Blockchain Assignment 2

Mayank Sharma, 160392

February 7 2019

1

Let us consider $H : P \times S \rightarrow \{0, 1...., 2^n - 1\}$ as a collision resistant hash function. Now, using a counter example, we will prove that any general collision resistant function of this form may not be proof of work secure. Let us construct a new hash function H' defined as follows,

$$H' : P \times S \rightarrow \{0, 1, 2..., 2^{n'} - 1\}, \qquad \text{where } n' = n + t$$

.
H' is a function whose target space (range) has a total of $2^{n'}$ possible values, hence the target space will be of n' bits.
The output of H' will have n' bits, so we can construct its output using the output of H and another t bits number. We choose that t bit number to have the value $(s \mod 2^t), s \in S$, where we take the binary representation of the string s.

$$H' = (S \mod 2^t) \mid H$$

We have assumed that H is collision resistant. Since H' is concatenation of 2 strings, one of them being H, we can easily see that H' will also be collision resistant.
If we take the value of $s = 0$, i.e. a string containing only null characters, then $(S \mod 2^t) = 0$, so effectively the target space of H' in such a setting reduces to 2^n values.

Hence we are able to find a solution (trivial) $s = 0 \in S$ such that

$$H'(p, s) < \frac{2^{n'}}{2^t}$$

. This shows that H' is not proof of work secure eve though it is collision resistant.

2 Beyond Binary Merkle Trees

2.1

Alice takes T_1, T_2 and T_3 and hashes them firstly so as to obtain $H(T_1), H(T_2)$ and $H(T_3)$, respectively. She does similar for the rest of the elements too, so as to obtain $H(T_i), i \in 1, 2, ..., 9$.

Now, she clumps $H(T_1), H(T_2)$ and $H(T_3)$ to form a new node which we call and then calculates a new hash from these three values which is $H(T_1 \mid T_2 \mid T_3)$.

Similarly, she gets $H(T_4 \mid T_5 \mid T_6)$ and $H(T_7 \mid T_8 \mid T_9)$, and she makes a node of these values. For the root node of k-ary Merkle Tree, she hashes the child nodes to get $H(H(T_1 \mid T_2 \mid T_3) \mid H(T_4 \mid T_5 \mid T_6) \mid H(T_7 \mid T_8 \mid T_9))$. This is how she commits to set S.

Suppose at a later stage Bob wants to check if T_4 is in S. So, she must have handed out a *verification token* to Bob which contained the values $H(T_1 \mid T_2 \mid T_3)$ and $H(T_5), H(T_6)$ and $H(T_7 \mid T_8 \mid T_9)$. Alice finds $H(T_4)$ from T_4, then finds $H(T_4 \mid T_5 \mid T_6)$, and then finally computes $H(H(T_1 \mid T_2 \mid T_3) \mid H(T_4 \mid T_5 \mid T_6) \mid H(T_7 \mid T_8 \mid T_9))$. If this value matches the earlier calculated hash value of root node of this merkle tree, it's confirmed that $T_4 \in S$.
This is because a root node of Merkle Tree indirectly stores the information about all the elements of set S in the form of Hash Values, and at max we perform $\lceil \log n \rceil$ hashes.

2.2

For finding the Merkle Tree Root node hash, we have to go all the way from leaf node to the root of tree performing hashes. For a k-ary tree, we have to go through the height of tree. Hence, the whole operation takes $O(\log_k(n))$ hashes which is the length of the proof.

2.3

For a k-ary tree, for verifying that some block is a part of the merkle tree, we need to perform a total of $(k-1)\log_k(n)$ computations.

Proof: Consider the parent (say A) of a leaf node (say B) of a k-ary tree. We'll be given the hash of the block below it by ther person who's claiming that the block (B) is in Merkle Tree, and then we have to take hashes of all the children of A.

So, in total, we'll have to perform (k-1) hashes (since 1 was to be given by the person trying to verify B). At each depth level, we'll have to perform (k-1) hashes to find the hash of the parent node. Hence, $(k-1)\log_k(n)$ computations will be needed.

$$= \frac{(k-1)\log_k(n)}{\log_2(n)}$$
$$= \frac{(k-1)}{\log_2(k)} > 1$$

The advantage of using a k-ary tree is that at the time of verification, we need to perform only $\log_k(n)$ hashes whereas for a binary tree, we'll have to perform $\log_2(n)$ hashes. It's a tradeoff between the initial time to perform computations vs the repetitive computations that might be needed in future.

3 Hiding vs Binding Components

3.1

Phone numbers are just 10 digits which corresponds to about 26 bits of **valid** phone numbers. Now, Bob can create a new phonebook which contains all the permutations of these valid phone numbers and she'll be able to know which all users are using BobCrypt app.

This way, he can know which all persons use the BobCrypt app and their contact numbers.

3.2

Suppose two different users Alice and Charlie have the same contact (say X) in their phonebooks. When the app will hash X's contact, it will generate a different (random) nonce (even for the same contact of X), and hence the finally generated hash values for the same contact number (X) will be different.

Hence, Bob's server won't be able to achieve the intended functionality of mapping a user's phone to a fixed hash because of this random nonce.

4 Bitcoin Script

4.1

ScriptSig :

< Password of Alice >

4.2

This method is not at all secure because scriptSig comes in the input field of every transaction, meaning that it is publicly visible to everyone. Anyone who knows which transaction (this is also public) Alice is using to store her password can easily see her password in plaintext.

4.3

Yes, a Pay-to-script-hash fixes the security issue although only temporarily. A P2SH Script is hidden from the sender as well as the Blockchain. Only the person who owns this 'redeemScript' and subsequently its hash can use this script to perform any operation.

So, here Alice can create a P2SH script for herself which saves her password in plaintext and she sends the hash of this script. She is the only one who owns this script so no worries of leaking password. When a sender processes such a P2SH transaction, he just pays to this script. But, for some other transaction, this script will act as SigScript and it will be revealed to the whole world when a redeemScript has been used for once. Alice will then have to change her password again.

5 Lightweight Clients

5.1

Alice should be sending all the block headers till the block which contains the last transaction which is put as the input of another transaction that she's trying to verify.

Moreover, Alice also needs to send the merkle root of the block which contains that input transaction. From Bob's perspective, he'll need to use the headers to verify that the transaction is indeed valid.

5.2

Number of transactions in a block,

$$n = 256$$

Number of blocks before head,

$$k = 8$$

Now, each block header is 80 bytes in length, and each SHA-256 hash is 32 bytes. The merkle proof for the last block which contains the transaction takes $\log_2(n)$ hashes.
Hence, total bytes Alice need to send

$$\begin{aligned}
&= 80 \times k + 32 \times \log_2(n) \\
&= 640 + 256 \\
&= 896 bytes
\end{aligned}$$

5.3

6 Bitcoin Lotto

6.1

To delay the redemption of lottery to Saturday, we can use locktime with an appropriate value. The value that locktime takes is that of a block height, so that once that blockheight has been exceeded, then only a miner can include this transaction in the next block.

6.2

To carry over the prize of week n over to week $n+1$, we can proceed as follows:

- When we are creating a new wallet with Prize Amount for week n, we will generate another public-private key pair for a new account for week $n+1$.
- At the time of printing the lotteries, we can create a new type of transaction which uses locktime, such that the sender address is the account of week n and the receiver is the week $n+1$ account.
- We put a locktime on this transaction such that the locktime corresponds to time 1-2 hr before when we are starting the lottery for week $n+1$.

Now, when private key of week n account gets lost, all the funds from week n account will automatically get transferred to week $n+1$ account just before the new week's lottery begins. This ensures that someone else doesn't have the chance to reclaim the previous week's (n) and this week's $(n+1)$ lottery both at the same time.