Assignment 1

Problem 1

1.

rkle_js > {} public.json > ...

```
[
  "6464294476958346139385024074008223400825166653076969388043746597957512245037",
  "1",
  "2",
  "3",
  "4"
]
```

2.

Input:

{} proof.json U   mimcsponge.circom U   merkle.circom U

{} input.json > [ ] leaves

```
{
  "leaves": [1, 2, 3, 4, 5, 6, 7 ,8]
}
```

Error:

```
[DEBUG] snarkJS: betaTauG1: fft 12 join  12/12  1/1 2/4
[DEBUG] snarkJS: betaTauG1: fft 12 join  12/12  1/1 0/4
matthew@matts-stonks-mac merkle_js % snarkjs groth16 setup ../merkle.r1cs pot12_final.ptau merkle2_0000.zkey

[ERROR] snarkJS: circuit too big for this power of tau ceremony. 9240*2 > 2**12
matthew@matts-stonks-mac merkle_js %
```

Share

This was solved by changing:

```
snarkjs powersoftau new bn128 12 pot12_0000.ptau -v
```

```
snarkjs powersoftau new bn128 15 pot12_0000.ptau -v
```

Output

```
e_js > {} public.json > ...
[
  "7457672556014162487472065518158328090252704233415054189820328174772177160972",
  "1",
  "2",
  "3",
  "4",
  "5",
  "6",
  "7",
  "8"
]
```

3.

No.

Yes, a smart contract can theoretically achieve the same outcome. However, it will require the user to publicly provide the information they want to verify is in the tree. Applications like Zcash benefit from this, as it allows for private transactions, while still proving that the transaction actually took place.

Problem 2

Minting of 2 NFTs

```
[vm] from: 0x5B3...eddC4 to: NFT.mint(bytes32[],uint256,address) 0x1c9...2b4bD value: 0 wei
data: 0x23c...1635c logs: 1 hash: 0xd66...c91e9
status              true Transaction mined and execution succeed
transaction hash    0xd6627ba1b5d36a9a04e76170e5a9752c04871bbd3b0972060ab6d2acb56c91e9
from                0x5B38Da6a701c568545dCfcB03FcB875f56beddC4
to                  NFT.mint(bytes32[],uint256,address) 0x1c91347f2A44538ce62453BEBd9Aa907C662b4bD
gas                 80000000 gas
transaction cost    107997 gas
execution cost      107997 gas
hash                0xd6627ba1b5d36a9a04e76170e5a9752c04871bbd3b0972060ab6d2acb56c91e9
input               0x23c...1635c
decoded input       {
                        "bytes32[] _merkleProof": [
                            "0x1e36a91400f78f385580bf271e487c1d0f1d9433d9d2333af854125862334ca0",
                            "0xffc0287a32bb2b465c05ecfb595cb1316f887b526ab2f926b04e9cb13cc1635c"
                        ],
                        "uint256 _tokenId": "1",
                        "address _recipient": "0xAb8483F64d9C6d1EcF9b849Ae677dD3315835cb2"
                    }
decoded output      {}
logs                [
                        {
```

```
[vm] from: 0x5B3...eddC4 to: NFT.mint(bytes32[],uint256,address) 0x1c9...2b4bD value: 0 wei      Debug
data: 0x23c...7fe35 logs: 1 hash: 0xfdf...a5b1f
status              true Transaction mined and execution succeed
transaction hash    0xfdf02d434a275924a3ccf2d7f1aee67ff1591cac095454c7900d16af7e5a5b1f
from                0x5B38Da6a701c568545dCfcB03FcB875f56beddC4
to                  NFT.mint(bytes32[],uint256,address) 0x1c91347f2A44538ce62453BEBd9Aa907C662b4bD
gas                 80000000 gas
transaction cost    90899 gas
execution cost      90899 gas
hash                0xfdf02d434a275924a3ccf2d7f1aee67ff1591cac095454c7900d16af7e5a5b1f
input               0x23c...7fe35
decoded input       {
                        "bytes32[] _merkleProof": [
                            "0x91078ba4b128b252bb01d1966d06c5d646f741c627b85f9f191b3c1540d95f3b",
                            "0x5ddfa65f0dea5533c994216abdb5f35db777e4fc6cf0555b1cf777d4d5d7fe35"
                        ],
                        "uint256 _tokenId": "2",
                        "address _recipient": "0x4B20993Bc481177ec7E8f571ceCaE8A9e22C02db"
                    }
decoded output      {}
logs                [
                        {
```

Problem 3

1.
SNARKs require a trusted setup, but STARKs do not.
This means that applications that utilise SNARKs will have to conduct a ceremony every time they make changes to their code base. This is rather computationally heavy, which hence reduces scalability.

2.
STARKs, which are hence more scalable, but their proofs take up more space, and hence more time is required to verify them. This means that each time the application requires a proof and verification, it might be slower than a SNARK.
The setup for Groth16 is not universal, but PLONK has a universal setup.
This means that Groth16 has to conduct a ceremony every time the smart contract is edited. PLONK's universal setup means that only one setup has to be done, and it will work for all future circuits, and this only costs 10% more gas than Groth16.

3.
ZK proofs and merkle trees can be used to prove that a certain wallet address is whitelisted. A merkle tree with all the whitelisted wallet addresses can be generated, and users that want to prove that their wallets are whitelisted can simply provide the root hash of the tree, proving that their wallet address was one of the leaf nodes.

Code: https://github.com/matthew-chua/ZKU.git