Engineering Blockchain and Web3 Apps Problem Set #1
Coulson Zhang — zz2980

Problem 1. ***A broken proof-of-work hash function.***

Solution. According to the definition of a secure scheme, there is negligible probability for an adversary to figure out a secret key. However, based on the given hash function, $H(x, y) = SHA256(x \oplus y)$, and the assumption that D is fixed ahead of time, a clever hacker may start solving the problem by finding the result of $x \oplus y$ before x even released. Supposed this hacker finds a key z, such that $H(z) < 2^n/D$, then once $x \in X$ is published, this hacker can find the desired solution $y \in Y$ by taking operation $z \oplus x = y$.

Therefore, once D is initialized, the hacker only needs to find one desired value z that satisfies the criteria and then this hacker can find solution $y \in Y$ with a time of one-bit operation once $x \in X$ is released. □

Problem 2. ***Beyond binary Merkle trees.***

Solution.

a) In the first level, Alice could make 9 leave nodes $node_{1,1}, node_{1,2} \ldots node_{1,9}$ whose value is the hash of the corresponding elements in S, such that $node_{1,1} = hash(m_1)$. In the second level, each node is a hash of the concatenation of its three children, such that $node_{2,1} = hash(node_{1,1}, node_{1,2}, node_{1,3})$, etc. Then in the third level, the binding commitment, $node_{3,1}$, of S of hash value of concatenation of all three nodes in second value which is $hash(hash(node_{2,1}, node_{2,2}, node_{2,3})$

   If Alica needs to prove that m_4 in S, then the value $m_5, m_6, node_{2,1}, node_{2,3}$ should be provided in the proof so that the verifier can re-calculate the commitment,$node_{3,1}$, of S using ternary Merkle tree when $k = 3$.

b) Start from the bottom level, we need $k - 1$ sibling values to calculate the next level value. Since there are n elements in S by the given assumption, there are $\lceil log_k n \rceil$ levels in Merkle tree. Therefore, the length of proof that proves $S[i] = m_i$ is $(k-1) * \lceil log_k n \rceil$

c) By the formula concluded in previous question, the length of proof of binary tree will be $(2 - 1) * \lceil log_2 n \rceil = \lceil log_2 n \rceil$ and the length of proof of ternary tree will be $(3-1) * \lceil log_3 n \rceil = 2 * \lceil log_3 n \rceil$. Seeing from those two formulas, it's safe to see that the growth rate of a ternary tree will be slower than a binary tree, but the coefficient of 2 makes the length of proof of a ternary tree larger than the length of proof of a binary tree if n is not significantly large. Therefore, to minimize the proof size, it's better to use a binary tree unless n is significantly large.

□

Problem 3. ***Bitcoin script.***

Solution.

a) The Following solution assumes that $1.ScriptSig$ will executed first and then $ScriptPubKey$ is executed. And $0xeb271cbcc2340d0b0e6212903e29f22e578ff69b$ is the hash of password P already.
$ScriptSig$ is just P. Then the full combined script will be

$$P\ OP\_SHA256\ < 0xeb271cbcc2340d0b0e6212903e29f22e578ff69b >\ OP\_EQUAL$$

Therefore, the password P will be pushed into the stack, hashed, and compared with the given hash of the password. Then the whole statement will return True if two hash values are matched and return False otherwise.

b) Since the password is only six characters, the scope of possible combinations is not computationally infeasible to prevent a brute force attack under current modern computational power. A hack may try all combinations to figure out Alice's password by hashing each combination and checking whose hashing value is the same as that in ScriptPubKey. Once the hacker figures out the password, then this hacker can redeem this UTXO without authorization from Alice, in other words, steal the bitcoint from Alice.
For example, let's assume that one laptop can perform x operations per second and it has only one processor. There are y valid characters at each position and it takes z operations to calculate the HASH256 function and compare the result. Then it takes $y^6 * z/x$ seconds to iterate all possible passwords and find the password for sure. Note that we can easily decrease this value by adding another processor or another more powerful computer. Therefore, once Alice posts her UTXO, a hacker can find out the password and steal her bitcoin under a reasonable time.

c) The $SriptPubKey$ fails to provide a secure way to protect Alice's bitcoins even if Alice sets up a strong 30-character passphrase P as the safety flaw remains the same. There are a few vulnerabilities within the $SriptPubKey$.
1. The password will be revealed if Alice wants to redeem this UTXO, then if Alice uses this $SriptPubKey$ again in the future, anyone can redeem the UTXO since the password is broadcast earlier.
2. If anyone somehow guesses the password or figures it out by brute force, then there is no way to prevent them from redeeming those bitcoins from Alice. The longer this UTXO is posted to the blockchain, the higher the risk of being stolen.

□

Problem 4. ***BitcoinLotto.***

Solution.

a) We can use a 2-out-of-2 multisig to achieve this design. To claim the jackpot, the winner has to use the unique private key printed on the ticket and a weekly updated private key from the authorization of Bitcoinia. Bitcoinia will not release their weekly updated private key until the end of the week. Therefore, no one can claim the jackpot before the end of the current week.

b) To prevent the unretrievable bitcoin due to the lost or destroyed ticket(s), a trusted, neutral judge is introduced into the design to achieve the rolling over mechanism. Instead of a 2-out-of-2 multisig, now we can use a 2-out-of-3 multisig design. The first possible private key is printed in the physical ticket, the second possible private key is released by Bitcoinia at the end of the week, and the third possible private key can only be provided by the judge to roll over the unclaimed jackpot at the end of the week.
Case 1: The winner has the private key, and winner can claim the jackpot with the private key released by Bitcoinia at the end of the week. No judge is involved in this case.
Case 2: The Winner lost/destroyed the ticket along with the unrevealed private key. Then judge offers the third possible key, along with the private key released by Bitcoinia to next week's jackpot.
As a result, any unclaimed jackpot from the week n can be claimed by the winner in week $n + 1$ and Bitcoinia can't embezzle funds on its own. All original features are reserved and unclaimed jackpots at week n can be claimed by the winner at week $n+1$.

□

Problem 5. ***Lightweight clients***

Solution.

a) Let's assume that Alice's payment is located in the block that Bob's lightweight Bitcoin client stores. Alice should provide the transaction of her payment, Merkle Proof that contains necessary hashes of the path used to compute the Merkle root. With all the given information, Bob can calculate the Merkle root value. With all the given information, Bob can calculate the Merkle root value and check whether it's the same as the one stored in the Blocker header of the only block Bob stores. If they are the same, then it's proved that Alice's transaction is included in the blockchain.

b) Let's assume that the transaction itself is 10 bytes, the block header is 80 bytes, and each hash value within Mergle proof is 32 bytes. Also, Bob can only download one previous block based on the metadata with the blockhead of the current block by looking at the *prev* data within the metadata. First, in order to retrieve the k blocks before the current head, Bob needs to download $k * 80$ bytes. Then, Bob needs to download the transaction itself, 10 bytes, and Merfle proof which contains $log_2(n)$ hash value. The later part is in total, $10 + 32 * log_2(n)$.
As a result, Bob needs to download $80k + 32log_2(n) + 10$ bytes to check whether Alice's payment is included in the blockchain or not.

c) This design can reduce the proof size since it provides a shortcut to fetch the k^{th} blocks before the current head, instead of downloading all k blocks before the current head.

It's very beneficial for lightweight clients with limited memory and bandwidth.
The worst case is to download all block headers without being able to use the extra field. For example, if we are current at head 8 and want to check transactions at head 6, we have to download node 7 and node 6. Therefore, the worst case of proof is $80k + 32log_2(n) + 10$.

□