The Theory Behind Blockchains (Spring 19)
Problem Set 1

Submission: By 15:00 on March 27th, 2019. To be graded by Nathan Geier.

Instructions: You can collaborate with each other and consult external resources. However, you should write the solution on your own, and for each question, list all collaborators and external resources. Otherwise, write explicitly "None". Do not discuss solutions in the course forums; you are welcome to ask for clarifications if needed.

1 Collision Resistance Hash (30 points)

In this exercise we show how to construct Collision Resistance Hash (CRH) families from the Discrete Log assumption. For $n \in \mathbb{N}$, let $p = p(n)$ be an n-bit prime and let $g = g(n)$ be a generator of $\mathbb{Z}_p^* = \{1, \ldots, p-1\}$ (Namely, for every $y \in \mathbb{Z}_p^*$ there exists $x \in \mathbb{Z}_p^*$ such that $g^x = y \bmod p$). The Discrete Log assumption states that:

$$\Pr_{x \leftarrow \mathbb{Z}_p^*}[A(p, g, g^x) = x] \leq \text{negl}(n)$$

For $y \in \mathbb{Z}_p^*$, let $h_{p,g,y} \colon \mathbb{Z}_p^* \times \{0,1\} \mapsto \mathbb{Z}_p^*$ be the following function: $h_{p,g,y}(x, b) = y^b g^x \bmod p$ (i.e., a one-bit compressing function). We now define $\mathcal{H} = \{\mathcal{H}_n\}$ where $\mathcal{H}_n = \{h_{p,g,y}\}_{y \in \mathbb{Z}_p^*}$ (i.e., sampling $h \leftarrow \mathcal{H}_n$ means to sample $y \leftarrow \mathbb{Z}_p^*$ and output $h_{p,g,y}$).

a. **(7 points)** Prove that for any $x, x' \in \mathbb{Z}_p^*$ it holds that $g^x \bmod p = g^{x'} \bmod p \iff x = x'$ (use the fact that g is a generator of $\mathbb{Z}_p^*$).

b. **(8 points)** Prove that the distribution of $g^x \bmod p$ when sampling uniformly $x \leftarrow \mathbb{Z}_p^*$ equals to the uniform distribution over $\mathbb{Z}_p^*$.

c. **(15 points)** Prove that $\mathcal{H}$ is a CRH family. (Show that any PPT algorithm A that is able to find a collision for $h_{p,g,y} \leftarrow \mathcal{H}_n$ with non-negligible probability, can be converted into an PPT algorithm A' that can break the Discrete Log assumption.)

2 Merkle Trees (35 points)

In Recitation 1 we showed how to construct a (static) Merkle Tree (MT) from a given set of data blocks. In this exercise we show how to construct a dynamic MT that also supports efficient insertions and deletions.

Assume that we take a data structure that implements a balanced binary search tree over data-blocks such that if it contains m nodes then its depth is $O(\log m)$ and insertion, deletion and searching takes $O(\log m)$ steps in the worst-case (e.g., AVL tree or Red-Black tree using some ordering function over data-blocks). Then we transform it into a dynamic MT by transforming its pointers into hash pointers using a hash function h. Namely, each node u contains two fields: the original data u.data and a hash pointer u.hash such that:

1. If u is a leaf, then u.hash $=\perp$.
2. If u has one child v, then u.hash $= h(v.\text{data}, v.\text{hash})$.
3. If u has two children v_1, v_2, then u.hash $= h(v_1.\text{data}, v_1.\text{hash}, v_2.\text{data}, v_2.\text{hash})$.

The publicly shared data that represents the tree is the hash of the root: $h(\text{root.data}, \text{root.hash})$.

Assume that computing h on any given input takes $O(1)$ time steps. Show how to implement Insertion, Deletion, Searching and Proof of Membership in a way that preserves the properties of the above dynamic MT (using the properties of the original balanced binary tree) such that:

- Searching for a data-block takes time: $O(\log m)$.
- Inserting a new data-block or deleting an existing one takes time: $O(\log^2 m)$.
- Proof of Membership size: $O(\log m)$.

In addition, show that given a fake proof of membership, it is possible to efficiently find a collision in h.

In your solution you can assume a specific implementation of the balanced binary search tree, like AVL-tree, but for the purpose of this exercises it should be suffice to only use the assumption that each operation in the original tree takes $O(\log m)$ steps, where the only steps that can change the tree are: (1) Creating a new node and adding a pointer to it, (2) Deleting an existing node and removing the pointer to it, and (3) Moving a pointer from one node to another.

3 Signature Schemes (35 points)

a. **(15 points)** The One-Time Signatures (OTS) we saw was such that keys and signatures grow with the length of messages. Suggest a way, using collision resistant hashing to circumvent this. Argue why your suggested scheme is secure (no need for formal reduction).

b. **(20 points)** Using the scheme from 3.a, we would like to construct a new scheme for messages of arbitrary length that is stateful. Specifically, each signer has a state that is updated after every time a message is signed. The state at time t (having already signed t messages) consists of two parts:

   – A private signing key SK_t.
   – A public verification key PK_t.

   We further require that the size of SK_t is independent of t, whereas the size of PK_t is allowed to grow with t. Describe how to generate the keys (SK_t, PK_t) at each time t and how to use them to sign and verify. Hint: The initial keys (SK_0, PK_0) are a pair of OTS keys (sk_0, pk_0). When signing the first message m_1, first sample a new pair of OTS keys (sk_1, pk_1), and use sk_0 to generate a signature σ_1 on the message (m_1, pk_1). Update $PK_1 = PK_0, (m_1, pk_1, \sigma_1)$.

c. **(Bonus: 10 points)** Suggest how to use the scheme from 3.b and a random oracle in order to create a standard (stateless) scheme. No need for analysis.