Programming Exercise 1

SHA-1 Hash Cracking (14 Pts)

Deadline: 12th February 2016 - 11:59 p.m.

Methods of User Authentication

Research Group: Mobile Security

Winter Semester 2015/16

Prof. Dr. Markus Dürmuth, Maximilian Golla, Jan Rimkus

Task: Write a program to invert SHA-1 hashes, i. e., given a hash, find a preimage. Performance should be your main concern.

- Preimage is limited to [a-z]^6, i. e., from `aaaaaa` to `zzzzzz`
- Your own SHA-1 implementation, no external libraries
- Single threaded
- We compile with the latest `gcc` (e. g., 4.8.4) running on Ubuntu 14.04
- Compile string: `gcc -O2 -Wall -fomit-frame-pointer -msse2 -masm=intel`
- Code has to be warning and error free (`-Wall`)
- Only one file (include your header definitions in your code)
- Naming: `SHA-1_<Team-xx>_<version-number>.c`
  (Example: Team 1, first version, `SHA-1_Team-01_1.c`)

Just upload your submission to the Moodle course, anytime you want; only the latest submission counts. Please only select one member of your team to upload it every time. We honor when you start coding right now and not wait until the submission deadline. Therefore we compare the latest submissions every two weeks, the fastest program will get some bonus.

Hints for grading 1.1 Schoolbook Implementation, 2 Points

The most easiest implementation of a six character password brute-force search. Input a SHA-1 hash, output the six character long preimage (lower-case). Please search from lowercase `a` to lowercase `z`.

Hints for grading 1.2 Code Optimization, 2 Points

For example use,

- Alternative f-functions
- Loop unrolling
- Macros
- Inline assembler
- Static inline functions
- Constants
- Multiple variable declaration
- Early variable definition

where it is reasonable. Be aware that some optimization maybe destroyed by `gcc -O2`.

Hints for grading 1.3 Hash-based Optimization, 7 Points

For example use,

- Compute the padding only once
- Zero-based optimization
- Initial-step optimization
- *Early-exit* optimization
- Precomputations
- SHA-1 Message expansion attack[1].

Some cues are given in the tutorial slides. Another great resource on that topic can be found here[2].

Hints for grading 1.4 SIMD Implementation, 3 Points

Implement the cracker with *Intel SSE2* and gain a 300% speed increase. Be aware of the *26/4 pitfall* and find a solution for it. To keep the hardware requirements low, please do not use a newer SSEx version and do not use AVX/2/512.

To enable easy correction on our side and help you to test and verify your program, we will provide a framework that has to be implemented. It can be downloaded via the Moodle course. More on that topic can be found in the tutorial slides.

Here is a small example how the final cracker may look like:

```
$> gcc -O2 -Wall -fomit-frame-pointer -msse2 -masm=intel testbench.c
SHA-1_Team-01_1.c -o crackSHA1

$> ./crackSHA1
Please provide a 160-bit hash.
Usage: ./crackSHA1 <HASH>

$> ./crackSHA1 30274c47903bd1bac7633bbf09743149ebab805f
Hash: 0x30274C47 903BD1BA C7633BBF 09743149 EBAB805F
Preimge: passwd
Cycles: 34812580311
Time: 12912.279000

$> ./crackSHA1 755bd810d2be0ebcbb6ce6f532b3d9cfcf9d9695
Hash: 0x755BD810 D2BE0EBC BB6CE6F5 32B3D9CF CF9D9695
Preimge: ananas
Cycles: 13021970649
Time: 4828.097000

$> ./crackSHA1 3854e277a37aee29bf9ecc86fb983737cf9d9695
Hash: 0x3854E277 A37AEE29 BF9ECC86 FB983737 CF9D9695
Preimge: qfucra
Cycles: 40910882356
Time: 15173.018000
```

[1] Jens Steube: Exploiting a Weakness in SHA1 Password Cracking, Passwords^12, December 2012, https://hashcat.net/events/p12/

[2] Jens Steube: Optimizing Computation of Hash-Algorithms as an Attacker, Passwords^13, May 2013, https://hashcat.net/events/p13/