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:

$k := \text{KeyGen}()$
$\underline{\text{Attack}():}$
  // Single block message
  $m_1 := \{0,1\}^\lambda$
  $m_2 := \{0,1\}^\lambda$
  $m_3 := \{0,1\}^\lambda$
  $h_1 := \text{MAC}(k, m_1)$
  $h_2 := \text{MAC}(k, m_2)$
  $h_3 := \text{MAC}(k, m_1 || m_3)$
  $h_4 := \text{MAC}(k, m_2 || m_3)$
  if $h_1 \oplus h_3 == h_2 \oplus h_4$:
    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 pictures:

 

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.

3

We can show that this library is not collision resistant by abusing the lack of second-preimage resistance, or weak hash collision resistance. For every value we generate through H^*, we track it in an array. Then, for each new output, we check it against the whole array. If there is a duplicate value, then we know we've found a collision:

```
COLLIDE(k, m)
  𝒯 := ∅
  // While we haven't found a collision, loop forever
  for(i = 0; i >= 0; i++) {
    if 𝒯.Contains(
      return true
    }
```

This system abuses the birthday attack, which will get a collision probability of P, according to the following formula: $P = 1 - \frac{n!}{(n-i)! * n^i}$. Because the denominator grows quite quickly, collision is highly likely in a relatively small number of attempts.