COMP0143 Coursework

Due: 15th February 2019 by 4pm

Instructions

Answer all questions. When you are done, convert the file to PDF (or otherwise ensure it is in this format), using your candidate number and the module code as the name (e.g., `KTJA9_COMP0143.pdf`).

You are allowed to discuss the questions at a high level with anyone, but you may not discuss specific answers. If your answer for a question is identical (modulo cosmetic tweaks) to an answer we find on the Internet or to the answer of another student, we will give no marks.

1 Types of clients [10 marks]

Which of the following types of Bitcoin clients exist? For the ones that do exist, describe in one or two sentences how they operate and what their properties are. Which one is the most secure?

- Simplified Payment Verification (SPV)
- Supernodes
- Full nodes
- Miners

2 Protocol decisions [20 marks]

For each of the questions below, keep your answer as brief as possible; i.e., limit yourself to at most a few sentences.

2.1 Gas [5 marks]

What is the purpose of gas in Ethereum? What could go wrong if gas were free?

2.2 Proof-of-work [5 marks]

Why is the difficulty of the proof-of-work in Bitcoin set to ten minutes? What would go wrong if it was changed to 12 seconds?

2.3 Selfish mining [5 marks]

One early proposal to mitigate temporary block withholding ("selfish mining") was to have Bitcoin nodes choose between competing blocks by mining on top of whichever block has the lowest hash value. Why is this not a good solution?

2.4 Creation [5 marks]

When the genesis block of Zcash, an alternative cryptocurrency, was created, the block header included the hash of a Bitcoin block mined on the same day. What were the Zcash creators trying to prove by including this value?

3 Miscellany [20 marks]

3.1 The cryptographic nature of transactions [5 marks]

Briefly explain why it is computationally infeasible for anyone to generate a Bitcoin transaction that references its own output as an input.

3.2 Solidity [5 marks]

Consider the following Solidity function contained in an Ethereum contract:

```
function foo(uint256 x, uint256 y) {
    while (uint256(sha3(x)) != y)
        x++;
}
```

Briefly explain how a malicious miner could waste the resources of other miners by using a specific choice of `x` and `y`, without spending any resources of its own.

3.3 Lottery security [5 marks]

Suppose that you want to generate the draw number for a lottery at UCL as $H(\mathsf{id}\|B_i)$, where id is each student's ID number and B_i is the i-th block header in the Bitcoin blockchain. To make the scheme more secure, your friend suggests using $H(\mathsf{id}\|B_{i-1}\|B_i)$. What extra security does this provide, in terms of protecting the lottery from manipulation by miners?

3.4 Eclipse attacks [5 marks]

If a malicious ISP completely controls a user's connections, can it launch a double-spend attack against the user? How much computational effort would be involved?

4 Beyond binary Merkle trees [10 marks]

In Week 2 we saw binary Merkle trees, which allow a user Alice to form a binding commitment to a set of elements $S = \{T_1, \ldots, T_n\}$ consisting of a single hash value (the root of the tree). Furthermore, she can later prove to Bob that some T_i is in S using an inclusion proof containing at most $\lceil \log_2(n) \rceil$ hash values (the path to the root). In this question, we explore how to do the same thing using a k-ary tree, where every internal node has up to k children rather than just two. The hash value for each of these nodes is computed as the hash of the concatenation of the values of all of its children.

1. Suppose $S = \{T_1, \ldots, T_9\}$. Explain how Alice computes a commitment to S using a ternary Merkle tree ($k = 3$). What does Alice later reveal to prove to Bob that T_4 is in S? **[2 marks]**

2. More generally, suppose S contains n elements. What is the length of the proof that some T_i is in S, as a function of n and k? **[3 marks]**

3. For large n, is it better to use a binary or ternary tree if our goal is to minimise the proof size? Briefly justify your answer. **[2 marks]**

4. Why do Bitcoin blocks contain a Merkle tree of transactions rather than a flat list? **[1 marks]**

permet d'avoir des SPV qui ont juste besoin du header (merkle root) et surtout much faster access

5 Very lightweight clients [10 marks]

Suppose Bob runs a very lightweight client which, in contrast to regular SPV clients, stores the header of only the most recent block in the Bitcoin blockchain.

1. If Alice wants to send money to Bob, what information must she include to prove to him that her payment has been included in the blockchain? **[3 marks]**

headers k block avant + info pour retrouver merkle root

2. If Alice's payment was included in a block k blocks before the current header and there are n transactions per block, how many bytes will the proof require in terms of k and n? Compute the concrete proof size for $k = 8$ and $n = 256$. **[3 marks]**

headers k block avant + info pour retrouver merkle root

3. One proposal is to add an extra field into block headers that points to the last block with a *smaller* hash value than the current one. Explain how this can be used to reduce the proof size. What is the expected size (in bytes, and in terms of its big-O complexity) of a proof in terms of k and n? What are the best- and worst-case sizes? [**4 marks**]