COMP9088: Cryptocurrencies and decentralized ledgers Semester 1 2022

Homework 1

due: Thursday 2022-03-17 22:00 AEDT

1. **Hash functions and signatures**: Signature schemes are typically defined to work for a finite domain of possible message (the message spaces). Suppose you have a signature scheme *S* which is correct and secure (existentially unforgeable) for domain $M=\{0,1\}^t$, that is, *S* can be used to sign any *t*-bit message. Suppose you also have a hash function *H* which outputs *t* bits and is collision-resistant.

   Consider a modified signature scheme *S'* which can sign messages of unlimited length, where:

   $$S'.\mathrm{Sign}(sk, m) = S.\mathrm{sign}(sk, H(m))$$

   Prove that this scheme is existentially unforgeable as long as *S* is existentially unforgeable and *H* is collision-resistant.

2. **Beyond binary Merkle trees:** Alice can use a binary Merkle tree to commit to a set of elements S = $\{T_1, \ldots, T_n\}$ so that later she can prove to Bob that some T_i is in S using a proof containing at most $\lceil \log_2 n \rceil$ hash values. The binding commitment to S is a single hash value. In this question your goal is to explain how to do the same using a *k-ary* tree, that is, where every non-leaf node has up to *k* children. The hash value for every non-leaf node is computed as the hash of the concatenation of the values of its children.
   a. Suppose S = $\{T_1, \ldots, T_9\}$. Explain how Alice computes a commitment to S using a ternary Merkle tree (i.e. *k*=3). How can Alice prove that T_4 is in S?
   b. Suppose S contains *n* elements. What is the length of the proof that proves that some T_i is in S, as a function of *n* and *k*?
   c. For large *n*, what is the proof size overhead of a *k*-ary tree compared to a binary tree? Can you think of any advantage to using a *k*>2? (Hint: consider computation cost)

3. **Bitcoin script**: Alice is on a kayaking trip and is worried that her phone (which contains her private key) might fall overboard. She would like to store her bitcoins in such a way that they can be spent with knowledge of a password. Accordingly, she stores them in the following ScriptPubKey address:

   ```
   OP_SHA1
   <0x13818a5684a7ed4dce8433c3f57e13b589b88852>
   OP_EQUALVERIFY
   ```

   a. Write a ScriptSig script that will successfully redeem this transaction. [Hint: it should only be one line long.]
   b. Explain why this is not a secure way to protect bitcoins using a password.

   c. Would implementing this using pay-to-script-hash (P2SH) fix the security issue(s) you identified? Why or why not?

4. **Lightweight clients**: Suppose Bob runs an ultra lightweight client which receives the current head of the blockchain from a trusted source. This client has very limited memory and so it only permanently stores the most recent blockchain header (deleting any previous headers).
   a. If Alice wants to send a payment to Bob, what information should she include to prove that her payment to Bob has been included in the blockchain?
   b. Assume Alice's payment was included in a block *k* blocks before the current head and there are *n* transactions per block. Estimate how many bytes this proof will require in terms of *n* and *k* and compute the proof size for *k*=8, *n*=256.
   c. One proposal is to add an extra field in each block header pointing to the last block which has a *smaller* hash value than the current block. Explain how this can be used to reduce the proof size from part (b). What is the expected size of a proof (in bytes) now in terms of *n* and *k*? To simplify your analysis, you may use asymptotic (Big O) notation. What are the best-case and worst-case sizes?

5. BitcoinLotto: Suppose the nation of Bitcoinia has decided to convert its national lottery to use Bitcoin. A trusted scratch-off ticket printing factory exists and will not keep records of any values printed. Bitcoinia proposes a simple design: a weekly run of tickets is printed with an address holding the jackpot on each ticket. This allows everybody to verify that the jackpot exists. The winning ticket contains the correct private key under the scratch material. There are no additional trusted parties (or judges) in the protocol.
   a. If the winner finds the ticket on Monday and immediately claims the jackpot, this will be bad for sales because players will all realize the lottery has been won. Modify your design to prevent this (of course, you can't prevent the winner from proving ownership of the correct private key outside of Bitcoin).
   b. Some tickets inevitably get lost or destroyed. So you'd like to modify the design to roll forward any unclaimed jackpot from Week *n* to the winner in Week *n+1*. Can you propose a design that works, without enabling the lottery administrators embezzle funds? Also, you want to make sure that the Week *n* winner can't wait until the beginning of Week *n+1* in an attempt to double their winnings.

1.Since we only want the winnings claimed by the end if the week, we would have to make sure the keys on the tickets are not activated until the end of the week, also verification of winnings occur by end of the week. This way someone cannot immediately unlock their bitcoin on a Monday but may just scratch the ticket.

2. Bitcoin wallets cannot be unlocked without the private key and so there is no other option when ticket is lost as bitcoin cannot be retrieved. Therefore it is important that tickets are secured properly during production and after when issued to contestants