ALGORITHMICS, CRYPTOGRAPHY & COMPUTER SECURITY, graduate programs 2023
CRYPTOGRAPHY, 2022, lab assignments list # 1, 8.03.2023 **updated** 17.03.2023

Familiarize yourself with MD5 hash function [1]. Your task for this list is to reproduce collision attack by Wang et al. [2, 3].

1. (1pt) To reproduce the collision attack you need a working MD5 implementation and its internals. Find an open source implementation, for instance MbedTLS [4] (you can chose any programming language you want). Familiarize yourself with the source code and identify inner workings, in particular F, G, H and I functions, A, B, C, D variables, and so on.

2. (3pts) Based on the implementation from the previous step (feel free to copy or link/reference, as long as the license permits!) try to compute step-by-step the example from Table 2 in [3]. Ensure that the data results in a collision, equal to the H from the table. Debug if not. In particular check for byte order (endianness). In general, we need to check if $MD5(MD5(IV, M_0), M_1) == MD5(MD5(IV, M_0'), M_1') = H$. Create an interface that allows you to verify this for any choice of M_0, M_0' and M_1, M_1'.

3. (3pts) Having this interface, implement the (easier) second step of the attack (Step 2 from Sect. 4.5 in [3]). Create a procedure that given M_0 and M_0' on input, finds M_1 and M_1' that result in a collision (same hash value). For now test with M_0 and M_0' from Table 2.

4. (3pts) Implement the first step of the attack (Step 1 from Sect. 4.5 in [3]) using "Single-message Modification" method (Sect. 4.4). This step should take slightly longer to compute, so give yourself ample time for potential debugging. Combine with Step 2 you implemented earlier for a full collision attack. Finally, generate a collision and try checking with an external software that you indeed obtain an MD5 collision (use for instance `md5sum` Linux command). Remember about potential encoding differences, byte order and so on.

5. (*, 2pts) Add an option to use "Multi-message Modification" method (Sect. 4.4)

Try to implement as much as you can during the labs. Discuss your findings and problems with your colleagues. Tasks you don't finish during the labs are your homework. At the next labs (your **2nd** full-time labs) present your progress and discuss obstacles. You should *at least* be able to complete tasks 1 and 2 on your own. By the 3rd labs you should be able to launch the attack.

Remember, the final implementation should be your own, with the exception of the open source fragments from Task 1.

Hints

1. The exhaustive search stage is pretty intensive. Make sure the environment you choose (language, libraries, execution environment, etc) can execute a 2^{40} search in a reasonable time (hours, not weeks).

2. Test as much as possible with the data from the paper. If you write a function that verifies if a message clears the requirements from Table 6 - check if the messages from Table 2 pass those checks.

3. Focus on correctness first, then on performance. Performance is still critical in the main search loop, but leaving the loop running 2^{40} times with a bug in code can be painful.

4. Byte order: MD5 loads data in *Little Endian* byte order, so the order on disk will be different from the order in paper (or the "natural", MSB first order), i.e. the first 4 bytes of M_0 (word `2dd31d1`) are `d1, 31, dd, 02`. All operations in MD5 are performed on 32-bit words so once the data is stored in that type, the order doesn't matter.

5. Section *4.4 Message Modification* describes how to modify a randomly generated message to fulfill *at least some* of the differential requirements (from tables 4 and 6). The remaining requirements still should be tested. Not all of them have to be, in fact you may just skip some of them and check if the results satisfy ΔH_1 / ΔH. Checking early allows for early detection of "bad" candidates.

6. Continuing the previous hint - *Single-message Modification* applies only to the first 16 rules. No hint regarding *Multi-message Modification* modification - it's a starred (*) task!

7. *Single-message Modification*, as presented in the paper, may be a bit imprecise, or straight-up invalid (rotation *IS NOT* distributive over addition!). To better understand how to apply it consider the following:

   (a) In a regular MD5 round, $a_{i+1} \leftarrow ((a_i + F(b_i, c_i, d_i) + m_i + k_i) \lll s_i) + b_i$, c.f. this image.

   (b) Since we need some specific properties out of a_{i+1} (let's call the modified version a'_{i+1}) we want to modify m_i into m'_i, s.t. $a'_{i+1} \leftarrow ((a_i + F(b_i, c_i, d_i) + m'_i + k_i) \lll s_i) + b_i$.

   (c) Now you have two equations with a single variable, which should be easy to solve by hand. Note that huge chunks of the two equations are common, but some parts need unwinding, like b_i needs to be subtracted from a'_{i+1} before the rotation can be inverted. Remember, here you solve for m'_i, all the other variables are known.

8. The condition tables are long and rewriting them by hand is very error-prone. Consider writing a script that can parse them into an easy-to-use form. For instance translate

   ```
   d1,2 = 0, d1,3 = 0, d1,6 = 0, d1,7 = a1,7, d1,8 = a1,8, d1,12 = 1,
   d1,13 = a1,13, d1,16 = 0, d1,17 = a1,17, d1,18 = a1,18, d1,19 = a1,19,
   d1,20 = a1,20, d1,21 = a1,21, d1,22 = 0, d1,26 = 0, d1,27 = 1,
   d1,28 = 1, d1,29 = a1,29, d1,30 = a1,30, d1,31 = a1,31, d1,32 = 1
   ```

   into:

   ```
   Set these bits to 0: 0x02208026
   Set these bits to 1: 0x8c000800
   Set these bits to their value in a1: 0x701f10c0
   ```

/-/ Marcin Słowik

References

[1] Ronald L. Rivest. *The MD5 Message-Digest Algorithm*. RFC 1321. Apr. 1992. DOI: `10.17487/RFC1321`. URL: `https://www.rfc-editor.org/info/rfc1321`.

[2] Xiaoyun Wang et al. "Collisions for Hash Functions MD4, MD5, HAVAL-128 and RIPEMD". In: *IACR Cryptol. ePrint Arch.* (2004), p. 199. URL: `http://eprint.iacr.org/2004/199`.

[3] Xiaoyun Wang and Hongbo Yu. “How to Break MD5 and Other Hash Functions”. In: *Advances in Cryptology - EUROCRYPT 2005, 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Aarhus, Denmark, May 22-26, 2005, Proceedings*. Ed. by Ronald Cramer. Vol. 3494. Lecture Notes in Computer Science. Springer, 2005, pp. 19–35. DOI: 10.1007/11426639_2. URL: https://iacr.org/archive/eurocrypt2005/34940019/34940019.pdf.

[4] TrustedFirmware and MbedTLS contributors. *MbedTLS MD5 implementation*. 2021. URL: https://github.com/Mbed-TLS/mbedtls/blob/3304f253d7aa4b5b18e772c455b2113f7af29ca5/library/md5.c.