Problem Set #6

CMSC/Math 456
Instructor: Daniel Gottesman

Due on Gradscope, Thursday, Nov. 10, 11:59 PM

Overview: The goal of this problem is to break a MAC. In problems 1 and 2, you will be attacking a MAC for small messages and for problem 3, you will attack it when used with a hash-and-MAC protocol.

Important reminder: You may collaborate on the problem set, but you *must* include a comment in your submission indicating anyone you worked with. You should also be writing your own code, not modeling it directly off of someone else's.

Instructions: You must upload to Gradescope a single Python file named "forge.py" containing all the functions with the specifications given below in the 3 problems. The file forge.py contains Python functions that implement the MAC and hash functions used for this problem. It also includes named functions with sample attacks (which don't succeed) for you to fill in the functions needed. You may use these functions to analyze the MAC and hash function, to test your attacks, and as subroutines in the attacks. You may also reuse code from them in your scripts.

Do not rely on uploading your file at the last minute. If there is a problem with your upload or a bug in your code that prevents the autograder from working, you will be left with a 0 score.

Scoring: Gradescope will pass your files through an autograder to give you a score and some feedback right away. You can resubmit as many times as you like to try to improve your score. However, I strongly recommend you do testing on your own system rather than try it repeatedly on the autograder, as it will be faster and consumes less communal resources.

Also, note that if we discover a bug in the autograder after the assignment is opened, it is possible we will have to rerun the autograder, in which case scores could potentially change.

The MAC: This description of how the MAC works is provided to help you understand it, but the true definition should be considered the Python code provided in forge.py, with one exception: The key in the true protocol is generated randomly.

The key k is a 16×16 array of numbers, each in the range 0 to 10,000. A message is a pair (m, n) with m and n both ranging from 0 to 15. Thus the pair specifies a location in the array. A tag is also a pair (s, t) given by the following formulas:

$$s = \sum_{i=0}^{m} k[i, n] \tag{1}$$

$$t = \sum_{j=0}^{n} k[m, j]. \tag{2}$$

The hash function and the hash-and-MAC protocol: Again, this description of the hash function is provided to help you understand, with the true definition given by the Python code in forge.py.

The hash function $H(x)$ is given by the Merkle-Damgard construction applied to a compression function $h(z, x)$ described below. The Merkle-Damgard construction was described in class and is in the book as well, sec. 6.2. The inputs and outputs x, z, and $h(z, x)$ are provided as lists of 6 bytes (i.e., integers in the range 0 to 255). Then

$$h(z, x)[i] = z[i] * x[i] + x[i + 1] \bmod 256. \tag{3}$$

Here the indices are considered to be cyclic, so the index $i + 1$ should be taken mod 6.

To tag a message m, m is padded as standard for the Merkle-Damgaard construction with a 1, then 0s enough to fill an even number of 6 byte blocks, and finally 2 bytes giving the length of the message in bytes, and then put through the hash function (i.e., through the compression function repeatedly). The initial value of z is a list of 6 bytes, all equal to 17.

Each byte in the output $H(m)$ of the hash function is broken up into a pair of two numbers 0–15 by taking the first and last 4 bits. This is in the form of a message above. Each such pair is used to generate a single tag t_i $(i = 1, \ldots 6)$.

The tag for the message m is then $(H(m), (t_1, t_2, t_3, t_4, t_5, t_6))$.

Python note: For those less familiar with Python, the name of the true MAC function (with the key which you are supposed to use to forge messages) is going to be passed to your attack functions as a variable named either MAC or MD in the example functions. You can call it by just treating it as a function named MAC or MD; the true name will be substituted in for that variable name.

Problem #1. Forge any Message (20 points)

For this problem, create a function "selective_forge(MAC)" which takes a function name as input MAC. The MAC function takes a message (as a pair (m, n), two numbers 0 to 15) and returns a tag (again a pair of two integers, although likely much larger than 15).

You may query the MAC one or more times and must return a pair (m, t) of a message m and a tag t. The tag should be a correct tag of the message you provided. The message can be anything of your choice, except that it cannot be a message that you used to query the MAC.

You may query the MAC as many times as you like but your score will be based on how many times you queried the MAC. If you successfully forged a tag with arbitrarily many queries, you will receive 10 points. This increases to 15 points if you can use less than 10 queries and increases further with fewer queries, to a maximum of 20 if you can match (or beat) the best attack we know.

Problem #2. Forge Specific Messages (20 points)

Define a function "universal_forge(MAC, msg)" that takes two inputs, a function name as input MAC and a message (a pair of two numbers from 0 to 15) as input msg.

You may query the MAC as in problem 1, but you now return only a tag (again, a pair of two integers). The tag must be a valid tag for the message your function received as input.

Be sure your function succeeds no matter which message you are passed. The exception is the message $(15, 15)$, which cannot be forged. You will likely need to deal with multiple cases.

You will be asked to forge messages for 4 different cases. Each is worth 3 points for a successful forgery, increasing to 5 points if you can use the minimum number of queries for the best attack we know.

Problem #3. Forge a Message for Hash-and-MAC (20 points)

Define a function "md_forge(MD)" that takes a single input, a function name MD which calls the hash-and-MAC protocol. The MD function takes as input a message which is now a list of up to 2^{16} bytes and returns a tag for that message.

You may query MD and must return a (message, tag) pair. The message is now a list of bytes (max length 2^{16}) and the tag is a list of 6 bytes plus 6 pairs of integers. The message you return cannot be one of the messages you queried.

You again get 10 points for a successfully forgery, rising with fewer queries to a maximum of 20 points with 1 query.