COMPUTER SECURITY, 2023/2024

LABORATORY EXERCISE 1: SYMMETRIC CRYPTOGRAPHY
11.03.2024.

INTRODUCTION

When users pick different strong passwords for different services, there is a need for a tool that will securely store those passwords, so they do not have to be memorized by the user. In this assignment, you will design and implement a *password manager* – a simple tool that uses symmetric cryptography for secure storage of user passwords.

FUNCTIONAL REQUIREMENTS

The tool needs to provide the following functionality to the user:

1. Initializing an empty password store.
2. Storing a pair *address, password* in the password store. If there is already a stored password for the given address, it should be replaced with the new address.
3. Retrieving a password stored for a given address.

The password manager should be a command line tool that stores the data on the disk and uses symmetric cryptography to guarantee the confidentiality and integrity of stored data. The data should be protected by a *master passphrase* specified by the user when using the tool.

An example interaction with the tool is as follows:

```
$ ./tajnik init mAsterPasswrd
Password manager initialized.
$ ./tajnik put mAsterPasswrd www.fer.hr neprobojnAsifrA
Stored password for www.fer.hr.
$ ./tajnik get mAsterPasswrd www.fer.hr
Password for www.fer.hr is: neprobojnAsifrA.
$ ./tajnik get wrongPasswrd www.fer.hr
Master passphrase incorrect or integrity check failed.
```

You may assume that the addresses and the passwords will consist of at most 256 non-whitespace printable ASCII characters (ASCII codes between 33 and 126 inclusive).

SECURITY REQUIREMENTS

We assume a *threat model* where the attacker can access and modify all the data stored on the disk. Hence, if the tool stores data directly on the disk with no protections, an attacker can easily retrieve all the stored passwords. Therefore, the security of the system must somehow rely on the master passphrase. Notice that the attacker can, over time, collect multiple versions of the files used by the password manager tool. Also, it is possible that the user will store *address, password* pairs chosen by the attacker. Assuming such threat model the tool needs to meet the following security requirements:

1. *Confidentiality of passwords*: the attacker cannot determine any information about stored passwords, not even the length of a particular password, not even if two passwords for two different addresses are same, not even if a new password is equal to an old password for an address.
2. *Confidentiality of addresses:* the attacker cannot determine any information about addresses, except, perhaps, how many different addresses are stored.
3. *Integrity of addresses and passwords:* it is impossible that the user obtains a password for a given address unless it has previously stored exactly that password for that address. Pay attention to a *password swap attack* – the attacker should not be able to swap a stored password with a different stored password.

TASKS

1. Using publicly available literature, research the notion of a *key derivation function* (KDF). Investigate the security guarantees provided by the KDFs and how KDFs are used to derive cryptographic keys from passphrases.
2. Design and briefly describe a password manager tool that meets the described functional and security requirements. Document how the data is protected before storage to disk and how the protection is verified when reading from disk. Document the procedures of generating keys from the master passphrase.

IMPLEMENTATION

You can implement the tool using a programming language of your choice. We recommend, and support either Python or Java. You need to make sure that the teaching staff can run and use your tool using standard compilers and interpreters on an Ubuntu Linux 22.04 system.

For the Java programming language, we recommend that you use standard cryptographic libraries that are included with Java SE 11, for Python we recommend the pycryptodome library. Below, we link the documentation for cryptographic primitives that you may find useful while designing and implementing the tool. Not all these need to be used in order to completely solve this assignment.

Symmetric encryption:

- https://docs.oracle.com/en/java/javase/11/docs/api/java.base/javax/crypto/Cipher.html
- https://pycryptodome.readthedocs.io/en/latest/src/cipher/cipher.html

Message authentication code:

- https://docs.oracle.com/en/java/javase/11/docs/api/java.base/javax/crypto/Mac.html
- https://pycryptodome.readthedocs.io/en/latest/src/hash/hmac.html

Cryptographic hash function:

- https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/security/MessageDigest.html
- https://pycryptodome.readthedocs.io/en/latest/src/hash/hash.html

Cryptographic random number generator:

- https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/security/SecureRandom.html
- https://pycryptodome.readthedocs.io/en/latest/src/random/random.html

Key derivation function:

- https://docs.oracle.com/en/java/javase/11/docs/api/java.base/javax/crypto/SecretKeyFactory.html
- https://docs.oracle.com/en/java/javase/11/docs/api/java.base/javax/crypto/spec/PBEKeySpec.html
- https://pycryptodome.readthedocs.io/en/latest/src/protocol/kdf.html

SUBMISSION

You need to submit a zip archive containing:

- Source code of your tool.
- Instructions for compiling and running your tool, ideally in a form of a *shell script* that will automatically build and run your solution in a way that demonstrates the required functionalities.
- Text file containing the description of your tool. In several sentences, describe your design and argue why it meets the security requirements above.
- The archive *must* not contain binaries, helper libraries or anything else not necessary for running your tool.

The regular deadline for submitting your solution is **24.03.2024. 23:59**.

If you are, for any reason, unable to submit the solution by the regular deadline, you are required to submit the solution by the extended deadline -- **31.03.2024. 23:59**. Valid solution submitted after the regular, but before the extended deadline will score 0 points, but will enable to student to satisfy the minimum requirements for the Laboratory exercises. If a solution is not submitted, this translates to an automatic failing grade for the Course.

In case of problems or questions, please contact the teaching staff in a timely manner (well before the submission deadlines) on the course mailing list srs@fer.hr. Always use your fer.hr email address for communication with the teaching staff.

Important: It is allowed and even desired that students discuss the problem and different approaches for solving the assignment. However, we require that all students work completely individually on their solutions. The teaching staff will check the similarity of submitted solutions. Any suspected plagiarism or other violations of the FER's Code of Conduct will be reported to the disciplinary committee, with possible additional consequences within the scope of this Course.

FREQUENTLY ASKED QUESTIONS

- *How and where to store the master password?*

  The master password (or its hash) should not be stored anywhere. Consequently, it is not required or possible to differentiate the case where an incorrect master password was used from the case where the integrity of stored data was compromised.

- *What happens if the tool is initiated twice?*

  Pick a reasonable behavior. For example, initialization can erase all data and start over with a new master password.