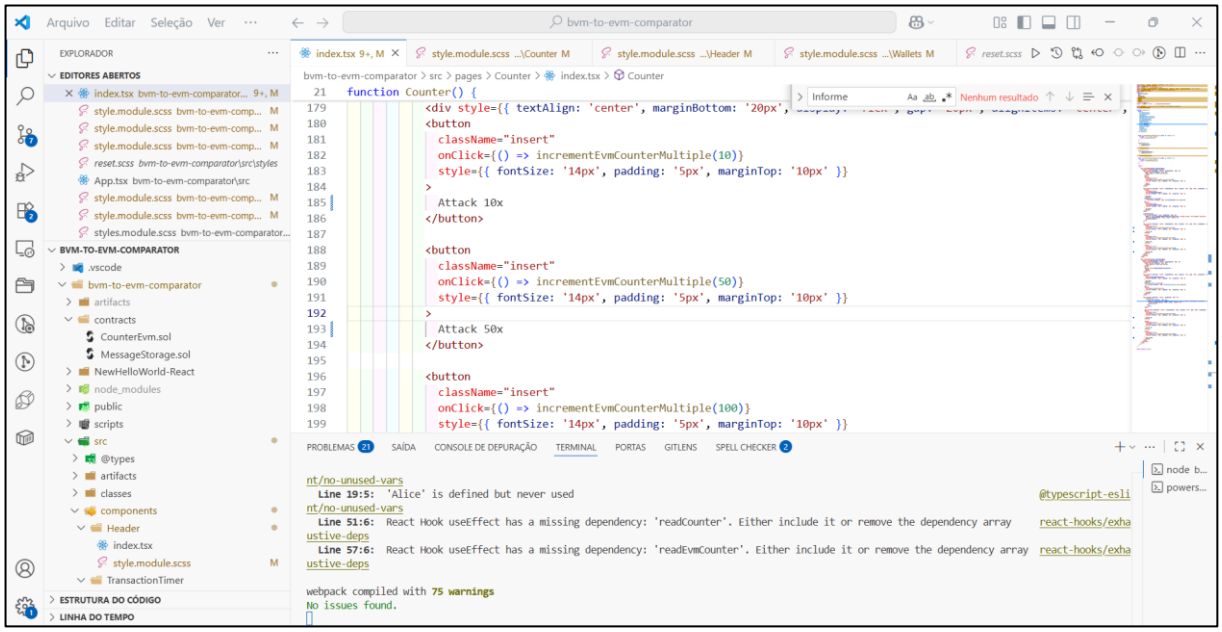
Fonte: própria.

A adoção desses dois contratos permite uma análise comparativa entre as redes *Binance Smart Chain* e *Bitcoin Satoshi Vision* sob diferentes perspectivas operacionais: desde interações simples até execuções em lote com maior complexidade computacional. Essa combinação fornece um conjunto de experimentos equilibrado entre leveza e stress computacional, contribuindo para uma avaliação mais abrangente do desempenho de cada plataforma.

2.2 AMBIENTE DE DESENVOLVIMENTO

Os testes experimentais desta pesquisa foram realizados por meio de um ambiente de desenvolvimento local estruturado com o editor *Visual Studio Code* (*VsCode*). Foram utilizados contratos inteligentes escritos em *Solidity* para a rede *B*SC e em *sCrypt* para a rede BSV. A plataforma de interação com os contratos foi desenvolvida utilizando o *framework React.js*.

Figura 10 - Ambiente de Desenvolvimento *VsCode*.



Fonte: própria.

O gerenciamento de versões e execução de *scripts* foi realizado com o auxílio da ferramenta *Hardhat*, utilizada para compilar, testar e implantar os contratos na BSC. Para os contratos em BSV, foi utilizado o SDK oficial do *sCrypt*. Todas as transações foram testadas em ambiente de *testnet*, de modo a garantir segurança e controle sobre os experimentos.

2.3 TECNOLOGIAS E FERRAMENTAS UTILIZADAS

O desenvolvimento dos contratos inteligentes e da interface de interação com as redes BSC e BSV exigiu o uso de um conjunto integrado de tecnologias, linguagens e *frameworks*. As ferramentas selecionadas foram fundamentais para a criação, implantação e testes dos contratos, bem como para o registro e análise de seu comportamento. A seguir, descrevem-se as principais tecnologias utilizadas e o papel de cada uma no projeto.

2.3.1 *Solidity*

Solidity é uma linguagem de programação orientada a objetos, projetada para o desenvolvimento de contratos inteligentes que são executados na EVM. No contexto deste projeto, *Solidity* foi utilizada para desenvolver contratos destinados à *B*SC, que é compatível com a EVM.

Os contratos escritos em *Solidity*, como o Counter.sol, implementam funcionalidades de incremento, decremento e ataques múltiplos (e.g., *attack10x, attack50x, attack100x*). Essas funções foram projetadas para testar o desempenho da rede sob diferentes cargas de transações.

A estrutura modular da linguagem permitiu a criação de funções específicas para manipulação de variáveis e simulação de cenários de stress, para facilitar a análise comparativa entre as redes BSC e BSV.

2.3.2 *sCrypt*

sCrypt é uma linguagem de programação de alto nível que transpila (processo de traduzir um código-fonte de uma linguagem de programação para outra linguagem de alto ou médio nível) para *Bitcoin Script*. Isso permite o desenvolvimento de contratos inteligentes na rede BSV. Neste projeto, *sCrypt* foi utilizada para criar contratos que simulam armazenamento de mensagens e operações de contagem.

A linguagem oferece maior expressividade e controle na criação de contratos inteligentes sobre a arquitetura da BSV. A sintaxe clara e tipagem forte de *sCrypt* facilitaram a definição de parâmetros de entrada e saída, utilizando estruturas como *PubKey* e *ByteString* para garantir a segurança das transações.

O ambiente de testes foi realizado na *testnet* da BSV, com integração ao *Whatsonchain*, permitindo a verificação da execução e status das transações.

2.3.3 *Hardhat*

Hardhat é um ambiente de desenvolvimento para *Ethereum* e redes compatíveis com EVM, utilizado no projeto para compilar e implantar contratos escritos em *Solidity* na *testnet* da BSC.

Sua principal vantagem é a capacidade de simular redes locais, executar *scripts* de testes e controlar o ciclo de vida do contrato de forma automatizada. No projeto, ele foi essencial para organizar os arquivos, compilar o *bytecode*, executar *deploys* e gerar registros (*logs*) de execução. O suporte a *plugins* e integração com bibliotecas como Ethers.js e Web3.js facilitou a interação com a aplicação *React*.

Além disso, o *Hardhat* permite a criação de ambientes de teste local, o que foi útil para validar funções e operações em lote antes da execução na *testnet* pública.

2.3.4 *React.js*

Essa biblioteca foi utilizada para criar a interface gráfica que permite ao usuário interagir com os contratos inteligentes implantados. A aplicação web foi organizada em componentes reutilizáveis, com separação clara entre lógica de negócios, chamadas aos contratos e exibição de resultados.

Através da integração com as bibliotecas Web3.js (para a BSC) e *sCrypt* SDK (para a BSV), a interface permite enviar transações, ler estados dos contratos e visualizar o histórico de execuções. A estrutura modular da aplicação foi projetada para possibilitar a inserção de dados, acionamento de funções e exibição de respostas da blockchain em tempo real.

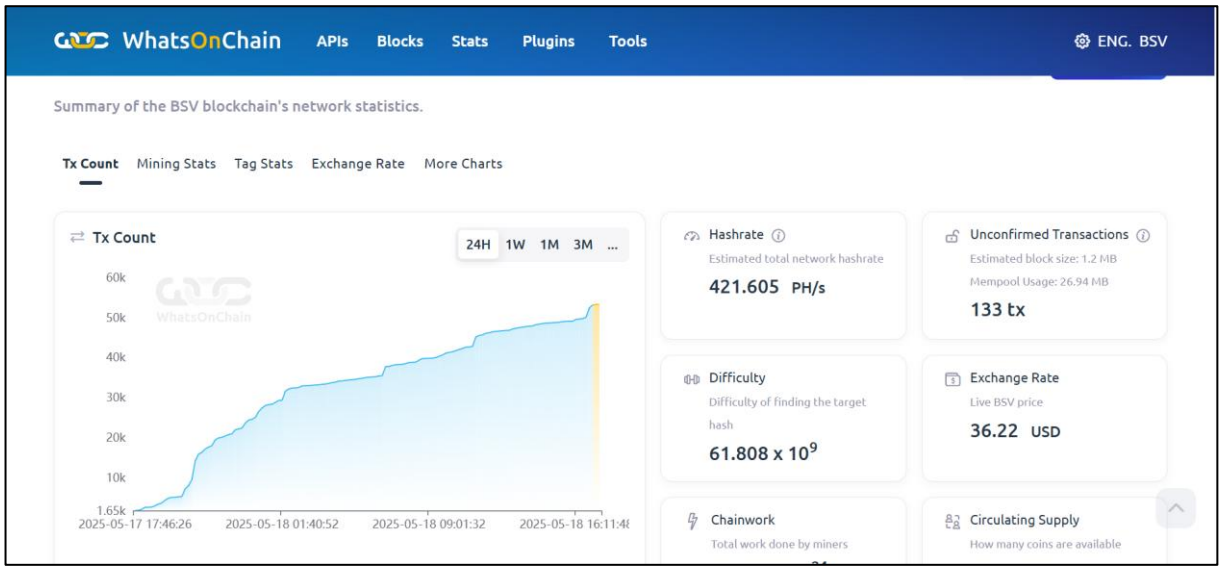
O *React* possibilitou também a inclusão de marcadores de tempo para análise de latência e performance, auxiliando nas métricas de avaliação do projeto.

2.3.5 *Whatsonchain e BSCscan Testnet*

Para o monitoramento das transações em blockchain, foram utilizados dois exploradores de blocos: o *Whatsonchain* (Figura 11), voltado para a *testnet* da BSV, e o *BSCscan* (

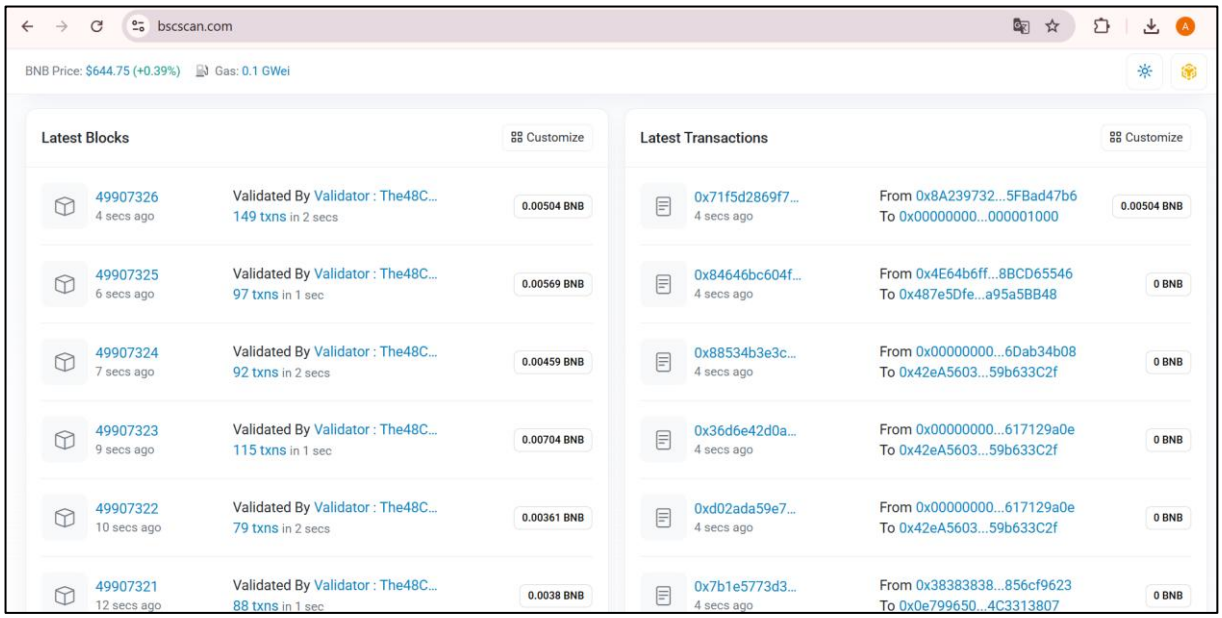
Figura 12), para a BSC.

Figura 11 - Visualização de transações na *Whatsonchain*.



Fonte: WHATSONCHAIN (2025).

Figura 12 – Visualização de transações na *BscScan*.



Fonte: BSCSCAN (2025).

Essas ferramentas permitiram rastrear cada transação executada pelos contratos, observar os tempos de confirmação e analisar os dados armazenados. No caso da BSC, a API do *BscScan* também foi usada para validar transações diretamente da interface. Para a BSV, o *Whatsonchain* ofereceu a validação de UTXO. Essa validação permitiu assegurar que os *scripts* foram corretamente processados.

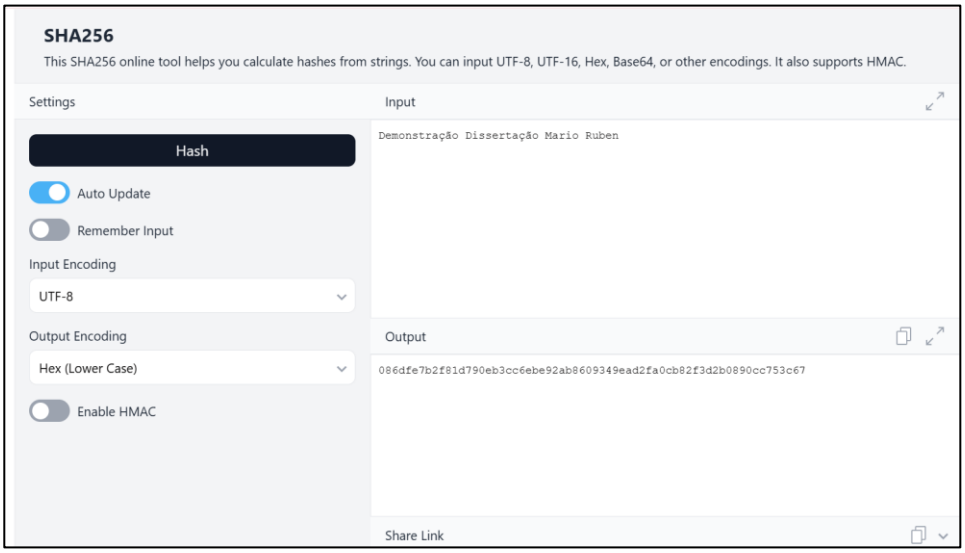
2.3.6 Carteiras e chaves privadas

Para permitir a execução dos contratos inteligentes e o envio de transações nas redes *testnet* da BSC e BSV, foi necessário o gerenciamento de carteiras digitais (*wallets*) com pares de chaves públicas e privadas.

A chave privada é essencial para assinar digitalmente transações, provando a propriedade dos fundos e autorizando a execução das funções dos contratos. Para cada rede, a geração e o tratamento dessas chaves seguem padrões criptográficos distintos, refletindo a arquitetura de cada blockchain.

No caso da BSV, a chave privada foi gerada utilizando uma função *hash* SHA-256 aplicada sobre uma *string* aleatória (entropia). Este processo é compatível com os princípios do protocolo *Bitcoin* e permite derivar a chave pública correspondente a partir da curva elíptica secp256k1, padrão do *Bitcoin*. A Figura 13 demonstra a interface usada para a geração da chave privada a partir de uma função *hash* SHA-256.

Figura 13 - Interface da ferramenta SHA256 Online.

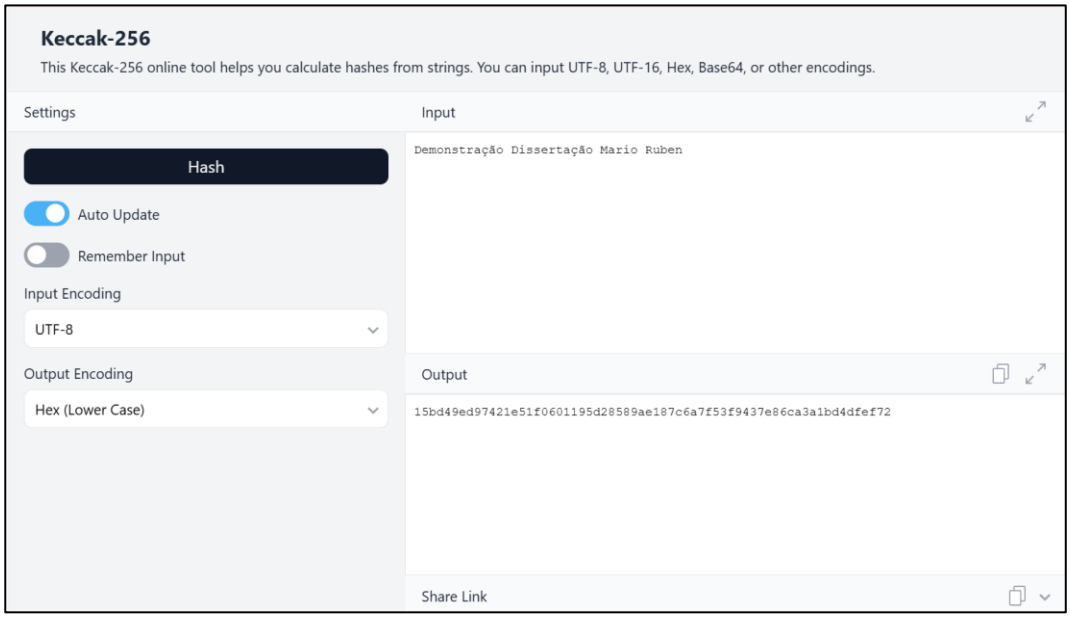


Fonte: EMN178 (2025).

Para a BSC, a geração da chave privada seguiu o modelo utilizado em blockchains compatíveis com a EVM. Foi utilizada a função de *hash* Keccak-256, que é empregada para derivar endereços *Ethereum* a partir de uma chave pública. A Keccak-256 é parte integrante do processo de derivação de endereços no padrão *Ethereum*, e, portanto, compatível com a BSC.

A Figura 14 demonstra a interface usada para a geração da chave privada a partir de uma função *hash* Keccak-256.

Figura 14 - Interface da ferramenta Keccak-256 Online.



Fonte: EMN178 (2025).

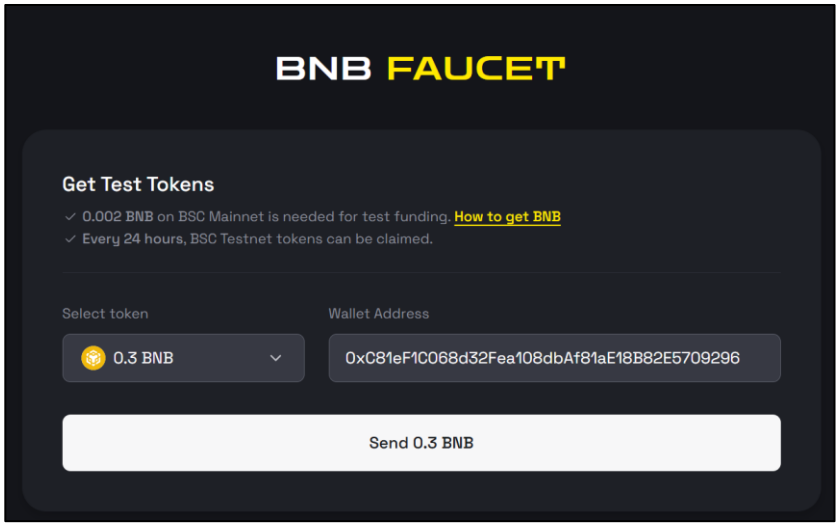
A gestão das chaves foi realizada de forma local e segura, garantindo que cada transação executada durante os testes fosse devidamente assinada e reconhecida pelas respectivas redes. A abordagem adotada para geração e uso das chaves permitiu simular cenários reais de operação e forneceu total controle sobre as transações enviadas, sem dependência de carteiras externas ou extensões de navegador.

2.3.7 *Faucets*

Como as transações nas redes BSC e BSV exigem saldo para pagamento de taxas (mesmo em ambientes de teste), foi necessário adquirir fundos fictícios por meio de *faucets*. Esses *faucets* são serviços gratuitos que enviam pequenas quantias de criptomoedas de teste para endereços válidos, permitindo o desenvolvimento e validação de contratos em *testnets* sem custo real.

Para a BSC, foi utilizado o *faucet* oficial da BSC *Testnet* (Figura 15), acessível em <https://www.bnbchain.org/en/testnet-faucet> no qual os endereços gerados foram inseridos e receberam BNB de teste. Esses fundos possibilitaram o *deploy* e a chamada das funções nos contratos inteligentes escritos em *Solidity*.

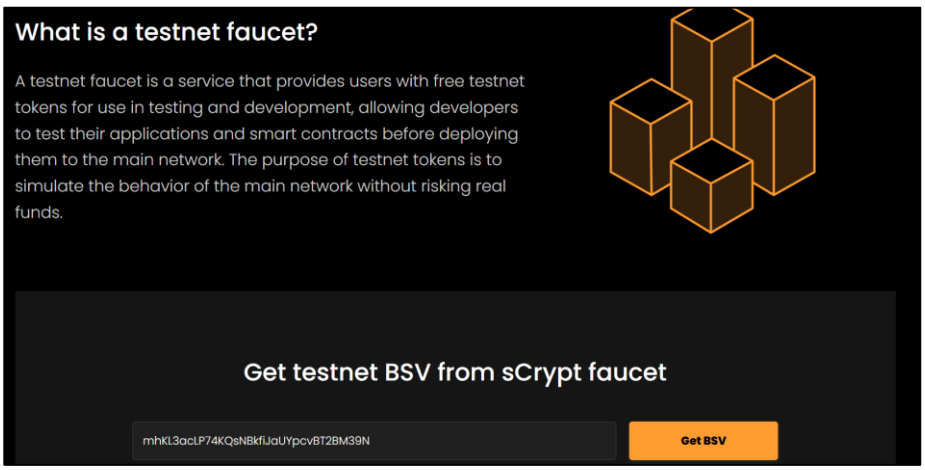
Figura 15 - Interface do BNB *Chain Testnet Faucet*.



Fonte: Adaptado de BNB CHAIN (2025).

Na rede BSV, os fundos de teste foram adquiridos no *faucet* da plataforma *sCrypt* (Figura 16), acessível em https://scrypt.io/faucet. O saldo foi essencial para simular cenários de atualização de estado, verificação de assinaturas e manipulação de UTXO na *testnet* pública, integrando-se ao explorador *Whatsonchain*.

Figura 16 - Interface do *sCrypt* BSV *Faucet*.



Fonte: Adaptado de SCRYPT (2025).

A utilização dos *faucets* garantiu a autonomia dos testes, sem depender de terceiros ou ambientes simulados. Isso assegurou que os contratos pudessem ser executados sob condições próximas às reais, com custos, assinaturas e validações representando o comportamento esperado em uma blockchain pública.

3 IMPLEMENTAÇÃO

3.1 CONFIGURAÇÃO DE AMBIENTE

Antes de iniciar a configuração do ambiente para trabalhar com a rede bitcoin e Ethereum, é importante fazer a pré configuração instalando na máquina o node.js conforme descrito na Listagem 1:

Listagem 1 – Pré configuração Node.js

```
fnm install 22 #download e instalação Node.js
node -v #Verificar versão do Node.js instalada
npm – v #Verificar versão npm instalada
```

Node.js é a base que conecta o seu ambiente de desenvolvimento com as ferramentas modernas de *blockchain* e *frontend*. Sem ele, ferramentas como *Hardhat, sCrypt SDK* e *React.js* não funcionam corretamente. A instalação desse pacote irá permitir a utilização de comandos como *npx hardhat, npm install scryptlib*, entre outros;

A correta configuração do ambiente foi fundamental para garantir a execução adequada dos contratos inteligentes, tanto na plataforma *Bitcoin Satoshi Vision* (BSV) quanto na *Binance Smart Chain* (BSC). A seguir, detalha-se a estrutura montada para cada rede, incluindo dependências, compiladores, carteiras e conexões com as respectivas *testnets*.

3.1.1 Ambiente BSV

Para iniciar o desenvolvimento dos contratos inteligentes na plataforma BSV utilizando a linguagem *sCrypt*, foi realizada a configuração de um projeto-base por meio da *Command-Line Interface* (CLI) oficial da ferramenta. A criação do ambiente seguiu uma sequência de comandos executados no terminal, conforme descrito abaixo na Listagem 2:

Listagem 2 – Configuração inicial de ambiente BSV

```
$ npx scrypt-cli Project helloworld
$ cd helloworld
```

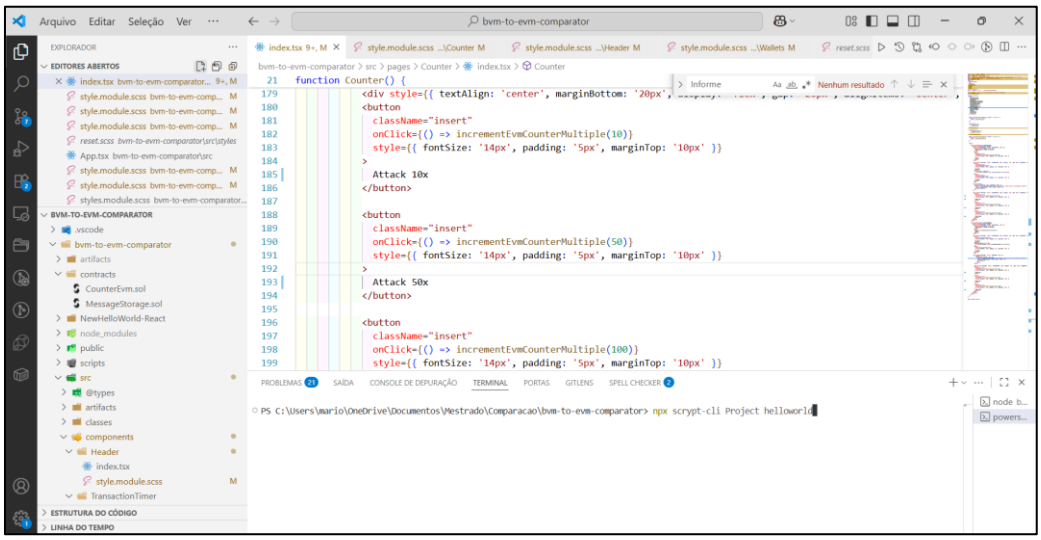
```
$ npm install
```

O comando *npx scrypt-cli project helloworld* é utilizado para inicializar um novo projeto baseado na linguagem *sCrypt*, voltada para contratos inteligentes na arquitetura *Bitcoin*.

Na linha seguinte, *cd helloworld*, o terminal é direcionado para o diretório recém-criado. Essa etapa é necessária para que os próximos comandos, como a instalação de dependências ou execução de *scripts*, sejam realizados dentro do ambiente correto do projeto.

Por fim, *npm install* é responsável por instalar todas as bibliotecas listadas no arquivo *package.json*. Isso inclui o compilador da linguagem *sCrypt*, o SDK para testes e *deploy* (*scrypt-ts*), e demais dependências. Após essa instalação, o ambiente estará completamente configurado para compilar, testar e publicar contratos inteligentes na *testnet* do BSV. A Figura 17 mostra o ambiente *VsCode* inicializando o projeto BSV.

Figura 17 – Interface *VsCode* inicializando projeto BSV.



Fonte: própria.

Devemos fazer algumas configurações rápidas como:

1. **Apagar pastas no projeto:**
   - Pasta `.cache` em `node_modules/.cache` caso exista essa pasta
   - Pasta `bsv` em `node_modules/@scrypt-inc/bsv`
   - Pasta `scrypt-ts` em `node_modules/scrypt-ts`

2. **Copiar pastas de dentro de `crack-scrypt` para o projeto:**

- Pasta `bsv` para dentro de `node_modules/@scrypt-inc/bsv`
- Pasta `scrypt-ts` para dentro de `node_modules/scrypt-ts`

Após a configuração inicial do ambiente, vamos fazer a escrita dos contratos inteligentes (Contrato de Mensagem e Contrato Contador). Para isso, devemos substituir o contrato padrão *HelloWorld.ts*, localizado no diretório /src/contracts/helloWorld.ts, e iniciar a implementação dos contratos propostos neste trabalho.

O primeiro contrato a ser implementado foi o **Contrato de Mensagem**, cuja funcionalidade consiste em armazenar uma *string* fornecida pelo usuário na blockchain, de forma segura e verificável. A estrutura completa do contrato e seu arquivo de *deploy* encontra-se descrito abaixo, conforme a Listagem 3 e a Listagem 4:

Listagem 3 – Contrato de mensagem em *Scrypt*.

```
import { assert, method, prop, SmartContract, Sig, PubKey, ByteString, toByteString, len } from 'scrypt-ts';

export class MessageStorageSCrypt extends SmartContract {
    @prop(true)
    message: ByteString;

    @prop(true)
    owner: PubKey;

    constructor(message: ByteString, owner: PubKey) {
        super(...arguments);
        this.message = message;
        this.owner = owner;
    }

    @method()
    public updateMessage(newMessage: ByteString) {
        this.message = newMessage;
        assert(true);
    }
}
```

Listagem 4 – Arquivo de *Deploy* em *Scrypt*.

```
import { writeFileSync } from 'fs'
```

```
import { BvmToEvmComparator } from '../src/contracts/bvmToEvmComparator'
import { privateKey } from './privateKey'
import { bsv, TestWallet, DefaultProvider, sha256, toByteString } from 'scrypt-ts'

function getScriptHash(scriptPubKeyHex: string) {
    const res = sha256(scriptPubKeyHex).match(/.{2}/g)
    if (!res) {
        throw new Error('scriptPubKeyHex is not of even length')
    }
    return res.reverse().join('')
}

async function main() {
    await BvmToEvmComparator.loadArtifact()

    // Prepare signer.
      // See https://scrypt.io/docs/how-to-deploy-and-call-a-contract/#prepare-a-signer-and-provider
        const signer = new TestWallet(privateKey, new DefaultProvider({
        network: bsv.Networks.testnet
    }))

    // TODO: Adjust the amount of satoshis locked in the smart contract:
    const amount = 100
    const message = toByteString('hello world', true);
    const messageHash = sha256(message);

    const instance = new BvmToEvmComparator(
        // TODO: Pass constructor parameter values.
        0n,
        messageHash
    )

    // Contract deployment.
    const deployTx = await instance.deploy(amount, messageHash)
    await instance.connect(signer)
```

```
        // Save deployed contracts script hash.
                                const           scriptHash           =
getScriptHash(instance.lockingScript.toHex())
        const shFile = `.scriptHash`;
        writeFileSync(shFile, scriptHash);

         console.log('BvmToEvmComparator contract was successfully
deployed!')
        console.log(`TXID: ${deployTx.id}`)
        console.log(`scriptHash: ${scriptHash}`)
    }

    main()
```

Após a implementação do contrato de mensagem, o próximo passo foi a construção do Contrato Contador, projetado para simular operações aritméticas em diferentes níveis de carga transacional. Este contrato tem como principal objetivo a manipulação de um valor numérico armazenado na blockchain por meio de funções de incremento, decremento e execuções múltiplas (denominadas ataques).

A implementação deste contrato exigiu a utilização de uma variável de estado para armazenar o valor atual do contador, além de funções específicas para realizar incrementos e decrementos controlados. Além disso, foram desenvolvidas variantes da função de incremento, como *attack10x*, *attack50x* e *attack100x*, que simulam um número elevado de chamadas consecutivas em uma única execução.

Essas funções foram implementadas de forma a testar os limites de desempenho da blockchain BSV em termos de tempo de execução, uso de memória e número de operações permitidas por transação. O comportamento do contrato sob diferentes intensidades de carga foi registrado por meio da interface *React* e monitorado em tempo real utilizando o explorador *Whatsonchain*. A Listagem 5 apresenta o código-fonte do contrato contador, conforme implementado no editor *Visual Studio Code*.

Listagem 5 – Contrato contador em *Scrypt*

```
    import {
        method,
        prop,
        SmartContract,
```

```
        hash256,
        assert,
        ByteString,
        SigHash,
    } from 'scrypt-ts'

    export class Counter extends SmartContract {
        // Stateful property to store counters value.
        @prop(true)
        count: bigint

        constructor(count: bigint) {
            super(count)
            this.count = count
        }

        @method(SigHash.ANYONECANPAY_SINGLE)
        public incrementOnChain() {
            this.increment()

            // make sure balance in the contract does not change
            const amount: bigint = this.ctx.utxo.value
            // output containing the latest state
            const output: ByteString = this.buildStateOutput(amount)
            // verify current tx has this single output
                assert(this.ctx.hashOutputs  ===  hash256(output),
'hashOutputs mismatch')
        }

        @method(SigHash.ANYONECANPAY_SINGLE)
        public decrementOnChain() {
            this.decrement()

            // make sure balance in the contract does not change
            const amount: bigint = this.ctx.utxo.value
            // output containing the latest state
            const output: ByteString = this.buildStateOutput(amount)
            // verify current tx has this single output
                assert(this.ctx.hashOutputs  ===  hash256(output),
'hashOutputs mismatch')
        }
```

```
        @method()
        public finish() {
            assert(this.count >= 3, 'Contract Not Old Enough')

            // build a change output to return funds to the sender
            const output: ByteString = this.buildChangeOutput()
                assert(this.ctx.hashOutputs  ===  hash256(output),
'hashOutputs mismatch')
        }

        @method()
        increment(): void {
            this.count++
        }

        @method()
        decrement(): void {
            this.count--
        }
    }
```

3.1.2 Ambiente BSC

A configuração do ambiente para o desenvolvimento de contratos inteligentes na BSC foi realizada com ferramentas compatíveis com a EVM, utilizando a linguagem *Solidity* e o *framework Hardhat* como base. O ambiente foi estruturado para garantir um ciclo completo de desenvolvimento.

A primeira etapa da configuração consistiu na inicialização de um projeto *Node.js*, por meio do seguinte comando no terminal:

```
$ npm init -y
```

Esse comando cria automaticamente o arquivo *package.json*, que serve como ponto central de gerenciamento das dependências, *scripts* e configurações do projeto. Em seguida, foi realizada a instalação local do *Hardhat* como dependência de desenvolvimento, utilizando o comando:

```
$ npm i -D hardhat
```

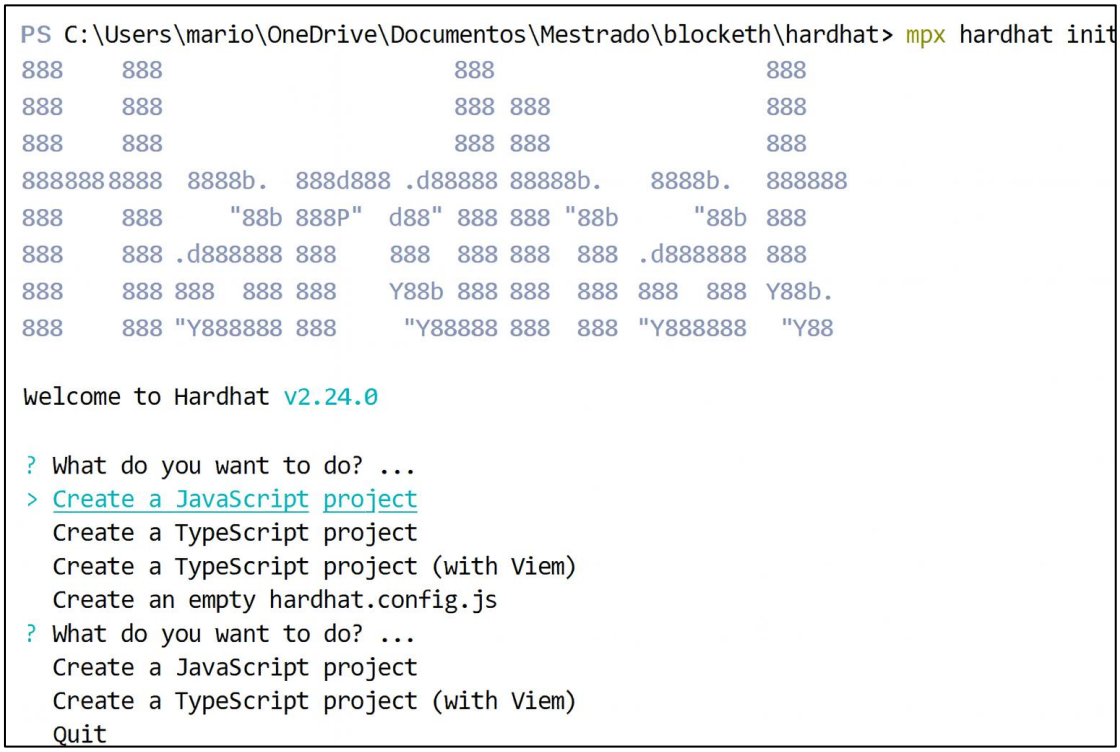
Após a instalação, foi executado o comando de inicialização do *Hardhat*:

```
$ npx hardhat init
```

Durante o processo interativo, foi selecionada a opção de criação de um projeto com suporte a *TypeScript* (Figura 18), na criação da seguinte estrutura de diretórios conforme aFigura 19:

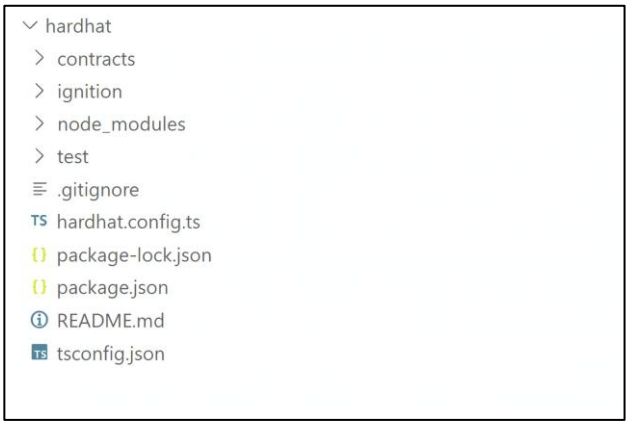
- /contracts: diretório onde ficam armazenados os contratos inteligentes escritos em *Solidity*.
- /test: pasta destinada aos arquivos de teste automatizado.
- /ignition: pasta introduzida pelas versões mais recentes do Hardhat, destinada à organização dos scripts de deploy, módulos de configuração e execução de cenários de testes personalizados.
- /node_modules: pasta gerada automaticamente pelo NPM, contendo todas as dependências instaladas do projeto.

Figura 18 - Configuração Hardhat com TypeScript.



Fonte: própria.

Figura 19- Dependências Hardhat.



Fonte: própria.

A seguir, foi realizada a configuração da rede BSC *testnet* no arquivo *hardhat.config.ts*, especificando o *endpoint* RPC público e a chave privada da carteira utilizada. A chave foi gerada a partir da função *hash Keccak-256*, conforme o padrão *Ethereum*. Para permitir a execução de transações, o endereço recebeu saldo por meio do *faucet* oficial da BSC *testnet*.

Após a configuração inicial do ambiente, procedeu-se com a escrita dos contratos inteligentes destinados à execução na BSC. Os contratos foram desenvolvidos na linguagem *Solidity* e inseridos no diretório /contracts, criado automaticamente durante a inicialização do projeto *Hardhat* com suporte a *TypeScript*.

O primeiro contrato implementado foi o Contrato de Mensagem, cuja função principal é permitir o armazenamento de um texto na blockchain. Esse contrato contém uma variável do tipo *string* que armazena a mensagem, além de uma função pública que permite ao usuário atualizar seu conteúdo. A lógica do contrato garante que o valor armazenado permaneça acessível e verificável, fornecendo uma base simples e funcional para mensuração de desempenho em operações de escrita e leitura de dados na blockchain. A Listagem 6 a seguir mostra a estrutura do contrato de mensagem.

Listagem 6 – Contrato de mensagem em Solidity

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

contract MessageStorage {
    string public message;
    address public owner;

    event MessageUpdated(string newMessage);
```

```
        constructor(string memory initialMessage) {
            message = initialMessage;
            owner = msg.sender;
        }

        modifier onlyOwner() {
            require(msg.sender == owner, "Only the owner can update
the message");
            _;
        }

        function updateMessage(string memory newMessage) public
onlyOwner {
            message = newMessage;
            emit MessageUpdated(newMessage);
        }
    }
```

Na sequência, foi desenvolvido o Contrato Contador, que mantém uma variável inteira e permite sua manipulação por meio de funções públicas. Além das funções básicas *increment*() e *decrement*(), foram implementadas funções adicionais que simulam operações em lote, denominadas *attack10x*(), *attack50x*() e *attack100x*(). Cada uma dessas funções executa múltiplas chamadas consecutivas de incremento, permitindo simular condições de uso intensivo da rede. O objetivo dessas funções foi gerar diferentes níveis de carga computacional sobre a rede, de modo a observar o comportamento da BSC em termos de tempo de execução, consumo de gás e estabilidade. A estrutura completa do contrato está ilustrada na Listagem 7, contendo as principais seções de lógica de negócio e definição de eventos.

Listagem 7 – Contrato contador em *Solidity*.

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

contract CounterEvm {
    int256 private counter;

    event CounterUpdated(int256 newValue);
```

```
    constructor(int256 initialValue) {
        counter = initialValue;
        emit CounterUpdated(counter);
    }

    function increment() public {
        counter += 1;
        emit CounterUpdated(counter);
    }

    function decrement() public {
        counter -= 1;
        emit CounterUpdated(counter);
    }

    function getCounter() public view returns (int256) {
        return counter;
    }
}
```

Após a implementação, os contratos foram compilados e implantados na *testnet* da BSC utilizando os *scripts* definidos no projeto *Hardhat*.

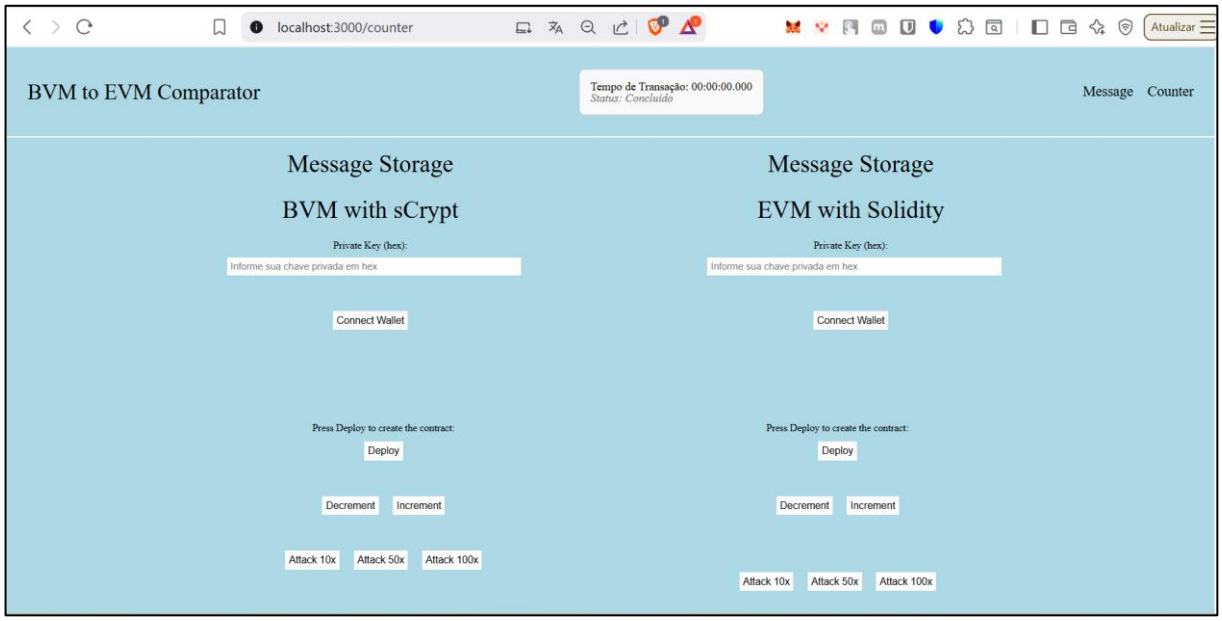
3.1.3 Interface Web

A camada de interface do projeto foi desenvolvida utilizando o *framework React.js*, com suporte a *TypeScript*, visando facilitar a interação do usuário com os contratos inteligentes implementados nas redes BSV (via *sCrypt*) e BSC (via *Solidity*). A aplicação foi estruturada em componentes modulares e executada localmente por meio do comando:

```
$ npm start
```

A interface gráfica foi projetada com foco comparativo, permitindo que o usuário visualize e interaja simultaneamente com contratos de mesma funcionalidade executados em arquiteturas distintas. A Figura 20 apresenta a interface da aplicação no navegador, já em execução local.

Figura 20 - Interface web para interação com contratos inteligentes BSV e BSC.



Fonte: própria.

No topo da interface, é possível observar um menu de navegação com abas denominadas *Message* e *Counter*, que permitem alternar entre os dois tipos de contratos implementados: o Contrato de Mensagem e o Contrato Contador. A visualização é separada horizontalmente, permitindo comparar diretamente a execução do mesmo tipo de contrato nas duas plataformas. À esquerda, o contrato implementado com *sCrypt* para a rede BSV, e à direita, o equivalente em *Solidity* para a rede BSC.

Ambos os lados da interface apresentam os mesmos campos funcionais: um campo para inserção da chave privada em formato hexadecimal, botão para conectar à carteira, botão para implantar o contrato (*Deploy*), e campos de entrada e botões para atualizar e consultar mensagens armazenadas na blockchain. A duplicação da estrutura funcional permite a realização de testes paralelos, com as mesmas ações sendo realizadas em ambientes distintos, proporcionando uma base visual e prática para a análise comparativa entre as arquiteturas BVM e EVM.

Essa interface representa uma parte essencial da aplicação, pois integra diretamente os contratos inteligentes com a experiência do usuário, permitindo validar a execução de funções, medir tempo de resposta, acompanhar estados e verificar resultados em tempo real.

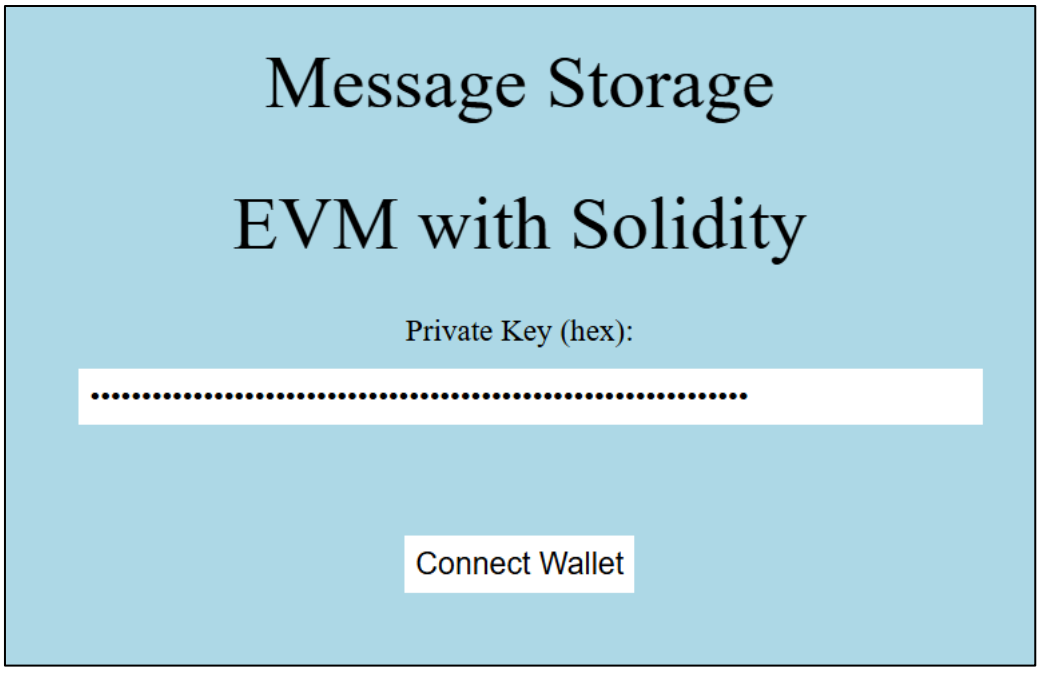
3.2 METODOLOGIA DE TESTES

A metodologia de testes adotada neste trabalho foi estruturada de forma a possibilitar a comparação sistemática entre contratos inteligentes executados em diferentes arquiteturas de blockchain. Para tanto, foram definidos cenários experimentais que contemplaram etapas de deploy, execução e monitoramento de contratos específicos, com variação no tipo de operação e no volume de dados processados. O objetivo central foi avaliar o desempenho das plataformas selecionadas sob condições controladas e replicáveis, assegurando a consistência dos resultados obtidos.

3.2.1 Execução do contrato de mensagem

A execução do contrato de mensagem teve início pela arquitetura EVM, utilizando a BSC como plataforma de testes. Para isso, foi necessário inserir a chave privada gerada por meio do algoritmo *Keccak-256*, representada em formato hexadecimal (ver Seção 4.3.6), associada a uma carteira previamente abastecida com fundos de teste (ver Seção 4.3.7). Esse procedimento é ilustrado na Figura 21.

Figura 21 - Inserção de chave privada em interface web.



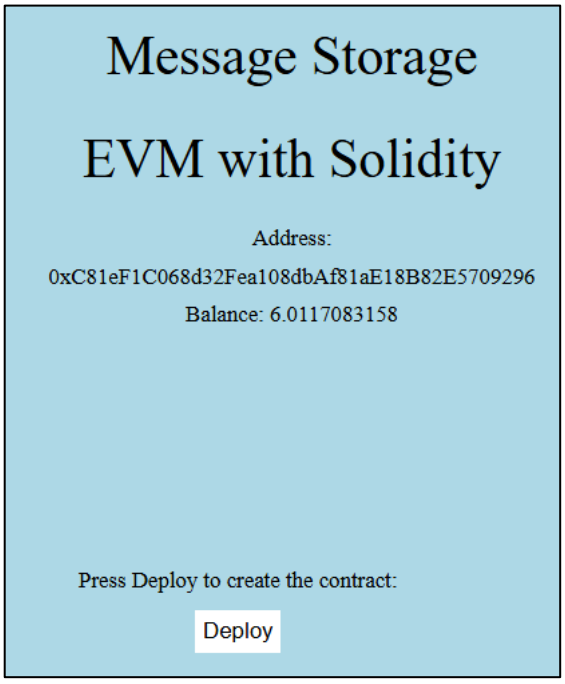
Fonte: própria.

Antes da implantação do contrato, foi realizada a conexão da chave privada à interface da aplicação. Essa chave é utilizada para assinar transações e autorizar a execução de comandos de escrita na blockchain, como a criação e atualização de contratos inteligentes, além de efetuar o pagamento das taxas de mineração correspondentes. Esse procedimento é essencial sempre que se realizam operações de gravação na blockchain, e foi aplicado de forma equivalente tanto à arquitetura baseada em *Ethereum* quanto à baseada em *Bitcoin*.

Com a carteira devidamente conectada, o usuário pode acionar o botão *Deploy*, responsável por realizar a implantação do contrato de mensagem na *testnet* da BSC. Nesse primeiro *deploy*, o contrato foi instanciado com a mensagem padrão "*HelloWorld from Solidity*", sendo gerado o seguinte identificador de transação (TxID): 0x8400f3da8a4da14c1dd3b6b13d6793d49b087ad590d2bd0b65b5fc21a351c371.

A Figura 22 ilustra o procedimento para deploy do respectivo contrato.

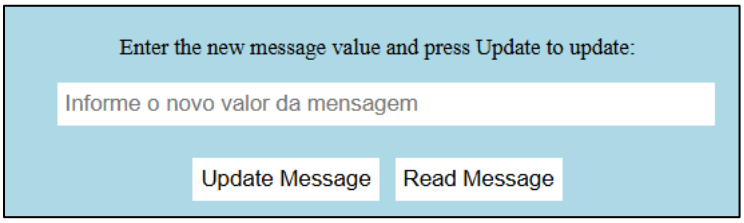
Figura 22 - Conexão de carteira no contrato de mensagem.



Fonte: própria.

Após a implantação, é possível realizar atualizações do conteúdo da mensagem por meio do campo de entrada exibido na interface, conforme demonstrado na Figura 23.

Figura 23 - Caixa de inserção de mensagem.

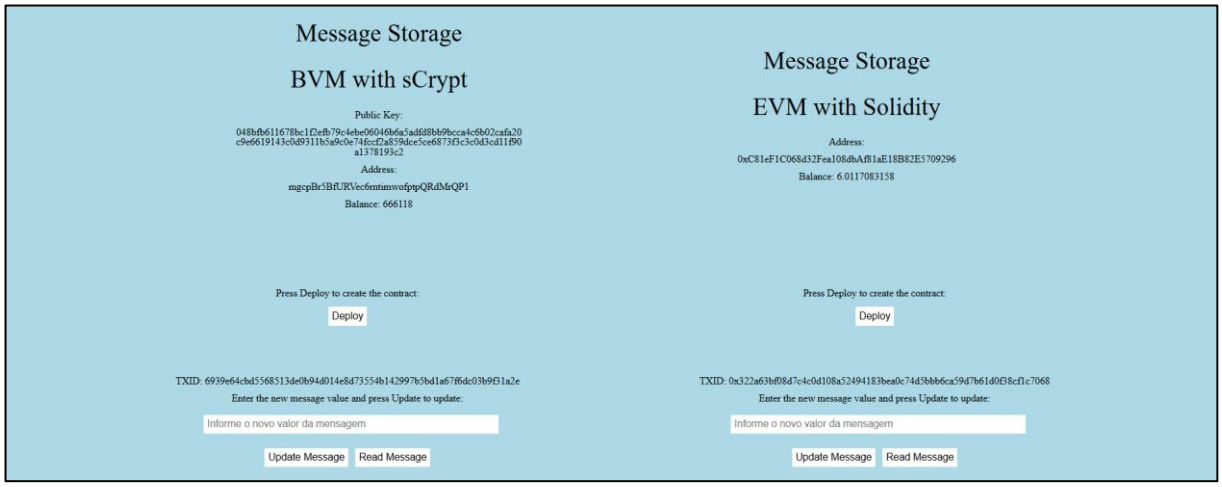


Fonte: própria.

O mesmo procedimento foi adotado para a arquitetura BVM, utilizando a rede BSV. Inicialmente, foi realizada a conexão da chave privada gerada com algoritmo SHA-256, responsável por autorizar transações e interações com o contrato inteligente implementado em *sCrypt*. Em seguida, ao acionar o botão *Deploy*, o contrato foi criado na *testnet* da BSV, gerando o TxID 930dc2f198a8d7bda040491f63006d13167f429e0b35eddbbc5d34fd349947c5.

A Figura 24 apresenta, lado a lado, a execução paralela do contrato de mensagem nas arquiteturas EVM e BVM, evidenciando que ambos os contratos realizam a mesma funcionalidade, embora executados sobre infraestruturas tecnológicas distintas.

Figura 24 - Execução simultânea do contrato de mensagem nas redes BSV (*sCrypt*) e BSC (*Solidity*) com interface web unificada para fins de comparação.



Fonte: própria.

Para comparar de forma equitativa as arquiteturas EVM e BVM, foram analisados três critérios principais: custo de transação, tempo de execução e ocupação em bloco. Para garantir a validade dessa comparação, optou-se por utilizar mensagens representativas com variações no tamanho do conteúdo textual (*payload*) e na frequência de interação com os contratos inteligentes.

A análise considerou diferentes cenários de uso, abrangendo mensagens curtas, médias e longas. Essa abordagem possibilita avaliar o desempenho das redes sob condições variadas de carga computacional, para identificar como cada arquitetura responde a operações simples e intensivas. Os resultados obtidos servirão de base para compreender os limites e a eficiência de cada plataforma em aplicações práticas baseadas em contratos inteligentes. Para isso será avaliado o comportamento sob diferentes condições, conforme a tabela 2:

Tabela 2 - Tipos de mensagens utilizadas na avaliação de desempenho das redes BSC e BSV.

Nível	**Tipo de Mensagem**	**Mensagem**	**Finalidade Comparativa**
Leve	Mensagem curta	“Teste”	Avaliar o tempo mínimo e taxa mais baixa possível
Média	Texto comum	"Mensagem de teste para análise de performance entre blockchains"	Verificar custo médio
Pesada	Texto longo	500 caracteres (5x): "uma mensagem longa para teste de desempenho e consumo de espaço em blocos nas arquiteturas BSC e BSV"	Avaliar limites de tamanho, taxas proporcionais e tempo de confirmação

Fonte: própria.

3.2.2 Execução do contrato contador

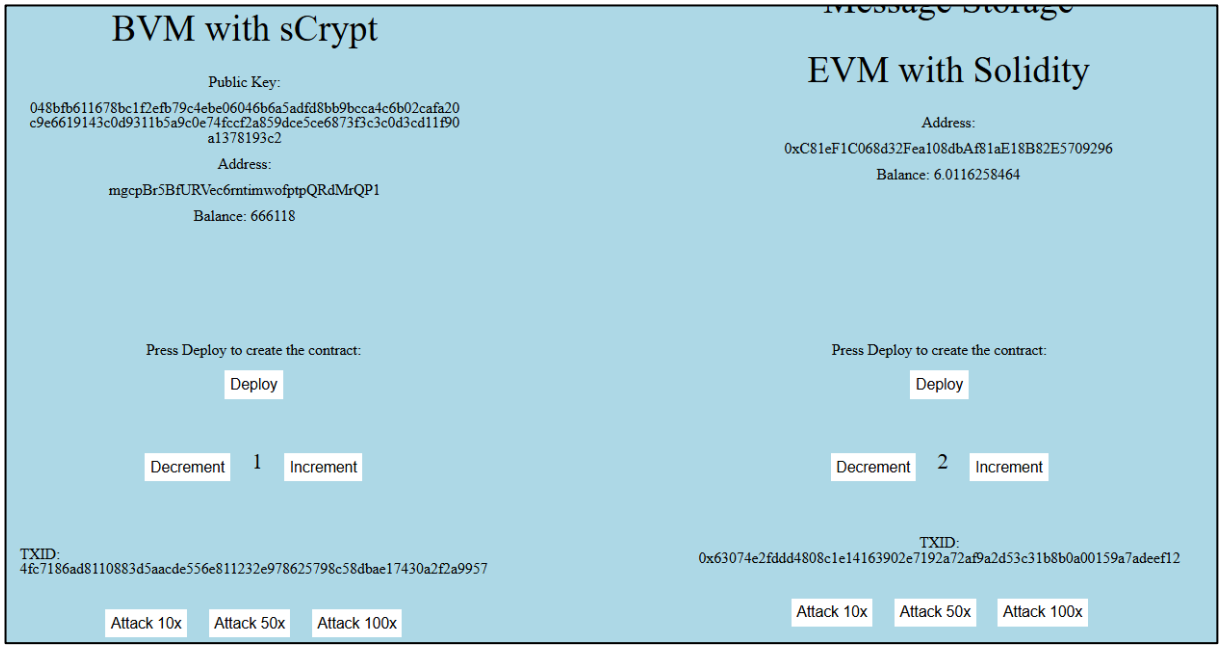
A execução do contrato contador foi realizada em ambas as arquiteturas de blockchain analisadas neste trabalho: a BVM, representada pela rede BSV, e a EVM, implementada na BSC. O objetivo dessa etapa foi validar o comportamento de um contrato com lógica aritmética básica (incremento e decremento) sob diferentes ambientes de execução, permitindo uma comparação inicial entre as duas plataformas quanto à funcionalidade e tempo de resposta.

Na arquitetura BVM, o contrato foi desenvolvido utilizando a linguagem *sCrypt* e implantado na *testnet* da BSV. A interface *React* permite ao usuário conectar sua chave privada, visualizar o endereço e saldo correspondente, e interagir com os botões de *Deploy, Increment* e *Decrement*. Após o envio da transação, é exibido o status da execução juntamente com o tempo de processamento.

Já na arquitetura EVM, o contrato foi implementado em *Solidity* e implantado na *testnet* da BSC. De forma similar à versão *sCrypt*, a interface permite a conexão da carteira, criação do contrato via botão *Deploy*, e execução de comandos de incremento e decremento sobre uma variável de estado. O valor atual do contador é exibido na tela após cada interação. Também é exibido o identificador da transação (TxID), permitindo o rastreamento e a validação dos dados diretamente no explorador de blocos (*BscScan Testnet*).

A interface gráfica foi projetada para permitir a execução paralela dos contratos nas duas redes, com o mesmo padrão visual e funcional, de modo a garantir isonomia nos testes e facilitar a análise comparativa futura. A Figura 25 mostra essa interface, destacando o equilíbrio visual entre as plataformas e a clareza na apresentação das ações disponíveis ao usuário.

Figura 25 - Execução simultânea do contrato contador nas redes BSV (*sCrypt*) e BSC (*Solidity*) com interface web unificada para fins de comparação.



Fonte: própria.

3.3 TESTES DE TEMPO DE EXECUÇÃO DOS CONTRATOS INTELIGENTES

Com o objetivo de avaliar o desempenho prático de contratos inteligentes em diferentes arquiteturas blockchain, foram conduzidos testes experimentais utilizando duas categorias de contratos: contrato de mensagem e contrato de contador. As implementações foram realizadas nas arquiteturas *Ethereum Virtual Machine* (EVM), por meio da *Binance Smart Chain* (BSC), e BVM (*Bitcoin Virtual Machine*), via *Bitcoin Satoshi Vision* (BSV), utilizando respectivamente as linguagens *Solidity* e *sCrypt*.

Os testes consideraram o tempo necessário para o *deploy* inicial dos contratos e a execução de transações com diferentes cargas de dados ou instruções.

3.3.1 Tempo de *Deploy* – Contrato de Mensagem

O primeiro teste consistiu no *deploy* do contrato de mensagem, responsável por armazenar e processar textos. Foram comparados os tempos de publicação na EVM e na BVM. Esse teste permite observar o tempo da alocação inicial do contrato em cada arquitetura.

A Tabela 3 apresenta a comparação dos tempos de execução medidos entre as duas arquiteturas:

Tabela 3 - Tempo de Deploy do Contrato de Mensagem.

Arquitetura	**Plataforma Blockchain**	**Tempo de Deploy (s)**	**Txid**
BVM	Bitcoin Satoshi Vision	4,662	220f380ca81af2f8c4b05de51fddb0b8a4 b59830c7b2ea678f17e2feaf2eff33
EVM	Binance Smart Chain	7,950	0xd15fa6dea0fc0e115f9601ba4211a35c 92b739127e076c97d925afb09061e7f3

Fonte: própria.

3.3.2 Tempo de Execução – Mensagens de Tamanho Variado

Após o *deploy*, foram realizadas transações contendo mensagens de três tamanhos distintos:

- Mensagem curta: "Teste"
- Mensagem média: "Mensagem de teste para análise de performance entre blockchains"

- Mensagem pesada: "Esta é uma mensagem longa para teste de desempenho e consumo de espaço em blocos nas arquiteturas BSC e BSV..."

Cada uma dessas mensagens foi enviada tanto na rede EVM quanto na BVM, com o tempo de execução registrado para análise de impacto do volume de dados na latência da transação.

As transações foram submetidas de forma manual, com medição do tempo desde o envio até a confirmação na rede. A Tabela 4 resume os tempos observados, juntamente com os identificadores das transações (txid) correspondentes.

Tabela 4 - Comparação de tempo de execução com variação de tamanho da mensagem.

Tamanho da Mensagem	**Tempo na BSC (EVM)**	**Tempo na BSV (BVM)**
Curta	9,16 s	3,89 s
Média	10,43 s	4,52 s
Pesada	8,44 s	4,47 s

Fonte: própria.

Txid das transações:

Mensagem Curta BSC:
0x8ae1b61365df9fed759c1ade96d333336a0613ad9b61581fc2e4e932fb5ebc54

Mensagem Curta BSV:
f93b27202ab9677eb52b278afa2489b44ee6cb7dbc172100ae25babe64527924

Mensagem média BSC:
0x79c1a48e0ec5fa00c4862a66cbd57d76d3b8e15a3bea89ab3f2d82059fe260d8

Mensagem média BSV:
fcc4a7434b2b4d421f67483fa5dec0874a1c82f7b40ac8ac426594ef166d7a87

Mensagem pesada BSC:
0xf944f09642974fdc3a2d15b58f985b7e7e84b8cfcb0b2e7a8074fbf3d5672db1

Mensagem pesada BSV:
fcc4a7434b2b4d421f67483fa5dec0874a1c82f7b40ac8ac426594ef166d7a87

3.3.3 Tempo de *deploy* - Contrato contador

O contrato de contador foi projetado para manter e modificar uma variável de estado interna por meio de operações explícitas de incremento e decremento. Sua estrutura apresenta maior complexidade lógica em comparação ao contrato de mensagem, o que pode influenciar no tempo de implantação nas diferentes arquiteturas blockchain.

A operação de *deploy* foi realizada manualmente em ambas as redes. O tempo de execução foi medido a partir do envio da transação até sua efetiva confirmação na blockchain. A Tabela 5 apresenta os identificadores das transações (txid) e os tempos observados.

Tabela 5 - Tempo de Deploy do Contrato contador.

Arquitetura	**Plataforma Blockchain**	**Tempo de Deploy (s)**	**Txid**
BVM	Bitcoin Satoshi Vision	4,53	0377220aa2ce2ca48aba9e50cf66cb4daf2765b2f51ec293b52f2051cb979259
EVM	Binance Smart Chain	10,28	0xdf5b5e00911546b97483b02766ca7cabb816c9827d947cf8b2c519e3570ee440

Fonte: própria.

Os dados obtidos evidenciam que o tempo de *deploy* do contrato de contador foi inferior na BVM, mesmo este apresentando maior complexidade lógica em relação ao contrato de mensagem. Esse resultado segue a mesma tendência observada anteriormente, no qual o tempo de *deploy* do contrato de mensagem também foi mais baixo na BVM quando comparado à EVM.

Essa consistência pode ser atribuída à arquitetura da rede BSV, que favorece a execução direta de contratos inteligentes na camada base da blockchain, com blocos de maior capacidade e menor sobrecarga de validação. Por outro lado, na EVM (via BSC), o *deploy* depende do modelo baseado em *gas*, o que torna o tempo de implantação mais sensível à complexidade do contrato e ao estado momentâneo da rede, podendo resultar em maiores tempos de confirmação mesmo para transações de estrutura relativamente simples.

3.3.4 Tempo de Execução – Incremento e Decremento

Após a implantação do contrato de contador em ambas as arquiteturas blockchain, foram conduzidos testes de execução para duas operações fundamentais: incremento e decremento da variável de estado interna. O objetivo foi avaliar a responsividade de cada rede à execução de instruções simples, mas que envolvem alteração de estado e, portanto, demandam validação e gravação no histórico imutável da blockchain.

As operações foram realizadas de forma isolada, com um incremento ou decremento por transação (1 a 1), nas redes *Binance Smart Chain* (EVM) e *Bitcoin Satoshi Vision* (BVM). A Tabela 6 apresenta os tempos medidos para cada operação, juntamente com os respectivos identificadores de transações.

Tabela 6 - Tempo de Execução das funções incremento e decremento do contrato contador.

Operação	**Plataforma Blockchain**	**Tempo de Execução (s)**	**Txid**
Incremento	Bitcoin Satoshi Vision	4,95	3967c270b08140deb6cfa178d12f8918f684c6545202f6d2fb95d0f9e267b422
Decremento	Bitcoin Satoshi Vision	4,12	e58577e52167cf5b1317fb77047e609b890160cb9b4c268727669a20a924885f
Incremento	Binance Smart Chain	9,82	0x461674acd97075465b6c26999d367f9e5f3d68b0dc1e4a3f0f697cf17af3b443
Decremento	Binance Smart Chain	9,84	0x1d153a2c07be1c6a92c5af144da95711c20f9cd2e254a1a0c0b7c0db7bda4a96

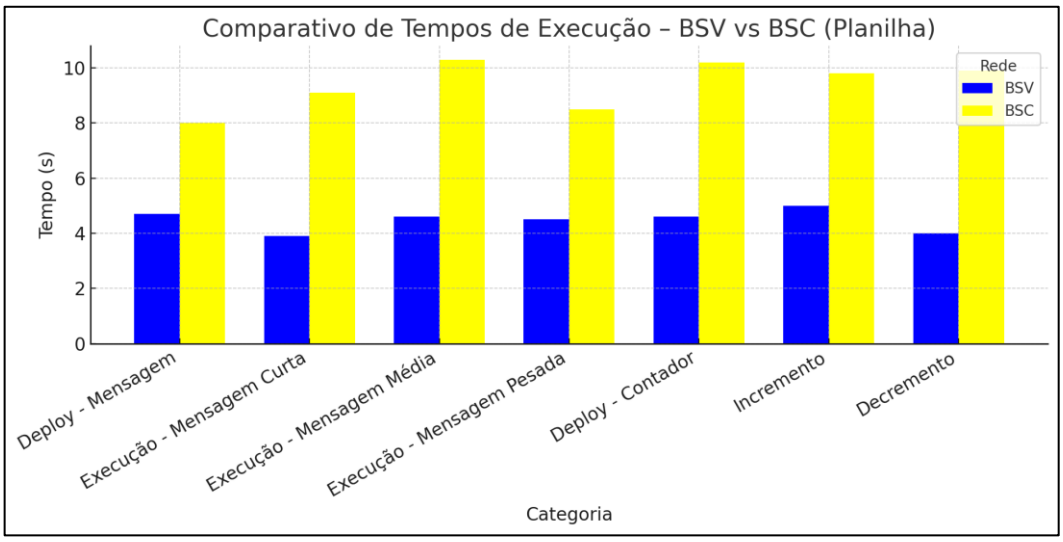
Fonte: própria.

Os resultados demonstram que a BVM (BSV) apresentou tempos de execução inferiores tanto para a operação de incremento quanto para a de decremento, em comparação à EVM (BSC). A diferença de desempenho pode ser atribuída à natureza da arquitetura BSV, que permite maior controle sobre o script de execução na camada base da blockchain e reduz o tempo de propagação e validação da transação.

Na EVM, o tempo de execução se mantém relativamente constante entre as operações, reflexo do modelo baseado em *gas*, onde as instruções básicas possuem custos computacionais definidos, mas estão sujeitas à latência da rede e à ocupação dos validadores.

A Figura 26 mostra o gráfico comparativo dos tempos de execução entre as redes BSV (BVM) e BSC (EVM), além da tabela interativa com todos os dados.

Figura 26 - Comparativo de tempos de Execução - BSV vs BSC.



Fonte: própria.

A análise dos resultados indica que a BVM (BSV) apresentou tempos de execução significativamente menores em todos os casos, com estabilidade na confirmação mesmo para mensagens mais extensas. Já a EVM (BSC) demonstrou tempos mais elevados e maior variação entre os testes, o que pode ser atribuído ao modelo de *gas fees*, congestionamento de rede e à latência do mecanismo de consenso PoSA.

3.4 COMPARAÇÃO DE CUSTOS ENTRE ARQUITETURA ETHEREUM E BITCOIN

A execução de contratos inteligentes em plataformas blockchain envolve custos relacionados à publicação (*deploy*) e interação com os contratos por meio de transações. Estes custos são pagos em moedas nativas das respectivas redes e podem influenciar significativamente a escolha da infraestrutura tecnológica, especialmente em aplicações que demandam alto volume de operações.

Nesta seção, é apresentada uma análise comparativa de custos entre duas arquiteturas de execução de contratos inteligentes: a *Binance Smart Chain* (BSC), que utiliza a *Ethereum Virtual Machine* (EVM), e a *Bitcoin Satoshi Vision* (BSV), que opera com a *Bitcoin Virtual Machine* (BVM). A comparação visa avaliar o custo operacional das transações realizadas em

cada plataforma, considerando aspectos como consumo de recursos, complexidade da execução e taxas associadas à interação com contratos inteligentes.

3.4.1 Metodologia e conversão de valores

Para a análise comparativa de custos entre as arquiteturas EVM (BSC) e BVM (BSV), foram consideradas as mesmas sete operações distintas citadas ao longo deste trabalho, estas foram executadas de forma equivalente em ambas as redes. As operações selecionadas refletem interações representativas com contratos inteligentes e incluem:

1. Implantação (*deploy*) do contrato de mensagem;
2. Execução com mensagem curta;
3. Execução com mensagem de tamanho médio;
4. Execução com mensagem longa;
5. Implantação do contrato contador;
6. Operação de incremento (1 a 1);
7. Operação de decremento (1 a 1).

Os valores das taxas de transação foram obtidos diretamente a partir dos exploradores oficiais de cada rede, *BscScan* para a *Binance Smart Chain* e *WhatsOnChain* para a *Bitcoin Satoshi Vision*, por meio da análise do campo *Transaction Fee* presente em cada transação registrada (TXID).

Para fins de padronização e conversão dos custos em moeda fiduciária (dólar americano), foram utilizados valores médios das cotações observadas nos últimos seis meses, com base até maio de 2025. As taxas de câmbio consideradas foram as seguintes:

- 1 BNB ≈ US$ 649,22
- 1 BSV ≈ US$ 77,80
- 1 BSV = 100.000.000 satoshis → 1 satoshi ≈ US$ 0.000000778

Foi realizada a conversão de BSV para satoshi para fins de simplificação. Esses valores serviram de base para a conversão dos custos unitários, permitindo uma análise comparável do custo efetivo em cada arquitetura de execução de contratos inteligentes.

3.4.2 Tabela Comparativa de custos

A Tabela 7 apresenta os custos consolidados por tipo de operação em cada arquitetura:

Tabela 7 - Tabela de Comparação de custos de *deploy*/execução de contratos das Arquiteturas EVM e BVM.

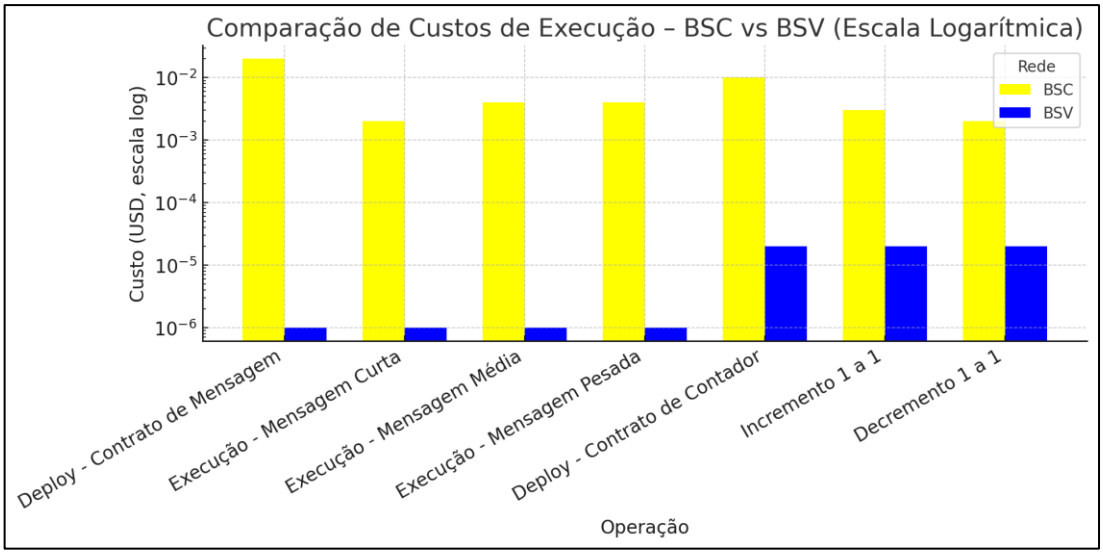
Operação	**Rede**	**TXID**	**Taxa de Transação**	**Equivalente em USD**
Deploy - Contrato de Mensagem	BSC	0xd15fa6dea0fc0e115f9601ba4211a35c92b739127e076c97d925afb09061e7f3	0,0000569196	US$ 0.03695
Deploy - Contrato de Mensagem	BSV	220f380ca81af2f8c4b05de51fddb0b8a4b59830c7b2eea678f17e2feaf2eff33	2 Satoshi	US$ 0,000001556
Execução - Mensagem Curta	BSC	0x8ae1b61365df9fed759c1ade96d333336a0613ad9b61581fc2e4e932fb5ebc54	0.0000032168	US$ 0.00209
Execução - Mensagem Curta	BSV	f93b27202ab9677eb52b278afa2489b44ee6cb7dbc172100ae25babe64527924	2 Satoshi	US$ 0,000001556
Execução - Mensagem Média	BSC	0x79c1a48e0ec5fa00c4862a66cbd57d76d3b8e15a3bea89ab3f2d82059fe260d8	0.0000077602	US$ 0.00504
Execução - Mensagem Média	BSV	fcc4a7434b2b4d421f67483fa5dec0874a1c82f7b40ac8ac426594ef166d7a87	2 Satoshi	US$ 0,000001556
Execução - Mensagem Pesada	BSC	0xf944f09642974fdc3a2d15b58f985b7e7e84b8cfcb0b2e7a8074fbf3d5672db1	0.0000078328	US$ 0.00509
Execução - Mensagem Pesada	BSV	fcc4a7434b2b4d421f67483fa5dec0874a1c82f7b40ac8ac426594ef166d7a87	2 Satoshi	US$ 0,000001556

Deploy - Contrato de Contador	BSC	0xdf5b5e00911546b97483b02766ca7cabb816c9827d947cf8b2c519e3570ee440	0.0000182866	US$ 0.01187
Deploy - Contrato de Contador	BSV	0377220aa2ce2ca48aba9e50cf66cb4daf2765b2f51ec293b52f2051cb979259	5 Satoshi	US$ 0,00000389
Incremento 1 a 1	BSC	0x461674acd97075465b6c26999d367f9e5f3d68b0dc1e4a3f0f697cf17af3b443	0.0000044866	US$ 0.00291
Incremento 1 a 1	BSV	3967c270b08140deb6cfa178d12f8918f684c6545202f6d2fb95d0f9e267b422	5 Satoshi	US$ 0,00000389
Decremento 1 a 1	BSC	0x1d153a2c07be1c6a92c5af144da95711c20f9cd2e254a1a0c0b7c0db7bda4a96	0.0000022919	US$ 0.00149
Decremento 1 a 1	BSV	e58577e52167cf5b1317fb77047e609b890160cb9b4c268727669a20a924885f	5 Satoshi	US$ 0,00000389

Fonte: própria.

A Figura 27 apresenta uma comparação dos custos de execução de contratos inteligentes nas redes BSC (Binance Smart Chain) e BSV (*Bitcoin Satoshi Vision*), considerando diferentes tipos de operação, como *deploy* de contratos, execução de mensagens e atualizações de estado.

Figura 27 - Gráfico de comparação de custos de execução – BSC vs BSV.



Fonte: própria.

O gráfico está representado em escala logarítmica no eixo vertical (eixo Y) para permitir a visualização adequada dos valores extremamente baixos observados na rede BSV. Sem esse ajuste, as barras correspondentes à BSV seriam praticamente imperceptíveis diante dos valores da BSC, devido à diferença de ordens de magnitude entre os custos das duas redes.

A análise revela diferenças marcantes entre os modelos econômicos das redes BSC e BSV. A BSC, baseada em EVM, apresenta valores de transação relativamente baixos quando comparada a outras redes *ethereum*, especialmente devido à sua arquitetura otimizada e baixa concorrência na *testnet*. Ainda assim, mesmo com essas otimizações, os valores absolutos em dólar das transações na BSC são notavelmente superiores àqueles observados na rede BSV.

A BSV demonstrou custos extremamente reduzidos em todas as operações. A maior taxa registrada foi inferior a 1.000 *satoshis*, o que corresponde a menos de US$ 0.001 por transação. Isso evidencia o modelo altamente escalável e eficiente de sua arquitetura baseada em *scripts* e execução direta na camada base, sem dependência de máquinas virtuais intermediárias ou mecanismos complexos de precificação de *gas*.

Do ponto de vista econômico, a BSV mostra-se mais vantajosa para aplicações que exijam alto volume de interações com contratos inteligentes. O baixo custo por transação pode ser decisivo em cenários de uso como microtransações, armazenamento de dados em cadeia e execução de lógica intensiva.

3.5 COMPLEXIDADE DE CÓDIGO E OCUPAÇÃO EM REDE

Nesta seção, será realizada uma análise comparativa entre os contratos inteligentes desenvolvidos nas linguagens *Solidity* (para a arquitetura EVM, via BSC) e *sCrypt* (para a arquitetura BVM, via BSV), com foco em dois aspectos principais: a complexidade estrutural do código-fonte e a ocupacão em *bytes* na blockchain durante o *deploy* dos contratos.

3.5.1 Análise Estrutural do Código

Do ponto de vista da estrutura de código, os contratos em *Solidity* apresentam uma sintaxe concisa, organizada em blocos de funções e eventos, com suporte direto a variáveis de estado e métodos de acesso. O contrato de mensagem inclui controle de permissão com *modifier*, eventos e interação com variáveis do tipo *string*, enquanto o contrato contador manipula variáveis inteiras e dispara eventos a cada operação.

Já os contratos em *sCrypt* a estrutura é altamente tipada, com uso de tipos como *ByteString* e *PubKey*, e a execução lógica se baseia no uso de *assert*, refletindo a lógica determinística da BVM.

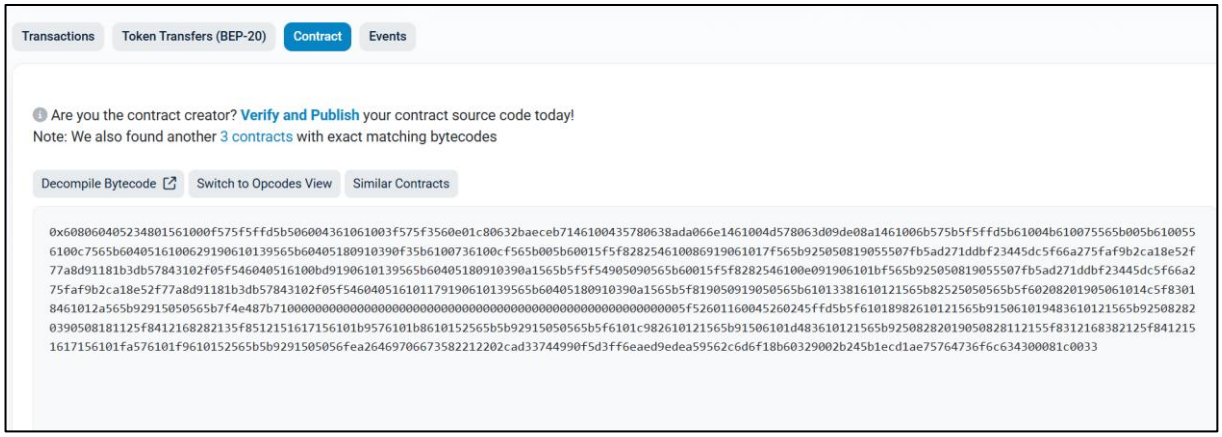
A complexidade conceitual em *sCrypt* tende a ser mais elevada por exigir compreensão de aspectos de baixo nível do script de Bitcoin, enquanto a linguagem *Solidity* abstrai boa parte dessas operações com interfaces de alto nível.

3.5.2 Ocupação de Rede: Tamanho das Transações de *Deploy*

Para mensurar o impacto da complexidade do contrato na ocupação da blockchain, foram comparados os tamanhos das transações de *deploy* na BSV e na BSC. No caso da BSV, o tamanho da transação (*Size*) foi obtido diretamente via explorador *WhatsOnChain*. Para a BSC, como o *BscScan* não fornece o tamanho diretamente, foi realizada a contagem dos caracteres hexadecimais do *bytecode* do contrato e posteriormente dividida por 2, considerando que 2 caracteres hex equivalem a 1 byte.

O contrato de contador na rede BSC gerou o seguinte *Bytecode* (conjunto de instruções que uma máquina virtual interpreta e executa) com tamanho 1.134 caracteres, isto é, 567 Bytes conforme Figura 28 ilustra:

Figura 28 - *Bytecode* do contrato contador na rede BSC



Fonte: própria.

A Tabela 8 apresenta os valores consolidados:

Tabela 8 - Ocupação em Bytes dos Contratos (Deploy)

Contrato	**Rede**	**Tamanho (Bytes)**
Mensagem	BSV	1.759
Mensagem	BSC	764
Contador	BSV	4.442
Contador	BSC	567

Fonte: própria.

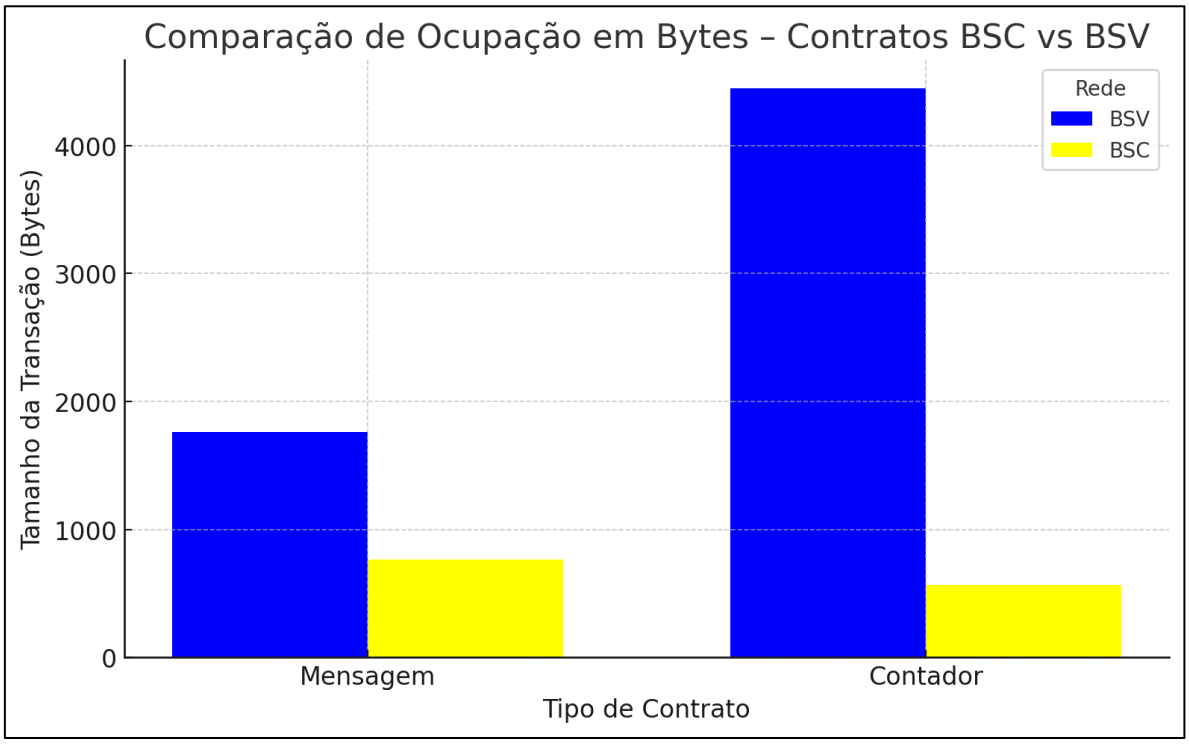
A análise dos dados revela que os contratos em *sCrypt* (BSV) têm, em geral, maior ocupação em bytes na blockchain quando comparados aos contratos em Solidity (BSC). Isso ocorre principalmente devido à inclusão detalhada de dados de inicialização, assinaturas, publicação de chaves e lógicas nativas no script, que exigem codificações mais extensas.

Por outro lado, os contratos em *Solidity* são implantados em *bytecode* EVM, altamente otimizados por compiladores como o solc, que realizam compressão e minimização do código. Além disso, muitos recursos internos da rede Ethereum (como gestão de estado, armazenamento e controle de permissão) são abstraídos de forma eficiente.

Ainda assim, é importante considerar que o maior tamanho de ocupação na BSV é compensado pelo baixo custo por byte e pela alta capacidade de bloco da rede. Isso permite transações mais pesadas sem impacto significativo na performance, o que pode ser vantajoso para aplicações que demandam scripts mais verbosos ou manipulação direta de dados.

A fim de aprimorar a visualização da diferença na ocupação de rede entre os contratos implementados em BSC e BSV, foi gerado o gráfico conforme figura 25. Ele ilustra o tamanho, em bytes, das transações de deploy para os contratos de mensagem e de contador em cada arquitetura, permitindo uma comparação direta e objetiva sobre o impacto do código na utilização da blockchain.

Figura 29 - Comparação De Ocupação Em Bytes – Contratos BSC Vs BSV



Fonte: própria.

3.6 TESTES DE ESTRESSE E COMPORTAMENTO SOB ATAQUES NO CONTRATO DE CONTADOR

Esta seção é dedicada à avaliação do desempenho e comportamento das arquiteturas EVM e BVM frente à execução de ataques de sobrecarga simulados em contratos inteligentes do tipo contador. Os testes foram projetados para estressar a infraestrutura dos contratos por

meio de execuções repetitivas de atualizações de estado em sequência, replicando cenários de uso intensivo ou malicioso.

Para isso, foram definidos três níveis de ataque: execução em lote de 10, 50 e 100 chamadas consecutivas às funções de incremento ou decremento. Esses testes visam observar o tempo de resposta, o impacto no estado interno do contrato, e os limites operacionais de cada arquitetura, fornecendo uma visão mais realista sobre sua resiliência e escalabilidade.

A implementação dos ataques foi automatizada por meio de scripts personalizados, que executam as transações de forma contínua e monitorada, simulando um comportamento hostil ou de estresse de sistema. Essa abordagem permite maior precisão na coleta de dados de tempo, custo de execução e integridade do estado final do contrato.

É importante destacar uma distinção fundamental entre as arquiteturas testadas:

- Na rede BSC, baseada na EVM, a mudança de estado de um contrato ocorre de maneira centralizada, com o armazenamento das variáveis internas diretamente associadas ao endereço do contrato. Assim, múltiplas chamadas afetam o mesmo *script* e a mesma instância de contrato, permitindo que o estado seja atualizado sequencialmente sem a necessidade de novos contratos.
- Na rede BSV, que adota o modelo UTXO e a Bitcoin Virtual Machine, cada modificação no estado do contrato implica na geração de um novo script associado a um novo UTXO. Ou seja, o estado é reconstruído a cada nova transação, criando uma cadeia de UTXOs que representam a evolução do contrato ao longo do tempo. Essa característica torna a execução de ataques em lote mais desafiadora, exigindo que o script seja constantemente reimplementado com o novo estado como entrada.

Esta distinção entre os modelos de execução, modelo baseado em conta EVM versus modelo baseado em UTXO BVM, impacta diretamente na forma como contratos reagem a execuções consecutivas, no controle de concorrência, e na forma de automação dos testes.

3.7 ATTACK 10X, 50X E 100X

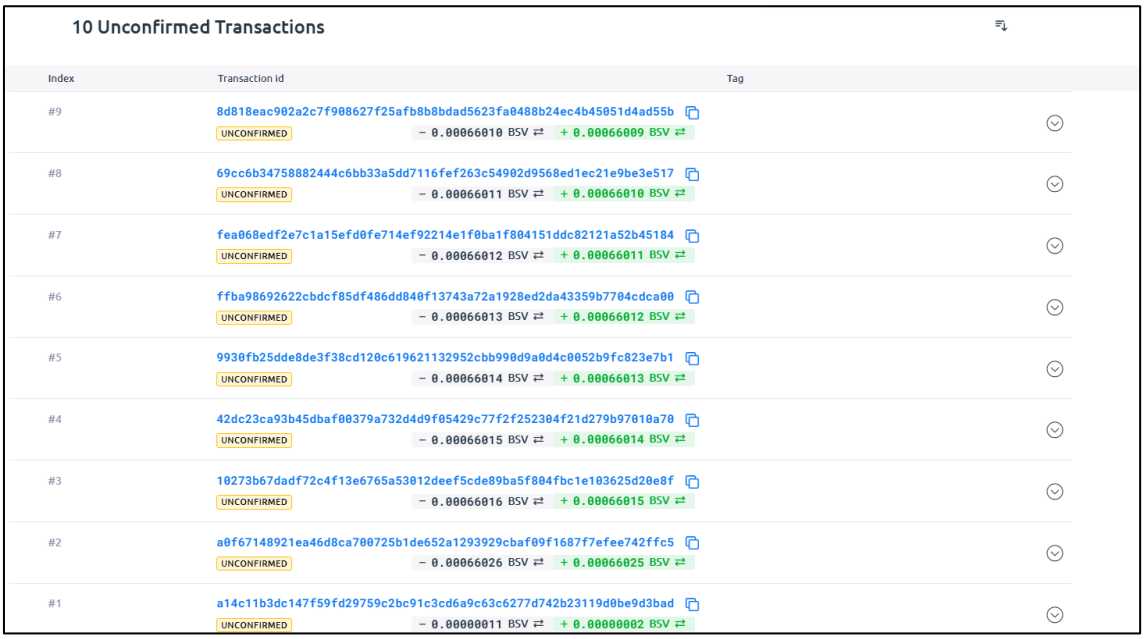
Os testes de ataque em lote foram inicialmente conduzidos na rede BSC, baseada na EVM, com o objetivo de avaliar o comportamento do contrato inteligente do tipo contador sob diferentes cargas de execução. As execuções foram realizadas com 10, 50 e 100 chamadas consecutivas à função de incremento, utilizando scripts automatizados com controle de tempo e tentativa contínua em caso de erro de rede.

É importante ressaltar que, nesta fase do experimento, não foram consideradas as taxas referentes à implementação do contrato nem a ocupação em rede, uma vez que essas métricas já foram analisadas em seções anteriores. Como o ataque consiste apenas na atualização de estado interno, o tamanho do contrato permanece constante, não tendo modificações estruturais no *bytecode* original.

A cada execução de ataque (*Attack* 10x, 50x, 100x), foi iniciado um contador de tempo para mensurar a duração total até a submissão da última transação, bem como o custo total envolvido em cada operação. Após a finalização dos testes na BSC, a mesma metodologia foi aplicada à rede BSV, que opera sob o paradigma UTXO e adota a BVM.

Uma distinção fundamental entre as redes foi evidenciada durante os testes: enquanto na BSC o saldo do contrato é atualizado imediatamente após a submissão da transação (modelo *account-based*), na BSV o novo estado só é efetivado após a confirmação em bloco, devido à natureza do modelo *UTXO*. Como resultado, as transações geradas durante os ataques na rede BSV permanecem inicialmente na *mempool* aguardando confirmação, conforme ilustrado na Figura 30, que apresenta as 10 transações não confirmadas resultantes de um dos ataques realizados.

Figura 30- Registro de ataques de incremento 10x na rede BNB.



Fonte: própria.

Este comportamento reforça a complexidade do controle de estado na BVM, onde cada novo incremento demanda a reconstrução de um script de contrato a partir do UTXO anterior, criando uma cadeia sequencial de dependências. Tal característica impacta diretamente no

tempo de execução e na robustez necessária para lidar com conflitos de *mempool* ou latências na propagação de transações.

Ao final da execução de todos os cenários de ataque em ambas as redes, os dados coletados foram consolidados e organizados na tabela 8, servindo como base para as análises comparativas de desempenho e custo.

Tabela 9 - Resultados obtidos em Ataques nos contratos

Operação	**Rede**	**Taxa de Transação (Cripto)**	**Equivalente em USD**	**Tempo total de execução**
Attack 10x	BSC	0.000029476 BNB	US$ 0.01914	01:15.08
Attack 10x	BSV	0.00000017	US$ 0.00001323	01:15.59
Attack 50x	BSC	0.00014054 BNB	US$ 0.09124	06:12.15
Attack 50x	BSV	0.00000057	US$ 0.00004434	06:36.56
Attack 100x	BSC	0.00027937 BNB	US$ 0.18141	11:21.19
Attack 100x	BSV	0.00000151	US$ 0.00011756	21:54.53

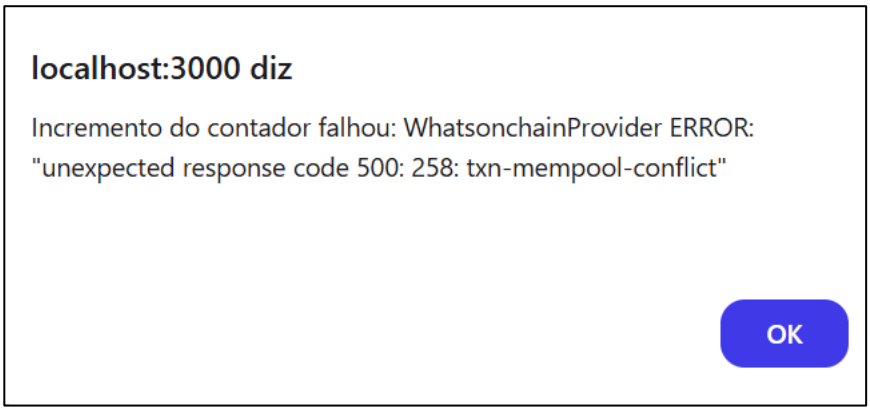
Fonte: própria.

Com os ataques em lote devidamente executados nas duas plataformas, foram observadas diferenças relevantes quanto à estabilidade, latência e comportamento das redes frente a operações contínuas de modificação de estado.

No ambiente da BSC, os contratos reagiram de forma estável às execuções em lote. Mesmo com 100 chamadas consecutivas, a arquitetura baseada em conta foi capaz de manter a consistência do estado interno sem apresentar falhas críticas. A atualização sequencial do contador ocorreu em uma única instância de contrato, com desempenho compatível com o volume de requisições, beneficiada por mecanismos de paralelismo e memorização do estado.

Já na BSV, os ataques exigiram um controle mais refinado, dado que cada transação cria um UTXO com um novo script. A tentativa de enviar múltiplas execuções consecutivas em intervalos curtos resultou em conflitos na *mempool* como o erro *txn-mempool-conflict* (Figura 31), exigindo lógica de reenvio, espera de propagação e monitoramento de confirmação. Isso se deve ao fato de que o novo script só pode ser validado quando o UTXO anterior já estiver reconhecido na rede como gasto.

Figura 31 - Conflito de transação em *mempool*.

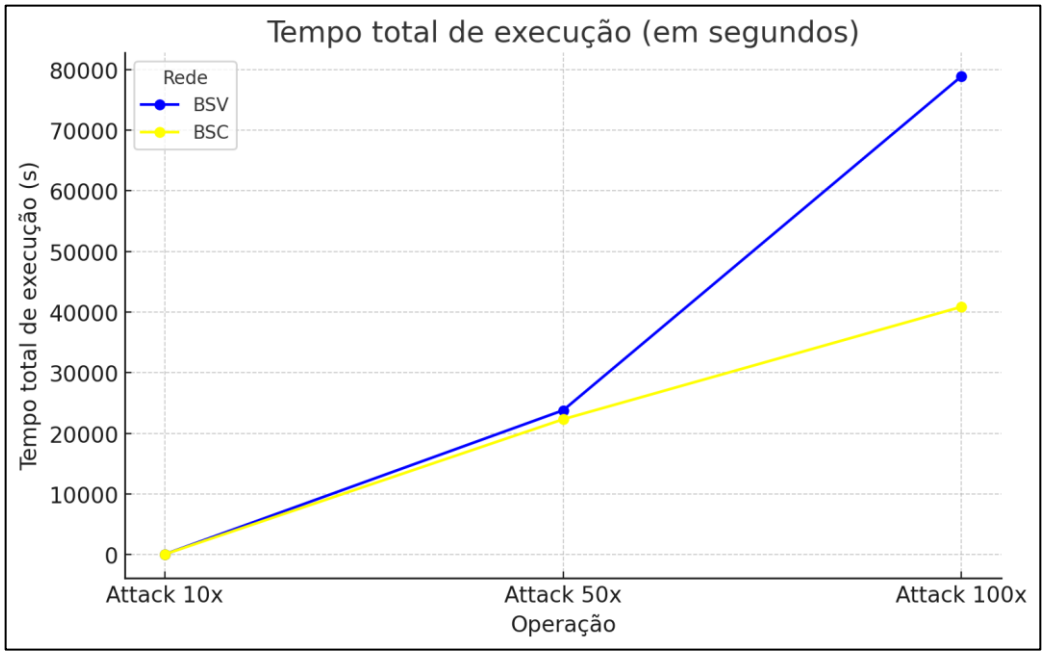


Fonte: própria.

Essa diferença estrutural refletiu-se nos tempos totais de execução, que foram maiores na BSV conforme o número de execuções aumentava, especialmente devido ao tempo de propagação de cada UTXO. Em contrapartida, a rede demonstrou maior previsibilidade na propagação, desde que respeitado um tempo mínimo de intervalo entre transações.

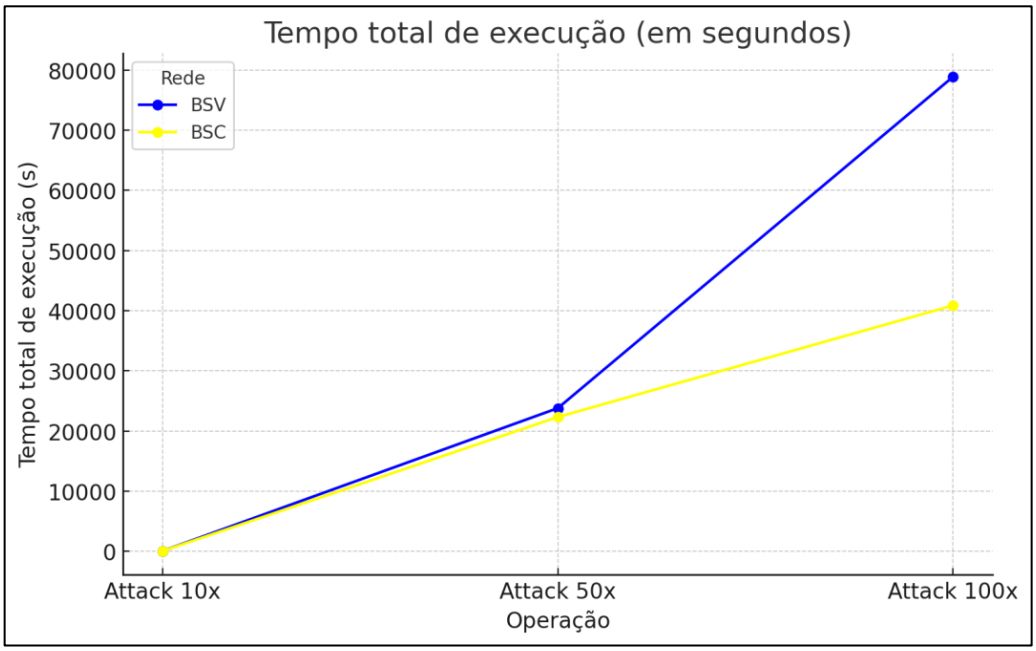
Os gráficos a seguir (Figura 32 e Figura 33) apresentam uma análise comparativa entre as arquiteturas EVM (BSC) e BVM (BSV) a partir dos testes de ataque realizados com 10, 50 e 100 execuções consecutivas da função de incremento.

Figura 32 - Tempo de execução nas operações de Ataque.



Fonte: própria.

Figura 33 - Custo de execução nas operações de Ataque



Fonte: própria.

Os dados evidenciam, em primeiro plano, o tempo total de execução, indicando que, embora a BSC tenda a manter uma linearidade no tempo com o aumento da carga, a BSV apresenta uma elevação mais acentuada devido ao modelo UTXO, que exige a reconstrução do contrato a cada transação. Em seguida, o gráfico de taxas de transação (em USD) demonstra que a BSC, apesar de custos médios maiores por transação, mantém valores absolutos mais baixos em cargas elevadas, enquanto a BSV, mesmo com taxas unitárias baixas, acumula valores crescentes à medida que o número de chamadas aumenta.

Por fim, a relação entre tempo de execução e custo destaca o impacto do modelo de execução em cada arquitetura, evidenciando as vantagens da BSC em cenários de alta demanda e a limitação operacional da BSV em ambientes com transações intensivas e contínuas. Esses resultados oferecem subsídios concretos para a escolha de uma arquitetura mais adequada conforme o perfil de uso e os requisitos de escalabilidade do sistema.

4 CONCLUSÃO

Os testes práticos conduzidos neste capítulo revelaram diferenças substanciais entre as arquiteturas EVM (BSC) e BVM (BSV) em quatro dimensões centrais: velocidade de execução, custo por transação, complexidade do código-fonte e ocupação em rede. A análise foi realizada com dois tipos de contrato – mensagem e contador – englobando operações de *deploy*, execução com diferentes cargas de dados e manipulação de estado por meio de incrementos e decrementos unitários.

De forma geral, a BVM apresentou tempos de execução consistentemente inferiores em todas as operações analisadas. No caso do *deploy* dos contratos, tanto o contrato de mensagem quanto o de contador foram implantados em menos de 5 segundos na BSV, ao passo que os mesmos contratos levaram entre 7,95 e 10,28 segundos na BSC.

A execução das transações de mensagem seguiu o mesmo padrão: mesmo com o aumento do volume de dados (mensagens curta, média e longa), a BVM manteve variação reduzida entre 3,89 e 4,52 segundos, enquanto a EVM apresentou maior instabilidade e tempos mais elevados, variando de 8,44 a 10,43 segundos. No contrato de contador, as operações de incremento e decremento 1 a 1 foram executadas abaixo de 6 segundos na BSV, contra aproximadamente 9,8 segundos na BSC.

Esses resultados indicam que a arquitetura da BVM é mais eficiente do ponto de vista de latência, especialmente em ambientes de teste e baixa concorrência. A possibilidade de execução direta de scripts na camada base, combinada com a estrutura de blocos de alta capacidade da BSV, favorece operações com maior volume de dados ou lógica mais densa. Já na BSC, a latência está diretamente atrelada ao modelo de *gas* e à demanda momentânea da rede.

No aspecto econômico, os dados observados na BSC demonstram alta eficiência: a maioria das transações apresentou custo inferior a US$ 0,01, graças ao modelo otimizado de *gas* e ao uso de contratos padronizados altamente compatíveis com o ecossistema EVM.

Entretanto, a rede BSV apresentou custos ainda mais reduzidos, com taxas de transação da ordem de 2 satoshi o que equivale a frações de centavo em dólar. Esse modelo é particularmente vantajoso para aplicações de alta frequência, microtransações ou armazenamento de dados em cadeia, onde o custo por byte se torna determinante.

A comparação entre os contratos desenvolvidos em *Solidity* (BSC) e *sCrypt* (BSV) evidencia diferenças importantes na filosofia de programação. *Solidity* oferece uma sintaxe de alto nível, compacta e orientada a objetos, com abstrações que facilitam o controle de estado,

segurança e permissões. Já o *sCrypt*, embora fortemente tipado e modular, exige familiaridade com conceitos de scripts Bitcoin e manipulação explícita de dados, o que aumenta a curva de aprendizado, mas proporciona maior controle lógico.

A ocupação de rede também refletiu tais diferenças: os contratos da BSV apresentaram tamanhos de transações de *deploy* significativamente maiores do que os da BSC. Por exemplo, o contrato de mensagem ocupou 1.759 bytes na BSV contra 764 bytes na BSC. O contrato de contador teve 1.390 bytes na BSV frente a apenas 567 bytes na BSC.

Essa diferença decorre da forma como cada rede representa os contratos: na BSC, o *bytecode* EVM é altamente otimizado pelo compilador; na BSV, os dados são mais explícitos e embutem lógica diretamente nos scripts. Apesar disso, a BSV compensa essa maior ocupação com uma política de taxas proporcionalmente mais baixa e maior capacidade por bloco.

Para melhor representar as diferenças na ocupação de rede, foi elaborado a tabela a seguir, que compara o tamanho das transações de *deploy* dos contratos de mensagem e contador entre as redes BSC e BSV.

Tabela 10 - Síntese Comparativa

Dimensão	**BSC (EVM)**	**BSV (BVM)**
Tempo de Execução	Maior (7–10 s)	Menor (3–5 s)
Custo por Transação	Baixo (< US$ 0,01)	Muito baixo (< US$ 0,001)
Complexidade do Código	Alta abstração, fácil adoção	Controle detalhado, maior curva de entrada
Ocupação em Bytes	Mais compacta	Mais extensa, mas compensada por custo

Fonte: própria.

Os testes de ataque em lote realizados com 10, 50 e 100 execuções consecutivas das funções de incremento revelaram importantes nuances sobre o comportamento das arquiteturas EVM e BVM sob condições de estresse. A principal métrica observada foi o tempo total necessário para concluir as operações, considerando-se um contrato previamente implantado e o uso de scripts automatizados para disparo das transações.

Na rede BSC, observou-se um crescimento quase linear do tempo total de execução conforme o aumento da carga. Os tempos para ataques 10x, 50x e 100x foram, respectivamente, 01:15.08, 06:12.15 e 11:21.19. A estabilidade da EVM nesse cenário reflete a capacidade do

modelo *account-based* de manipular estado interno do contrato de forma sequencial e síncrona, sem a necessidade de reimplantação de script.

Por outro lado, a rede BSV, ao operar sobre o modelo UTXO, demonstrou comportamento mais complexo. Cada incremento exigiu a geração de um novo script com estado atualizado, resultando em maior tempo de execução acumulado: 01:15.59 (10x), 06:36.56 (50x) e 21:54.53 (100x). Apesar da sobrecarga estrutural, a BVM manteve a execução estável e com elevado índice de sucesso, mesmo diante de conflitos de *mempool* ocasionais, que foram tratados por meio de tentativas automáticas com monitoramento.

Quanto ao custo, ambas as redes mantiveram taxas acessíveis. A BSC apresentou um pequeno aumento conforme o volume de chamadas, com destaque para o ataque 100x, que totalizou aproximadamente 0.00027937 BNB (cerca de US$ 0,18). Já a BSV seguiu com taxas extremamente baixas, inferiores a US$ 0,001 em todos os casos — mesmo no cenário mais intenso, o custo foi de apenas 151 satoshi (aproximadamente US$ 0.000117). Esse diferencial evidencia a eficiência econômica da BVM, especialmente para aplicações com alta frequência de atualização de estado.

A BSC, com seu modelo baseado em contas (*account-based*), demonstrou desempenho mais linear e previsível, com tempos de execução menores que os da BSV nos testes mais longos (50x e 100x). Sua estrutura facilita a manipulação direta do estado interno do contrato, tornando-a ideal para aplicações interativas em tempo real, como jogos blockchain, sistemas de votação instantânea, ou plataformas *DeFi* com alta taxa de chamadas por segundo. Além disso, a atualização imediata do saldo após a submissão da transação permite respostas rápidas na interface do usuário, o que é desejável em experiências web3 fluidas.

Em contrapartida, a BSV demonstrou notável robustez e estabilidade mesmo sob ataques com alto número de transações consecutivas, ainda que com maior tempo acumulado, O custo extremamente reduzido por transação (frações de centavo) posiciona a rede como altamente competitiva para aplicações que exigem escalabilidade econômica, como microtransações recorrentes, rastreamento de ativos, registros de auditoria imutáveis, ou sistemas de certificação de dados com necessidade de grande volume e baixo custo por operação.

Em resumo:

Tabela 11 – Recomendações de contratos para redes

Rede	Recomendada para	Justificativa
BSC (EVM)	Aplicações com alta interatividade e resposta em tempo real (jogos, DeFi, DApps front-end intensivos)	Atualização de estado imediata, modelo simplificado e alta compatibilidade
BSC (EVM)	Aplicações com alta frequência e baixo valor por transação (micro pagamentos, certificações, rastreamento imutável)	Custo unitário extremamente baixo, alta escalabilidade em volume, e encadeamento de estado mais seguro

Fonte: própria.

Os resultados obtidos demonstram que não há um modelo único superior, mas sim adequações diferentes para necessidades distintas. A escolha entre EVM e BVM deve considerar o tipo de lógica contratual, o volume de atualizações, o custo tolerável por transação e a criticidade da rastreabilidade do estado ao longo do tempo.

5 TRABALHOS FUTUROS

Diante dos resultados obtidos, identificam-se diversas possibilidades de expansão e aprofundamento da pesquisa em futuras investigações.

Em primeiro lugar, recomenda-se a inclusão da rede Solana em testes comparativos de desempenho. Solana tem se destacado no cenário global por sua alta taxa de transações por segundo (TPS) e seu modelo híbrido de consenso *Proof of History* (PoH) combinado ao *Proof of Stake*. A análise de contratos inteligentes em Solana permitiria explorar o impacto do processamento paralelo em larga escala sobre operações de leitura e escrita de estado, além de avaliar como o modelo baseado em contas desta blockchain difere estruturalmente da BSC (EVM) e da BSV (BVM). Testes com o framework *Anchor*, bastante utilizado na Solana, também poderiam ampliar a análise da complexidade de desenvolvimento frente às demais linguagens avaliadas (*Solidity* e *sCrypt*).

Ampliar os testes de ataques (já iniciados com stress 10x, 50x, 100x) para vulnerabilidades conhecidas em contratos inteligentes (*reentrancy*, *overflow*, *frontrunning*).

Outro trabalho futuro relevante consiste na implementação de testes sob condições de carga real, com múltiplas transações simultâneas emitidas por diferentes endereços, simulando cenários de concorrência em ambiente de rede pública. Isso permitiria uma avaliação mais precisa da resiliência e escalabilidade das arquiteturas frente a ataques de negação de serviço (DoS) ou sobrecarga transacional, complementando os testes sequenciais realizados neste trabalho.

Por fim, seria valioso desenvolver um módulo automatizado de benchmarking de contratos inteligentes *multi-chain*, com interface gráfica e capacidade de gerar relatórios técnicos de desempenho, visando fomentar a adoção desse tipo de análise por desenvolvedores, pesquisadores e empresas que desejam selecionar a arquitetura mais adequada a seus casos de uso.

Tais extensões não apenas ampliariam a robustez da comparação entre diferentes arquiteturas blockchain, como também promoveriam uma compreensão mais ampla dos desafios e oportunidades envolvidos no desenvolvimento de aplicações descentralizadas de alto desempenho

6 REFERÊNCIAS

ANTONOPOULOS, Andreas M.; WOOD, Gavin. *Mastering Ethereum: Building Smart Contracts and DApps*. 1. ed. Sebastopol: O'Reilly Media, 2018. Acesso em: 18 março 2025.

BINANCE. *Analysis of BscScan concept and usage*. 2025. Disponível em: https://www.binance.com/en/square/post/43215. Acesso em: 12 maio 2025.

BINANCE. *BNB Smart Chain (BSC): bring smart contracts to BNB Chain*. 2025. Disponível em: https://www.bnbchain.org/en/bnb-smart-chain. Acesso em: 14 março 2025.

BNB CHAIN. *BNB Smart Chain Testnet Faucet*. Disponível em: https://www.bnbchain.org/en/testnet-faucet. Acesso em: 29 maio 2025.

BSCSCAN. *BscScan – Binance Smart Chain Explorer*. Disponível em: https://bscscan.com/. Acesso em: 29 maio 2025.

BUTERIN, Vitalik. *A next-generation smart contract and decentralized application platform*. 2014. Disponível em: https://ethereum.org/en/whitepaper/. Acesso em: 10 fevereiro 2025.

CACHIN, Christian; GUERRAOUI, Rachid; RODRIGUES, Luís. *Introduction to Reliable and Secure Distributed Programming.* 2. ed. Berlin: Springer, 2011.

CHEN, Y.; PENDLETON, M.; NJILLA, L.; XU, S. *A survey on Ethereum systems security: vulnerabilities, attacks and defenses*. ACM Computing Surveys, v. 53, n. 3, p. 1-43, 2020.

CHRISTIDIS, K.; DEVETSIKIOTIS, M. Blockchains and smart contracts for the Internet of Things. *IEEE Access*, v. 4, p. 2292-2303, 2016.

CHRISTIDIS, Konstantinos; DEVETSIKIOTIS, Michael. *Blockchains and smart contracts for the Internet of Things.* IEEE Access, v. 4, p. 2292–2303, 2016. DOI: 10.1109/ACCESS.2016.2566339

CRUZ, Carlos. *UTXO vs Account-Based Blockchains: Entenda as diferenças entre os dois modelos*. Dio.me, 2022. Disponível em: https://www.dio.me/articles/utxo-vs-account-based-blockchains. Acesso em: 04 jun. 2025.

DANNEN, Chris. *Introducing Ethereum and Solidity: Foundations of Cryptocurrency and Blockchain Programming for Beginners*. Berkeley: Apress, 2017.

EMN178. *Keccak-256 Online*. Disponível em: https://emn178.github.io/online-tools/keccak_256.html. Acesso em: 19 maio 2025.

EMN178. *SHA256 Online*. Disponível em: https://emn178.github.io/online-tools/sha256.html. Acesso em: 19 maio 2025.

HIRAI, Y. *Formal verification of smart contracts*. 2017. Disponível em: https://yoichihirai.com/papers/Formal_Verification_of_Smart_Contracts.pdf. Acesso em: 05 abril 2025.

IBAÑEZ, Juan Ignacio; FREIER, Alexander. Bitcoin's Carbon Footprint Revisited: Proof of Work Mining for Renewable Energy Expansion. *Challenges*, v. 14, n. 3, p. 35, 2023. DOI: 10.3390/challe14030035.

MENDLING, Jan et al. *Blockchains for Business Process Management—Challenges and Opportunities.* ACM Transactions on Management Information Systems (TMIS), v. 9, n. 1, p. 1–16, 2018. DOI: 10.1145/3183367.

NAKAMOTO, Satoshi. *Bitcoin: a peer-to-peer electronic cash system*. 2008. Disponível em: https://bitcoin.org/bitcoin.pdf. Acesso em: 14 março 2025.

NARAYANAN, Arvind et al. *Bitcoin and cryptocurrency technologies: a comprehensive introduction*. Princeton: Princeton University Press, 2016.

NASDAQ. Research: Bitcoin Consumes Less Than Half the Energy of the Banking or Gold Industries. Nasdaq.com, 2023. Disponível em: https://www.nasdaq.com/articles/research%3A-bitcoin-consumes-less-than-half-the-energy-of-the-banking-or-gold-industries. Acesso em: 11 agosto 2025.

SCHNEIER, Bruce**.** *Applied Cryptography: Protocols, Algorithms, and Source Code in C.* 2. ed. New York: John Wiley & Sons, 1996.

SCRYPT. *sCrypt BSV Faucet*. Disponível em: https://scrypt.io/faucet. Acesso em: 29 maio 2025.

ŠINKEC, Matija. *What people get wrong about Bitcoin and smart contracts*. BSV Blockchain, 2022. Disponível em: https://bsvblockchain.org/what-people-get-wrong-about-bitcoin-and-smart-contracts/. Acesso em: 05 de abril 2025.

SIPSER, Michael. *Introduction to the Theory of Computation*. 3. ed. Boston: Cengage Learning, 2012.

SZABO, Nick. *Smart Contracts: Building Blocks for Digital Markets.* 1996. Disponível em: https://www.fon.hum.uva.nl/rob/Courses/InformationInSpeech/CDROM/Literature/LOTwinterschool2006/szabo.best.vwh.net/smart_contracts_2.html. Acesso em: 12 ago. 2025.

WHATSONCHAIN. *Bitcoin SV Block Explorer*. Disponível em: https://whatsonchain.com/. Acesso em: 29 maio 2025.

WOOD, Gavin. *Ethereum: A Secure Decentralised Generalised Transaction Ledger (Yellow Paper).* Ethereum.

XU, Xiao et al. *A Survey on Blockchain Oracles: Technical Challenges, Commercial Applications and Security*. IEEE Access, v. 7, p. 164 830–164 847, 2019. Acesso em: 18 março 2025.