"Crowds and charlatans have entered the cryptocurrency and smart contract spaces that not only lack cypherpunk overtones, but hate cypherpunk values, including values like trust minimization that give market value to certain cryptocurrencies like Bitcoin."

Nick Szabo

ABSTRACT

In a world where financial privacy is increasingly vulnerable and economic freedom is under threat, it is crucial to have options to protect our assets and keep our transactions safe. With the growing importance of privacy in the digital age, it is necessary to have tools that allow users to have full control of their data and safeguard their privacy. This is where Algorithm X's transaction obfuscation contract comes into play, a solution for those looking for greater privacy in their transactions and a way to protect their digital freedom.

Introduction

The introduction of a smart contract should begin with a clear presentation of its purpose and functionality, including the specific problems it solves. In technical terms, it could refer to the implementation of a set of rules and protocols programmed in a programming language to automate the execution and fulfillment of a contract. In addition, it can be mentioned that smart contracts are a type of decentralized application that runs on a blockchain and is used to ensure security, privacy and transparency in financial and commercial transactions. Importantly, these contracts eliminate the need for trusted intermediaries and third parties in transactions, providing greater efficiency, security and confidence in the process.

This smart contract, called Algorithm X

, has been created with the aim of offering a solution to the privacy and financial freedom issues faced by cryptocurrency users. In an increasingly digitized world, it is essential to have tools that protect our personal and financial information from malicious third parties, as well as having absolute control over our digital assets.

Algorithm X, built on Polygon MATIC blockchain technology, offers a high level of privacy by using a transfer feature that hides the origin and destination of transactions. In addition, it has a function to generate random codes that allow users to withdraw their funds safely and privately. All this without compromising the security and transparency that characterize blockchain technology.

We strongly believe that access to financial freedom is a fundamental right, and Algorithm X is our contribution to this goal. In this contract, you will not only find a solution to your privacy and security needs in the handling of cryptocurrencies, but also an invitation to reflect on the importance of financial freedom and the need to protect it.

ADVANTAGES

Privacy is one of the biggest advantages offered by our transaction obfuscation contract. On the Polygon network, transactions are public and anyone can track transactions from any address. However, our contract offers a solution for those users who want to keep their transactions private. By obfuscating transactions, our contract ensures that transactions cannot be traced to a specific address. This is achieved by generating random addresses and dividing the tokens into equal parts that are transferred to these addresses. In this way, you cannot link a specific address to a particular transaction.

Security is another important advantage of our contract. By dividing tokens equally and transferring them to random addresses, our contract reduces the risk of token loss or theft. In case an attacker could gain access to one of the random addresses, they would only have access to a fraction of the tokens, rather than the totality.

Our contract also offers a solution for those users who want to send large amounts of tokens without revealing the address of the final recipient. By dividing the tokens equally and transferring them to random addresses, our contract ensures that the final recipient receives the full amount of tokens, without their address being revealed.

In addition, our contract is easy to use and can be accessed from any wallet supported by the Polygon network. Users simply have to generate a transfer code and send their tokens to the address provided. Once transactions have been confirmed, users can claim their tokens using the same transfer code. The tokens will be sent to your specified wallet address.

In short, our transaction obfuscation contract offers a solution for those users who want to keep their transactions private and secure. In addition, it offers a solution for those users who want to send large amounts of tokens without revealing the address of the final recipient. Our contract is easy to use and can be accessed from any wallet supported by the Polygon network.

Special Features

Obfuscated classic transfer

Generate 10 random addresses ghost

```
    uint256[] memory addresses = new uint256[](10);
for (uint256 i = 0; i < 10; i++) {
addresses[i] = uint256(keccak256(abi.encodePacked(block.timestamp, i,
msg.sender))) % 2**160;
    }
```

Divide the amount of tokens into 10 equal parts

```
uint256 numChunks = 10;
uint256 chunkSize = amount / numChunks;
```

Send tokens to random addresses

```
for (uint256 i = 0; i < numChunks; i++) {
address recipient = address(uint160(addresses[i]));
_transfer(msg.sender, container, chunkSize);
    }
```

Transfer phantom address tokens to recipient

```
for (uint256 i = 0; i < numChunks; i++) {
address recipient = address(uint160(addresses[i]));
_transfer(recipient, to, chunkSize);
    }

return true;
 }
```

```
//funcion de tranferencia ofuscada
//Obfuscated transfer function
function transfer(address to, uint256 amount) public override returns (bool) {
require(balanceOf(msg.sender) >= amount, "Not enough tokens");

//Generar 10 direcciones aleatorias
//Generate 10 random addresses
uint256[] memory addresses = new uint256[](10);
for (uint256 i = 0; i < 10; i++) {
    addresses[i] = uint256(keccak256(abi.encodePacked(block.timestamp, i, msg.sender))) % 2**160;
}

//Dividir la cantidad de tokens en 10 partes iguales
//Divide the amount of tokens into 10 equal parts
uint256 numChunks = 10;
uint256 chunkSize = amount / numChunks;

//Enviar los tokens a las direcciones aleatorias fantasmas
//Send the tokens to the phantom random addresses
for (uint256 i = 0; i < numChunks; i++) {
    address recipient = address(uint160(addresses[i]));
    _transfer(msg.sender, recipient, chunkSize);
}

//Transferir los tokens de las direcciones fantasmas al usuario real
//Transfer the tokens from the phantom addresses to the real user
for (uint256 i = 0; i < numChunks; i++) {
    address recipient = address(uint160(addresses[i]));
    _transfer(recipient, to, chunkSize);
}

return true;
}
```

First of all, this function is responsible for transferring a number of tokens from the sender's address to the recipient's address, but it does so in an obfuscated way to increase the privacy of the transaction.

To achieve this, the function uses two main steps:

1. Generate random addresses: 10 random addresses are generated using the hash of a combination of the block's timestamp, an index, and the sender's address. These addresses are "ghosts" and do not belong to any natural or legal person.
2. Transfer tokens to the random addresses and then to the recipient: The number of tokens is divided into 10 equal parts and each is sent to one of the random addresses generated above. The tokens are then transferred from the random addresses to the final recipient in equal parts.

This process creates multiple transactions that are difficult to trace, in the chain of blocks the source account A will never be linked with

the destination B here a data masking occurs, only 20 transfers will be reflected in the polygonscan thus increasing the privacy of the transaction. In addition, the division of tokens into equal parts makes it more difficult for attackers to obtain information about the total amount of tokens transferred.

Transfer functions by bytes32 algorithmic code

```
function generateTransferCode(uint256 amount) public returns (bytes32) {

        require(balanceOf(msg.sender) >= amount, "Insufficient
balance");

        bytes32 code = keccak256(abi.encodePacked(msg.sender,
block.timestamp, amount));

        transferCodes[msg.sender] = code;

        transferAmounts[code] = amount;

        _burn(msg.sender, amount);

        return code;

    }
```

generateTransferCode(uint256 amount): This function is responsible for generating a transfer code based on the provided amount of tokens. Here's how it works:

First, it checks if the sender has a sufficient balance of tokens to generate the transfer code. If the balance is not enough, it throws an error.

Next, it uses the keccak256 hash function to create a unique code by encoding the sender's address, the current block timestamp, and the amount of tokens.

The generated code is stored in the transferCodes mapping, where the sender's address is the key and the code is the value.

Additionally, the amount is stored in the transferAmounts mapping using the generated code as the key.

Finally, it burns (removes) the specified amount of tokens from the sender's balance and returns the generated code.

```
 function getCode() public view returns (bytes32) {
        return transferCodes[msg.sender];
    }
```

getCode(): This function allows a user to retrieve their transfer code. Here's how it works:

It simply returns the transfer code stored in the transferCodes mapping for the sender's address. The transfer code serves as a reference for the generated transfer.

```
function withdrawWithCode(bytes32 code) public {
        require(transferAmounts[code] > 0, "Invalid code");
        uint256 amount = transferAmounts[code];
        transferAmounts[code] = 0;
        _mint(msg.sender, amount);
    }
```

withdrawWithCode(bytes32 code): This function is used to withdraw the balance associated with a given transfer code. Here's how it works:

First, it checks if the provided code exists in the transferAmounts mapping and if it corresponds to a positive amount. If the code is invalid or has already been used, it throws an error.

If the code is valid, it retrieves the amount of tokens associated with the code.

The amount is then set to zero in the transferAmounts mapping to mark the code as used.

Finally, it mints (creates) the specified amount of tokens and adds them to the sender's balance.

Tokeninfo

Information about the Token

Algorithm X: First privacy token on Polygon network, offers confidential transactions and user privacy protection.

- **Name:Algorithm X**
- **Symbol:AlgoX**
- **Max Supply: 1.000.000**
- **Decimals:18**
- **Contract 0x**
- **Blockchain Polygon**

Tokenomics

Public Sale (50%): During the public sale, 50% of the total tokens (500,000 tokens) will be made available for the general public to purchase. This allows for wider participation and distribution among the public.

Team Allocation (10%): 10% of the tokens (100,000 tokens) will be allocated to the project team. These tokens will be used to incentivize and reward the team's efforts in the development and growth of AlgorithmX.

Private Mint (15%): 15% of the tokens (150,000 tokens) will be allocated for private minting. This allows for controlled distribution among strategic partners, early contributors, or investors who are actively involved in the project.

Foundation Reserve (10%): 10% of the tokens (100,000 tokens) will be reserved for the project's foundation or responsible entity. These tokens will be used to support the long-term sustainability, community initiatives, and further development of AlgorithmX.

Partnerships and Alliances (10%): 10% of the tokens (100,000 tokens) will be allocated to establishing strategic partnerships and collaborations with other projects, businesses, or entities in the ecosystem. This will foster growth, adoption, and mutual benefits within the AlgorithmX network.

Rewards and Liquidity Program (5%): 5% of the tokens (50,000 tokens) will be dedicated to rewards and liquidity programs. These tokens will be used to incentivize community engagement, provide liquidity support, and reward users for their active participation in the AlgorithmX ecosystem.

Conclusion

In short, the transaction obfuscation contract provides an innovative and effective solution for those users who want to maintain their privacy and security while transacting with cryptocurrencies. Through the generation of random addresses and the division of tokens into equal parts, this contract provides greater security and privacy to users, allowing them to make transactions without worrying about the exposure of their personal or financial information.

In addition, the code transfer feature provides an additional way for users to transfer tokens securely and privately, avoiding the need to share public or private addresses.

Together, the transaction obfuscation contract and code transfer feature offer users greater security and privacy when transacting with cryptocurrencies, opening up new possibilities and opportunities in the cryptocurrency space.

Dedication:

This project is dedicated to Nick Szabo, a pioneer in the field of cryptocurrencies and blockchain technology. His contributions and visions have been instrumental in the development and evolution of this field, inspiring many to move forward in exploring the possibilities and potential of this revolutionary technology.

AlgorithmX@gmx.com