2024

Yellow Paper ICO SLS

Version 1.1

This document provides a comprehensive technical overview of the SLS token, its underlying technology, protocols, and algorithms. It serves as a technical guide for developers and specialists, enabling them to understand the inner workings of the SLS ecosystem.

SlonPlus Company

19.07.2024

1. Introduction

1.1 Purpose

The SLS token is designed to facilitate transactions within the SlonPlus ecosystem, providing a secure and efficient means of exchanging value. This Yellow Paper outlines the technical architecture and operational mechanisms of the SLS token.

1.2 Scope

This document covers mathematical models, encryption algorithms, consensus protocols, and other technical specifications necessary for the SLS token.

2. SLS Token Technical Specifications

2.1 Overview

The SLS token is a digital asset built on the TRON blockchain, compliant with the TRC-20 standard. This document provides a detailed technical specification of the SLS token, including its architecture, functionalities, and security measures.

2.2 General Information

- **Token Name:** Shares SlonPlus
- **Token Symbol:** SLS+
- **Decimals:** 18
- **Total Supply:** 560,000,000 SLS+
- **Blockchain Platform:** TRON
- **Token Standard:** TRC-20

2.3 Token Contract Details

2.3.1 Contract Address

- **Contract Address:** To be provided upon deployment

2.3.2 Compiler Version

- **Solidity Version:** ^0.8.20

2.3.3 Key Variables

- **name:** string - The name of the token.
- **symbol:** string - The symbol of the token.
- **decimals:** uint8 - The number of decimal places the token uses.
- **_totalSupply:** uint - The total number of tokens in existence.
- **TokenPrice:** uint256 - The price of one token in wei (18 decimal places).

- **owner:** address - The address of the contract owner.
- **newOwner:** address - The address of the new owner when ownership is being transferred.
- **paused:** bool - The state of the contract, indicating if it is paused.

2.3.4 Mappings

- **balances:** mapping(address => uint) - Maps addresses to their respective token balances.
- **allowed:** mapping(address => mapping(address => uint)) - Maps addresses to their allowances for spending tokens on behalf of others.

2.4. Core Functionalities

2.4.1 Total Supply

- **Function:** totalSupply()
- **Description:** Returns the total number of tokens in circulation, excluding those held by the zero address.
- **Return Type:** uint

2.4.2 Balance Of

- **Function:** balanceOf(address tokenOwner)
- **Description:** Returns the token balance of a specified address.
- **Parameters:**
    - tokenOwner: address - The address whose balance is to be retrieved.
- **Return Type:** uint

2.4.3 Transfer

- **Function:** transfer(address to, uint tokens)
- **Description:** Transfers tokens from the sender's address to the recipient's address.
- **Parameters:**
    - to: address - The recipient's address.
    - tokens: uint - The number of tokens to transfer.
- **Return Type:** bool

2.4.4 Approve

- **Function:** approve(address spender, uint tokens)
- **Description:** Approves the spender to transfer tokens from the owner's address.
- **Parameters:**
    - spender: address - The address authorized to spend the tokens.
    - tokens: uint - The number of tokens to approve.
- **Return Type:** bool

2.4.5 Transfer From

- **Function:** transferFrom(address from, address to, uint tokens)
- **Description:** Transfers tokens on behalf of the owner.
- **Parameters:**
    - from: address - The address from which the tokens are to be transferred.

    - to: address - The recipient's address.
    - tokens: uint - The number of tokens to transfer.
- **Return Type:** bool

2.4.6 Allowance

- **Function:** allowance(address tokenOwner, address spender)
- **Description:** Returns the remaining number of tokens approved for transfer.
- **Parameters:**
    - tokenOwner: address - The owner of the tokens.
    - spender: address - The address authorized to spend the tokens.
- **Return Type:** uint

2.4.7 Buy Tokens

- **Function:** buyTokens()
- **Description:** Allows users to buy tokens by sending Ether to the contract.
- **Return Type:** void

2.4.8 Sell Tokens

- **Function:** sellTokens(uint256 amount)
- **Description:** Allows users to sell tokens and receive Ether in exchange.
- **Parameters:**
    - amount: uint256 - The number of tokens to sell.
- **Return Type:** void

2.4.9 Pause

- **Function:** pause()
- **Description:** Pauses the contract, disabling buy and sell functions.
- **Return Type:** void

2.4.10 Unpause

- **Function:** unpause()
- **Description:** Unpauses the contract, enabling buy and sell functions.
- **Return Type:** void

2.4.11 Transfer Ownership

- **Function:** transferOwnership(address _newOwner)
- **Description:** Initiates the transfer of ownership to a new owner.
- **Parameters:**
    - _newOwner: address - The address of the new owner.
- **Return Type:** void

2.4.12 Accept Ownership

- **Function:** acceptOwnership()
- **Description:** The new owner accepts ownership of the contract.

- **Return Type:** void

2.5 Security Measures

2.5.1 Only Owner Modifier

- **Modifier:** onlyOwner
- **Description:** Ensures that certain functions can only be called by the contract owner.

2.5.2 SafeMath

- **Library:** SafeMath
- **Description:** Ensures safe arithmetic operations to prevent overflow and underflow.

2.5.3 Reentrancy Protection

- **Pattern:** Checks-effects-interactions
- **Description:** Protects critical functions from reentrancy attacks.

2.6 Events

2.6.1 Transfer Event

- **Event:** Transfer(address indexed from, address indexed to, uint tokens)
- **Description:** Emitted when tokens are transferred.

2.6.2 Approval Event

- **Event:** Approval(address indexed tokenOwner, address indexed spender, uint tokens)
- **Description:** Emitted when a token owner approves a spender.

2.7 Operational Mechanics

2.7.1 Token Price

- **Token Price:** 1 USD in wei (18 decimal places)

2.7.2 Buying Tokens

- Users send Ether to the contract to purchase tokens.
- The number of tokens purchased is calculated based on the current token price.
- Tokens are transferred from the contract's balance to the user's balance.

2.7.3 Selling Tokens

- Users can sell their tokens back to the contract.
- The Ether equivalent of the sold tokens is calculated and sent to the user.
- Tokens are transferred from the user's balance to the contract's balance.

2.7.4 Pausing and Unpausing

- The contract owner can pause the contract to disable buying and selling of tokens.
- The contract owner can unpause the contract to re-enable buying and selling of tokens.

The SLS token leverages the TRC-20 standard on the TRON blockchain to provide a secure, efficient, and user-friendly token. Its architecture includes robust security measures, operational protocols, and functionalities to ensure smooth interactions within the TRON ecosystem.

2.8 Smart Contract

The SLS token is implemented as a TRC-20 compliant smart contract on the TRON blockchain. The smart contract manages token creation, transfers, and other operations.

```
pragma solidity ^0.8.20;

interface TRC20 {
    function totalSupply() external view returns (uint);
    function balanceOf(address tokenOwner) external view returns (uint balance);
    function allowance(address tokenOwner, address spender) external view returns (uint remaining);
    function transfer(address to, uint tokens) external returns (bool success);
    function approve(address spender, uint tokens) external returns (bool success);
    function transferFrom(address from, address to, uint tokens) external returns (bool success);

    event Transfer(address indexed from, address indexed to, uint tokens);
    event Approval(address indexed tokenOwner, address indexed spender, uint tokens);
}

contract SharesSlonPlus is TRC20 {
    string public name = "Shares SlonPlus";
    string public symbol = "SLS+";
    uint8 public decimals = 18;
    uint public _totalSupply = 560000000 * 10**uint(decimals);
    uint256 public TokenPrice = 1e18; // 1 USD in wei (18 decimal places)
    address public owner;
    address public newOwner;
    bool public paused = false;

    mapping(address => uint) balances;
    mapping(address => mapping(address => uint)) allowed;

    modifier onlyOwner() {
        require(msg.sender == owner, "Only the owner can call this function");
        _;
    }

    constructor () {
        owner = msg.sender;
        balances[owner] = _totalSupply;
        emit Transfer(address(0), owner, _totalSupply);
    }

    function transferOwnership(address _newOwner) public onlyOwner {
        require(_newOwner != address(0), "New owner cannot be zero address");
        newOwner = _newOwner;
    }

    function acceptOwnership() public {
        require(msg.sender == newOwner, "Only the new owner can accept ownership");
        owner = newOwner;
        newOwner = address(0);
    }

    function totalSupply() public view override returns (uint) {
        return _totalSupply - balances[address(0)];
    }

    function balanceOf(address tokenOwner) public view override returns (uint balance) {
        return balances[tokenOwner];
    }

    function transfer(address to, uint tokens) public override returns (bool success) {
        require(to != address(0), "Cannot transfer to the zero address");
        require(balances[msg.sender] >= tokens, "Insufficient balance");
        balances[msg.sender] -= tokens;
        balances[to] += tokens;
        emit Transfer(msg.sender, to, tokens);
        return true;
    }

    function approve(address spender, uint tokens) public override returns (bool success) {
        allowed[msg.sender][spender] = tokens;
        emit Approval(msg.sender, spender, tokens);
        return true;
    }

    function transferFrom(address from, address to, uint tokens) public override returns (bool success) {
        require(to != address(0), "Cannot transfer to the zero address");
        require(balances[from] >= tokens, "Insufficient balance");
        require(allowed[from][msg.sender] >= tokens, "Insufficient allowance");
        balances[from] -= tokens;
        allowed[from][msg.sender] -= tokens;
        balances[to] += tokens;
        emit Transfer(from, to, tokens);
        return true;
    }

    function allowance(address tokenOwner, address spender) public view override returns (uint remaining) {
        return allowed[tokenOwner][spender];
    }

    function buyTokens() public payable {
        require(!paused, "Contract is paused");
        uint256 amount = msg.value / TokenPrice;
        require(amount > 0, "You need to send some TRX");
        require(amount <= balances[owner], "Insufficient balance in the contract");
        balances[owner] -= amount;
        balances[msg.sender] += amount;
        emit Transfer(owner, msg.sender, amount);
    }

    function sellTokens(uint256 amount) public {
        require(!paused, "Contract is paused");
        require(amount > 0, "You need to sell at least one token");
        require(amount <= balances[msg.sender], "Insufficient balance");
        uint256 trxAmount = amount * TokenPrice;
        balances[msg.sender] -= amount;
        balances[owner] += amount;

        // Using call instead of transfer
        (bool success, ) = msg.sender.call{value: trxAmount}("");
        require(success, "Transfer failed.");

        emit Transfer(msg.sender, owner, amount);
    }

    function pause() public onlyOwner {
        paused = true;
    }

    function unpause() public onlyOwner {
        paused = false;
    }

    // Fallback function to handle TRX sent to the contract
    receive() external payable {
        // Custom logic for receiving TRX can be added here
    }
}
```

3. Technical Architecture and Operational Mechanisms of the SLS Token

3.1 Technical Architecture

3.1.1 Blockchain Platform

The SLS token is implemented on the TRON blockchain, leveraging its high throughput, low transaction costs, and robust smart contract capabilities. The choice of TRON ensures efficient and scalable operations for the SLS token.

3.1.2 Smart Contract

The SLS token smart contract is a TRC-20 compliant contract. It manages token issuance, transfers, and other functionalities essential for the token's operation.

3.1.3 Key Components

- **Token Details:**
  - **Name:** Shares SlonPlus
  - **Symbol:** SLS+
  - **Decimals:** 18
  - **Total Supply:** 560,000,000 SLS+
- **Contract Variables:**
  - TokenPrice: The price of the token set at 1 USD in wei.
  - owner: The address of the contract owner.
  - newOwner: The address of the proposed new owner for ownership transfer.
  - paused: A boolean indicating whether the contract is paused.

3.1.4 Data Structures

- **Mappings:**
  - balances: Tracks the balance of SLS tokens for each address.
  - allowed: Tracks allowances for token transfers on behalf of other addresses.

3.1.5 Modifiers

- **onlyOwner:** Ensures that only the contract owner can call certain functions.

3.1.6 Functions

- **Core Functions:**
  - totalSupply(): Returns the total supply of the token.
  - balanceOf(address tokenOwner): Returns the balance of tokens for a given address.
  - allowance(address tokenOwner, address spender): Returns the remaining tokens that the spender is allowed to spend on behalf of the token owner.
  - transfer(address to, uint tokens): Transfers tokens to a specified address.
  - approve(address spender, uint tokens): Approves a spender to spend a specified number of tokens.
  - transferFrom(address from, address to, uint tokens): Transfers tokens from one address to another using an allowance.
- **Ownership Management Functions:**
  - transferOwnership(address _newOwner): Initiates the transfer of ownership to a new owner.
  - acceptOwnership(): Accepts ownership by the new owner.
- **Token Sale Functions:**

    - buyTokens(): Allows users to buy tokens by sending TRX to the contract.
    - sellTokens(uint256 amount): Allows users to sell tokens in exchange for TRX.
- **Contract Control Functions:**
    - pause(): Pauses contract operations.
    - unpause(): Resumes contract operations.
- **Fallback Function:**
    - receive(): Handles receiving TRX sent to the contract.

3.2 Operational Mechanisms

3.2.1 Token Distribution

The total supply of SLS tokens is pre-minted and allocated to the contract owner. The owner can distribute tokens through various means such as sales, rewards, or other mechanisms.

3.2.2 Token Pricing

The SLS token is priced at 1 USD in wei. This price mechanism ensures a stable valuation of the token and simplifies the buying and selling process for users.

3.2.3 Buying Tokens

Users can buy SLS tokens by sending TRX to the contract. The contract calculates the number of tokens to be issued based on the sent TRX amount and the token price. The user's balance is updated, and a Transfer event is emitted.

3.2.4 Selling Tokens

Users can sell their SLS tokens back to the contract in exchange for TRX. The contract calculates the amount of TRX to be sent based on the token price. The user's balance is updated, and the TRX amount is transferred using the call method to ensure security against reentrancy attacks.

3.2.5 Token Transfers

Users can transfer tokens to other addresses. The contract ensures that the sender has sufficient balance and that the recipient is not the zero address. The balances are updated, and a Transfer event is emitted.

3.2.6 Allowance and TransferFrom

Users can approve other addresses to spend tokens on their behalf using the approve function. The transferFrom function allows the spender to transfer tokens from the token owner's address to another address within the approved limit.

3.2.7 Contract Pausing

The contract owner can pause and unpause contract operations using the pause and unpause functions. When paused, critical functions such as buying, selling, and transferring tokens are disabled to protect against potential threats or vulnerabilities.

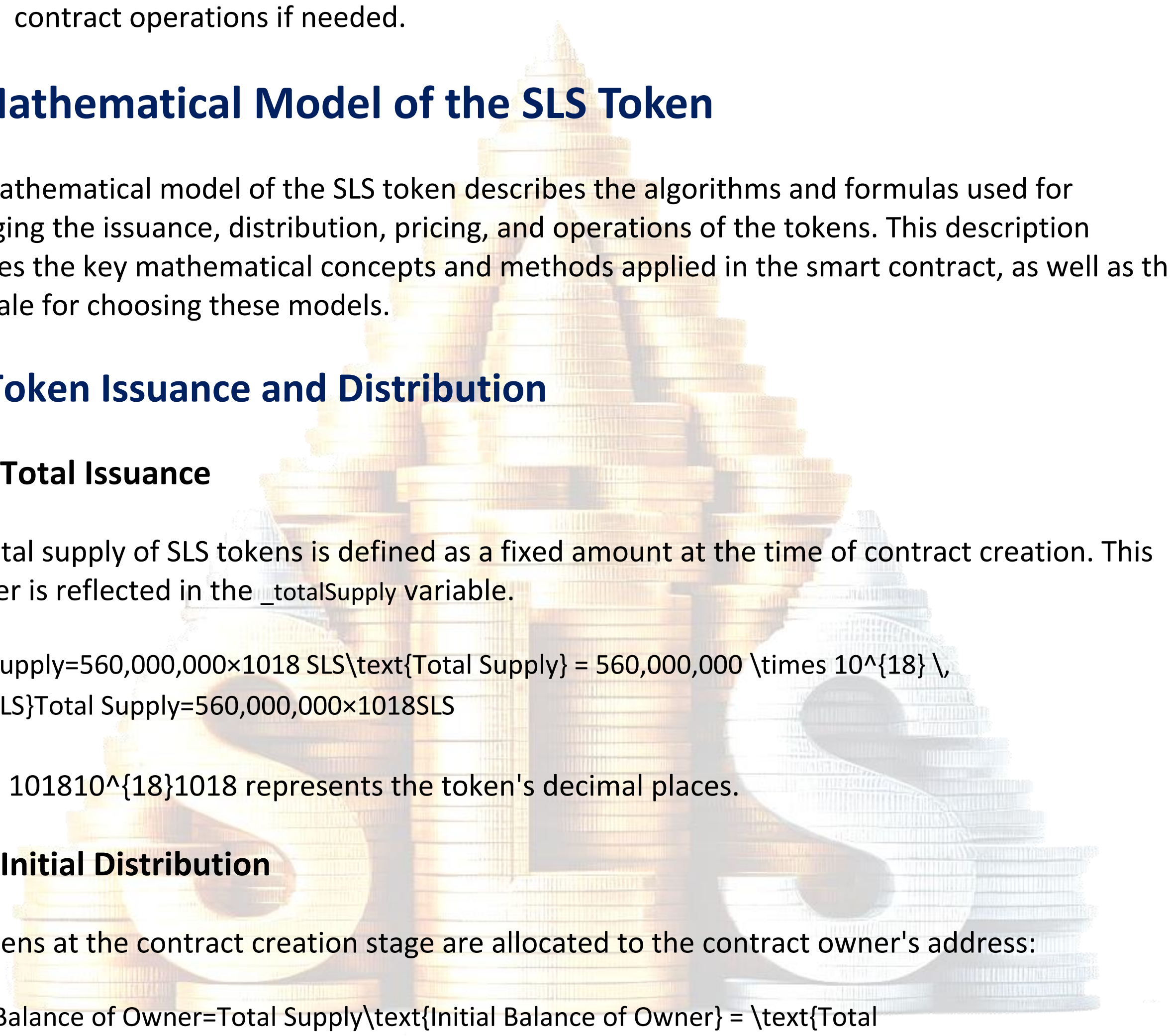
3.2.8 Ownership Transfer

The contract supports secure transfer of ownership through a two-step process. The current owner initiates the transfer by calling transferOwnership with the new owner's address. The new owner must then accept the transfer by calling acceptOwnership.

3.2.9 Security Measures

- **Reentrancy Protection:** The contract uses the call method for transferring TRX, which mitigates the risk of reentrancy attacks.
- **Zero Address Check:** Transfers and ownership changes are validated to prevent operations involving the zero address.
- **Emergency Stop:** The pausing mechanism provides an emergency stop capability to halt contract operations if needed.

4. Mathematical Model of the SLS Token



The mathematical model of the SLS token describes the algorithms and formulas used for managing the issuance, distribution, pricing, and operations of the tokens. This description includes the key mathematical concepts and methods applied in the smart contract, as well as the rationale for choosing these models.

4.1 Token Issuance and Distribution

4.1.1 Total Issuance

The total supply of SLS tokens is defined as a fixed amount at the time of contract creation. This number is reflected in the _totalSupply variable.

Total Supply=560,000,000×1018 SLS\text{Total Supply} = 560,000,000 \times 10^{18} \, \text{SLS}Total Supply=560,000,000×1018SLS

where 101810^{18}1018 represents the token's decimal places.

4.1.2 Initial Distribution

All tokens at the contract creation stage are allocated to the contract owner's address:

Initial Balance of Owner=Total Supply\text{Initial Balance of Owner} = \text{Total Supply}Initial Balance of Owner=Total Supply

4.2 Token Pricing

4.2.1 Token Price

The price of one SLS token is set at the equivalent of 1 USD in wei. In the context of TRON, 1 wei equals 10−1810^{-18}10−18 TRX.

Token Price=1 USD×1018 wei\text{Token Price} = 1 \, \text{USD} \times 10^{18} \, \text{wei}Token Price=1USD×1018wei

4.2.2 Converting TRX to Tokens

The amount of tokens a user can purchase for the sent TRX is calculated as follows:

Amount of Tokens=Amount of TRXToken Price\text{Amount of Tokens} = \frac{\text{Amount of TRX}}{\text{Token Price}}Amount of Tokens=Token PriceAmount of TRX

where Amount of TRX\text{Amount of TRX}Amount of TRX is the amount of TRX sent.

4.2.3 Converting Tokens to TRX

The amount of TRX a user receives when selling tokens is calculated as follows:

Amount of TRX=Amount of Tokens×Token Price\text{Amount of TRX} = \text{Amount of Tokens} \times \text{Token Price}Amount of TRX=Amount of Tokens×Token Price

where Amount of Tokens\text{Amount of Tokens}Amount of Tokens is the amount of tokens being sold.

4.3 Token Operations

4.3.1 Token Balance

The token balance for any address is calculated as the sum of all incoming tokens minus all outgoing tokens.

Balance(a)=∑Received(a)−∑Sent(a)\text{Balance}(a) = \sum \text{Received}(a) - \sum \text{Sent}(a)Balance(a)=∑Received(a)−∑Sent(a)

where aaa is the user's address.

4.3.2 Token Transfer

When transferring tokens from one address to another, the balance of each participant is updated as follows:

Balance(sender)=Balance(sender)−Amount\text{Balance}(sender) = \text{Balance}(sender) - \text{Amount}Balance(sender)=Balance(sender)−Amount
Balance(receiver)=Balance(receiver)+Amount\text{Balance}(receiver) = \text{Balance}(receiver) + \text{Amount}Balance(receiver)=Balance(receiver)+Amount

where Amount\text{Amount}Amount is the number of tokens transferred.

4.3.3 Approval and transferFrom

Approval to transfer tokens on behalf of another address is done through the `approve` function. The approved amount is recorded in the `allowed` table.

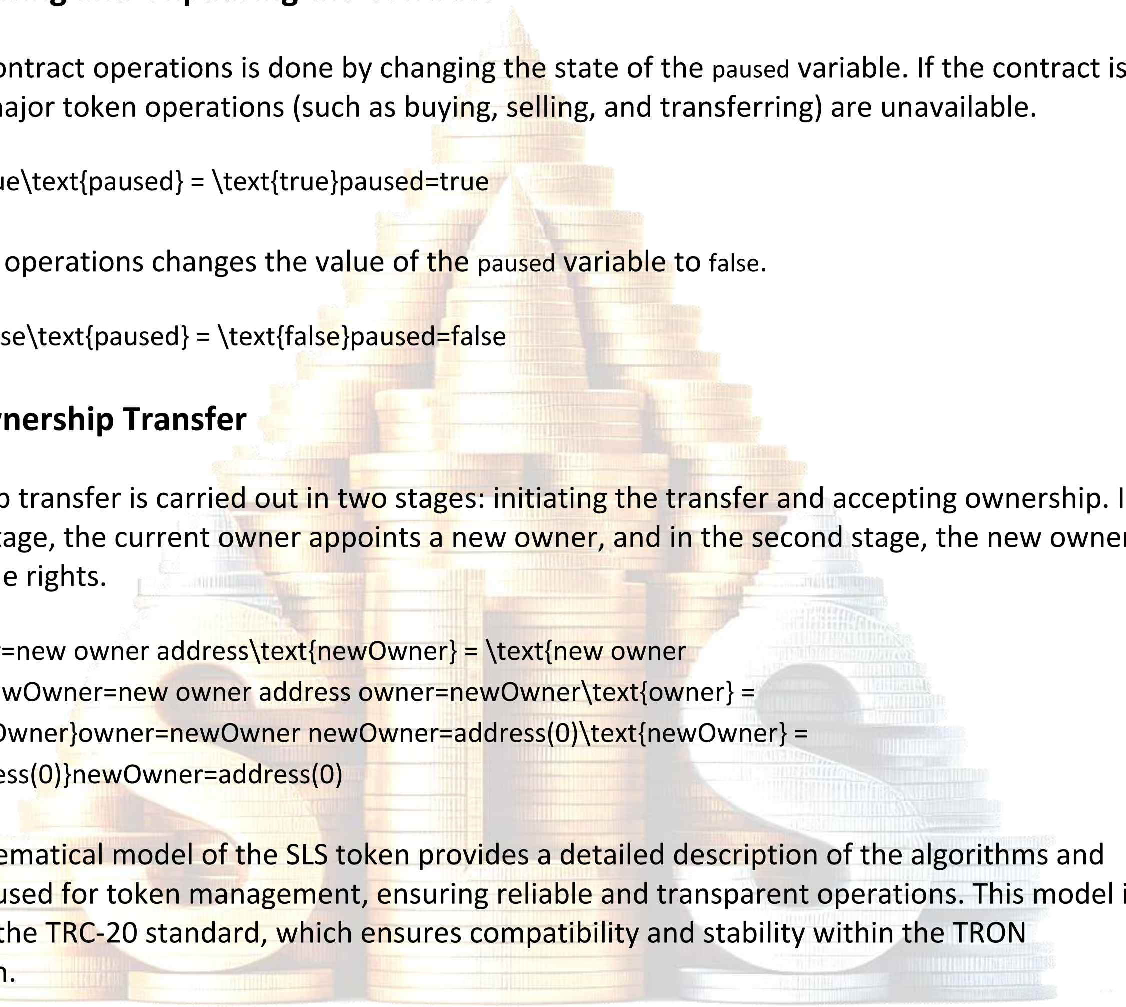
allowed[owner][spender]=Amount\text{allowed}[owner][spender] = \text{Amount}allowed[owner][spender]=Amount

Transferring tokens via `transferFrom` decreases the allowed amount and the sender's balance, increasing the recipient's balance:

allowed[owner][spender]=allowed[owner][spender]−Amount\text{allowed}[owner][spender] = \text{allowed}[owner][spender] - \text{Amount}allowed[owner][spender]=allowed[owner][spender]−Amount
Balance(owner)=Balance(owner)−Amount\text{Balance}(owner) = \text{Balance}(owner) - \text{Amount}Balance(owner)=Balance(owner)−Amount
Balance(receiver)=Balance(receiver)+Amount\text{Balance}(receiver) = \text{Balance}(receiver) + \text{Amount}Balance(receiver)=Balance(receiver)+Amount

4.4 Management Mechanisms

4.4.1 Pausing and Unpausing the Contract



Pausing contract operations is done by changing the state of the paused variable. If the contract is paused, major token operations (such as buying, selling, and transferring) are unavailable.

paused=true\text{paused} = \text{true}paused=true

Resuming operations changes the value of the paused variable to false.

paused=false\text{paused} = \text{false}paused=false

4.4.2 Ownership Transfer

Ownership transfer is carried out in two stages: initiating the transfer and accepting ownership. In the first stage, the current owner appoints a new owner, and in the second stage, the new owner accepts the rights.

newOwner=new owner address\text{newOwner} = \text{new owner address}newOwner=new owner address owner=newOwner\text{owner} = \text{newOwner}owner=newOwner newOwner=address(0)\text{newOwner} = \text{address(0)}newOwner=address(0)

The mathematical model of the SLS token provides a detailed description of the algorithms and methods used for token management, ensuring reliable and transparent operations. This model is based on the TRC-20 standard, which ensures compatibility and stability within the TRON blockchain.

5. Encryption Algorithms of the SLS Token

The SLS token employs cryptographic algorithms to ensure the security, integrity, and privacy of transactions within the TRON blockchain. This document provides a detailed overview of the encryption algorithms and security mechanisms used by the SLS token.

5.1 Public Key Cryptography

5.1.1 Elliptic Curve Cryptography (ECC)

TRON uses Elliptic Curve Cryptography (ECC) for public key cryptography. ECC provides high levels of security with relatively small keys, making it efficient and secure.

- **Algorithm:** SECP256k1
- **Key Size:** 256 bits

Key Generation:

1. **Private Key:** A randomly generated 256-bit number.
2. **Public Key:** Derived from the private key using the ECC algorithm.

Public Key=Private Key×G\text{Public Key} = \text{Private Key} \times GPublic Key=Private Key×G

where GGG is the generator point on the curve.

5.1.2 Digital Signatures

Digital signatures are used to verify the authenticity and integrity of transactions. TRON employs the Elliptic Curve Digital Signature Algorithm (ECDSA).

- **Algorithm:** ECDSA
- **Curve:** SECP256k1

Signature Generation:

1. **Message Hashing:** The message is hashed using the SHA-256 algorithm.
2. **Signature Creation:** The hash is signed using the private key.

Signature=(r,s)\text{Signature} = (\text{r}, \text{s})Signature=(r,s)

where r\text{r}r and s\text{s}s are derived from the private key and the message hash.

Signature Verification:

1. **Hash Calculation:** The received message is hashed using SHA-256.
2. **Signature Verification:** The public key and the signature are used to verify the hash.

Verify(Public Key,Message Hash,(r,s))=true/false\text{Verify}(\text{Public Key}, \text{Message Hash}, (\text{r}, \text{s})) = \text{true/false}Verify(Public Key,Message Hash,(r,s))=true/false

5.2. Hash Functions

5.2.1 SHA-256

The Secure Hash Algorithm 256 (SHA-256) is used for generating cryptographic hashes of data. It ensures data integrity by producing a fixed-size 256-bit hash value from an arbitrary amount of input data.

Properties:

- **Output Size:** 256 bits
- **Collision Resistance:** Very high
- **Pre-image Resistance:** Very high

Hash Calculation:

Hash=SHA-256(data)\text{Hash} = \text{SHA-256}(\text{data})Hash=SHA-256(data)

5.2.2 KECCAK-256

KECCAK-256, a variant of the SHA-3 algorithm, is used in certain TRON operations. It produces a 256-bit hash value and is known for its security and efficiency.

Hash Calculation:

Hash=KECCAK-256(data)\text{Hash} = \text{KECCAK-256}(\text{data})Hash=KECCAK-256(data)

5.3 Secure Communication

5.3.1 Transport Layer Security (TLS)

TLS ensures secure communication between clients and servers interacting with the TRON blockchain. It provides encryption, data integrity, and authentication.

- **Version:** TLS 1.2 or higher
- **Encryption Algorithms:** AES-256, ChaCha20
- **Key Exchange:** ECDHE (Elliptic Curve Diffie-Hellman Ephemeral)
- **Authentication:** X.509 certificates

TLS Handshake Process:

1. **Client Hello:** The client proposes an encryption method.
2. **Server Hello:** The server agrees on an encryption method.
3. **Certificate Exchange:** The server sends its certificate to the client.
4. **Key Exchange:** The client and server exchange keys.
5. **Finished Messages:** Both parties confirm the handshake and start encrypted communication.

5.4. Token Operations Security

5.4.1 Transaction Encryption

All transactions on the TRON blockchain are encrypted using the sender's private key. The transaction details are hashed using SHA-256, and the hash is signed with the private key to create a digital signature.

Transaction Structure:

1. **Sender Address:** Public key of the sender.
2. **Receiver Address:** Public key of the receiver.
3. **Amount:** Number of tokens to transfer.
4. **Signature:** Digital signature of the transaction hash.

5.4.2 Smart Contract Security

The SLS token smart contract includes several security mechanisms to prevent common vulnerabilities such as reentrancy, integer overflow/underflow, and unauthorized access.

- **Reentrancy Guard:** Prevents reentrant calls.
- **SafeMath Library:** Ensures safe arithmetic operations.
- **Access Control:** Only the contract owner can perform critical operations like pausing or unpausing the contract.

The SLS token employs robust cryptographic algorithms to ensure the security and integrity of transactions on the TRON blockchain. Public key cryptography, secure hash functions, and transport layer security collectively provide a comprehensive security framework for the SLS token and its operations.

6. SLS Token Protocol Description

The SLS token, deployed on the TRON blockchain, uses a variety of protocols to ensure reliable operation, security, and interaction with users and other smart contracts. This document presents the key protocols used by the SLS token.

6.1. TRC-20 Protocol

6.1.1 Description

TRC-20 is the standard protocol for creating and issuing tokens on the TRON blockchain. This standard ensures compatibility with various smart contracts and dApps (decentralized applications) within the TRON ecosystem.

6.1.2 Key Functions of TRC-20

- **totalSupply:** Returns the total number of issued tokens.
- **balanceOf:** Returns the token balance of a specified address.
- **transfer:** Moves the specified number of tokens from the sender to the recipient.
- **approve:** Allows the token owner to authorize another address to spend tokens on their behalf.
- **transferFrom:** Moves tokens from one address to another using the approval granted by the approve function.
- **allowance:** Returns the remaining number of tokens that have been approved for transfer to a specific address.

6.1.3 TRC-20 Events

- **Transfer:** Records the transfer of tokens from one address to another.
- **Approval:** Records the approval to transfer tokens on behalf of the owner.

6.2. Security Protocols

6.2.1 Access Control

- **onlyOwner Modifier:** Ensures that critical functions can only be called by the contract owner.
- **transferOwnership:** Allows the owner to transfer ownership rights to another address.
- **acceptOwnership:** Allows the new owner to accept ownership rights.

6.2.2 Overflow and Underflow Protection

Using the SafeMath library ensures safe arithmetic operations, preventing integer overflow and underflow.

6.2.3 Reentrancy Protection

Critical functions are protected from reentrancy attacks using the "checks-effects-interactions" pattern.

6.3 Pause and Resume Protocols

6.3.1 Pause

- **pause:** Suspends all critical contract functions such as buying and selling tokens. Can only be called by the owner.
- **unpause:** Resumes all suspended functions. Can only be called by the owner.

6.3.2 Application

Pausing is used to prevent operations in case of detected vulnerabilities or when performing updates and maintenance.

6.4. Token Trading Protocols

6.4.1 Buying Tokens

- **buyTokens:** Allows users to buy tokens by sending Ether to the contract. The number of tokens purchased is calculated based on the established token price.

6.4.2 Selling Tokens

- **sellTokens:** Allows users to sell tokens and receive Ether in exchange. The amount of Ether sent is calculated based on the established token price.

6.5 User Interaction Protocols

6.5.1 Sending Tokens

- **transfer:** Users can send tokens to other users by specifying the recipient's address and the number of tokens.

6.5.2 Approval and Transfer

- **approve:** Users can authorize third parties to spend tokens on their behalf.

- **transferFrom:** Third parties can transfer tokens on behalf of the user given the necessary approval.

6.5.3 Retrieving Information

- **balanceOf:** Users can find out the number of tokens at any address.
- **allowance:** Users can find out the number of tokens approved for transfer to a specific address.

The SLS token, implemented on the TRON blockchain, uses TRC-20 standards and protocols to ensure compatibility, security, and ease of use. The token contracts include advanced security protocols, attack protection, and functions for trading and managing tokens, ensuring reliability and convenience for users.

7. Future Improvements

7.1 Upgradable Contracts

Plans are in place to introduce upgradable contract mechanisms to allow for future enhancements and features without disrupting the existing ecosystem.

7.2 Additional Security Audits

Regular security audits and penetration testing will be conducted to ensure the contract's security and compliance with the latest security practices.

8. Conclusion

This Yellow Paper provides a detailed technical description of the SLS token, its architecture, and operational mechanisms. Adhering to strict technical standards and implementing robust security measures, the SLS token aims to provide a secure and efficient platform for value exchange within the SlonPlus ecosystem.

This document is intended to serve as a comprehensive guide for developers and specialists wishing to understand the technical foundations of the SLS token. For further details and updates, please refer to the official SlonPlus documentation and repositories.