Chair of IT Security
Brandenburg University of Technology

Asya Mitseva, M.Sc.
Fabian Mildner, B.Sc.
Prof. Dr.-Ing. Andriy Panchenko

Summer Term 2025

Part I: Cryptography

Deadline: 30th April 2025

Cyber Security Lab (Ethical Hacking)
MD5 Collision Attacks

Introduction

A cryptographically secure hash function must exhibit (i) *one-wayness* and (ii) *collision resistance*. A one-way hash function is a function that can easily compute $hash(M) = h$ for an input M, but it is hard to find a suitable input M that will produce a given hash value h. A collision resistant hash function ensures that it is computationally infeasible to find two different inputs M_1 and M_2 that hash to the same hash value h. Several popular one-way hash functions fail to provide a sufficient degree of collision-resistance. In [1], the authors demonstrate a practically feasible collision attack against MD5.

In this laboratory exercise, we study the usage of collision attacks and the damage that can be caused if the collision-resistance property of a widely-used one-way hash function is broken. In particular, we launch real collision attacks against the one-way hash function MD5. The goal of these attacks is to modify two different files such that they share the same MD5 hash value despite having completely different contents and format.

Preparation of the Experimental Setup

To fulfill this laboratory exercise, you should use an *Ubuntu Desktop* operating system. If necessary, please create your own virtual machine. In addition, we will be using the tools developed by the project *HashClash* [2]. You are required to install these tools by following the instructions provided by the project *HashClash*. The source code and installation guidelines can be found here:
`https://github.com/cr-marcstevens/hashclash`.

1 Tasks

Task 1: Generating Two Different Files with the Same MD5 Hash

First, you should get familiar with the tools provided by the project *HashClash*. The Git repository [3] and the website of the project [2] provide several guidelines about the operation of the tools. For this task, you should use the tool `md5_fastcoll` to generate two different files with the same MD5 hash value. For now, the first part of these files should be the same, i.e., they should share the same prefix. To this end, you can utilize the option to specify a prefix file with arbitrary contents. The operation of this tool is shown in Figure 1. The following command generates two output files, `msg1.bin` and `msg2.bin` for a given a prefix file `prefix.txt`:

```
$ ./md5_fastcoll -p prefix.txt -o msg1.bin msg2.bin
```

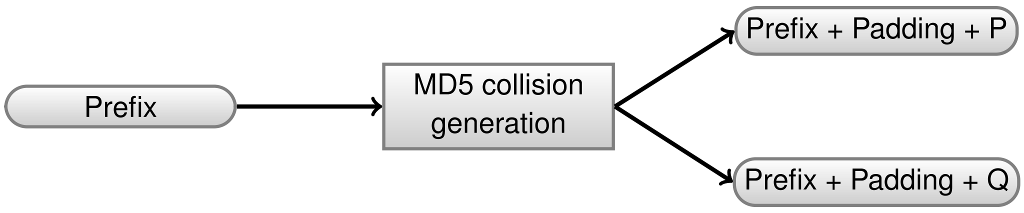

Figure 1: Generation of MD5 collision by using a predefined prefix.

To check whether the output files are different, you can use the command `diff`. To check the MD5 hash value of each of the output files, you can use the command `md5sum`.

```
$ diff msg1.bin msg2.bin
$ md5sum msg1.bin
$ md5sum msg2.bin
```

To read (and edit) both output files `msg1.bin` and `msg2.bin`, you need a binary editor, e.g., `hexedit`, `bless`. Please open both output files by using a binary editor and describe your observations. In particular, you should answer the following questions:

1. What happens if the byte length of your prefix file is not a multiple of 64?
2. Create a prefix file that has *exactly* a length of 64 bytes and run the collision generation described above once more. Describe what happens.
3. Is the data (128 bytes) generated by `md5_fastcoll` completely different for both output files? Please identify all the bytes that are different.

Task 2: Understanding MD5's Property

The goal of this task is to study some of the properties of the MD5 algorithm. These properties are important for the execution of the next task. As shown in Figure 2, MD5 divides the input data into blocks of 64 bytes and, then, computes the hash of these blocks iteratively. The most important part of the MD5 algorithm is the compression function, which needs two inputs: (i) the current 64-byte data block and (ii) the output from the previous iteration of this function. In other words, the compression function produces a 128-bit Intermediate Hash Value (IHV) and this output is then fed into the next iteration. If the current iteration is the last one, the IHV will be the final hash value. The IHV input for the first iteration IHV_0 is a predefined fixed value.

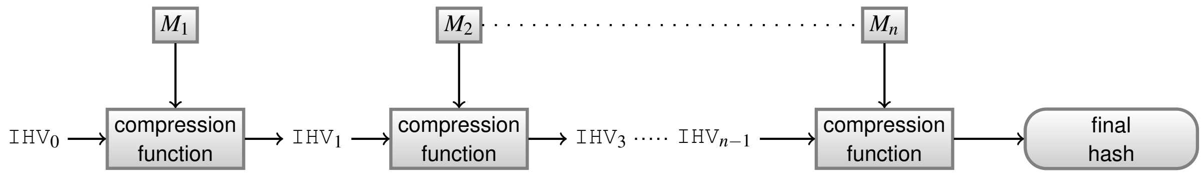

Figure 2: An overview of the MD5 algorithm.

Based on the operation of the MD5 compression function, the following property can be derived: If the MD5 hash values of two different inputs M and N are the same, i.e., $\mathtt{MD5}(M) = \mathtt{MD5}(N)$, then $\mathtt{MD5}(M \mathbin{||} T) = \mathtt{MD5}(N \mathbin{||} T)$ for any input T, where || represents a concatenation. In other words, if the inputs M and N produce the same hash value, adding the same suffix T to them will result in two outputs that have the same hash value as well. This property holds not only for the MD5 hash algorithm, but also for many other hash algorithms.

In this task, you should design and execute an experiment that demonstrates this property for *two differing files*. To this end, you can use the command `cat` to concatenate two files (binary or text files) into a single one. For instance, the following command concatenates the content of `file1.bin` and the content of `file2.txt` into a single file `file3.bin`:

```
$ cat file1.bin file2.txt > file3.bin
```

Task 3: Generating Two Files with the Same MD5 Hash

Nowadays, many online resources still use an `MD5` checksum for integrity checking. This is sufficient to detect accidental changes to the file, e.g. due to networking issues. However, due to the broken collision-resistance property, there are scenarios in which a deliberate attacker can make changes to a resource without the awareness of the user, even despite their efforts to secure data integrity using `MD5`.

In this task, you will receive two PNG files and their corresponding checksums. Your goal is to change these files in such a way that a user who executes integrity checking cannot identify these changes. In other words, you should modify both files such that they have the same `MD5` checksum. We will consider two scenarios for the execution of the attack:

- **Scenario 1:** The attacker can modify *exactly one* of both files. In this scenario, we assume that the attacker knows only one PNG file and can modify the picture without any restrictions beside that the (visible) content of the PNG file should not change. On the other hand, the adversary does not have any chance to modify the other PNG file. The second PNG file can be viewed, but not be modified by the attacker. He is able to calculate its `MD5` checksum.
- **Scenario 2:** The attacker can modify both files. This scenario is common for binary executable files, when the attacker provides a developed program for security auditing before he gets a certificate/allowance to publish the software. Thus, the adversary can pass the audit and change the program to a malicious version afterwards. In this scenario, the attacker is allowed to modify the binary of both files to achieve his goal of having the same `MD5` checksum. However, the (visible) content of both files should not be modified.

In particular, you should answer the following questions and/or tasks:

1. Which property of a secure hash function must be broken in order to be able to execute the attack? Elaborate on this question for each of the scenarios described above.
2. Is this property broken in case of the MD5 algorithm? Elaborate on this question for each of the scenarios described above.
3. Choose exactly one of the scenarios presented above and implement the corresponding attack. Describe and demonstrate each step of the development of the attack precisely. Finally, beside both files with the same MD5 checksum, you should also provide a writeup which describes the implementation of your attack and can be used to reproduce it. Please note that the use of existing scripts as a black-box solution will not be accepted when grading the task.

Preparation of Your Submission

Your submission for this lab should contain any and all source code you have written and files you have generated, along with a detailed writeup about the tasks you solved. Your writeup should be given in an appropriate format, containing detailed descriptions of how you solved the tasks, what theoretical insights you gained, what observations you have made as well as images / screenshots / listings to aid the understanding of your approach. Your writeup should be submitted in PDF format.

Please submit your solutions in a packed zip file named after your full name, with the following structure and minimum required contents. In particular, ensure you submit the colliding but differing files generated for task 1 as well as the modified PNG file(s) you generated for task 3.

```
firstname_lastname.zip
├─ Writeup.pdf and/or other source code
├─ task01
│  ├─ prefix.txt
│  ├─ msg1.bin
│  └─ msg2.bin
└─ task03
   ├─ collision1.png
   └─ collision2.png
```

Preparation for the Q&A Session

Prepare yourself for a Q&A session of up to 40 minutes. During the Q&A session, you should be prepared to *explain and demonstrate* how you solved the tasks. Take special care to explain how your solution works and why you took the approach you took. Note that simply presenting a piece of code without any explanation will not yield any points! Last but not least, you should be ready to answer spontaneous (theoretical) questions asked by the reviewer in relation to the topics of this lab.

References

[1] J. Black, M. Cochran, and T. Highland. “A Study of the MD5 Attacks: Insights and Improvements”. In: *Proceedings of the 13th International Conference on Fast Software Encryption*. FSE’06. Graz, Austria: Springer-Verlag, 2006, pp. 262–277 (cit. on p. 1).

[2] M. Stevens. *Project HashClash*. URL: https://marc-stevens.nl/p/hashclash/ (visited on 04/14/2020) (cit. on pp. 1, 2).

[3] M. Stevens. *Project HashClash - Git*. URL: https://github.com/cr-marcstevens/hashclash (visited on 04/14/2020) (cit. on p. 2).