CS 427, Assignment 6

Cody Malick
malickc@oregonstate.edu

March 3, 2017

1

The weakness here is that we are xoring the results of the PRF F with the previous results of the PRF. The seed of the value t being the key. If the length of m is of size 1, then the returned value t will always be the result of the PRF F. While this is an issue, it becomes a significant issue when we use this fact our advantage when we take another message, concatinate it with a known message resulting in a string of size 2. We can then attack the fact that a message of two block sizes doesn't gain any extra encryption from the xor:

```
k := KeyGen()
Attack():
  // Single block message
  m_1 := {0,1}^λ
  m_2 := {0,1}^λ
  m_3 := {0,1}^λ
  h_1 := MAC(k, m_1)
  h_2 := MAC(k, m_2)
  h_3 := MAC(k, m_1||m_3)
  h_4 := MAC(k, m_2||m_3)
  if h_1 ⊕ h_3 == h_2 ⊕ h_4:
    return true
  return false
```

The attacking function will return true with a probability 1. But the important distinguisher here is that the xor does effectively nothing with block size one and two strings.

2

After some fun experimentation, I was able to find a collision in the first five bytes after 2616776 attempts! The original file hashes are:

gcat.jpg - b4a00bd5ce01c34f9faf62142d51e810
doge.jpg - 2a23c1bc0108eceaa0a6e8837414803d

And the collision is:

newGcat.jpg - 3fa488457e868f688209d0725baaca3a
newDoge.jpg - 3fa488457ebc3e6fe988b774e67062d5

And, of course, the collided pictures:


Sorry if the pictures are a little blury, but I picked 50x50 size pictures because my first attempt involved storing all newly generated pictures. That ended up never terminating and using 350 GBs of storage.

I've attached a file with this submission, main.go, which contains the code I used to generate and check the collision. It's written in the Go programming language. I used weak hash collision by generating a random 20 bytes of information, and appending it to either the cat or dog picture. Alternating between the two, I kept a dictionary of created hashes, along with the 20 bytes I needed to append to generate that hash. Once I find a collision, I recreate the collided picture using the dictionary.

3

Per the class text, a function is "collision-resistant" if no polynomial-time program can find a collision in H. With that in mind, the attack on this function must be executable in polynomial time. More specifically, a given function should "self-destruct" if it detects a collision. To show that the hash isn't resistant, we should show that we can trigger a self-destruct a non-neglible amount of times.

We can attack this function by taking advantage of the fact that the first block is not xored with another value:

```
COLLIDE()
  m_1 := 0^n
  h_1 := H*(m_1)
  // first and second block
  b_1, b_2 := H*(m_1 || (h_1 ⊕ m_1))
  if h_1 == b_2 :
    return true
  return false
```

By xoring the value of the first hash, and combining it with the second block, we can negate, with consistency, the value change that is supposed to happen in the xor. This attack can be extended to an arbitrary number of blocks.

This function will return 1 when it calls H^*, and will return false the majority of the time with a collision resistant hash. Specifically:

If H^* is called: $Pr[Collide\ returning\ true] = 1$
If a collision resistant hash is called: $Pr[Collide\ returning\ true] = \frac{1}{2^\lambda}$