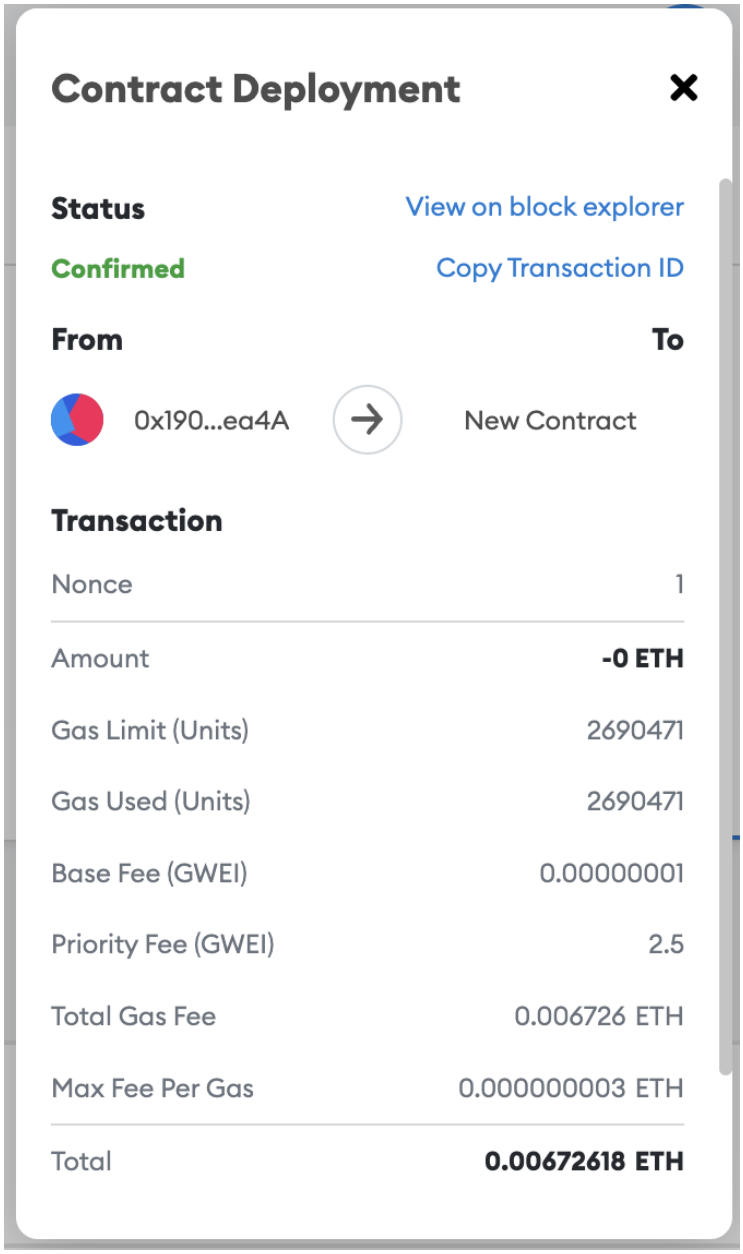
Contract Deployment
Status
View on block explorer
Confirmed
Copy Transaction ID
From
To
0x190...ea4A
New Contract
Transaction
Nonce 1
Amount -0 ETH
Gas Limit (Units) 2690471
Gas Used (Units) 2690471
Base Fee (GWEI) 0.00000001
Priority Fee (GWEI) 2.5
Total Gas Fee 0.006726 ETH
Max Fee Per Gas 0.000000003 ETH
Total 0.00672618 ETH

Program:

```
pragma solidity 0.8.6;

//Importing two repos that will allow the minting process
import "https://github.com/0xcert/ethereum-erc721/src/contracts/tokens/nf-token-metadata.sol";
import "https://github.com/0xcert/ethereum-erc721/src/contracts/ownership/ownable.sol";

//Creating our smart contract
contract newNFT is NFTokenMetadata, Ownable
{

  constructor()
  {
    nftName = "Image Of Me Sebu";
    nftSymbol = "IOMS";
  }

//function mint, takes in address, tokenId, uri
//external function to owner
  function mint(address _to, uint256 _tokenId, string calldata _uri) external onlyOwner
  {
    super._mint(_to, _tokenId);
    super._setTokenUri(_tokenId, _uri);
  }

}
```

(Does not belong to me and isn't altered) found at this link
Vvvvvv

```
pragma solidity ^0.8.5;
//https://www.youtube.com/watch?v=n6nEPaE7KZ8
//into the function you pass in two addresses ["address1","address2"] then the merkle array of bytes32
//then we pass ion the root "top hash", the leaf, and the index
contract MerkleProof
{
    function verify(bytes32[] memory proof, bytes32 root, bytes32 leaf, uint index) public pure returns (bool)
    {
        bytes32 hash = leaf;
        //Recompute Merkle Root
        //We'll utilize a for loop in order to iterate through the merkle root
        for (uint i = 0; i < proof.length; i++)
        {
            ///If you look at a merkle tree it takes two hashs to get to the next level so this is just checking
            //That the left hash is even (it always is) otherwise it's the other way around
            if (index%2==0)
            {
                hash = keccak256(abi.encodePacked(hash, proof[i]));
            }
            else
            {
                hash = keccak256(abi.encodePacked(proof[i], hash));
            }
            ///This is our steps to working our way up the merkle tree, so 16,8,4,2,1(merkle root)
            index = index/2;
        }
        //This is the boolean that is returned if true!
        return hash == root;
    }
}
```