# Transição para Criptografia Pós-Quântica em Redes Blockchain de Alta Performance: Uma Análise da Arquitetura Solana

**Arthur T. Oliveira**[1] **, Ana Cristina dos Santos**[1]

[1]Instituto de Tecnologia e Liderança
Av. Professor Almeida Prado, 520, Butantã – 05508-901 – São Paulo – SP – Brasil

arthur.oliveira@sou.inteli.edu.br, anacris.santos@prof.inteli.edu.br

***Abstract.*** *The emergence of cryptographically relevant quantum computers poses an existential threat to the cryptographic infrastructure underlying blockchain networks. Solana, a high performance blockchain processing thousands of transactions per second, relies on the Ed25519 digital signature algorithm, which is vulnerable to Shor algorithm. This work investigates the technical feasibility of transitioning Solana to post-quantum cryptography through two complementary approaches: (1) the implementation of a hybrid program combining Ed25519 authentication with Falcon 512 verification using a two layer architecture (off-chain for PQC key management and on-chain for signature verification), and (2) a proposal for modifying the Solana runtime through a native syscall for post-quantum signature verification.*

## 1. Introdução

O advento de computadores quânticos criptograficamente relevantes (CRQCS) representa uma ameaça existencial à infraestrutura criptográfica que sustenta a economia digital. O algoritmo de Shor[1], demonstrado em 1994, provou que computadores quânticos suficientemente poderosos podem resolver o problema do logaritmo discreto em curva elipticas e da fatoração de número inteiros em tempo polinomial, comprometendo algoritmos amplamente utilizados como RSA, ECDSA e suas variantes [2]. Esta vulnerabilidade torna-se particularmente crítica diante de estratégias do tipo ”Harvest Now, Decrypt Later”, nas quais os atacantes capturam dados criptográficos atuais para decifrá-los posteriormente, quando os CRQCS estiverem disponíveis [3]. A urgência dessa transição é aumentada pelo fato de que sistemas de alta criticidade, como blockchains, dependem fundamentalente da segurança de longo prazo dessas primitivas criptográficas devido a natureza imutável de suas ledgers [4].

A rede Solana fundamenta sua segurança no algoritmo de assinatura digital Ed25519 [5]. Como variante do EdDSA baseada em curvas elípticas, o Ed25519 oferece desempenho e tamanhos compactos de chave (32 bytes) e de assinatura (64), características essenciais para uma rede de alta performance que necessita processar milhares de transações por segundo [6]. Entretanto, essa primitiva criptográfica é vulnerável ao algoritmo de Shor, um computador quântico relevante seria suficiente para comprometer a segurança do Ed25519[2]. Diferentemente de sistemas tradicionais onde vulnerabilidades podem ser mitigadas retroativamente, a natureza imutável e publicamente auditável das

blockchains torna toda transação historicamente registrada um alvo perpétuo para ataques quânticos futuros [4].

Diante dessas ameaças, o National Institute of Standards and Technology (NIST) formalizou uma resposta, publicando em 2004, os primeiros padrões de Criptografia Pós-Quântica: FIPS 203 (ML-KEM para encapsulamento de chaves), FIPS 204(ML-DSA para assinaturas digitais) e FIPS 205 (SLH-DSA baseado no SPHINCS+)[8]. De forma crucial, o NIST recomenda explicitamente uma abordagem de criptografia híbrida durante o período de transição, na qual sistemas combinas algoritmos clássicos com algortimos pós-quânticos.[8]. Tal abordagem, oferece segurança contra ameaças tanto clássicas quanto quânticas, mantendo a compatibilidade retroativa e mitigando riscos de vulnerabilidades ainda não descobertas nos novos algoritmos pós-quânticos.

A implementação da arquitetura híbrida em redes blockchain de alta performance apresenta desafios excepcionais. Os algoritmo pós-quânticos padronizados pelo NIST possuem estruturas de dados maiores que seus predecessores clássicos. Enquanto que o Ed25519 opera com chaves públicas de 32 bytes e assinaturas de 64 bytes [9], os algoritmos PQC como o Falcon-512 opera com chave públicas de 897 bytes e assinaturas de aproximadamente 666 bytes [10]. Essas diferenças impactam diretamento o modelo econômico e operacional de blockchains de alta performance, como a solana, que possui o tamanho máximo de 1232 bytes para um transação [24].

Dessa forma, este trabalho investiga a viabilidade técnica da transição híbrida da rede Solana para criptografia pós-quântica através de abordagens complementares: (1) a implementação de uma programa híbrido que combina autenticação Ed25519 com verificação Falcon em uma arquitetura composta por duas camadas(off-chain e on-chain) e (2) a proposta de modificações do runtime da Solana através de adição de um syscall nativo para verificação de assinatura pós-quânticas. A analise foca especificamente no algoritmo Falcon-512, selecionado por apresentar o menor tamanho de assinatura entre os finalistas do NIST para esquemas baseados em reticulados [11], característica crítica para operação dentro dos limites de transaçõds da Solana.

### 1.1. Objetivos

O objetivo geral deste trabalho é analisar a viabilidade técnica da transição da rede blockchain de alta performance Solana de sua criptografia baseada em curvas elípticas 25519 para algoritmos de criptografia pós-quântica, propondo e implementando soluções que permitam essa migração dentro das restrições operacionais da rede.

Para atingir este objetivo, foram definidos os seguintes objetivos específicos:

- Analisar a arquitetura criptográfica atual da rede Solana, identificando os componentes vulneráveis a ataques quânticos e as restrições técnicas impostas pelo modelo de transações e computação da rede.
- Implementar um programa que combine o esquema de assinatura clássico Ed25519 com o algoritmo pós-quântico falcon-512, utilizando uma arquitetura de duas camadas(off-chain para gestao das chaves PQC e on-chain para verificação das assinaturas).
- Avaliar as limitações da Solana Virtual Machine (SVM) para execução de operações criptográficas baseadas em reticulados, identificando as incompatibili-

dades técnicas (suporte a ponto flutuante, dependências de sistema) que inviabilizam a verificação PQC em programas on-chain convencionais.
- Propor uma modificação no runtime da Solana através de um syscall nativo (`sol_falcon512_verify`) que permita a verificação eficiente de assinaturas falcon-512 dentro dos limites computacionais da rede.

## 2. Fundamentação Teórica

Essa seção apresenta os conceitos fundamentais da arquitetura da rede Solana, com ênfase nos componentes relevantes para o processo de transição criptográfica.

### 2.1. Arquitetura da Rede Solana

A Solana é uma blockchain de alta performance projetada para suportar aplicaçÕes descentralizadas. Diferentemente de outras blockchain tradicionais, a Solana introduz uma série de inovações na arquitetura que permitem um throughput superior a 65000 transações por segundo [6].

#### 2.1.1. Proof of History (PoH)

O proof of History constitui a principal inovação da Solana em comparação com as outras redes. Trata-se de uma função de atraso verificável (VDF) [7] que cria um registro histórico criptográfico provando que um eveneto ocorreu em um momento específico no tempo [6]. O PoH é composto por uma cadeia sequencial de hashes SHA-256, em que cada hash incorpora o hash anterior e um contador incremental:

$$h_n = SHA256(h_{n-1}||contador_n||dados_n) \tag{1}$$

Esse comportamento da estrutura permite que os validadores ordenem as transações existentes sem a necessidade de uma comunicação síncrona entre si, reduzindo drasticamente a latência do consenso, otimizando o protocolo Tower BFT.

#### 2.1.2. Tower BFT

Tower BFT consiste na implementação do consenso Byzantine Fault Tolerant [18] da Solana, otimizado para aproveitar o registro criptográfico proveniente do Proof of History (PoH) [19]. Os validadores emitem votos assinados utilizando o Ed25519 sobre o estado da rede, os quais serão registrados no histórico do PoH. Assim, o protocolo garante finalidade após um número determinado de confirmaçoes, com cada voto representando o compromisso do validador. A segurança do Tower BFT depende diretamente da integridade das assinatura digitais dos validadores. Com o avanço da computação quântica, um atacante capaz de forjar assinaturas Ed25519 poderia comprometer o consenso da rede, identificando a necessidade de migração para primitivas resistentes a ataque quânticos.

### 2.1.3. Solana Virtual Machine (SVM)

Assim como as outras blockchain como Ethereum e as suas Layers2, a Solana, também, possui um ambiente de execução para os programas on-chain, entretanto o que difere essa virtual machine das outras redes é que a SVM(Solana Virtual Machine) baseia-se no Berkeley Packet Filter [20]. Os programas são compilados para bytecode BPF a partir de linguagens como Rust ou C, e executados em um ambiente isolado. Com o objetivo de garantir uma previsibilidade e um alto troughput, a SVM impõe limites de Compute Units (CUs) por transação durante a geração de blocos. O limite padronizado é de 200000 CUs por instrução, sendo o máximo 14000000 CUs por transação [21]. Para que seja possível realizar operações mais complexas como verificação de assinaturas Ed25519, foram implementadas syscalls com custo fixo de CUs, otimizadas para execução fora do ambiente BPF.

### 2.1.4. Syscall Nativos

Os syscalls são funções implementadas diretamente no runtime da Solana, executadas fora do ambiente BPF com acesso a recursos do sistema [23]. Exemplos incluem `sol_log` para logging, `sol_sha256` para hashing, e `sol_ed25519_verify` para verificação de assinaturas. Os syscalls consomem um número fixo de CUs independentemente dos dados de entrada, oferecendo previsibilidade de custo. A existência desta interface de syscalls constitui o mecanismo proposto neste trabalho para integração de verificação PQC ao runtime da Solana, contornando as limitações do ambiente BPF.

## 3. Materiais e Métodos

O presente estudo adota uma abordagem experimental para investigar a possibilidade da transição da primitiva criptográfica na rede Solana. A metodologia foi estruturada em duas fases que se complementam. (1) Implementação de um programa hibrido como prova de conceito em ambiente de desenvolvimento(devnet) e (2) proposta de modificação do runtime da Solana para suporte nativo a verificação PQC. Em ambas as abordagens, utilizou-se o algoritmo falcon 512 como primitiva pós-quântica.

### 3.1. Seleção do Algoritmo Pós-Quântico

Entre os algoritmos padronizados pelo NIST para assinaturas digitais, o Falcon512 foi selecionado para este estudo com base em critérios específicos de adequaç˜ão à arquitetura da Solana:

- **Tamanho de Assinatura:** O Falcon512 possui assinaturas de aproximadamente 666 bytes[10], o menor entre os algoritmos baseados em reticulados. Em comparação, o Dilithium gera assinaturas de 2420 bytes a 4627 bytes a depender do nível de segurança [12].
- **Compatibilidade com limites da transação:** O tamanho máximo de uma transação na solana é 1232 bytes [24]. Uma transação de verificação utilizando o Falcon512 requer aproximadamente 899 bytes (1 byte da instrução + 32 bytes da hash + 666 bytes da assinatura + 200 bytes de overhead), disponibilizando uma margem de 333 bytes.
- **Nível de Segurança:** O Falcon512 oferece segurança equivalente ao nível 1 do NIST, comparável à segurança de 128 bits contra atacantes clássicos e quânticos

### 3.2. Ambiente de Desenvolvimento

O desenvolvimento foi realizado utilizando as seguintes tecnologias e ferramentas:

- **Linguagem de programação:** Rust, linguagem nativa do ecossistema Solana.
- **Framework para contratos inteligentes:** Anchor, framework que simplifica o desenvolvimento de programas Solana.
- **Biblioteca criptográfica PQC:** pqcrypto-falcon, implementação em Rust dos algoritmos Falcon conforme as especificações do NIST.
- **Ambiente de Teste:** Solana Devnet para testes de integração de programas on-chain.

### 3.3. Implementação da Carteira Híbrida

A arquitetura foi projetada para contornar as limitações atuais da Solana Virtual Machine (SVM) em relação a operações criptográficas complexas. O modelo aborda uma separação de responsabilidaades entre componentes off-chain e on-chain.

#### 3.3.1. Componente Off-Chain: Gestão das Chaves PQC

A camada do cliente opera fora da blockchain e é responsável pelas operações criptográficas que não demandam da rede:

- **Geração de Chaves Hívrida:** O cliente gera localmente um par de chaves Falcon512 em conjunto com o par de chaves Ed25519 nativo das carteiras da Solana. A chave privada falcon permanece estritamente no dispositivo do usuário, enquanto a chave pública é enviada ao um programa.
- **Assinatura de Transações:** Paara autorizar uma transfer6encia, o cliente produz duas assinaturas distintas: uma assinatura clássica Ed25519, responsável por pagar as taxas da rede, e uma assinatura falcon512 que será enviada no payload da instrução, juntamente, com a mensagem.

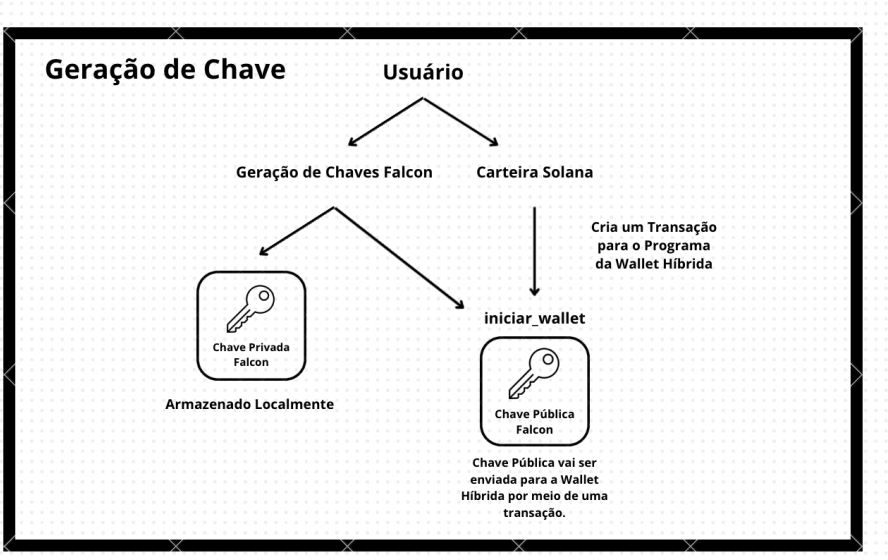


**Figure 1. Geração da Chave PQC**

A geração do par de chaves do Falcon512 é realizada conforme o pseudocódigo:

```
fn generate_falcon_keypair() -> (Vec<u8>, Vec<u8>) {
    let (pk, sk) = falcon512::keypair();
    (pk.as_bytes().to_vec(), sk.as_bytes().to_vec())
}
```

### 3.3.2. Componente On-Chain: Programa Verificador

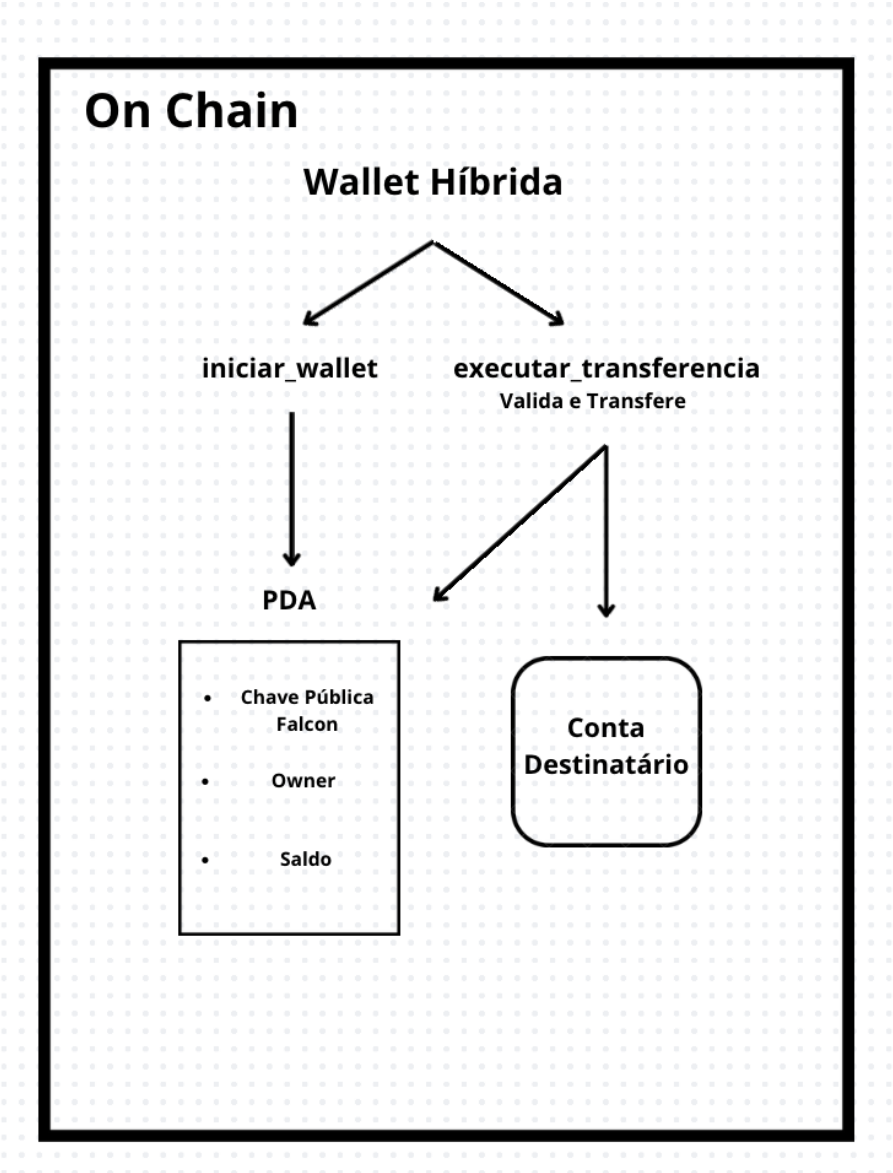


**Figure 2. Arquitetura da Wallet Híbrida**

O componente on-chain é um contrato inteligente desenvolvido com o framework Anchor, que atua como encapsulador seguro e verificador híbrido. Suas principais funções são:

**Vinculação de Identidade (PDA):** Na inicialização, o programa cria uma conta do tipo Program Derived Address(PDA)[14] que armazena de forma imutável a chave pública Falcon512 associada ao endereço Ed25519 do usuário. Isso estabelece o vínculo criptografico entre a identidade clássica e a pós-quântica da carteira.

```
pub fn iniciar_wallet(ctx: Context<Initialize>, pqc_pubkey: Vec<u8>) ->
     Result<()> {
   require!(pqc_pubkey.len() <= 2048, HybridError::ChavePQCMuitoGrande)
      ;

   let wallet = &mut ctx.accounts.wallet;
   wallet.owner = ctx.accounts.owner.key();
   wallet.pqc_pubkey = pqc_pubkey;

   Ok(())
}
```

**Estrutura da Conta PDA:** A conta híbrida armazena os seguintes campos:

```
#[account]
pub struct WalletHibrida {
   pub owner: Pubkey, // 32 bytes -> Endereco Ed25519
   pub bumpL u8, // 1 byte - Bump Seed do PDA
   pub pqc_pubkey: Vec<u8>, // Chave publica do Falcon
}
```

### 3.3.3. Limitação Identificada: Incompatibilidade com BPF

O Berkeley Packet Filter(BPF), originalmente desenvolvido para filtragem de pacotes de rede no kernel Linux [15], foi adaptado pela Solana como ambiente de execução para programas on-chain [20]. A variante utilizada pela Solana, denominada Solana BPF(SBF) é uma máquina virtual baseada em eBPF que executa bytecode compilado a partir de código Rust [23]. Contudo, o ambiente BPF impõe restrições significativas em comparação com a execução: não suporta operações de ponto flutuamente, loops, variavéis globais e passagem de estruturas como argumentos de função [16]

Mesmo diante dessa limitação, o programa da carteira híbrida foi desenvolvido e implantado com sucesso na Devnet ( Ambiente de desenvolvido da Solana). O deploy do bytecode BPF ocorreu sem erros, e a instrução `iniciar_wallet`, que armazena a chave pública Falcon no PDA, executou corretamente. O problema manifestou-se ao tentar implementar a verificação real do Falcon512 on-chain. A implementação da função de verificação seria:

```
use pqcrypto_falcon::falcon512;
use pqcrypto_traits::sign::{DetachedSignature, PublicKey};

pub fn verificar_assinatura_falcon(
   ctx: Context<ExecutarPqc>,
   mensagem: Vec<u8>,
   assinatura_pqc: Vec<u8>
) -> Result<()> {
```

```
    let wallet = &ctx.accounts.wallet;

    // Recuperar chave publica do PDA
    let pubkey = falcon512::PublicKey::from_bytes(&wallet.pqc_pubkey)
        .map_err(|_| Erros::ChavePQCInvalida)?;

    // Deserializar assinatura
    let signature = falcon512::DetachedSignature::from_bytes(&
        assinatura_pqc)
        .map_err(|_| Erros::AssinaturaPQCInvalida)?;

    // Verificar assinatura - FALHA: BPF nao suporta ponto flutuante
    falcon512::verify_detached_signature(&signature, &mensagem, &pubkey)
        .map_err(|_| Erros::VerificacaoFalhou)?;

    Ok(())
}
```

Ao tentar compilar este código, o compilador retorna erros relacionados à incompatibilidade da biblioteca `pqcrypto-falcon` com o ambiente BPF. A biblioteca utiliza internamente operações de ponto flutuante no Algoritmo Falcon [17], operações estas que não são suportadas pelo conjunto de instruções do BPF[16].

Esta incompatibilidade fundamental motivou a segunda abordagem deste trabalho: a implementação de um syscall nativo que executa fora do ambiente BPF, diretamente no runtime do validador.

### 3.4. Proposta de Syscall Nativo

Para viabilizar a verificação PQC dentro dos limites operacionais da Solana, propõe-se a implementação de um syscall nativo `sol_falcon512_verify` no runtime da rede.

#### 3.4.1. Arquitetura do Syscall

As syscalls são funções nativas executadas diretamente pelo runtime da Solana, fora do ambiente de execução da SVM [23], oferecendo duas vantagens principais:

1. **Performance:** O Código de verificação executa como código nativo compilado, eliminando as limitações impostas pela validação do BPF (Berkerley Packet Filter).
2. **Custo Fixo:** O consumo de CUs (Compute Units) é definido estaticamente, independente da complexidade da operação, similar ao que ocorre na verificação nativa do Ed25519 (Ed25519SigVerify111111111111111111111111111).

#### 3.4.2. Implementação do Syscall

Para implementação de um novo syscall foi realizado um clone do repositório agave `https://github.com/anza-xyz/agave`. Com a clonagem concluída, o syscall foi implementado utilizando a macro `declare_builtin_function` do runtime da Solana. A função recebe como parâmetros os endereços de memória da mensagem, assinatura e chave pública:

```
pub const FALCON512_PUBKEY_LEN: usize = 897;
pub const FALCON512_SIGNATURE_LEN: usize = 666;
pub const FALCON512_VERIFY_COST: u64 = 25_000;

declare_builtin_function!(
   SyscallFalcon512Verify,
   fn rust(
      invoke_context: &mut InvokeContext,
      message_addr: u64,
      message_len: u64,
      signature_addr: u64,
      pubkey_addr: u64,
      _arg5: u64,
      memory_mapping: &mut MemoryMapping,
   ) -> Result<u64, Error> {
      consume_compute_meter(invoke_context, FALCON512_VERIFY_COST)?;

      // Mapeamento de memoria VM -> processo
      let message = translate_slice(memory_mapping, message_addr,
         message_len, check_aligned)?;

      let signature_bytes = translate_slice(memory_mapping,
         signature_addr, FALCON512_SIGNATURE_LEN as u64, check_aligned
         )?;

      let pubkey_bytes = translate_slice(memory_mapping, pubkey_addr,
         FALCON512_PUBKEY_LEN as u64, check_aligned)?;

      // Validacao e verificacao
      let pubkey = falcon512::PublicKey::from_bytes(pubkey_bytes)?;

      let signature = falcon512::DetachedSignature::from_bytes(
         signature_bytes)?;

      match falcon512::verify_detached_signature(&signature, message, &
         pubkey) {
         Ok(()) => Ok(0), // Sucesso
         Err(_) => Ok(3), // Verificacao falhou
      }
   }
);
```

A função é dividida em três partes principais: Separação das informações, Verificação da Chave pública e Assinatura e Validação da mensagem assinada.

**Etapa 1: Separação das Informações** A primeira etapa consiste na extração dos dados da memória da máquina virtual. A função `translate_slice` realiza o mapeamento entre os endereços virtuais da SVM e os endereços reais de memória do processo validador. Esta operação é necessária porque o programa é executado em um ambiente isolado com seu próprio espaço de endereçamento, necessitando desse mapeamento.

```
let message = translate_slice(
   memory_mapping, message_addr, message_len, check_aligned,
```

```
)?;

let signature_bytes = translate_slice(
    memory_mapping, signature_addr,
    FALCON512_SIGNATURE_LEN as u64, check_aligned,
)?;

let pubkey_bytes = translate_slice(
    memory_mapping, pubkey_addr,
    FALCON512_PUBKEY_LEN as u64, check_aligned,
)?;
```

O parâmetro `check_aligned` garante que os dados estejam alinhados corretamente na memória, prevenindo erros de acesso. Ao final desta parte, três slices de bytes estão disponíveis: a mensagem, a assinatura de 666 bytes e a chave pública de 897 bytes.

**Etapa 2: Validação da Chave Pública e da Assinatura** Com essas variavéis extraídas, a segunda etapa utiliza-se da biblioteca pqcrypto para validar as estruturas da chave pública e da assinatura. A função `PublicKey::from_bytes` verifica se os 897 bytes correspondem a uma chave pública Falcon-512 válida, checando a estrutura interna e os parâmetros do reticulado.

```
let pubkey = match falcon512::PublicKey::from_bytes(pubkey_bytes) {
    Ok(pk) => pk,
    Err(_) => return Ok(1),
};

let signature = match falcon512::DetachedSignature::from_bytes(
    signature_bytes) {
    Ok(sig) => sig,
    Err(_) => return Ok(2),
};
```

**Etapa 3: Validação da Mensagem Assinada** A etapa final é crucial, sendo responsável pela verificação criptográfica da mensagem. A função `verify_detached_signature` recebe a assinatura, a mensagem original e a chave pública recuperada, executando as operações matemáticas sobre reticulados que comprovam a autenticidade da assinatura.

```
match falcon512::verify_detached_signature(&signature, message, &pubkey
    ) {
    Ok(()) => Ok(0),
    Err(_) => Ok(3),
}
```

Se a assinatura foi gerada pela chave privada correspondente à chave pública fornecida, e a mensagem não foi alterada desde a assinatura, a função retorna 0 (sucesso). Caso contrário, retorna 3 indicando que a verificação criptográfica falhou, o que pode ocorrer se a mensagem foi modificada ou se a assinatura foi gerada por uma chave privada diferente.

### 3.4.3. Códigos de Retorno

O syscall utiliza códigos de retorno padronizados indicando o resultado da operação:

| Código | Descrição |
|---|---|
| 0 | Assinatura válida (sucesso) |
| 1 | Chave pública inválida |
| 2 | Formato de assinatura inválido |
| 3 | Verificação criptográfica falhou |
| 4 | Mensagem inválida |
| 5 | Budget de computação excedido |

**Table 1. Códigos de retorno do syscall `sol_falcon512_verify`**

### 3.4.4. Registro via Feature Flag

Com o syscall implementado, é necessário ativá-lo, para isso, foi implementado uma feature flag no módulo `svm-feature-set`:

```
pub mod enable_falcon512_syscall {
    solana_pubkey::declare_id!("
        Falcon512Verify111111111111111111111111111");
}

register_feature_gated_function!(
    result,
    enable_falcon512_syscall,
    "sol_falcon512_verify",
    SyscallFalcon512Verify::vm,
)?;
```

## 4. Resultados

Nesta seção apresentamos os resultados obtidos nas duas abordagens investigadas e suas implicações para a transição da rede Solana para uma criptografia pós-quântica.

### 4.1. Resultados da Carteira Híbrida

O programa da carteira híbrida foi deployado com sucesso na Devnet, demonstrando a capacidade de armazenar a chave pública do Falcon512 on-chain através de uma Program Derived Address (PDA). A partir dos testes realizados, foi constatado que as estruturas de dados do algoritmo Falcon512 se adequam aos limites impostos pela rede: a chave pública de 897 bytes e a assinatura de 666 bytes permitem a construção de transações dentro do limite máximo de 1232 bytes estabelecido pela rede.

| Transação | Tamanho | Limite | Status |
|---|---|---|---|
| TX Initialize (PDA + PK Falcon) | 1.098 bytes | 1.232 bytes | Sucesso |
| TX Verify (hash + assinatura) | 899 bytes | 1.232 bytes | Sucesso* |

**Table 2. Tamanhos das transações híbridas na Devnet**

Contudo, a verificação criptográfica real do Falcon512 não pôde ser implementada on-chain. Ao tentar incorporar a biblioteca `pqcrypto-falcon` para realizar a validação matemática da assinatura, o processo de compilação falhou devido às restrições do ambiente de execução da própria Solana Virtual Machine. Especificamente, em relação as operações de ponto flutuantes, as quais não sao suportadas pelo conjunto de instruções do BPF. Esta incompatibilidade impossibilitou a execução criptográfica do Falcon.

### 4.2. Resultados da Implementação de um novo Syscall

A implementação do syscall `sol_falcon512_verify` foi desenvolvida como prova de conceito, com o objetivo de demonstrar a viabilidade técnica da abordagem.

| Especificação | Valor |
|---|---|
| Nome do syscall | `sol_falcon512_verify` |
| Tamanho da chave pública | 897 bytes |
| Tamanho da Assinatura | 666 bytes |
| Feature Flag | `enable_falcon512_syscall` |

**Table 3. Especificações do syscall `sol_falcon512_verify`**

Tendo em vista que as funções nativas não sofrem das limitações impostas pelo BPF, a criação de uma nova chamada nativa que utiliza-se de bibliotecas PQC coerentes a implementação sinalizada pelo NIST, tornaria possível a verificação PQC on-chain, permitindo que uma transição hibrida poderia ser inicializada.

### 4.3. Discussão

#### 4.3.1. Viabilidade da transição

Os resultados demonstram que a transição da Solana para criptografia pós-quântica é tecnicamente viável, porém requer algumas modificações no runtime da rede. A abordagem de syscall nativo apresenta-se como a solução maais promissora oferecendo compatibilidade de tamanho, tendo em vista que as transações falcon512 cabem dentro do limite de 1232 bytes por transação e uma segurança hibrida, já que a arquitetura atual seria mantida( até que a transição PQC fosse totalmente implementada.

#### 4.3.2. Desafios Identificados

A implementação identificou alguns desafios significativos.

1. **Modificação do Runtime:** Qulquer alteração no código-fonte do validador Solana necessita de uma aprovação da comunidade de desenvolvedores e coordenação para ativação da feature flag.
2. **Incompatibilidade do BPF:** A ausência de suporte a ponto flutuante e outras estruturas no BPF é uma limitação fundamental que afeta não apenas o Falcon, mas outros algoritmos PQC que dependem de operações aritméticas complexas.
3. **Custo de Armazenamento:** O aumento de 28x no tamanho de chaves públicas (32 bytes paara 897 bytes) causa um impacto direto nos custos de rent para contas que armazenam identidades PQC. Além de afetar a questão do througput da rede, fazendo com que ela processo menos transações por segundo.

## 5. Conclusão

Este trabalhou investigou a viabilidade técnica da transição da rede Solana para criptografia pós-quântica, com foco no algoritmo Falcon 512, através de duas abordagens complementares: A implementação de uma carteira híbrida e a proposta de um syscall nativo. Foram identificado os principais desafios e soluções levando em consideração ambas as abordagens em um processo de migração criptográfica.

A principal contribuição deste trabalho é a demonstração prática de que redes blockchain de alta performance podem ser adaptadas para criptografia pós-quântica sem comprometer suas características de throughput e custo. A implementação do syscall proposto requer um trabalho em conjunto da comunidade de desenvolvedores da Solana, representando uma oportunidade para que a rede se posicione na vanguarda da segurança criptográfica.

## References


[1] Shor, P. W. (1994). "Algorithms for quantum computation: discrete logarithms and factoring". In: *Proceedings 35th Annual Symposium on Foundations of Computer Science*, p. 124-134.

[2] Roetteler, M., Naehrig, M., Svore, K. M., Lauter, K. (2017). Quantum resource estimates for computing elliptic curve discrete logarithms. In *International Conference on the Theory and Application of Cryptology and Information Security* (pp. 241-270). Springer.

[3] Mosca, M. (2018). "Cybersecurity in an Era with Quantum Computers: Will We Be Ready?" *IEEE Security & Privacy*, 16(5), 38-41.

[4] Aggarwal, D., Brennen, G. K., Lee, T., Santha, M., Tomamichel, M. (2018). Quantum attacks on Bitcoin, and how to protect against them. *Ledger*, 3.

[5] Bernstein, D. J., Duif, N., Lange, T., Schwabe, P., Yang, B. Y. (2012). High-speed high-security signatures. *Journal of Cryptographic Engineering*, 2(2), 77-89.

[6] Yakovenko, A. (2018). "Solana: A new architecture for a high performance blockchain v0.8.13". `https://solana.com/solana-whitepaper.pdf`

[7] D. Boneh, J. Bonneau, B. Bünz, and B. Fisch, *Verifiable Delay Functions*, in Proceedings of CRYPTO 2018, pp. 757–788. `https://eprint.iacr.org/2018/601`

[8] National Institute of Standards and Technology. (2024). "Post-Quantum Cryptography: FIPS Approved". `https://csrc.nist.gov/news/2024/postquantum-cryptography-fips-approved`

[9] S. Josefsson and I. Liusvaara, Edwards-Curve Digital Signature Algorithm (EdDSA), RFC 8032, IETF, Jan. 2017. `https://www.rfc-editor.org/rfc/rfc8032.html`

[10] Falcon - Post-Quantum Signatures `https://falcon-sign.info/`

[11] M.Abbasi, et al. Cryptography Across Heterogeneous Computing Environments, Cryptography, vol. 9, no. 2, Art. 32, 2025. `https://www.mdpi.com/2410-387X/9/2/32`

[12] Lyubashevsky, V., Ducas, L., Kiltz, E., Lepoint, T., Schwabe, P., et al. (2018). "CRYSTALS-Dilithium: A Lattice-Based Digital Signature Scheme". *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2018(1), 238-268.

[13] QuReady, NIST PQC Standards Guide. Overview of the NIST Post-Quantum Cryptography standardization process and security level equivalences, 2025. Available at: https://quready.com/resources/nist-pqc-standards-guide

[14] Solana Documentation, *Program-Derived Address (PDA)*, Official Solana documentation describing program-derived addresses and their properties. https://solana.com/docs/core/pda

[15] S. McCanne and V. Jacobson, *The BSD Packet Filter: A New Architecture for User-level Packet Capture*, in Proceedings of the USENIX Winter 1993 Conference, San Diego, CA, USA, Jan. 1993, USENIX Association.

[16] Linux manual page for bpf(2), *bpf(2) — Linux manual page*, Manual page for the BPF syscall describing the restricted C language and its omissions, including lack of floating-point support. Available at: https://man7.org/linux/man-pages/man2/bpf.2.html

[17] T. Pornin, *New Efficient, Constant-Time Implementations of Falcon*, Technical report discussing Falcon implementations, including the use of floating-point arithmetic for Gaussian sampling and alternatives for platforms without an FPU. https://falcon-sign.info/falcon-impl-20190918.pdf

[18] M. Castro and B. Liskov, *Practical Byzantine Fault Tolerance*, in Proceedings of the Third Symposium on Operating Systems Design and Implementation (OSDI), USENIX Association. https://www.usenix.org/legacy/publications/library/proceedings/osdi99/full_papers/castro/castro_html/castro.html

[19] Agave Tower BFT. *Tower BFT — Agave* Disponível em: https://docs.anza.xyz/implemented-proposals/tower-bft

[20] Solana Foundation. (2024). Programs https://solana.com/pt/docs/core/programs

[21] Solana CUs. *Solana CUs* Disponível em: https://solana.com/pt/docs/core/fees/#limite-de-unidade-de-computa%C3%A7%C3%A3o

[22] Falcon Reticulados. *Falcon Reticulados* Disponível em: https://pqc-cma.gitlab.io/cma/docs/schemes/nist/all/falcon

[23] SolanaRuntime. *SolanaRuntime* Disponível em: docs.solanalabs.com/ar/validator/runtime

[24] SolanaTransacao. *SolanaTransacao* Disponível em: https://solana.com/pt/docs/core/transactions#instruction-format