
NFT
ANONYMOUS
COMMUNICATOR

Objective: Enable NFT owners to communicate anonymously with wallets to show interest in their asset for purchase or for collaboration.

Enable sending a signed message to wallet

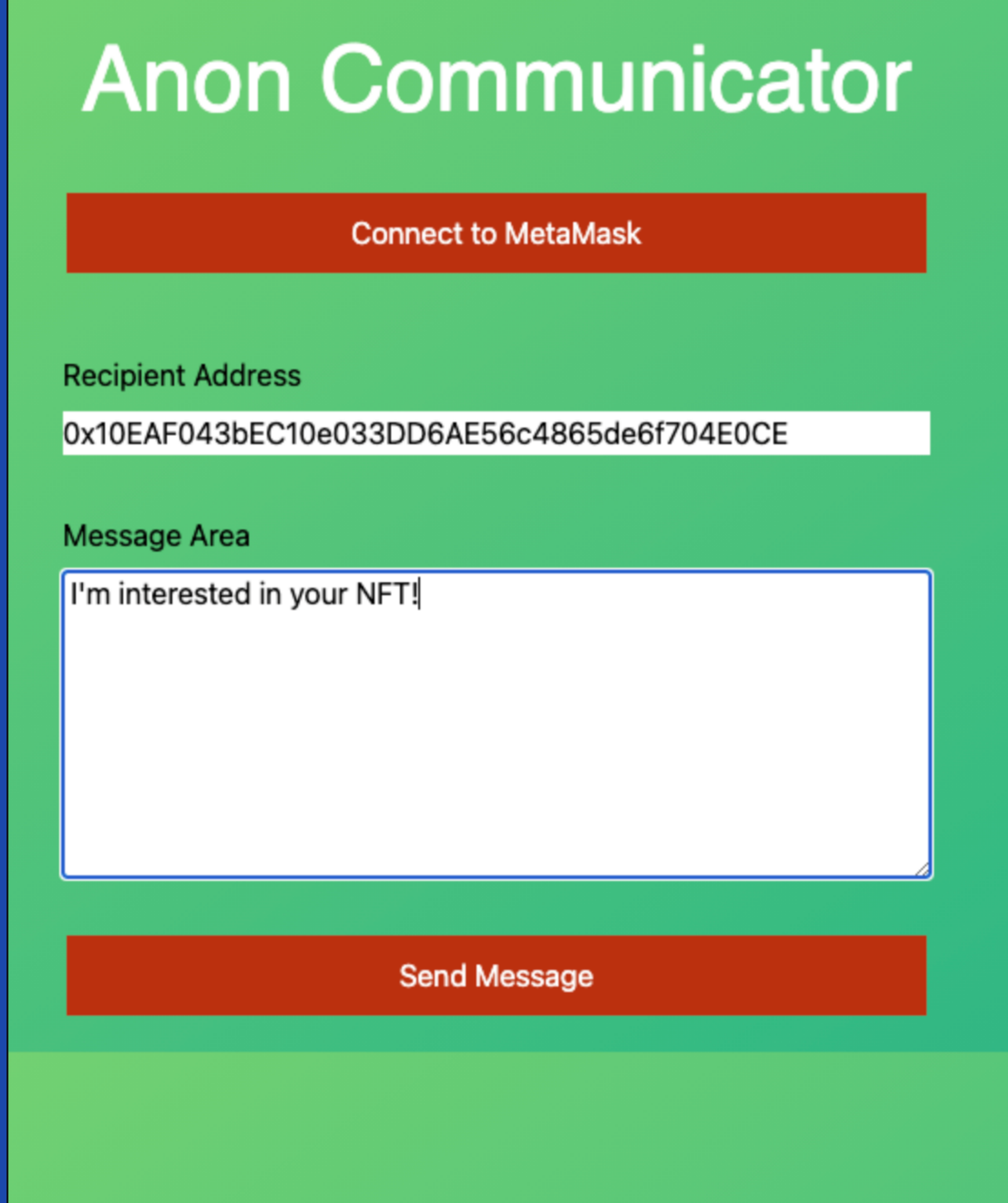
The signed message is readable only by the wallet owner

Mint a token to show a message waiting for the owner.

1. **Front end app to login using wallet (eg. Metamask: ETH, Phantom: SOL)**
2. **Personal user experience**
   - A. Message sender -
     1. Enter the public wallet address
     2. Enter the text message
     3. Hit Send Message
   - B. Message Recipient
     1. Login to the app
     2. Displays # of messages waiting for your wallet address
     3. Enable access to read the message.
     4. Choose to reply (switch role as message sender)
3. **Encrypt the message and store it on IPFS**
4. Mint token and send token to recipients wallet address
   a. Token metadata – IPFS link to the message

Application homepage

- Text box to input the wallet address of the message recipient
- Text box for the message content
- Send message button that calls the mint function in the smart contract. The minted token is then sent to the recipient address.
- The message content is parsed to a json file that can then be pinned to the Pinata IPFS.
- The recipient see's the token in their wallet prompting them to login to the Anon communicator where they can view messages in their inbox.



MetaMask Login function

```
// Login with Web3 via Metamasks window.ethereum library
async function loginWithEth() {
  if (window.web3) {
    try {
      // We use this since ethereum.enable() is deprecated. This method is not
      // available in Web3JS - so we call it directly from metamasks' library
      const selectedAccount = await window.ethereum
        .request({
          method: "eth_requestAccounts",
        })
        .then((accounts) => accounts[0])
        .catch(() => {
          throw Error("No account selected!");
        });
      window.userAddress = selectedAccount;
      window.localStorage.setItem("userAddress", selectedAccount);
      showAddress();
    } catch (error) {
      console.error(error);
    }
  } else {
    alert("No ETH brower extension detected.");
  }
}
```

On the home page, a login button allows the user to connect their MetaMask wallet.

Solidity Smart Contract

This contract mints the token to be deposited into the wallet address that holds the NFT the user is interested in.

```
pragma solidity ^0.5.0;

import "https://github.com/OpenZeppelin/openzeppelin-contracts/blob/release-v2.5.0/contracts/token/ERC721/ERC721Full.sol";

contract ArtToken is ERC721Full {
    constructor() public ERC721Full("ArtToken", "ART") {}

    function registerArtwork(address owner, string memory tokenURI)
        public
        returns (uint256)
    {
        uint256 tokenId = totalSupply();
        _mint(owner, tokenId);
        _setTokenURI(tokenId, tokenURI);

        return tokenId;
    }
}
```

mintNFT function with parameters "address" and "tokenURI"

```
const nftContract = await new web3.eth.Contract(abistring, contractAddress);

 const nonce = await web3.eth.getTransactionCount(accounts[0], 'latest'); //get latest nonce

 //the transaction
 const tx = {
   'from': accounts[0],
   'to': contractAddress,
   'nonce': nonce,
   'gas': 500000,
   //'maxPriorityFeePerGas': 2000000,
   'data': nftContract.methods.registerArtwork(address, tokenURI).encodeABI()
 };

 //step 4: Sign the transaction
 const signedTx = await web3.eth.accounts.signTransaction(tx, PRIVATE_KEY);
 const transactionReceipt = await web3.eth.sendSignedTransaction(signedTx.rawTransaction);

  console.log(`Transaction receipt: ${JSON.stringify(transactionReceipt)}`);
```

Build Json Function

Build Json function used to create the json file to be passed to Pinata IPFS.

```
function buildJson() {
  var y = document.getElementById("messageArea").value;
  var x = document.getElementById("recepientAddress").value;
  var f = '{"address" :"'+x +'", "message" :"' +y +' "}'
  console.log(f)
  //const myJSON= '{"address":"0x123456", "message":"I want to buy your NFT"}';
  //const myObj = JSON.parse(myJSON);

  const myObj = JSON.parse(f);
  document.getElementById("demo").innerHTML = myObj.address;
  //document.getElementById("demo").innerHTML = myObj.message;
}
```

Pin the message to IPFS and return the IPFS hash and timestamp

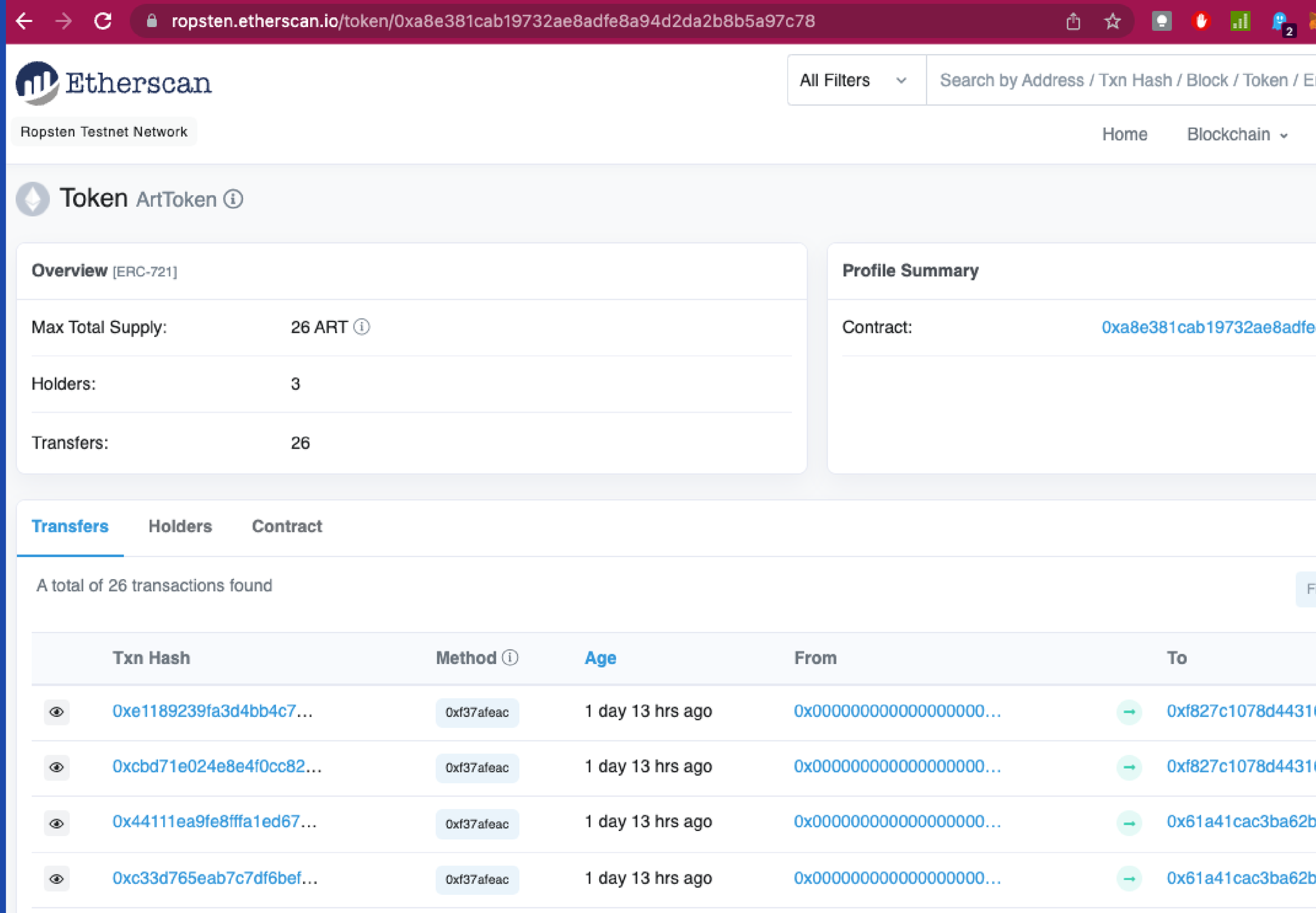
```
function URI_Fetch(myCallback)
{
    var y = document.getElementById("messageArea").value;
    var x = document.getElementById("recepientAddress").value;
    var f = '{"address" :"'+x +'", "message" :"' +y +' "}'
    console.log(f)

    //const myJSON= '{"address":"0x123456", "message":"I want to buy your NFT"}';
    //const myObj = JSON.parse(myJSON);

    const myObj = JSON.parse(f);
    document.getElementById("demo").innerHTML = myObj.address;
    var pinData;
    fetch('https://api.pinata.cloud/pinning/pinJSONToIPFS', {
    method: 'POST', // or 'PUT'
    headers: {
      'Content-Type': 'application/json',
      'pinata_api_key': "26689a58bbd3e05207c5",
      'pinata_secret_api_key': "622c0634c379b1139f92dfd22f6dd79dd600df1eb09bec13cb70456a5e3835fa"
    },
    body: JSON.stringify(myObj),
    })
    .then(response => response.json())
    .then(data => {

    console.log('Success:',data);
    pinData=data;
    console.log('IpfsHash:',pinData.IpfsHash);
    console.log('Timestamp',pinData.Timestamp);
    myCallback(data)
    })
    .catch((error) => {
    console.error('Error:', error);
    });
```

ART on Ropsten Testnet

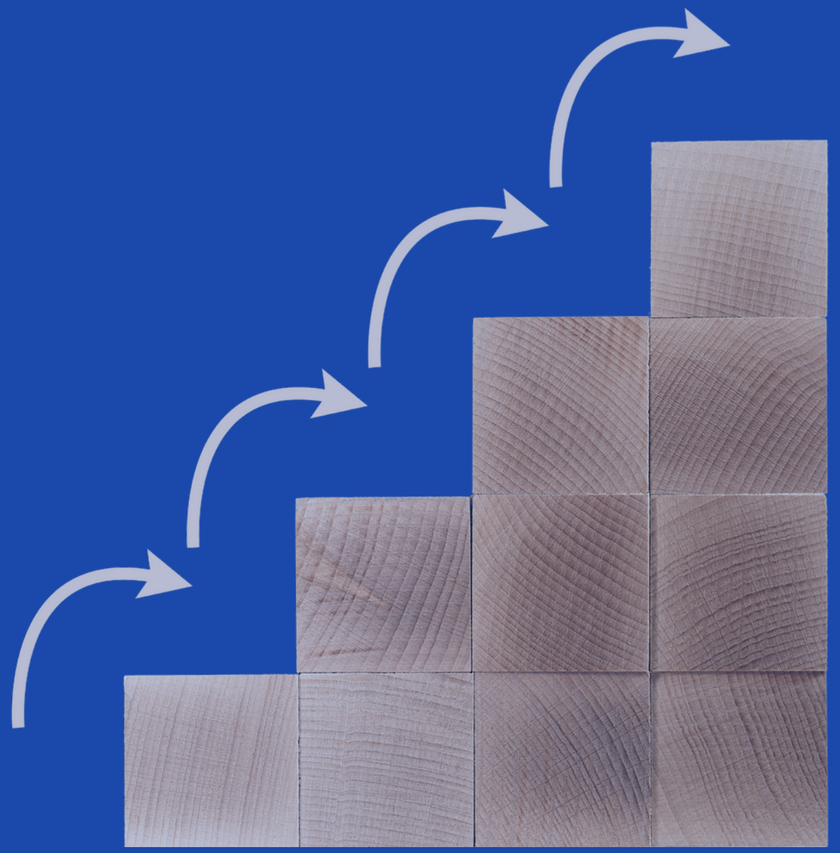


Obstacles

- Integrating MetaMask in Streamlit
- Pinning custom user input to Pinata IPFS.
- Connections between UI, Pinata, Alchemy

Next Steps

- Message encryption so that only the intended recipient of the message can view it.
- Non transferrable tokens



Creators

- Noah Beito
- Owen Wardlaw
- Israel Fernandez
- Stephen Miyumo
- Aheesh Nagraj