David Magrefty September 30, 2022

6.1600 Problem Set 1

Collaborators: *none*

Problem 1

(a) The function h' is not a one way function. We will prove that by example. Given $h'(x) = 0||\{0,1\}^n$ an adversary will be able to easily reveal x since it knows that if the solution starts with (ie most significant bit is 0), the key -x- is just the everything that comes after the most significant bit.

(b) We will prove h'(x) is collision resistant by cases:
Case 1:
$x \in \{0,1\}^n$: Only if $x_1 = x_2$ then $h'(x_1) = h'(x_2)$ since from the construction $h'(x) = 0||\{0,1\}^n$ **thus only equal x's will have the same solution.**
Case 2:
$x \notin \{0,1\}^n$: h(x), guarantees given us a collision resistant solution to the hash, **thus the solution given to us will be collision resistant.**

Now since for each case, we concat a different bit at the beginning, even if h(x) returns us an n bit string that equals to some $x \in \{0,1\}^n$ we have previously inserted, the concatenated bit guarantees that we will have different solutions.
Exploring the two cases proves h'(x) is collision resistant.

Proof by Reduction
Suppose there exists an efficient adversary A that breaks the collision resistance property of h'(x). We will use A to construct an efficient adversary B that breaks the collision resistant property of h(x):

B will choose a random string x in the domain $\{0,1\}^{n+1}$, compute h'(x) and will ask A to invert h'(x) getting back x' such that h'(x')=h'(x). Now since we chose superficially x as an n + 1 length string, h'(x) will have to use h(x) to compute the result. Now since A was able to find a collision, it basically means it found a collision of h(x), contradicting the assumption that h(x) is collision resistance which is false.

(c) This statement is a stronger statement than the statement in part b, thus since part b is false, this is also false. This is true since statement b allows the adversary find any pair, while statement c requires the adversary to find a specific pair. Regardless, we will still prove h is target collision resistant.
Assumming h(x) is target collision resistant, we will prove h'(x) is also TCR by cases:
Case 1:

$h'(x) = 0||\{0,1\}^n \rightarrow x = x'$ IFF the resulted hash starts with 0, the only collision that can happen is if x=x'.
Case 2:
$h'(x) = 1||\{0,1\}^n = 1||h(x) \rightarrow$ the adversary found a collision in h(x). But this contradicts our initial assumption that h(x) is TCR, thus it is false.

Problem 2

(a) Since A now has the two different answers Bob can supply, he basically can do replay attack. In the attack the Adversary will send the answer he wants back to Alice. Alice will not have an option to know if this is Bob's current answer or previous answer from the day before. In order to fix that, there are few options. One can be that Bob must include the question he is answering in the message he is sending back to Alice. This solution will not work if Alice asks the same question every day. To fix that, we can basically add a unique index to every question Alice asks and Bob will have to include that in his message back to Alice. Basically every time Alice will ask a question she will add one to her question counter, concat the counter to the msg, MAC the msg and Bob will do the same with his answer. Unpacking the message Alice will be able to see which question Bob is answering.

(b) It is not a secure MAC. Since XOR is associative, the server will be able to do mix and match, and create new messages. Basically cut the original message to 128 bit segments, shuffle the segments and send that as a new message. The MAC will be the same due the associative nature of XOR.

(c) The rule is that for any padded message we should have unique unpadded message. Imagine a case where we have a message that is shorter than 128 bits and it ends with zero. The MAC of that message with zero padding will be equal to a MAC of a message without that zeroth bit with padding. The two messages are complete different but they will have the same MAC. An attacker can use that knowledge by basically finding a message that ends with one or more zeros, remove those zeros and use the same MAC for the newly constructed message without the zeros.
$m_1 = 100101101010000000 \rightarrow pad(m_1) = 100101101010000000....00000$
$m_2 = 10010110101 \rightarrow pad(m_2) = 100101101010000000....00000$
$m_1 \neq m_2$ but $pad(m_1) = pad(m_2) \rightarrow MAC(m_1) = MAC(m_2)$

A solution for this will be: $MSG : m, len(m) = n < 128 \rightarrow pad(m) = m||1||\{0\}^{128-n-1}$ IFF $len(m) = 128 \rightarrow pad(m) = m$ In addition add a dummy block to the message with length 128 which holds the number of bits added for padding. Thus, the dummy block will be mostly 0's with value holding $128 - len(m)$

Problem 3

(a) This property is not optional since we don't have any guarantee that the server which is distributing the data to our devices will stay loyal and not either send us fake data or will only send us some data and not data uploaded from all devices.

(b) If the server has unbounded computational power it will be able to compute the secret key used for authentication and then it will start adding logs apparently signed by a user that holds a secret key.

(c) Since the photos are just a string with 0 and 1 values, the server will be able to basically take the entire concatenated string, cut in different places, and create as many segments as the number of photos. Then the server will connect each segment it created to a log and it will basically be a new set of photos but concatenated together it will be the same as all the original photos and client won't be able to know.

(d) We basically need to find a way where my local history is broadcast and all devices are able to verify that they have completely received my local history and vice versa.
A possible fix is to implement a solution like blockchain where we treat every image added, log added, as a block. Each block contains a pointer, i.e. it is signed with the hash of the previous block. Then me as a user that receives a bunch of logs, every log I unpack contains the signature uploaded with it, I construct the block and check that the hash of the block plus the hash of the last block I have in my history match the signature added to the constructed block. If so, I know that my I have the correct chain, and I add the new block to my history. If not, I know there is some block of data I have missed or didn't receive and raise an exception.
In this scheme, the signature scheme is:
$sign(block_x) = sign(hash(block_x))$ where $block_x$ contains that signature of $block_{x-1}$ and the genesis block, i.e. block 0 can just be the empty block value.

(e) **This is part e and f**:
In the above blockchain scheme, our runtime is based only and number of images the devices uploaded and are synced each time. When I upload a picture, I hash and sign at each point I upload the picture which takes O(U) where U is the number of images I have uploaded. When I sync, I check each log before I add it to my chain (history) or continue to the next one. Thus, if I check a given log and see the signatures don't match, I stop since I know there is a problem with the chain provided to me.