Cryptography and Network Security Make-up Exam

Jacques Becker

October 27, 2018

Problem 1

The example attack described in the textbook is based on the birthday paradox. It explains that an attacker can find a pair of messages m_1 and m_2 such that $H(m_1) = H(m_2)$ with probability 0.5 by generating $2^{n/2}$ variations of each message with identical meaning, where n is the number of bits in each hash. In order to implement this attack, I need to define a procedure for generating many variations of a given message without altering its meaning, decide which hash function to use, and find an efficient way to store and compare the message/hash pairs.

To generate variations of a given message without altering its meaning, I made a simple algorithm to insert spaces in between words in various combinations. This way, the text is still readable, as only the whitespace of the message is altered. Here is a description of the algorithm:

Input: m, the original message represented as a list of words, w, the number of words in the message, and n, the number of variations to be generated
Output: M, a list of n variations of the original message m

1. Create an empty list L
2. Create an empty queue Q
3. Create an array A of size w and initialize each value to 1
4. $Q \leftarrow A, L \leftarrow A$
5. While L contains fewer than n elements:
   (a) $X = Q.\text{pop}()$
   (b) for $i = 0$ to w:
      i. $X' := X$
      ii. $X'[i] := X'[i] + 1$
      iii. $Q \leftarrow X', L \leftarrow X'$
6. for each array A in L:
   (a) Create an empty string v
   (b) for $i = 0$ to w:
      i. Append $A[i]$ space characters to v
      ii. Append the i^{th} word of m to v
   (c) $M \leftarrow v$
7. return M

Although this procedure generates some duplicate variations of the message, the number of unique variations is close enough to n for this to be a viable procedure in this kind of attack.

The next step is to decide which hash algorithm to use. I initially tried to use MD-5 because it was mentioned the problem's text, but quickly realized that generating and storing 2^{32} variations of the message in the problem requires more memory than my machine has available. After a short bit of research into other hash algorithms, I decided to use MurmurHash3, a very fast 32-bit hashing algorithm which is better suited for the resources I have available.

The last step is finding an effective way to search for message pairs with identical hashes. I did this by creating an empty python dictionary, then hashing all variations of the message, and using the hash value as the key in the dictionary, and storing the index of the message as the associated value. Then, after generating variations of the fraudulent message, hashing each fraudulent variation and checking if the hash value has already been used as a key in the dictionary. If it has, then a collision has been found. Since the python dictionary is implemented as a hashmap, adding key-value pairs and looking up keys is a constant time operation in the average case. Therefore, the whole collision search process takes linear time with respect to the number of message variations generated.

My program, when run using the provided message and my arbitrarily-chosen fraudulent message `"This is a fraudulent message. This is a fraudulent message. This is a fraudulent message."`,

found a collision between this variation of the original message:

```
"  More efficient attacks are possible by employing cryptanalysis
to specific hash functions. When a collision  attack is discovered
and is found to be faster than a birthday attack, a hash function is
often denounced as "broken". The NIST hash function competition was
largely induced by published collision  attacks against two very commonly
used hash functions, MD5 and SHA-1.  The collision attacks against
MD5 have improved so much that, as of 2007, it takes just a few seconds
on a regular computer. Hash collisions created this way are usually
constant length and largely unstructured, so cannot directly be applied
to attack widespread document formats or protocols."
```

and this fraudulent message:

```
" This is   a fraudulent message. This is a   fraudulent message.
This is  a fraudulent message."
```

both of which have the same hash value: 1903964890

If you'd like to run my code and verify that it works, first run `pip install mmh3` to acquire the MurmurHash libraries, then run

`python collision.py [num variations]` to execute the code. For this particular example, I had it make $2^{21} = 2097152$ message variations before it found a collision, so simply run `python collision.py 2097152` to see the output presented above. On my machine, it took about two minutes to terminate and it used up ≈ 7 GB of memory.

Problem 2:

Problem 10-12: Consider the elliptic curve $E_7(2,1)$. Determine all the points in $E_7(2,1)$

The points are:

- $(0,1)$
- $(0,6)$
- $(1,2)$
- $(1,5)$
- The point at infinity

Problem 10-13: What are the negatives of the following elliptic curve points over Z_7? $P = (3,5), Q = (2,5), R = (5,0)$

$-P = (3,-5) = (3,-5 \bmod 7) = (3,2)$
$-Q = (2,-5) = (2,-5 \bmod 7) = (2,2)$
$-R = (5,0)$

Problem 10-14: For $E_{11}(1,7)$, consider the point $G = (3,2)$. Compute the multiples of G from $2G$ through $13G$.

The multiples of G are:

- $G = (3,2)$
- $2G = (10,4)$
- $3G = (1,8)$
- $4G = (5,4)$
- $5G = (4,8)$
- $6G = (7,7)$
- $7G = (6,8)$
- $8G = (6,3)$
- $9G = (7,4)$
- $10G = (4,3)$
- $11G = (5,7)$
- $12G = (1,3)$
- $13G = (10,7)$

Problem 10-15: This problem performs elliptic encryption/decryption using the scheme outlined in Section 10.4. The cryptosystem parameters are $E_{11}(1, 7)$ and $G = (3, 2)$. B's private key is $n_B = 7$.

(a) Find B's public key P_B.
$P_B = n_B G = 7(3, 2) = (6, 8)$

(b) $C_m = \{kG, P_m + kP_B\}$
$kG = 5G = (4, 8)$
$kP_B = 5(6, 8) = (4, 8)$
$P_m + kP_B = (10, 7) + (4, 8) = (1, 8)$
$C_m = \{(4, 8), (1, 8)\}$

(c) Show the calculation by which B recovers P_m from C_m.

$$
\begin{aligned}
P_m &= (P_m + kP_B) - n_B kG \\
P_m &= (1, 8) - 7(4, 8) \\
P_m &= (1, 8) - (4, 8) \\
P_m &= (1, 8) + (4, -8) \\
P_m &= (1, 8) + (4, 3) \\
P_m &= (10, 7)
\end{aligned}
$$

Problem 3: Consider the following integers: 31531; 520482; 485827; 15485863

- 31531 is prime
- $520482 = 2 \cdot 3 \cdot 223 \cdot 389$
- 485827 is prime
- 15485863 is prime

My source code for the Miller-Rabin and Pollard-Rho algorithms is linked in the piazza submission. To run the code, use
`python miller_rabin.py [number] [num_trials]` or use
`python pollard_rho.py [number]`.