UNIVERSITY OF EDINBURGH

Assignment #1

Student name: *Jure Sternad (s2450797)*

Course: *Blockchains & Distributed Ledgers*
Due date: *October 17, 2022*

1

a.

Formally prove (or disprove) the following proposition: "It is infeasible to forge a proof of inclusion for a Merkle Tree that uses a collision resistant hash function."

Answer. Let $a_1, ..., a_n$ be the chunks of data of the Merkle Tree and H collision resistant hash function. Let MTR be the Merkle tree root of this tree ($MTR = H(H(...(H(H(a_1) \parallel H(a_2))...) \parallel H(...(H(H(a_{n-1}) \parallel H(a_n))...))$. Assume that for chunk x which is not in the tree, prover claims that it is and provides proof-of-inclusion Π. Let us assume that prover claims that x is on the position of i of the tree (instead of a_i) and let i be odd. Let the MTR^* be the new root of the tree. Now let us show that $MTR \neq MTR^*$.

Because H is collision resistant, $H(a_i) \neq H(x)$ for every $a_i \neq x$. That implifies that $H(H(a_i) \parallel H(a_{i+1})) \neq H(H(x) \parallel H(a_{i+1}))$. If we repeat the procedure untill we get to the root, we can see that $MTR = H(H(...H(a_i)...)) \neq H(H(...H(x)...)) = MTR^*$. Proof is very simillar in the case that i is even (we use chunk a_{i-1} instead of a_{i+1}) and the case that i is the only child of its parent (we can imagine that hashes on the pre-last level are the new leaves).

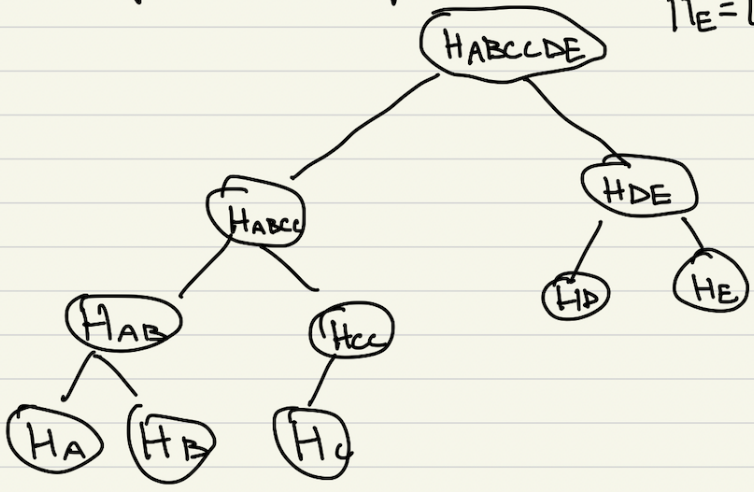
b.

Describe (or sketch) i) a binary full Merkle Tree, ii) a binary complete Merkle tree, for the following chunks of data: A, B, C, D, E. Provide the proof of inclusion for E in both cases.

Answer.

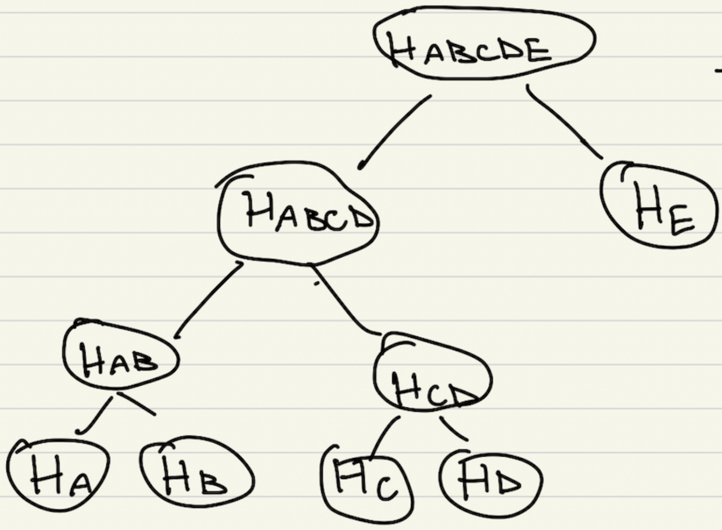
1. Complete (not full)

$\pi_E = [H_D, H_{ABCC}]$



2. Full (not complete)

$\pi_E = [H_{ABCD}]$



2

Derive the formula for the birthday paradox (show your work, explaining every step) and calculate the approximate number of elements needed to find a collision with at least 50probability. Apply this to find out approximately how many Bitcoin wallets needed to initialize their seed, which is a randomly sampled sequence of 12 words from the list in https://github.com/bitcoin/bips/blob/master/bip-0039/english.txt, to have the event that, with probability at least 50both with and without replacement of sampled words).

Answer. Let $P(n,k)$ be probability that in the group of k people, 2 have birthday on the same day if there are n possible days. The probability can be represented by
1 - "probability that no one has the birthday on same day as someone else".

$$P(n,k) = 1 - \frac{n}{n} \cdot \frac{n-1}{n} \cdots \frac{n-k+1}{n} = 1 - \prod_{i=0}^{k-1}(\frac{n-i}{n})$$

We can derive the second part of equation and use the fact that for $f(x) = 1 - x$ and $g(x) = e^{-x}$, $f(x) \leq g(x)$ for any $x \geq 0$.

$$\prod_{i=0}^{k-1}(\frac{n-i}{n}) = \prod_{i=0}^{k-1}(1 - \frac{i}{n}) \leq \prod_{i=0}^{k-1}(e^{-\frac{i}{n}}) = e^{-\frac{1}{n}\cdot\sum_{i=0}^{k-1} i} = e^{-\frac{k\cdot(k-1)}{2n}}$$

We now put the derivited part into our original equation and approximate the behaviour of function $e^{-\frac{k\cdot(k-1)}{2n}}$ with $e^{-\frac{k^2}{2n}}$.

$$P(n,k) = 1 - \prod_{i=0}^{k-1}(\frac{n-i}{n}) \geq 1 - e^{-\frac{k\cdot(k-1)}{2n}} \approx 1 - e^{-\frac{k^2}{2n}}$$

Now we just need to find out when is $1 - e^{-\frac{k^2}{2n}} = \frac{1}{2}$

$$1 - e^{-\frac{k^2}{2n}} = \frac{1}{2}$$
$$\frac{1}{2} = e^{-\frac{k^2}{2n}}$$
$$\ln\frac{1}{2} = -k^2 2n$$
$$k^2 = 2n \cdot \ln 2$$
$$k = \sqrt{2n \cdot \ln 2} \approx 1.17\sqrt{n}$$

Second part:

a) with replacement:

$$n = 2048^{12} = 5444517870735015415413993718908291383296$$

$$\Rightarrow k = 1.17 \cdot \sqrt{5444517870735015415413993718908291383296} = 86330762264960701562.88$$

b) without replacement:

$$n = \binom{2048}{12} = 1100526171791803717565934919116 8$$

$$\Rightarrow k = 1.17 \cdot \sqrt{11005261717918037175659349191168} \approx 3881378977329835.681$$

3

Assume that all Bitcoin miners are honest except one (the attacker). A miner creates a block B which contains address α, on which they want to receive their rewards. The attacker changes the contents of B, replacing α with an adversarial address α'. What needs to happen for the attacker to receive the rewards for B?

Answer. Attacker could perform a version of "sellfish mining attack". He would need to have enough computatuional power and have a sufficient enough connection to the network. He would mine blocks on his own private chain and publish them at the right time. If the attacker started mining his own private chain before the block B was published and there exists a time when his chain is ahead of the official chain, he could publish his chain and because it would be longer, the nodes would assume it as the correct chain. By that he would steal all of the miners block rewards for blocks that are not the part of the chain anymore.
(Note: his chain should be longer in the sense of longest chain - having more computation/work done for it, which also usually results in having more blocks).

4

Give two detailed examples of how an orphan block can be created in Bitcoin.

Answer.

- **First example**: Two or more miners mine a block at almost the same time, both with valid proof of work. Because of the time delay and network connectivity, some nodes will approve one of those blocks while the other nodes will approve other blocks. The chain forks into two or or more branches and the conflict is resolved by the next block that is mined and gets approved. When one of those branches becomes longer (in terms of longer chain), the nodes choose that branch now becomes part of the official chain while other branches get orphaned. The blocks on other branches are known as the orphan blocks. The majority of transactions that are in orphan blocks is also in the block that has been put on the chain while other transactions become candidate transactions for the future blocks.

- **Second example**: (51% attack) A miner or a group of miners that control more than 51 % of the computational power in the network would mine secretly and not publish the mined blocks to the network. By having that much of the power, they would mine blocks faster and build a version of blockchain that is longer then the official one. At chosen time, they would publish their chain and all of the nodes would recognize it as the longest chain. Although the goal of the attack should be double spending (sending transactions to the public chain, meanwhile building private chain and after the publishment of their chain having the transactions on the publich branch reversed), the side result is that blocks that are not included on their chain after the attack would get orphaned.

5

5. Construct a digital signature scheme, based on a hash function, that is one-time secure (i.e., it is secure if each private key signs only a single message). Describe in detail how the keys are generated and how signatures are issued and verified.

Answer. Based on the Winternitz Scheme:

- **Prepreparation**:
  1. Choose parameter t (in the original Winternitz Scheme $w = 2^t$)
  2. Define $N = \lceil \frac{n}{t} \rceil + \lceil \frac{\lceil \log_2 n - \log_2 t \rceil + t}{t} \rceil$
- **Key generation**
  1. **Private key**: Choose random n-bits size number x_i for $i = 1, \dots, N$.
  2. **Public key**: Iterate collision resistant hash function H $(2^t - 1)$-times for every x_i:
     $y_i = H^{2^t - 1}(x_i)$ for $i = 1, \dots, N$.
     Define public key $Y = H(y_1 \parallel y_2 \parallel \cdots \parallel y_N)$.
- **Issuing signature**:
  1. Let m be the message.
  2. Split hash of message $H(m)$ into $\lceil \frac{n}{t} \rceil$ t-bits parts $b_1, \dots, b_{\lceil \frac{n}{t} \rceil}$.
  3. Issue signature $s_i = H^{b_i}(x_i)$ for $i = 1, \dots, N$.
- **Verifying signature**:
  1. Split message m into $\lceil \frac{n}{t} \rceil$ t-bit parts $b_1, \dots, b_{\lceil \frac{n}{t} \rceil}$.
  2. Compute $v_i = H^{2^t - b_i - 1}(s_i)$ for $i = 1, \dots, N$.
  3. Check that $Y = H(v_1 \parallel v_2 \parallel \cdots \parallel v_N)$.

The signature in the scheme is secure because it generates an unique hash for every part of the message which is then used for encryption with the "secret" private key. Because the hash is unique, anykind of changing the message would result in different hash. The private key also stays hidden from everyone else except the owner, resulting that signature can be veryfied without knowing the private key.

6

Using the course's Ethereum testnet, send 1 ETH to the address of a fellow student. Write a small description on how you conducted the payment, including the transaction's id and addresses which you used.

Answer. I have sent 1 ETH to the address `0xB7967E81DA3d7D3fec5d63106B909f51672F7919`. I used MetaMask on which I just followed the automated parameters for gas limit and gas price. The transaction id is
`0x87df7914ff6a0e06314b77d68d29db69b17ec071dbab12058164b1bf513a1373`.

Here is the additional data of the transaction:

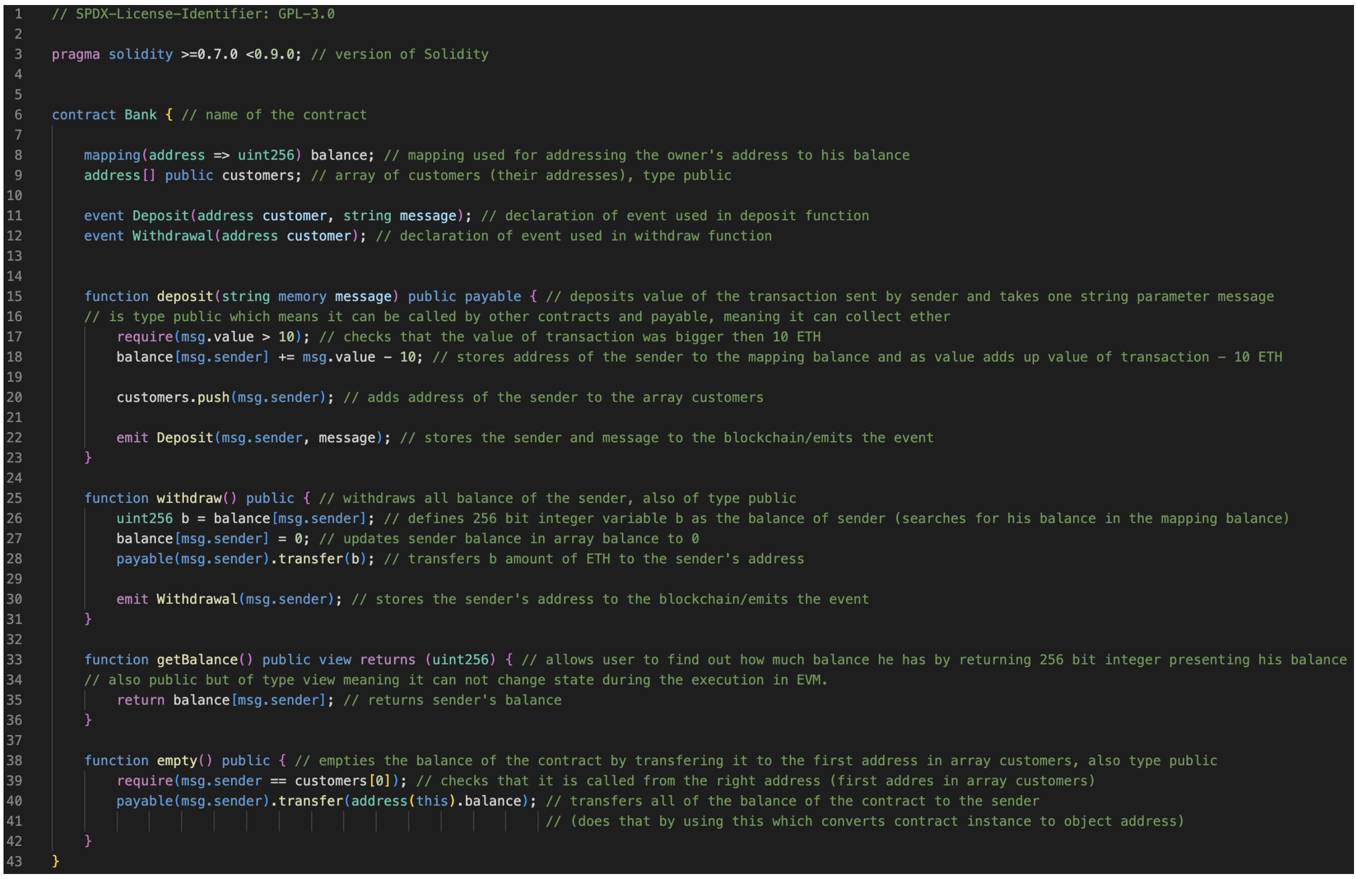
- Gas Limit (Units): 21000
- Gas Used (Units): 21000
- Gas price: 1
- Total: 1.000021 BTL_ETH

7

A smart contract has been deployed on the course's testnet. Its code is available here and its deployed address is: 0xA8D9D864dA941DdB53cAed4CeB8C8Bcf53aFe580 You can compile and interact with it using Remix. You should successfully create a transaction that interacts with the contract, either depositing to or withdrawing from it some coins. Describe the contract's functionality (that is, the purpose of each variable, function, and event) and provide the id of the transaction you performed and the address you used.

Answer. For both, deposit and withdrawal, I used the address:
`0xb892aEC81166aaeA7B13054C7908d8B64A9C27Da`.

- deposit's transaction id:
  `0x20fe44ae78011f9c4419ba5d4d3fbb7aed55e7dd0c0dbf0b9fee1962ef4e66fc`
- withdrawal's transaction id:
  `0x1f3f738a35fddc79d49a5d7f180c19e33a750cdbccc2ee070ccabb8a24a1010e`



```solidity
// SPDX-License-Identifier: GPL-3.0

pragma solidity >=0.7.0 <0.9.0; // version of Solidity


contract Bank { // name of the contract

    mapping(address => uint256) balance; // mapping used for addressing the owner's address to his balance
    address[] public customers; // array of customers (their addresses), type public

    event Deposit(address customer, string message); // declaration of event used in deposit function
    event Withdrawal(address customer); // declaration of event used in withdraw function


    function deposit(string memory message) public payable { // deposits value of the transaction sent by sender and takes one string parameter message
    // is type public which means it can be called by other contracts and payable, meaning it can collect ether
        require(msg.value > 10); // checks that the value of transaction was bigger then 10 ETH
        balance[msg.sender] += msg.value - 10; // stores address of the sender to the mapping balance and as value adds up value of transaction - 10 ETH

        customers.push(msg.sender); // adds address of the sender to the array customers

        emit Deposit(msg.sender, message); // stores the sender and message to the blockchain/emits the event
    }

    function withdraw() public { // withdraws all balance of the sender, also of type public
        uint256 b = balance[msg.sender]; // defines 256 bit integer variable b as the balance of sender (searches for his balance in the mapping balance)
        balance[msg.sender] = 0; // updates sender balance in array balance to 0
        payable(msg.sender).transfer(b); // transfers b amount of ETH to the sender's address

        emit Withdrawal(msg.sender); // stores the sender's address to the blockchain/emits the event
    }

    function getBalance() public view returns (uint256) { // allows user to find out how much balance he has by returning 256 bit integer presenting his balance
    // also public but of type view meaning it can not change state during the execution in EVM.
        return balance[msg.sender]; // returns sender's balance
    }

    function empty() public { // empties the balance of the contract by transfering it to the first address in array customers, also type public
        require(msg.sender == customers[0]); // checks that it is called from the right address (first addres in array customers)
        payable(msg.sender).transfer(address(this).balance); // transfers all of the balance of the contract to the sender
                                                             // (does that by using this which converts contract instance to object address)
    }
}
```