FB12 — AG IT-Sicherheit
Prof. Dr. Elmar Tischhauser
Alexander Bille

Philipps Universität Marburg

IT-Security

Homework 1
Breaking and Protecting a KeePass Database

General notes:

- Please submit your solutions as a ZIP archive containing the answers as a PDF document as well as the corresponding source code and any other files relevant to your answers. The PDF can be any combination of text, photographed notes, program code, etc. Clarity is desirable, there are no beauty points.
- Explain your answers briefly but in a comprehensive way (if necessary with references to the source code, you do not have to write a novel).
- Upload your solutions to the corresponding homework exercise on Ilias for submission.
- Please work in groups of 1–3 students and provide the names and matriculation numbers of all group members.
- Deadline for submission is **Sunday, June 16, 2024 at 23:59 CET**.

Exercise 1 (Breaking a KeePass Password Database)

a) Write a program that performs a brute force attack on a KeePass 2 database, i.e. determines the user password with which it can be unlocked. There is a separate database for each student in the archive belonging to this note (named after the university account name, e.g. `tischhau.kdbx`). Each group must crack **at least one database of its group members** and provide the corresponding password as the answer to this question. Please note:
   - The database is only encrypted with a password; there is no key file.
   - The password has a maximum of 4 digits and consists only of numbers.
   - A description of the database format can be found in Appendix A on the following page.
   - Your program must contain your own code for parsing, key derivation, trial decryption, etc. Copying from the KeePass sources is not the purpose of the exercise. Of course you can use libraries for computing SHA-256, AES etc.

b) Use the cracked password to open the database (e.g. with KeePass, KeePassX, KeePassXC, ...). What is the combination of login and password stored *in* the password database?

Exercise 2 (Discussion of the Database Format)

a) What is the purpose of the key transformation (calculation of *transformedCredentials*)? What influence does the number of rounds *transform_rounds* have?

b) Does the use of ECB mode weaken the security of the transformation?

c) Measure the number of passwords that you can test with your program from Exercise 1 in one second. Estimate the approximate time for cracking the database if the key space is expanded to also include

   a) lowercase letters

   b) lowercase letters and uppercase letters

   with the number of rounds set to both 10000 (as in the example database) and 1000000 (newer versions).

d) What is the purpose of the *stream start bytes* field in the header? Is there another way to achieve the same functionality (without repeating the first 32 plaintext bytes)?

e) In older KeePass versions, the header is not integrity protected. Show that without the new integrity check it would be possible to create a second valid file based on an existing database file, which has the same content (but a modified header) and can still be successfully decrypted. Include the modified database file in your solution.

   (*Note*: Use a hex editor of your choice. The modified file can no longer be opened with KeePass, as the header is checked nowadays, but your program from task 1 should still work in the same way.)

f) AES is much faster than SHA-2 on modern CPUs, so repeating the AES in the key transformation seems to make less sense than a good hash-based key derivation, for example based on PBKDF2 (see RFC 8018):

$$key = \text{PBKDF2-HMAC-SHA256}(password, master\_seed, transform\_rounds, 32 \text{ bytes})$$

   Implement an alternative key derivation with PBKDF2-HMAC-SHA256 and measure the number of iterations (*transform_rounds*) with which a key derivation would take about 1 second on your CPU. How does this change your estimates from subtask (c)?

Appendix A. Description of the KeePass 2 database format

Appendix A.1. Structure

KeePass database files are structured as follows:

1. Signature (each field has 4 bytes):

`0x9AA2D903`	`0xB54BFB67`	`<Version>`

2. List of header fields. Each header field has the following structure:

`ID` (1 byte)	`Length` (2 bytes)	Data (`<Length>` bytes)

   There are the following header fields (other fields can be ignored):

ID	Description
0	`end of header` (the encrypted database starts after this header field.)
4	`master seed`
5	`transform seed`
6	`transform rounds`
7	`encryption initialisation vector` (IV)
9	`stream start bytes` (the first 32 bytes of the decrypted database)

Note that header fields can appear in any order until an `end of header` field appears.

3. The data stream encrypted with AES-256 in CBC mode. (The derivation of the key is explained below).
   - The first 32 bytes of the decrypted data stream are randomly selected bytes and are copied into the header field `stream start bytes` when the file is created.
   - If the content of this header field matches the decrypted data, the password is correct.
   - This is followed by the actual database (a compressed XML document), but this is not relevant for this homework.

Note that on Intel CPUs we have little-endian byte order: the 32-bit hexadecimal value 0x9aa2d903 is encoded in memory (and thus also in the file) by the byte sequence 03 D9 A2 9A.

Appendix A.2. Key derivation

The key *key* for the AES-256 CBC encryption of the database is derived from the password as follows:

$$\begin{aligned} \mathit{credentials} &= \text{SHA-256}(\text{SHA-256}(\mathit{password})) \\ \mathit{transformed\_credentials} &= \text{SHA-256}\left(\text{AES-256}^{\mathit{transform\_rounds}}_{\mathit{transform\_seed}}(\mathit{credentials})\right) \\ \mathit{key} &= \text{SHA-256}(\mathit{master\_seed} \parallel \mathit{transformed\_credentials}), \end{aligned}$$

where $\parallel$ denotes string concatenation and the notation

$$\text{AES-256}^{t}_{K}(m)$$

means that we apply the block cipher AES-256 with the key K in ECB mode t-fold to m, i.e. for $t = 2$ this means $\text{AES-256}_K(\text{AES-256}_K(m))$.