BDL Assignment 1

Tianjiao Wang s1912558

October 24, 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."

"It is infeasible to forge a proof of inclusion for a Merkle Tree that uses a collision resistant hash function." is true.

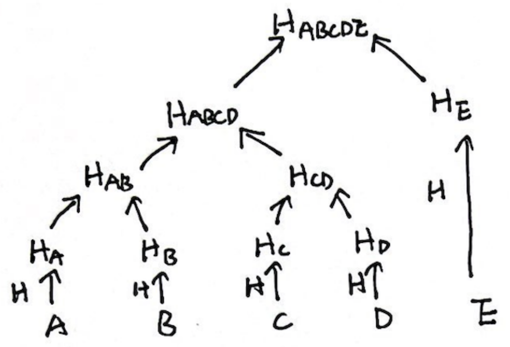
Collision resistance hash function means a hash function is collision resistant if it is hard to find two inputs that hash to the same output. To establish proof of inclusion of Merkle Tree, we need to hash a hash's corresponding hash together and climb up the tree until obtaining the root hash which is or can be publicly known.

For example in a Merkle Tree, there is node A and B, according to the feature collision resistant, we can't find another input A' that can hash to the same output as A (H(A) != H(A')), as the tree climbs up H(A'B) will be different from H(AB), same as the root hash, so the forge is infeasible.

(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.

1. b.

i). binary full Merkle Tree



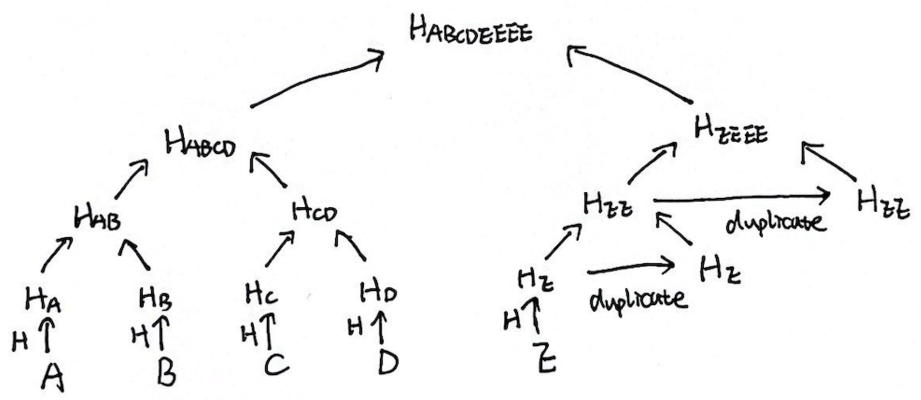
Proof of inclusion for Z:

Data Z get hashed and have H_Z

H_Z ~~hase~~ hashed with H_{ABCD} leads to H_{ABCDZ}, that is the publically available Merkle root.

We hence proven that data Z is indeed present in our merkle tree by making use of H_{ABCD} without having to reveal Z or any of the data.

ii). binary complete Merkle Tree



Proof of inclusion for Z:

Data Z get hashed ~~with~~ and have H_Z

H_Z duplicates itself and hashed with H_Z lead to H_{ZZ}

H_{ZZ} duplicates itself and hashed have H_{ZZZZ}

H_{ZZZZ} ~~hased~~ hashed with H_{ABCD} have $H_{ABCDZZZZ}$. that is the publically available Merkle ~~a~~ root.

We hence proven that data Z is indeed present in our Merkle Tree by making use of H_Z, H_{ZZ}, & H_{ABCD} without having to reveal Z or any of the data.

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 50% probability. 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 50%, at least two wallets end up with exactly the same seed (calculate both with and without replacement of sampled words).

2. Suppose there are n possible dates and k people.
The first ~~pe~~ person has n choices from n dates, to avoid collision, the next person has n-1 choices from n dates... and the kth person will have n-(k-1) choices from n dates.
Thus, the probability will be:

$$\Pr[\neg Col] = \frac{n}{n} \times \frac{n-1}{n} \times \frac{n-2}{n} \times \ldots \times \frac{n-(k-1)}{n}$$

$$= \prod_{l=1}^{k-1} (1 - \frac{l}{n})$$

~~Eq~~ According to Taylor expansion of e^x,

~~e~~ $e^x = 1 + x + \frac{x^2}{2!} + \ldots$

Ignore the high order term, when $|x| < 1$:

$$e^x \gtrsim 1 + x$$

Replace $-\frac{l}{n}$ with x:

$$e^{-\frac{l}{n}} \gtrsim 1 - \frac{l}{n}$$

$$\Pr[\neg Col] = \prod_{l=1}^{k-1} (1 - \frac{l}{n}) = \prod_{l=1}^{k-1} e^{-\frac{l}{n}}$$

$$= e^{-[\frac{1}{n} + \frac{2}{n} + \ldots + \frac{k-1}{n}]} = e^{-\frac{k(k-1)}{2n}}$$

$\Pr[\neg Col] \leq \frac{1}{2}$, so

$$\ln(\tfrac{1}{2}) \geq -\frac{k(k-1)}{2n}$$

$$-\ln(\tfrac{1}{2}) \leq \frac{k(k-1)}{2n}$$

$k^2 - k \gtrsim -\ln(\frac{1}{2}) \cdot 2n \quad \therefore k \gtrsim 1.177\sqrt{n}$

①. Random selection of 12 random words from list (2048 words) with replacement.

$n = 2048^{12}$

$k \gtrsim 1.177 \times 2048^{6}$

1.177×2048^{6} users are needed to end up with the same seed, with replacement.

②. Without replacement

$n = 2048 \times 2047 \times 2046 \times \cdots \times 2037 = \frac{2048!}{2036!}$

$\therefore\ k \gtrsim 1.177 \times \frac{2048!}{2036!}$

$1.177 \times \frac{2048!}{2036!}$ users are needed to end up with the same seed without replacement.

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

When the attacker changed the address from a to a', the transaction hash is changed and the Merkle hash will be different from block B. So the attacker must calculate a new Merkle hash based on the transaction with address a' and the block hash will be different, which would not satisfy the POW. So the attacker has to calculate a new block hash to prove his work.

In this case, if a majority of the hash rate is controlled by honest miners, the honest chain will grow the fastest. If the attacker wants to modify a past block, he/she would have to redo the POW of the block and all blocks after it to catch up with and surpass the work of the honest miners. As subsequent blocks are added on the honest chain, the probability of a slower attacker catching up will be decreased.

However, if the attacker controls most of the hash rate, creates a new block hash (based on the changes attacker made) that satisfies POW and builds the longest chain based on it, he/she might get the rewards.

Another case is the attacker could create their own private branch of the blockchain based on block B. The rest of the network continues to build on block B, while the attacker builds on top of this new chain. The attacker needs to remain at least one block ahead of the rest of the chain. Nodes accept the chain with the most accumulated proof of work as the valid blockchain. When the attacker reveals their chain if it is longer than the one followed by the rest of the network, the existing blocks will be discarded, and transactions reversed. The attacker could collect all the rewards from block B.

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

One scenario orphan block is generated is when two miners produce a block around the same point in time and they are all verified and valid. Because miners are constantly generating new blocks, some of these may be broadcasted to the network almost simultaneously. Since the network is distributed, the transmission of information between nodes takes some time. Miners would accept the valid block they first receive and try to obtain the next block based on this block. There is a possibility that a group of nodes will choose to validate one block, while another group will choose to validate the other. With continuous calculation, new blocks would be constantly created and there would exist one time when one blockchain would be longer than another one. According to the consensus, the longest chain would be regarded as the main chain and all the miners will work on this chain. The other block which used to be in the other chain will be discarded as the orphan block.

Another scenario orphan blocks may be generated is when one single entity or organization is able to obtain more than 50% of the hashing power and seek to produce blocks with double spends and reverse transactions by generating alternative desired blocks. Also called 51% attack. An attacker can use their majority share of the network hash rate to create their own version of the blockchain. The Bitcoin network will consider the longest chain as the main chain, a miner who owns 51% of the network is able to work on their own blockchain at a faster rate than everyone else and that miner could create their own chain of orphaned blocks to take control of the ledger.

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.

   Lamport One Time Signature is a method for constructing a digital signature based on a hash function. It is a one-time signature scheme because each private key signs only a single message.

   Key generation Imagine Alice wants to digital sign her message to Bob, she needs a private key and a public key. To create the private key, Alice uses a secure random number generator to generate 256 pairs of random numbers. Each number consists of 256 bits. To create the public key, Alice hashes the 256 pairs of random numbers' private key and derived 512 numbers each consisting of 256 bits, that is the public key. Alice securely stores the private key and shares her public with everyone.

   Signature issued Alice hashes the message she wants to send to Bob using a 256-bit cryptographic hash function (like SHA 256) to obtain a 256 bits digest. For every bit in the digest, Alice chooses the corresponding number from the 256 pairs of numbers of her private key. For example, if the bit value for the nth number is 0, she chooses the first number in the nth pair of numbers; if the bit value is 1 she chooses the second number in the nth pair of numbers. This will produce 256 bits of numbers, which is the signature. In this case, Alice won't use the private key again, she should delete the 256 pairs of random numbers that she used to create her signature.

   Signature verified After Alice sends her message and signature to Bob, Bob hashes the message using the same 256-bit cryptographic hash function that Alice used to obtain a 256-bit digest. For each bit in the digest, following the way Alice picks each number for her signature, Bob picks the corresponding number from the public key sent by Alice. That is as stated above if the nth bit of the message is 0 he picks the first hash in the nth pair and if it is 1 picks the second hash in the nth pair. Finally, Bob will have a sequence of 256 numbers. He hashes each number in Alice's signature to obtain a 256-bit digest, if the digest matches the sequence of 256 numbers that Bob picked from the public key, the signature is verified.

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.

   In the MetaMask homepage press Send, enter the address of the receiver. Put the amount of Ether (which is 1 as required) in the required place, keep Gas prices and Gas limit as default, press confirm. Check the details of the transaction in the confirmation page, transaction id is also shown in that page. Transaction id is 0x72c58d911d561523c12782e6bb38f7601c4e6fed250b67285ccfe2936a95d7d2 and address I'm using is 0xBE1A323bD614B7feca731aB1e3e8f9A14BE7612d.

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.

   Purpose of each variable

   (a) balance

   The balances variable is a HashMap which defined as mapping from address to unit256 (which datatype is integer), which stores the balance of customers each address has.

   (b) customers

   Datatype address, stores customer's address. Public means that the variable customers can be accessed by the contract and by other smart contracts.

   Purpose of each function

   (a) deposit

   When someone call the function, it deposit money sent to the balance but with 10 Wei less as transaction fee.

   The function deposit takes message, which datatype is string, as parameter. 'memory' is a keyword used to store data into EVM's memory for the execution of a contract. 'public' means everyone could use this function. 'payable' allows the function to send and receive ether and run code to account for this deposit. When deposit happens, 'message' allows sender to leave a message for the transaction.

   'require(msg.value > 10)' means the function only runs if the value sent is greater than 10. 'msg. sender' is the address of the person who is using the function; 'msg. value' is the number of Wei sent by this person. When requirement satisfied, update address of the sender and his/her balance with 10 less and record the sender as customer.

   At the end of the function, it emit the event Deposit with parameter of sender's address and message input.

   (b) withdraw

   When someone called the function, it withdraw all the money user deposited to user's balance.

   'public' means everyone could call this function, this is no parameter for this function.

   Local variable b type integer, is assigned to the sender's balance value and the balance value is set to 0. Convert the sender's address to payable and transfer b ether to the sender.

   Then it sets the balance of this account to 0. After that, it firstly uses a type conversion, to make the address of the message-sender payable, and transfer b Ether back to the sender.

   At the end of the function it emit the event Withdrawal with parameter of sender's address.

   (c) getBalance

   When someone called the function he/she could check their balance.

’public’ means everyone could call this function, this is no parameter for this function. ’view’ means the function only reads but doesn’t alter the state variables defined in the contract and there is no fees for calling this function outside the contract.

’returns (unit256)’ means the function will return a 256 bits integer. It uses sender’s address to get the balance and return it from the variable balance.

(d) empty

The first customer use this function to transfer all ether in the balance to his/her address.

’public’ means everyone could call this function, this is no parameter for this function.

’require(msg.sender==customers[0])’ means the function only allows the first customer in this contract to call it, other customers could not run this function.

’payable(msg.sender).transfer(address(this).balance)’ means the all ether in the balance of this contract will be transferred to the user’s address, which is the first customer.

Purpose of each event

(a) Deposit

This event will record the customer’s address and message from customer into transaction’s log.

(b) Withdrawal

This event will record the customer’s address into transaction’s log when withdraw happens.

Id of the transaction is 0x2ff05da6565e187746dc66134c726a55aa46da49482c1595b66bdcefd72ab121 and the address I’m using is 0xBE1A323bD614B7feca731aB1e3e8f9A14BE7612d.