Find a collision

HW2 - CNS Sapienza

Pietro Spadaccino 1706250

6 November 2018

1 Introduction

Our goal is to find a collision between two different outputs on 'weakened' SHA-1, where 'weakened' means that only last k bits of the two inputs' hash must be equal in order to have a collision. We will describe how to generate such a collision between two different string using $k = 61$ bits.

2 SHA-1 hashing function

The SHA-1 is a gradually disappearing hashing method. It maps its input to a fixed-length output of 160 bits, hence capable of generating 2^{160} (or about 10^{48}) different hashes. It is getting replaced by the more secure SHA-2, a family of hashing functions with minimum output of 256 bits, counting almost 10^{29} times more possible hashes. SHA-1 is considered broken since researchers from Google and CWI Amsterdam found the first collision [1], performing about 10^{19} hash compressions, more than one billion billion hashes. Note that this number is far smaller than the number of all possible outputs. To find a collision in such a 'tiny' number of attempts, the team exploited cryptographic weaknesses alongside a collision method called the 'birthday attack', described in the next section.

3 Birthday attack

The birthday attack it's a probabilistic consequence that greatly reduces the number of hashes to be done in order to find a collision. By being a mathematical result, this attack is suitable for every hashing functions, even for methods more secure than SHA-1.

The birthday paradox states that if 23 people are chosen random, then there is more than 50% chance that at least two of them have the same birthday (assuming every person has the same probability of being born in all days). The idea to understand this principle is that every time I pick a new person there are two possibilities: or it has the same birthday of one of the people already selected, so I've found a birthday match, or I increase the set of distinct birthdays, thus increasing the probability of a match at the next pick. More formally, the probability p of having two matching birthdays by choosing n people is given by:

$$p = 1 - \left(\frac{364}{365}\,\frac{363}{365}\,...\,\frac{365-(n-1)}{365}\right) \;=\; 1 - \frac{365!}{365^n(365-n)!} \tag{1}$$

The birthday attack is this same principle applied to hashing function, picking distinct strings instead of people and considering their hashes instead of their birthdays. This attack can be very powerful, since, in order to have a non-negligible chance for a collision, I don't have to search on all the output space Π but only on $\approx \sqrt{\Pi}$. As reported in [2], I can use the following approximations of p and n:

$$p \approx 1 - e^{-n^2/2|\Pi|} \tag{2}$$

$$n \approx \sqrt{2|\Pi| \ln \frac{1}{1-p}} \tag{3}$$

where $|\Pi|$ is the number of all possible outputs of the chosen hashing function, and p (n) is the expected probability (number of steps) to generate a collision.

4 Approaching the problem

It is clear that to perform a collision for sufficiently large values of k we need to exploit the birthday paradox, otherwise it would not be unfeasible; a method that finds a collision on a particular string, where hash are generated until one of them is equal to the starting string's hash, *does not* exploit the birthday attack, because every guess is independent from the previous guesses. To estimate how many steps are needed I can use this equation:

$$p = 1 - \left(1 - \frac{1}{2^k}\right)^n \tag{4}$$

where p is the expected probability of finding a collision, n is the number of steps. As an example, I can calculate the expected number of steps to have a collision chance of at least $p = 75\%$, using $k = 56$ bits:

$$n = \log_{1-\frac{1}{2^k}}(1-p) = \log_{1-\frac{1}{2^{56}}}(0.25) \approx 10^{17} \tag{5}$$

Definitely too large.

In order to take advantage of the birthday paradox we need to keep track of all previous calculated hashes, and to do that we need a data structure capable of fast lookups and fast insert: a hashmap should do the trick.
Next, the algorithm is quite easy: initialize an empty hashmap $m : H \to S$, where H is the set of k bits sub-hashes and S is the string generating that hash. Loop by calculating hashes h of different strings s: if h is not in the hashmap then add it, otherwise we have found a collision between s and $m(h)$. With this algorithm, using the approximation in equation (3) and setting $k = 56$ bits, we have an expected number of guesses approximately equal to 10^9 to have a collision chance of $p = 99.9\%$.

5 Implementation time complexity

The chosen language was C++, because of performance reasons and the standard library built-in.
Let's talk about time complexity. Let's assume that we have to perform N hashes in order to generate a collision. The cost of calculating a SHA-1 can be considered constant, since the strings have similar length, so we have $O(N)$ for hash calculations. Then, extracting k bits of one hash is constant too, by the use of a binary mask. Finally, I used an `unordered_map` as data structure to save the hash entries: it has an amortized cost of $O(1)$ for both finding and inserting a key, so the cost of managing N hashes is again $O(N)$. Therefore the total asymptotic time complexity is $O(N)$.

6 Expected performances

The coded algorithm performs in the exact way as described in section 4, with some little differences in order to take in consideration memory as a limited resource. When running the algorithm, a lot of RAM must be used by the hashmap, ≈ 70 bytes per entry, quickly saturating system resources. By having 16GB available, a hashmap would reach a maximum size of around

14GB before crashing the entire operating system and it would contain 215 millions entries. If we use $k = 61$ bits, the probability of finding a collision in $n = 215$ million steps is calculated using equation (2): $p \approx 1\%$.
What if a collision doesn't happen? A solution is starting a new iteration by computing collisions again with a different set of strings. If the strings used are different from the previous ones, then the probability of finding a collision is independent from the previous iteration, and it would have the same colliding chance of $p \approx 1\%$. From a probabilistic point of view, we have an expected probability of at least $p = (75\%, 90\%, 99\%)$ of finding a collision after about respectively $n \approx (138, 230, 459)$ independent iterations, or equivalently (29.6, 49.4, 98.7) billion hashes, calculated using the same reasoning of equation (4).

7 Found collision

The collision was found using $k = 61$ bits during the 246^{th} iteration after calculating ≈ 52.7 billion hashes. The strings are

```
ATTACK AT TIMESTAMP=52676963044
ATTACK AT TIMESTAMP=52696959107
```

both *with trailing newline*. This can be verified by launching the SHA-1 sum of the attached files from bash with

```
sha1sum secret_*.txt
```

or launching the SHA-1 sum directly with the following:

```
echo "ATTACK AT TIMESTAMP=52676963044" | sha1sum |
   xxd -r -ps | xxd -b &&
echo "ATTACK AT TIMESTAMP=52696959107" | sha1sum |
   xxd -r -ps | xxd -b
```

and checking that the last 7 bytes + 5 bits are equal.

References

[1] Marc Stevens, Elie Bursztein, Pierre Karpman, Ange Albertini, Yarik Markov, *The first collision for full SHA-1*, 2017

[2] wikipedia.org/wiki/Birthday_attack, *Birthday attack*