Collision detection in weakened SHA-1

HW2 - CNS Sapienza

Luigi Russo 1699981

11/11/2018

1 Goals

The goal of this homework is to find a collision on a weakened form of SHA-1: more in detail, SHA-1 produces a 160 bits digest, thus resulting in 2^{160} combinations; a weakened form of SHA-1 can be obtained just considering the last k (< 160 of course) bits of this digest, thus resulting in 2^k combinations.

1.1 The birthday paradox

Given k as the length (in bits) of the digest, there is a well-known paradox that states that, for a brute force attack, we don't actually need 2^k attempts, but we need just $2^{k/2}$ attempts to have a success rate greater than 0.5.

2 Methodology

I've used 3 different algorithms in order to find the collision for the maximum possible (given the resource constraints of my machine) value of k. I will show in details each of these in the next sections. The source code is available at *https://www.gitlab.com/lrusso96*.

3 Naive Approach

I've written a Python 3 script in order to solve the problem: first I've generated all the possible combinations of *ASCII_printable*, i.e. the characters that can be printed (letters, numbers, white-spaces, etc). Then I've stored the pair *(word, digest)* in a hash-map, s.t. map[*digest*] = *word*. When the script is running and generates a new word, say *word_a*, it computes the SHA-1 of word_a, say *digest_1*, then it removes the first 160 - k bits; at this point it searches for an occurrence of *digest_1* in the map: if there is already a value stored in map[*digest_1*] and this value is, say, word_b, then I've found a collision between word_a and word_b on the common value *digest_1*.

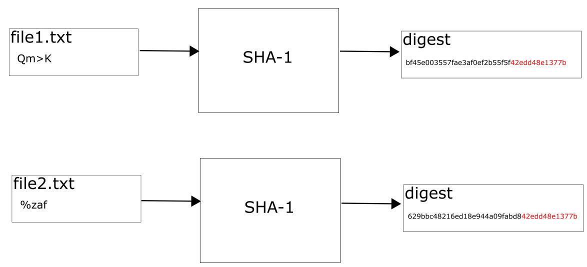


Figure 1: *Example of collision on weakened SHA-1. The digests above are represented in hexadecimal.*

3.1 Memory issues

I've run the script on my 8 GB RAM laptop: for k = 52, for instance, the number of digests computed before finding the collision is greater than 67 millions, that is $\approx 2^{26}$ (cfr. the birthday paradox). In order to run the script on values > 52, e.g. 56, I had to slightly change it.

4 Need for space: a smarter approach

The totality is not, as it were, a mere heap, but the whole is something besides the parts stated Aristotle in his Metaphysics. So, instead of building a huge hash-map of digests, I partitioned the digests set in 16 different sets, each associated to a unique map: each of these maps behaved as an index; in fact, map_0 contained all the digests whose first 4 bits were 0000, map_1 contained all the ones with 0001 at first, and so on. The sets of the keys contained in each map were (built to be) disjoint. Since I could not have in main memory all the 16 sets, the script simply made 16 iterations, focusing each time on one specific prefix (map_index) that was effectively analyzed.

4.1 Time complexity and parallelization

Partitioning the digests set in 16 disjoint sets is *asymptotically* irrelevant for the time complexity of the algorithm, since 16 is a multiplicative constant that I had to pay (in my trade-off with RAM usage) in order to achieve

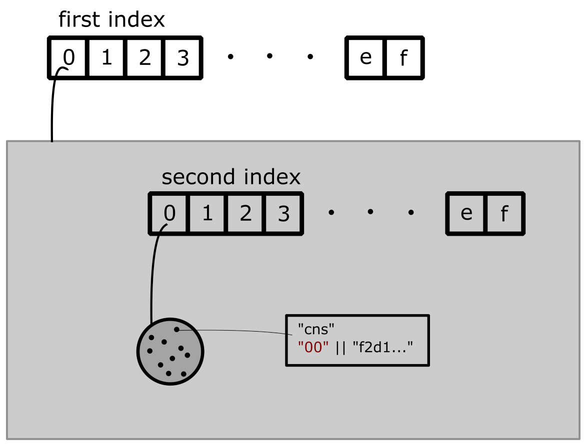


Figure 2: *Simple schema of the double indexing used to handle multiple hash-maps. The string in red, i.e. the prefix, is not stored of course because unnecessary.*

the goal. Moreover, due the fact that each iteration is independent from the others (because each one searches the collision with a specific pattern) it is possible to parallelize the jobs: e.g given 16 laptops, each of these can process a specific prefix (i.e. the first 4 bits of digests).

5 Floyd's algorithm

Even the modified version of the Naive Algorithm requires a lot of RAM to run for very big values of k. For this reason I've decided to adopt a well-known algorithm able to detect a cycle in a graph: the Floyd's algorithm. It requires constant usage of RAM and after a number of iterations ($\approx 2^{k/2}$) converges and finds the collision. I've implemented this algorithm in Python 3, and I've been able to find a collision for k = 66 bits.

6 Results

6.1 Main goal

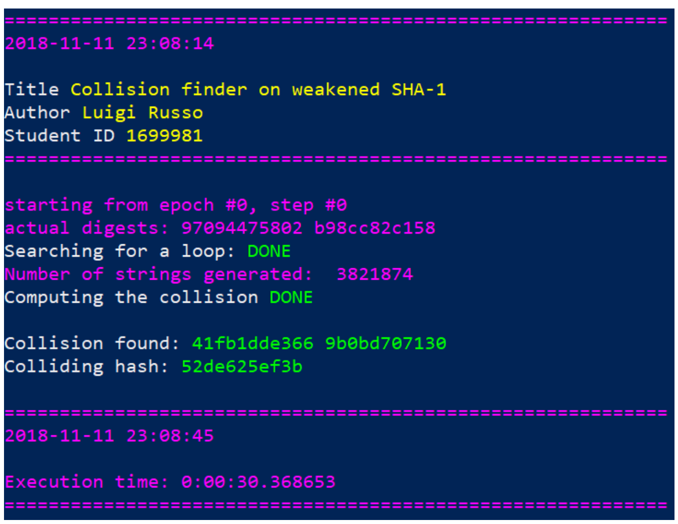
I've managed to generate two files with same weakened SHA-1, with **k = 66**. The files are attached to this report, as file1.txt and file2.txt. In the next table there is a summary of the results obtained with each method and the resources needed to reach the goal.

	k	RAM peak (MB)	Iterations (M)
Naive	52	≈ 4000	≈ 67
Modified Naive	56	≈ 3000	≈ 500
Floyd	**66**	< 20	≈ 1300

Table 1: *Results and resource usage for each method.*

6.2 Other considerations

To be honest, finding the collision with k = 66, was only one of the steps of the script. Indeed, the algorithm can brute-force whatever value: for completeness I've attached the file collisions.txt, where it is possible to find collisions for k $\in$ {4,8,12,16,20,24,28,32,36,40,44,48,52,56,60,64,66}: each row reports the number k used, the two words whose digests collide, and the number of digests computed before finding the collision.



```
=============================================================
2018-11-11 23:08:14

Title Collision finder on weakened SHA-1
Author Luigi Russo
Student ID 1699981
=============================================================

starting from epoch #0, step #0
actual digests: 97094475802 b98cc82c158
Searching for a loop: DONE
Number of strings generated:  3821874
Computing the collision DONE

Collision found: 41fb1dde366 9b0bd707130
Colliding hash: 52de625ef3b

=============================================================
2018-11-11 23:08:45

Execution time: 0:00:30.368653
=============================================================
```

Figure 3: *Example of run, with $k = 44$.*