Homework 5

1 Problem 1 (16 points)

In this problem, we will explore how to pad messages in order to make Merkle-Damgard collision resistant, even if the messages are different lengths. Let $h : \{0,1\}^{512} \to \{0,1\}^{256}$ be a compression function. We will apply the following padding:

- First, pad the message to a multiple of 256 bits by appending a string of the form $100\ldots00$.
- Suppose after the previous step, the message is of length $256 \times \ell$. Next, write down ℓ as a 256-bit number, and append ℓ to the message

For example, a string x of length 500 will get padded to $x10^{11}0^{254}10$, where 10^{11} is the padding from the first step, and $0^{254}10$ is the representation of $\ell = 2$ as a 256-bit number.

The actual hash function H applies the padding above to get a padded string whose length is a multiple of 256 bits, and then applies the Merkle-Damgard hash function.

(a) Show that any collision for H can be turned into a collision for h, even if the original collision consisted of messages of different lengths.

(b) What happens if we replace the first step with a padding of just 0s instead of $100\ldots00$?

(c) Suppose we only apply the first step (appending $100\ldots00$), but do not append the length ℓ before applying Merkle-Damgard. Show that the collision resistance of h is *not* enough to demonstrate the security of H. That is, show that it is possible for h to be collision resistant, but nevertheless (for a bad choice of IV) it is possible to find collisions in H.

(d) What happens if we put the length ℓ at the *beginning*, instead of at the end?

2 Problem 2 (12 points)

One problem with Merkle-Damgard is that it is sequential, and cannot take advantage of parallelism to speed up the computation.

An alternative hash function is described as follows.

Let $h : \{0,1\}^{512} \to \{0,1\}^{256}$ be a compression function. Define $H_i : \{0,1\}^{256 \times 2^i} \to \{0,1\}^{256}$ as follows:

- $H_1 = h$
- H_{i+1} does the following. On input a $256 \times 2^{i+1}$-bit message x, write x as $x_0 x_1 \ldots x_{2^i - 1}$, where each x_j is a 512-bit string. Let $y_j = h(x_j)$ for all j, and let $y = y_0 y_1 \ldots y_{2^i - 1}$. Output $H_i(y)$ as the hash.

In other words, H_i is defined by a binary tree containing 2^i leaves, which contain the various 256-bit blocks of x. The value at each node is set to be the hash under h of the concatenation of its two children. The output is the value of the root.

Given sufficiently many processors, one can compute H_i in about i time steps.

(a) Show how, given a collision for H_i, you can construct a collision for h

(b) The above hash functions only work for messages consisting of an number of blocks that is a power of 2. Explain how to make the hash function work for arbitrary length messages. Your hash function should use essentially the same number of hashes as Merkle-Damgard would use on the same message (it can be an additive logarithm more, but shouldn't be double).

3 Problem 3 (10 points)

Here, we will use the hash functions H_i from problem 2 to solve the following problem.

Suppose you have outsourced the storage of some extremely large database x (which for our purposes we will think of as an enormous bit-string) to some third party storage provider (say Amazon or Dropbox). Unfortunately, you do not trust the provider to maintain your database correctly, so you have kept a hash of the database to yourself.

When you are accessing some portion of the database (say, a 256-bit block), you would like the provider to give you some sort of proof that this block was the block you originally stored. Of course, they could send you the entire database, which you would hash and compare to the hash you've stored. If the hashes are different, you know something is wrong with the database; on the other hand, if the hashes are the

same, you will be convinced that the database was unchanged. This check is of course extremely expensive, and you don't want to download the entire database every time you want to access some small portion of it. You would instead like some way of just checking that the block you are accessing.

(a) Suppose your database has size 256×2^i, and you stored the hash $H_i(x)$. Now, you ask the database for block j; show that the provider can send you x_j along with a "proof" that x_j is what you originally stored. This proof should have size $O(i)$ (namely, logarithmic in the overall database size). Prove that if provider sends you a different x'_j along with a valid proof for x'_j, then the provider can find a collision for the underlying compression function h. In other words, the collision-resistance of h guarantees that the provider can only give you the correct x_j.

[Hint: The tree structure discussed in problem 2 will be useful here. Your proof will contain the values of certain nodes in the tree]

(b) Sometimes you will want to update your database. As with reading the database, you will typically only update a single 256-bit blocks at any time. You would like to simultaneously update your hash of the database, but without downloading the entire database.

Show how the provider can provide you with the new hash instead, along with a proof that the new hash is correct. Again, your proof should have size $O(i)$.

4 Problem 4 (12 points)

Explain how to efficiently find collisions in the following hash functions:

(a) $H_a : \{0,1\}^{512} \to \{0,1\}^{256}$ is defined as follows. Let F, F^{-1} be a secure block cipher with block size and key length 256. $H_a(x, y) = F(y, x \oplus y) \oplus y$ That is, y is interpreted as a key for F, and to compute H_a, we first XOR y with x, apply the block cipher to the result, and then XOR with y one more time

(b) $H_b(x, y) = F(y \oplus x, x)$, where F, F^{-1} is as in part (a)

(c) H_c is a sponge function where the message block size is equal to the internal state size, so that when we XOR in a block of the message, it affects the entire state (so in other words, in the lecture slides the blue boxes represent the entire state and the orange boxes are non-existant). Here, the round function f is an un-keyed SPN network.

(d) $H_d : \{0,1\}^{257} \to \{0,1\}^{256}$ is defined as follows. Let $H : \{0,1\}^* \to \{0,1\}^{256}$ be a collision-resistant hash function for arbitrary-length messages, and let $H_d(x, b) = H(x)$ if $b = 0$ and $H(H(x))$ if $b = 1$.