TECHNISCHE HOCHSCHULE MITTELHESSEN

CAMPUS GIESSEN

MNI Mathematik, Naturwissenschaften und Informatik

Bachelorarbeit

Server-Dependent File Access Systems
Definition, Model Implementation, and Analysis

zur Erlangung des akademischen Grades

Bachelor of Science

vorgelegt dem
Fachbereich Mathematik, Naturwissenschaften und Informatik
der Technischen Hochschule Mittelhessen

Lars Schmitt

im Mai 2023

Referent: Prof. Dr. Ing. André Rein

Korreferent: Prof. Dr. Steffen Vaupel

Eidesstattliche Erklärung

Hiermit versichere ich, die vorliegende Arbeit selbstständig und unter ausschließlicher Verwendung der angegebenen Literatur und Hilfsmittel erstellt zu haben.
Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungsbehörde vorgelegt und auch nicht veröffentlicht.

Gießen, den 1. Mai 2023

Acknowledgment

I want to thank Professor Rein and Professor Vaupel for supporting me, both during my studies and in writing this thesis. Furthermore, I thank my friends and family for their support. Last but not least, I thank Ján Mojžiš for providing me with access to his helpful paper about memory dumping attacks against Microsoft's SDFA clients [1].

Abstract

This work provides a clear, technical definition of Server-Dependent File Access (SDFA) systems and compares it to the definitions of Enterprise Resource Management (ERM) and Digital Rights Management (DRM) systems. Furthermore, a prototype implementation of a general-purpose model SDFA system written in Python 3 is provided to underline the concepts and potentially help developers to implement production-ready SDFA systems. Moreover, several different attack vectors against SDFA systems are discussed. The author concludes that SDFA systems are a valuable tool against several attack types and that the most effective attacks against SDFA systems can be performed by malicious insiders, using both internal weaknesses and external devices.

Contents

1 Introduction

1.1 Background

Information security aims at the protection of information from external threats. If information is stored digitally, the task of protecting the information from all kinds of threats is carried from the physical to the digital realm. The properties of digitally-stored information change the potential attack scenarios dramatically. Among other things, electronic data can be transferred and replicated easily. This is a clear advantage for regular use. However, it also opens up a wide variety of potential threats to information. Subsequently, the main security goals of information security are summarized in the so-called CIA Triad (see section 2.1), which consist of confidentiality, integrity, and availability (CIA).

When it comes to threats affecting electronically-stored data, the loss of sensitive data proposes a major risk to private people, but also companies and other organizations. However, “data loss” is an ambiguous term as it can describe both the unavailability of data (affecting the third CIA goal) as well as a breach of confidentiality (affecting the first CIA goal), which in some literature [2, 3] is also referred to as “data leakage” to avoid confusion. Therefore, Data Loss/Leakage Prevention (DLP) refers to the specific prevention of such a breach of confidentiality, even though it is often confused with measures to protect the availability (like backup systems).

While in personal environments, the amount of data, the number of users, and the need for secure data transmission to external parties is limited, corporate environments face a multitude of problems accomplishing the CIA goals for all their data, both internally, as well as externally. Many organizations use databases, email servers, and Enterprise Resource Planning (ERP) systems for their everyday business. However, many also try to improve their workflow using dedicated tools to manage and share files, roll-out software to endpoint devices, improve their security posture or optimize their business processes. This increases the complexity of their infrastructure.

1.2 Motivation

The abstract introduction of data classification is a strategy to limit data access. It is meant to improve information security by preventing members of an organization from accessing and distributing data assets of increased value. However, the complexity of modern companies' digital infrastructure (as well as human error) can lead to sensitive data being treated in a wrong way. Not long ago, the German train company Deutsche Bahn was allegedly sabotaged with confidential information that was publicly available on the internet, leading to train cancellations for several hours [4]. This basic example underlines the importance of correct and consistent data classification, but also the strong need to enforce restrictions tied to a sensitivity class.

One technical solution to solve several problems regarding information security are systems that provide advanced controls and technical measures for sensitive files. Depending on the business application, such systems are commonly referred to as Enterprise Resource Management (ERM) systems, Information Rights Management (IRM) systems, or Digital Rights Management (DRM) systems. Microsoft provides a particularly interesting line of ERM services called Rights Management Service (RMS) that assures confidentiality, even when an attacker copies or exports a sensitive file. This concept is highly interesting and deserves closer inspection.

1.3 Problem Description

While there are several definitions to describe systems that can help with the protection of data assets in company environments, there is no definition that concerns the exact technical functionality in a descriptive way. Furthermore, while work exists on security aspects of single systems, there is no work known to the author that focuses on such systems in general regarding their security aspects, their effectivity, or how to construct them from scratch. This makes it difficult to discuss the properties and abilities of such systems in a general matter. At the same time, discussing their use cases, abilities, and limits is highly relevant to potential users, organizations, and security professionals.

1.4 Research Questions

The goal of this work is to provide a definition and discussion of Server-Dependent File Access (SDFA) systems. Furthermore, a basic yet functional (Python 3) prototype model implementation of such a system will be provided and discussed. Moreover, the general use case for such systems will be outlined from both the defensive and the offensive side.

Different attack scenarios are to be applied theoretically to the idea of such systems and the effectiveness in defense against these systems will be discussed. Subsequently, different points of view (such as corporate employee, attacker, forensic specialist, ...) are going to be implicitly considered. Finally, the strong suits and weaknesses of such systems will be summarized and future work will be suggested. By that, the following research questions will be answered:

1. How can an Server-Dependent File Access (SDFA) system be defined?
2. How does an SDFA system work and what security features does it use and provide?
3. How does a simplified SDFA model compare against an existing productive system like Microsoft's RMS?
4. Against what forms of attack do SDFA systems provide an effective solution?
5. Against what scenarios do SDFA systems fail to provide a sufficient protection?
6. How can an organization treat sensitive data using an SDFA system?
7. How will the future availability of quantum computers with a significant amount of qubits affect the security of SDFA systems?

1.5 Contributions

This work aims to contribute a definition of Server-Dependent File Access (SDFA) systems. This definition will consider existing definitions of Rights Management Service (RMS), Enterprise Resource Management (ERM) systems, as well as Digital Rights Management (DRM) systems. Furthermore, a simplified prototype implementation of an SDFA system, based on the concepts of Microsoft's RMS will be provided. The differences between the prototype implementation and Microsoft RMS will be outlined. Subsequently, both the concept of SDFA systems and the model implementation (with its features and limitations) will be analyzed and faced with different attack scenarios. The future challenges, including cryptographic attacks towards the used cryptographic systems as well as the potential danger of post-quantum cryptographic attacks, will be considered.

2 Technical Background

2.1 CIA Triad

An asset in the context of IT security is something worth protecting. Assets can be physical items like a computer or a secret blueprint, but they can also be digital (like customer data or a piece of software) or abstract (like concepts or processes). For assets, security goals can be defined. The CIA security goal triad (CIA Triad) is the industry standard model for computer security and used in information security [5]. It consists of the goals confidentiality, integrity, and availability. The goals are also referred to as CIA goals. Different attacks and threats aim at different of these security goals. While some attacks such as Denial of Service (DoS) attacks only affect one of the security goals (the availability in this case), some attacks manage to overcome several or all security goals. The CIA Triad is often extended by further security goals, depending on the context. As an example, while in a web- and communication context, the goal of authenticity is often added, the goal of non-repudiation is sometimes used in the domain of digital forensics.

2.2 Cryptography

2.2.1 Symmetrical and Asymmetrical Cryptography

Cryptographic systems (also referred to as cryptosystems) aim at the obfuscation of information in a way that a crucial amount of the information is transferred into a key. The basic premise is that the key holds the information necessary to access the original piece of information. Vice versa, following Kerckhoffs's principle, without the key, the information cannot be accessed, even if the cryptosystem itself and the obfuscated information are known. In other words, access to the entire original information depends solely on the key. The transformation from the original information (also called plaintext) into the obfuscated version (also called ciphertext) is called encryption. The process of restoring the plaintext using the ciphertext and the (correct) key is called decryption.

For symmetric cryptosystems, the key used for encrypting and decrypting information is the very same. With the encryption function E and the decryption function D, for symmetric encryption, the formula for any supported plaintext m and the key k is:

$$\begin{gathered} c = E(m, k) \\ m = D(c, k) \\ m = D(E(m, k), k) \end{gathered}$$

However, for asymmetric encryption, two different keys are required to each encrypt and decrypt a certain plaintext. These keys are mathematically connected, but not the same. The key that is used for encryption is commonly referred to as public key whereas the key used for decryption is called private key:

$$\begin{gathered} c = E(m, k_{public}) \\ m = D(c, k_{private}) \\ m = D(E(m, k_{public}), k_{private}) \end{gathered}$$

While asymmetric encryption has many advantages in different use cases, it also has some downsides. Asymmetric encryption is usually much slower than symmetric encryption. Furthermore, the key generation process can cost many computing resources, and using cryptographic block modes (see section 2.2.4) to extend the supported message size is very uncommon. Therefore, when dealing with larger amounts of data in an asymmetric cryptosystem, a layered approach is used to encrypt the randomly generated key of the symmetric encryption with the asymmetric public key. For the encryption, this means:

$$\begin{gathered} k_{symmetric} \mid \text{randomly generated} \\ c_1 = E_{symmetric}(m, k_{symmetric}) \\ c_2 = E_{asymmetric}(k_{symmetric}, k_{public}) \\ c = (c_1, c_2) \end{gathered}$$

For the decryption of m, the counter operations must be executed in reverse order:

$$\begin{gathered} c_1, c_2 = c \\ k_{symmetric} = D_{asymmetric}(c_2, k_{private}) \\ m = D_{symmetric}(c_1, k_{symmetric}) \end{gathered}$$

This concept is implemented for the request communication in the later discussed model implementation (see figure 4.5).

Symmetric cryptosystems are also referred to as private key cryptosystems (because they only have a private key). Equally, asymmetric cryptosystems are also called public key cryptosystems.

2.2.2 Hash Functions

While hash functions (also hashing functions) fall into the domain of cryptography, they do not provide encryption or decryption functionality. Hash functions are one-way-functions that are used to create unique, fixed-length hash values of data (of any length). The resulting hash values are also called hashes or digests. These digests are irreversible due to the nature of hash functions, and no key is required. For a message m, the hash digest h can be calculated with the function H:

$$h = H(m)$$

Hash values are often used to verify the integrity of information as described in the CIA goals. The uniqueness of hash values has a special role in this scenario. When two different messages result in the same hash value, this is defined as a collision. Collisions are hard to find for modern hash functions. However, finding a collision or a way to generate them can result in significant problems for the cryptosystem using the hash function. Depending on the use case, not only the integrity, but also the confidentiality can be affected.

2.2.3 Certificates

A certificate is a construct that can be used to sign and verify data in addition to encryption and decryption. For certificates of known actors, this provides the additional goal of authenticity. Among other data, certificates contain an asymmetric cryptosystem with a public and a private key. However, the process of signing or verifying information works differently than the usual process of encryption or decryption in a public key cryptosystem. In the case that an actor with a certificate wants to sign an information, the information is (usually) hashed. The resulting hash is then encrypted with the private key (instead of the public key). The receiving actor can then use the (known) public key to decrypt the hash and check the integrity of the information by comparing the decrypted hash value with a self-calculated hash value of the information. The process of creating a signature s for the message m can be defined as:

$$h = H(m)$$
$$s = E(h, k_{private})$$

The process of verifying the information on the receiving end can be described as:

$$h = D(s, k_{public})$$
$$h_{calculated} = H(m)$$

If the calculated hash ($h_{calculated}$) is the same as the received one (h), the integrity of the message m is considered verified. Note that the content of the message m is not encrypted in this case, so if desired, confidentiality must be ensured by another layer of encryption. It is also important to note that a matching signature only means the message came from a trusted source when the public key itself is trusted; a signature itself only provides information about the integrity of the message, not the identity of the source. Therefore, a signature does not prevent an attacker from performing a Man-in-the-Middle (MitM) attack during the retrieval of the public key. Only the trusted identity of the certificate holder of the used public key can create authenticity as well. One potential solution for this problem is the usage of a Public Key Infrastructure (PKI) with certificates only issued by a trusted Certification Authority (CA).

2.2.4 Cryptographic Block Modes

Cryptosystems are divided into stream ciphers and block ciphers. Stream ciphers iterate over single bits of the plaintext while block ciphers require a certain number of bits to encrypt or decrypt at once. Therefore, to encrypt a plaintext with a block cipher, the plaintext must be divided into blocks of a size supported by the cryptosystem. With a block size of x bits, as long as the plaintext is longer than x, the first (or next) x bits are taken as a block. When the remaining plaintext is shorter than x, a padding scheme is applied to complete the last block. In case that a plaintext has length n and $n\%x = 0$, the text can be perfectly split into blocks in theory. However, in some cases, a full block of the selected padding scheme is added to mark the end of the message. In any case, the resulting data blocks (each with the length x) can be encrypted with the block cipher.
When all the generated blocks are each simply encrypted with the cipher, this is called the Electronic Code Book (ECB) mode. This is the most intuitive way of encrypting data with a block cipher. However, it also raises major problems. When encrypting the same block several times, the lacking level of information obfuscation achieved by this mode is highly problematic. Furthermore, there is no way of detecting errors when encrypting the blocks independently from each other. More complex modes like the Cipher Block Chaining (CBC) mode use additional operations (like XOR) to solve both problems. However, this also opens up the possibility for side side channel attacks, like the CBC mode padding oracle attack that (for instance) can be observed in the domain of SSL/TLS encryption [6]. Modern modes like the Galois/Counter mode (GCM) are even more complex in their functionality, using several layers of operations, additional data, and hashing functions. However, they also provide a greater set of features to the one using them, like verifying additional unencrypted data. The latter is known as Authenticated Encryption with Associated Data (AEAD) [7].

2.3 Data Classification

Data classification is a process of information security that assigns a class to certain data. This can be done based on the sensitivity of the data (which is probably the most common classification attribute), but it can also happen based on other factors (such as availability requirements). As suggested by George Firican [8], for the scope of this paper, the distinction between data categorization and data classification is defined in the relationship between the category or class and the data. While a piece of data can be a member of several categories, it can only belong to one class. This can be represented by defining the category-data relationship as n-to-n, while the class-data relationship can be defined as 1-to-n. Subsequently, categorized data can be seen as an intermediate step between unclassified and classified data. In the field, categories and classes are often represented as labels, which also associate the member data with a set of rules that need to be followed. These can be both technical measures (such as encryption or content marking) or behavioral rules (like limited access, disposal regulations, or others).

2.4 Microsoft Products

2.4.1 Active Directory

Microsoft Active Directory (AD) is a directory management service that is widely used. According to Nestori Syynimaa, in 2014, AD was used by about 95% of the so-called fortune 500 companies [9]. AD provides a multitude of different authentication methods and services and can be deployed on premise. It is integrated into different versions of Microsoft's Windows operating system (OS) since Windows 2000 and Windows Server 2003 [10], but is also increasingly supported by Linux distributions. AD is the basis for many other services, including but not limited to Azure Active Directory (Azure AD) and Microsoft Rights Management Service (RMS).

2.4.2 Azure

Microsoft Azure is a cloud service platform that can be used in many ways. Among many other services and features, it also enables companies to host an instance of Active Directory in the Azure cloud environment. This is referred to as Azure Active Directory (Azure AD).

2.4.3 Microsoft 365

Microsoft 365 (M365) [11] is a Microsoft software product family for both private and corporate customers. Depending on the version, different services are included, the most important of them being the classic office applications like Microsoft Word, Microsoft PowerPoint, and Microsoft Excel. Further services reach from Microsoft's communication platform Microsoft Teams to complex business and security services, such as PowerBI or Azure Information Protection (AIP). It is notable that the term Microsoft 365 is somewhat wide-spanning, especially since the assimilation of the former Microsoft Office 365 (O365) product line into the namespace of Microsoft 365 that happened in April 2020 [12].

2.4.4 Microsoft Rights Management Service

Microsoft Rights Management Service (RMS) is a family of Enterprise Resource Management systems (defined in section 3.1.2). In the context of Microsoft Azure, it is part of Microsoft Azure Information Protection (AIP). It is both available on local Active Directory instances (AD RMS) and instances of Azure Active Directory (Azure RMS) [2]. The distinction between AD RMS and Azure RMS is significant when analyzing potential security threats against the security goals of Enterprise Resource Management (ERM) services (see section 5.1.2). However, Azure Rights Management Service is a part of Azure Information Protection (AIP), which can be found in some versions of Microsoft 365 subscriptions. A list of definitions and services included in the namespace of AIP is provided by Microsoft [13].

3 Server-Dependent File Access Systems

3.1 Related Definitions

3.1.1 Digital Rights Management

Digital Rights Management (DRM) systems are systems that aim to provide limited access to a piece of (physical or digital) media for a user, mostly a customer. They are mostly used to prevent a user from replicating and distributing content that the user is allowed to access for himself, but not to share with other users. Nowadays, DRM systems are commonly used in streaming services such as Netflix, which uses its DRM to prevent users from extracting (and potentially sharing) the movies and series played. However, also local media such as DVDs are protected with DRM measures. DRM systems are often used in the specific relationship between a customer and a service provider, and often protect the service provider from copyright infringement. Depending on the exact definition, DRM can include a variety of measures (like watermarking) that are not necessarily associated with encryption [14].
The Amazon Kindle file access system (one of the systems mentioned by Grothe et al. [15]) is one example of DRM protection. While both the cryptographic cipher and the Kindle (SDFA) client have been broken in the past and are prone to the concept of all-or-nothing protection [16] (which is discussed in section 5.1.1), Amazon is keen to protect the copyright of the books provided to the customers using DRM.

3.1.2 Enterprise Resource Management System

Enterprise Resource Management (ERM) systems protect company data by encrypting and digitally signing it [2]. One synonym for ERM systems, which is removing the term from its necessarily corporate background and putting it into the context of Information Protection, is Information Rights Management (IRM) systems [2].
One example of ERM systems is Microsoft Rights Management Service (RMS). Microsoft RMS (not to confuse with Microsoft Dynamics Retail Management System) span several products and licenses, covering ERM functionality across different scenarios and configurations. In the Microsoft Azure context, RMS is part of the Azure Information

Protection (AIP) Suite. However, RMS services can also be used outside of the Azure cloud in a different AD environment. Across hosting platforms, the ERM functionality is based on labels. These labels can statically represent a sensitivity class, but they can also be used to give the user ad-hoc control over which rights and limitations are assigned to an object, depending on the label configuration. The labels are then published in a policy and can then be applied to different data objects such as files, Outlook emails, groups, or schematized data assets. When it comes to files, Microsoft's main focus lies on Microsoft's native office file formats like ".docx", ".pptx", or ".xlsx". However, while some file formats like ".exe" are also purposefully excluded, more file formats like ".pdf" and ".png" are supported as well [17].

3.1.3 Rights Management Services

Rights Management Service (RMS) is a term mostly defined by Microsoft to describe their ERM services (see section 3.1.2).

3.2 Definition

Unfortunately, the terms listed above as well as their common synonyms are insufficient to describe the specific kind of system this works aims to discuss. The relatively wide array of measures that can be defined as part of a DRM system results in the term DRM being somewhat unspecific from a technical point of view. Moreover, the previously mentioned definitions of ERM/IRM and DRM systems focus on the context in which they are applied. Nonetheless, judging by their functionality, ERM systems can be used both for internal, as well as external users (both employees and customers), depending on the exact use case. Furthermore, ERM systems are sometimes defined based on their cryptographic functionality [2], but the term doesn't provide a clear description of the involved parties and their role in the system. To discuss the properties of Microsoft RMS-like ERM systems without focusing too much on a certain product family, a more general definition of such systems is required. Therefore, the following definition of Server-Dependent File Access systems provides a more technical description of the functionality of these kinds of system, independent of their precise business use case.

For the scope of this work, Server-Dependent File Access (SDFA) systems will be defined as systems that:

- protect digital files from unauthorized access, even if the entire protected file is obtained by an unauthorized party
- use a client-server architecture, in which only the server can grant access to a protected file
- provide users with the ability to set specific access rights for entities like users and groups
- ensure that the server never obtains the body of the protected files

Crucially important to this definition and the protective capabilities of SDFA systems is that the SDFA server itself (or a different one of the same providing organization) never stores or even sees the file content (not counting the restriction-representing metadata in the container). Furthermore, what really distinguishes SDFA systems from other definitions is that even if an attacker obtains the desired asset in its permanently stored state (including encrypted body and access rights), the attacker still cannot use the asset without server permission. On the disk, the body of the file only exists on the client side in an encrypted format. Nevertheless, the server decides whether the client has access to a file at all and what this access looks like for an authenticated user. An abstract activity diagram over a data asset's lifecycle in an SDFA environment is provided in figure 3.1.

It can also be argued that other assets like database entries are also admissible targets for SDFA systems since they are also stored in files. Also, additional server features do not necessarily violate the definition of SDFA systems. Such features could include file location tracking, access monitoring, or content-specific access denial. However, the latter must be exercised with caution. However, the server is not supposed to have or store any information about the unencrypted file content itself. For instance, while the server-side storing of file content's hash values would be "just" a potential attack vector, further information about the content of the files would violate the SDFA client-server data separation requirements.

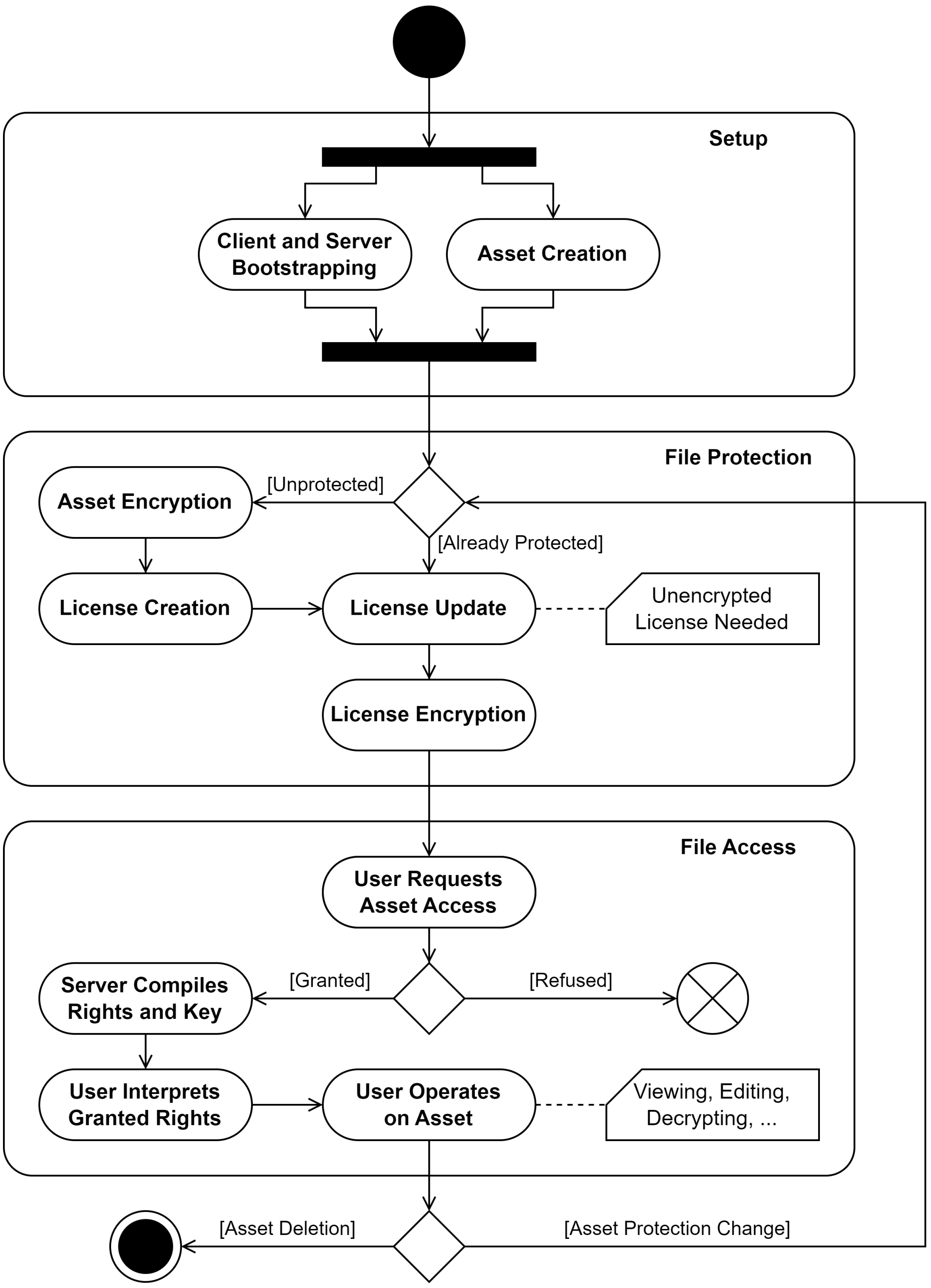

Setup
Client and Server Bootstrapping
Asset Creation
File Protection
Asset Encryption
[Unprotected]
[Already Protected]
License Creation
License Update
Unencrypted License Needed
License Encryption
File Access
User Requests Asset Access
Server Compiles Rights and Key
[Granted]
[Refused]
User Interprets Granted Rights
User Operates on Asset
Viewing, Editing, Decrypting, ...
[Asset Deletion]
[Asset Protection Change]

Figure 3.1: Abstract activity diagram of the lifecycle of a data asset in an SDFA system.

3.3 Use Cases

SDFA systems can be used in different environments with several users to technically instantiate data classes. Use cases for this reach from very small companies not wanting an intern to access certain sensitive documents to large organizations with a complex system of authorized groups, users, compartments, and security levels. The following steps could be taken by an organization that wants to use an SDFA system:

- **Definition of Categories**: In a first step, the organization has to think about what kind of data it holds and which categories of data deserve protection. Categories can be derived from both internal sensitivity (secrets, confidential information, etc.) and from external factors such as laws regulating the security requirements for certain categories of data (such as personal identifiable information, credit card information, passwords, health information, and so on).
- **Definition of Classes**: From the list of categories, classes can be created. These classes should have the type of data (respectively the categories) and information attached to them, who (inside or outside the organization) should be able to access certain information. As an example, a company could decide to only make their department of human resources able to access contractual data.
- **SDFA Implementation**: The classes are technically instantiated in the selected SDFA by creating templates for the defined classes and the attached behavior. If supported, employees are assigned group memberships to represent extended or limited rights over certain classes of files. Certain trusted users are assigned extensive rights.
- **SDFA Usage**: During business operation, the templates are applied to the files and other data assets. Restricted access for those assets is enforced throughout the organization and to the outside world. Templates, group memberships, user rights, and file permissions are changed dynamically.

The previous example covered the case of organizations and can be applied to any ERM system. However, since the definition of SDFA systems in theory could also include DRM systems that work in a similar way (and respect the four points of the definition), it is important to point out that the same steps can also be used for much different kinds of relationships. A streaming service for instance could design its categories based on the products offered. A licensed piece of media could for instance be accessible to all employees of the IT department, but also to customers that (internally) are members of a group that represents access to this asset. This is also an example of the fact that SDFA systems can be used for ERM and DRM functionality at the same time. Overall, schools, foundations, governmental organizations (including but not limited to military and intelligence organizations), online communities, online-class providers, and a multitude of other institutions could benefit from the usage of SDFA systems. In addition, SDFA systems can be useful on many types of devices. As an example, an SDFA-driven access right revocation would be a quick, effective, and reversible first step for a service that aims at data protection for lost devices.

While without a doubt the main advantages of SDFA systems lie in environments with several users, in special cases, it can also be advantageous to use an SDFA system as a single user. This is the result of the SDFA server being a Single Point of Failure (SPOF). While this intuitively poses a risk to the availability of assets protected with an SDFA system, it can also be an intended weakness, as access to any amount of protected files can very easily be revoked. In this context, the usage of an SDFA system with a "kill switch" for the server can even be seen as a potentially effective anti-forensic measure concerning secret data [18]. However, even without the case of single user access revocation to prevent external access, SDFA systems can be useful to prevent the access of unauthorized third parties. An example would be a burglar (as attacker), obtaining copies of sensitive files from a victim. If the victim protected the files with an SDFA system (no matter the access rights for authorized users), the attacker would not be able to access the files without a compatible SDFA client as well as the ability to authenticate as an authorized user.

3.4 Existing SDFA-Like Systems

Many existing ERM solutions only focus on providing users and groups with granular access to files and directories. They do not fall under the definition of SDFA systems because if an attacker manages to obtain control over a file (for instance by copying it to an external hard drive), the protection is null and void. However, on the other end of the spectrum, DRM systems often fail to comply with the definition of SDFA systems as they neither provide users with the ability to set the rights (by class-based labeling or ad-hoc), nor a clean separation of hosting service and SDFA service. While the Amazon Kindle DRM does not fulfill the requirements of an SDFA system, it arguably falls under the definition of an SDFA-like DRM system. The reason the definition is not fully fulfilled is that the user does not set the rights and there is no clean separation between hosting service and SDFA service. However, as for any pure DRM system, the risk of combining the file hosting service and the SDFA system is given in theory, but it is not relevant since in cases of DRM protection, the confidentiality towards the hosting service is not a desirable goal. Therefore, this aspect should not be considered when discussing SDFA-like DRM systems as opposed to "real" SDFA systems with a strong focus on ERM functionality.

On the other hand, Microsoft's RMS (which in some versions can also be used for DRM-like functionality) is the main and prime example of existing and productive SDFA systems. While it is important to note that there are cases in which Microsoft is in control of both the file hosting and the SDFA system - which in theory proposes a severe security risk as explained in section 5.1.2 - the protection service environment is among the most successful implementations of customer-oriented ERM systems as of now. Micrsosoft's RMS is the only example of a fully compliant SDFA system known to the author if hosted on premise (AD RMS). Subsequently, it is also the main orientation point for the following SDFA prototype implementation. As for the functionality of Azure RMS, useful information is provided both by Microsoft [19] and external sources [20, 21].

4 Model Implementation

4.1 Concept

4.1.1 Purpose

The SDFA implementation discussed is meant to be a non-productive model implementation. For reasons outlined in section 4.5, the implementation is not meant to be used in any form of productive environment. The sole purpose is to provide insight on how such a system can be implemented as well as an easy-to-understand code example. With the implementation, the concept of SDFA can be better understood, as well as details regarding its behavior in different attack scenarios, discussed in section 5.

4.1.2 Derived Goals

The goals of the model implementation have been defined as follows:
The purpose of the implementation is to provide a general-purpose prototype of an Server-Dependent File Access (SDFA) system (as defined in section 3.2). The used definition of a prototype is defined in Mary Shaw's 2003 paper on software development papers [22]. The implementation should be easy to understand and extendable. The project should have a working client application as well as a working server providing main functionality and demonstrating the concept of SDFA. The implementation should provide protection functionality for all types of files, as well as the operations *view*, *edit*, *protect*, and *decrypt*. While not aiming to be a production ready implementation by any means, basic safety and security practices should be applied. Common technology standards and libraries should be used. The usage of non-Python code should be minimized. Performance is almost entirely disregarded to improve simplicity and readability. The project should follow the object-oriented paradigm.

4.1.3 Terminology

The following terms have to be defined in order to understand the provided SDFA implementation:

- **Posting License:** A Posting License (PL) is an object that defines the access different users and groups have to a protected file. A PL contains the rights over a file in an encrypted and signed format, as well as the body key to access it. The PL in the prototype is the equivalent to Microsoft's Publishing License.
- **Server Certificate**: The Server Certificate (SCert) is the asymmetric certificate representing the server's identity. It is responsible for encrypting the PLs and the double-encrypted communication to the server. It is the equivalent of Microsoft's Server Licensor Certificate (SLC).
- **Container**: A container is a file that contains a file that is protected with the prototype SDFA. In a container that is persistent on the disk, the original file is only present in an encrypted state, as well as the key stored in the PL (status 4). However, containers can be in less secure states (1-3) at runtime. Containers have the standard file extension ".container" in addition to the original file name and extension.
- **User Certificate**: The User Certificate (UCert) is the asymmetric certificate representing the user device's identity. It is responsible for encrypting the double-encrypted communication from the server to the client. It combines the functionality of the Microsoft user certificates Secure Processor Certificate (SPC), Rights Account Certificate (RAC), and Client Licensor Certificate (CLC).
- **Content/Body Key**: The content key (also body key) is a randomly generated symmetrical key used to encrypt the body of protected files. The key itself is stored in the PL. When a container is protected, the PL is encrypted itself with the SCert.
- **List of Rights**: A List of Rights (LoR) is an entity-based list of permissions that is compiled by the server to grant client access. It is encrypted with the requesting user's UCert, along with the content key, and sent back. Microsoft calls this construct a Use License (UL).

4.2 Implementation Process

4.2.1 Methodology

The following goals for the implementation process have been defined:
For the development, a prototype will be created, serving as a proof of concept and providing the author with useful experience to develop the main project. After the prototype has been finished, the actual model will be implemented. For the development process, the waterfall model will be used. Agile approaches are rejected since the product requirements are static. UML-like diagrams will be used to support the development process.

4.2.2 Code Conventions

To achieve the goal of having an easily-understandable code, the code complies with the pep-8 style guide for Python [23]. Furthermore, type annotations with the module typing have been used, both to enhance the readability of the code, as well as to support the debugging process (visually and with static code analysis).

4.2.3 Used Software

The following software has been used to implement this project:

- **Python 3:** Python [24] is a general-purpose programming language that is defined by its focus on readability and outstanding support for external libraries and modules. It supports object-oriented programming and provides optional typing functionality. It works on all major operating systems and is sometimes automatically included in Linux distributions such as Debian and Ubuntu. Also, several dialects exist that can be run on compatible microcontrollers. Known downsides of Python are its lacking speed (which was ignored in this project) and the non-existing type safety (which was counteracted by using type annotations throughout the project).

- **pip:** Pip [25] is the package manager of Python that can be used to download and install external modules and packages.

- **PyCharm Professional:** PyCharm [26] is an Integrated Developing Environment (IDE) by JetBrains. For this project, the professional version was used with a student license.

- **VSCodium:** VSCodium [27] is an IDE supporting several programming languages. It is forked from Microsoft's Visual Studio Code, removing tracking functionality.
- **diagrams.net client:** The diagrams.net client [28] is the offline version of the web service, allowing one to create and export a wide variety of different diagrams. It was used for the creation of diagrams supporting the development process, as well as the diagrams used in this work. Diagrams.net is also known as draw.io.
- **mypy:** Mypy [29] is a type checking software for Python, compatible with the typing annotations used in this project.
- **DB Browser for SQLite:** DB Browser for SQLite [30] is a cross-platform, open-source manipulation tool for sqlite databases with a Graphical User Interface (GUI).
- **Wireshark:** Wireshark [31] is a powerful network analyzer that has been used to verify and debug the communication of the model implementation.

4.2.4 Used Technology Standards

The following technology standards have been used in the project:

- **AES:** The Advanced Encryption Standard (AES) is a definition of a symmetrical block cipher encryption that can be used with several key lengths and modes of operation. It is the successor of the Data Encryption Standard (DES) as well as its variant Triple DES (3DES). The most common key lengths for AES are 128 and 256 bit. The latter will be used for this project. For file encryption, the CBC mode of AES will be used. For the client-server-communication, the Galois/Counter mode will be used, additionally wrapped by a layer of asymmetric certificate encryption.
- **RSA:** Rivest-Shamir-Adleman (RSA) is an asymmetric encryption algorithm named after its founders Ron Rivest, Adi Shamir, and Leonard Adleman, who proposed it in 1978 [32]. It is one of the most widely used asymmetric encryption systems today and can be used with several key lengths. While other key lengths are possible, the most common ones are 1024, 2048, and 4096 bits. Its security is based on the principle that it is relatively easy for a computer to multiply large (prime) numbers, but it is not easy to reconstruct them from their multiplication result. Nowadays, this premise is under attack by new approaches to factorize numbers efficiently (see section 5.5.1). However, as of now, RSA is considered safe and runs on many different device types.

- **HTTP:** Hypertext Transfer Protocol (HTTP) is the most widely used protocol for data exchange in the world wide web and is used in the model implementation for the client-server communication. It is extended by Hypertext Transfer Protocol Secure (HTTPS), a wrapper around HTTP that extends its encryption with SSL/TLS certificates. However, this is not used in the model implementation. Furthermore, HTTP supports a basic authentication scheme with username and password [33] which in the model implementation is used to authenticate users at the server.

- **JSON:** JavaScript Object Notation (JSON) is a data notation format which is sometimes seen as a successor of Extensible Markup Language (XML) in the web context. It is faster than XML [34] and supported in HTTP-requests for data exchange. However, its main feature is that data objects can be easily represented in a JSON format, allowing to easily export and import data objects in several languages.

- **JWT:** JSON Web Token (JWT) is a standard defining web tokens that can be used for authentication and authorization of clients on a server. It is defined in rfc7519 [35] and implemented in libraries across several programming languages [36]. JWTs are generated and signed by a server, based on an internal private key. The payload can not be modified by the user without breaking the integrity of the signature. When used, JWTs are sent with HTTP-requests to ensure the user's identity and carry additional information that can be verified by the server, using a range of different signing functions.

- **Argon2:** Argon2 is a modern password hashing algorithm that is recommended since it won the Password Hashing Competition, running from 2013 to 2015 [37,38].

- **SHA 256/512:** General-purpose hash algorithms of the Secure Hash Algorithm (SHA) family are defined in the Secure Hashing Standard (SHS) FIPS 180-4 [39]. While SHA-1 and its predecessor MD5 should no longer be used [40], other hash algorithms like SHA-512 and SHA-3 are recommended. The model implementation uses several SHA-family algorithms like SHA-256 and SHA-512 for general-purpose hashing; passwords are hashed with Argon2 instead.

4.2.5 External Libraries

The following external libraries have been used in the project:

- **Flask:** Flask [41] is a Python library for the creation of (mainly RESTful) web servers and APIs. In this project, it is used as the base for the server application.

The server itself is a Flask application, using the database package in object format. Flask natively uses decorated functions for its route definitions.

- **requests:** Requests [42] is a Python library for sending HTTP-requests. The client configuration file handler module binds the required requests into methods that are used by the client backend while also dealing with the authentication with web tokens, respectively credentials.
- **Python-RSA:** Python-RSA is a pure Python implementation of the RSA algorithm [43] that is used to provide cryptographical functionality to the certificates, including generation, encryption, decryption, signing, and verification.
- **pyAesCrypt:** PyAesCrypt [44] is a pure Python module that encrypts and decrypts data with AES (256) in CBC mode. The resulting cipher texts are compatible with the AES Crypt file format [44]. In the model implementation, PyAesCrypt is used for the content encryption of files.
- **PyCryptodome:** PyCryptodome [45] is a Python package for several cryptographic purposes. It includes several AES modes, Elliptic Curve Cryptography (ECC), hashing algorithms, and cryptographically safe random number generators (CSPRNGs). In the model implementation, it is used to encrypt and decrypt request data with AES GCM.
- **argon2-cffi:** Argon2-cffi [46] is a Python library that implements Argon2, the used password hashing algorithm on the server side.
- **PyJWT:** PyJWT [47] is a Python module that supports the generation and verification of JSON Web Token (JWT), used for user authentication after the initial authentication with credentials.
- **psutil:** Psutil [48] is a Python module for operating on operating system (OS) processes. In the model implementation, it is used to determine the number of threads the system CPU has, which is used for optimized RSA key generation.
- **universal-startfile:** The module universal-startfile [49] implements the functionality of the os.startfile command for operating systems other than Windows (Windows is supported as well). It is used to ensure cross-platform compatibility in the generic file handler.

4.2.6 Notable Python Features

Python is a programming language with some remarkable syntactical shortcuts and features. Two of them are particularly worth mentioning as they provide an easier understanding of how the project code works.

Context Managers help to ensure that when a certain context is left, post conditions are fulfilled. The most commonly used case is the open context manager, which guarantees that an opened file is closed, even if an error (inside the context) occurs:

```
with open("test.txt", "w") as opened_file:
    opened_file.write("content")
    raise Exception("artificially raised exception")
# the file is properly closed after the context, despite the exception
```

A custom context manager is defined in the server database handler. It ensures the database file is in a usable state, even when a database error occurs. A code example for the usage is:

```
def retrieve_id_from_name(self, name: str) -> str:
    assert name
    result: str
    with DBCursor(self.database_file) as cursor:
        cursor.execute(f"SELECT id FROM entities WHERE name=?", (name,))
        result = cursor.fetchone()
    return result[0] if result is not None else ""
```

Decorators are the second Python feature that proved to be especially useful in the project and are commonly used by Flask [41]. They are based on the principal that in Python, functions, methods, and lambda expressions are all objects of the type callable. As such, they can be passed on as parameters. Decorators can make use of this concept by treating a function as an object and extending (or otherwise modifying) its functionality. The most notable usage of decorators in the project is the “needs authentication”-decorator for server routes. The simplified implementation is:

```
def needs_authentication(function: Callable):
    def wrapper(*args, **kwargs):
        try:
            user_id = verify_user_id(request.headers) # pseudo function
        except Exception:
            abort(403)  # "forbidden"
        try:
            return function(user_id, *args, **kwargs)
        except Exception:
            abort(500) # "internal server error"
    return wrapper
```

If the user is authenticated, the verified user ID is provided to the function as an additional parameter as seen in this example of a server route:

```
@app.get("/profile", endpoint=generate_endpoint())
@needs_authentication
def send_userid(user_id: str):
    return respond(f"You are authenticated as {user_id}", user_id)
```

In early versions of the project, decorators were also used for debugging functionality (runtime measurement, function stack tracking). In the project, they are mainly used for exception management and the server routing functions.

4.3 Result

4.3.1 Structure

The file structure of the project is outlined in figure 4.1. The main components of the project are Python packages (white), Python applications (black), Python packages used

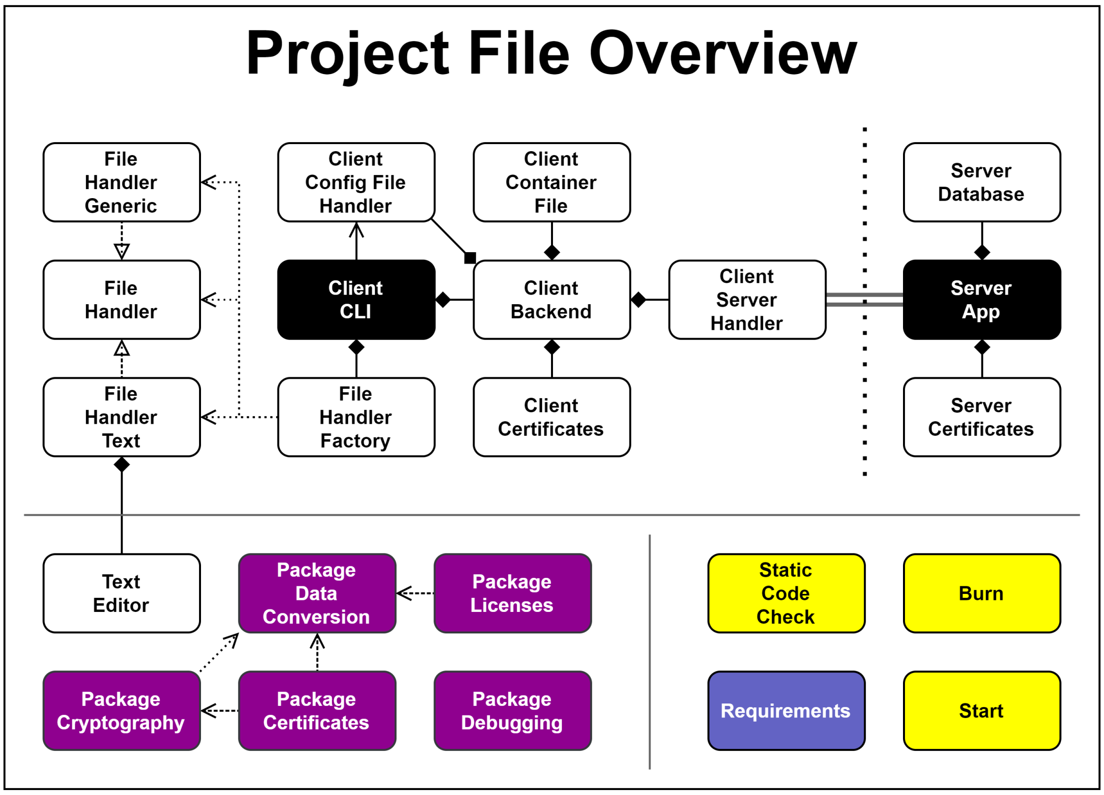


Figure 4.1: File overview of the SDFA prototype implementation project.

by both the client and the server (purple), installation files (blue), and development-supporting scripts (yellow). The components of the projects are roughly discussed in the bullet points below; the full Python code as well as the installation files and the burn scripts can be found in the appendix.

- **Client CLI:** The client Command Line Interface (CLI) is an interactive frontend prototype application, enabling the user to use the client to perform operations on files. These files can be both plain or protected.

- **Client Backend:** The client backend bundles all functionality of the SDFA client in a class that can be used by various backends. It is also able to perform the client bootstrapping process, generating a client certificate, authenticating at the server, and creating all necessary configuration files for operation.

- **Client Config File Handler:** The client configuration file handler file provides a class to the backend that is used to handle the credentials and necessary configuration data of the client. This includes their generation when the client is being bootstrapped, but also the retrieval during normal operation.

- **Client Container File:** The client container file provides the definition of client container files, which is crucial to the SDFA client functionality. The definitions include the loading, generation, encryption, and decryption of container files while checking preconditions and removing original plain files after protecting a container.

- **Client Certificates:** The client certificate file uses the certificate package to define server and client certificate functionality on the client side. While treating the own certificate as full certificate with full cryptographic functionality, the server certificate is treated as a public certificate with limited functionality due to the unknown private key.

- **Client Server Handler:** The client server handler file provides a class for the client backend that is able to communicate with the server. The main abilities of the client server handler lie in its ability to send authenticated requests, either using an existing JWT or obtaining one with authentication credentials.

- **File Handler Factory:** The file handler factory file contains a class that selects the correct file handler based on a given container. It uses all available definitions of file handlers.

- **File Handler:** The file handler file provides an abstract definition of a file handler. File handlers are responsible for providing file-driven operations (*view*, *edit*) to the user for the file type of the original file in a container.

- **File Handler Generic:** The file handler generic file provides a default implementation of a file handler for all unknown file types. It uses the standard programs of the operating system as well as the super class' functionality to create unencrypted temporary files for consumption and editing. However, this proposes an additional security risk as mentioned in section 4.5.7.

- **File Handler Text:** The file handler text file provides an example for a specific file handler for utf-8-compatible files. Editing a file is supported by the usage of a GUI text editor while viewing the file results in a simple console output.

- **Text Editor:** A Tkinter-based text editor for text-like files (of which the content is utf-8 compatible). The editor is roughly based on a forum post by martineau and wolf [50].

- **Server App:** The server app is the main server application, bundling and providing all functionality of the SDFA server. The first time it is called (or after a reset), it also performs the server bootstrapping process, generating a server certificate, a database with a standard admin, and all necessary configuration for operation.

- **Server Database:** The server database file contains classes that provide the server with database functionality. The most important class is the database handler class. The model implementation uses sqlite for a simple database file in which all permanent operational data are stored.

- **Server Certificates:** The server certificate file uses the certificate package to define server and client certificate functionality on the server side. While treating its certificate as full certificate with full cryptographic functionality, the client certificates are treated as a public (half) certificates with limited functionality due to the unknown private key.

- **Package Data Conversion:** The data conversion package is responsible for converting different data formats into each other throughout the project as well as a function with decorating capabilities to deep-copy data objects. The string formats supported are base64, hexadecimal bytes, JSON, and utf-8. Furthermore, plain bytes and dictionaries are supported.

- **Package Cryptography:** The cryptography package bundles all cryptographic operations used in the project into classes. The most important definition is the one of cryptosystems, which is inherited and extended by the different encryption and decryption methods, namely AES (CBC and GCM) and RSA.

- **Package Certificates:** The license package provides the class definitions for full and half certificates. The “full” and “half” name refers to the cryptographic abilities; while a half certificate is only able to perform operations with a given public key, the full certificate can also perform operations with the private key. Both types of certificates are further implemented in the client and server certificate files.

- **Package Licenses:** The license package provides important classes for both client and server. The classes include the definition of permission objects, rights objects, Posting License (PL) objects, and a static permission merger.

- **Package Debugging:** The debugging package is used for debugging purposes during development. It provides functionality in the form of function decorators that provide information about the function call.

- **Static Code Check:** The static code script uses mypy to perform a static code analysis of the project, which can help identify bugs and type inconsistencies during development. It is not OS-independent.

- **Burn:** The scripts starting with the prefix “burn” remove all configuration files and data associated to the client and or the server being bootstrapped on the current system. Therefore, they can be used to “start over” the bootstrapping processes; it could be said that the scripts “unbootstrap” the project. Unlike the Python program(s) and text file(s), these scripts are no longer OS-independent. The “burn”-scripts are provided in the appendix.

- **Start:** The scripts starting with the prefix “start” generate a process of one or more applications of the project and are used for development. They might include additional features by calling one of the burn scripts. Unlike the Python program(s) and text file, these scripts are no longer OS-independent.

- **Requirements:** The requirements file is a pip-compatible text file listing all external requirements that need to be installed in order to run the model SDFA prototype. While there is no distinction between client and server requirements, comments are used to describe the purpose of each requirement. The requirements file is provided in the appendix.

4.3.2 Program Overview

An abstract overview of the provided SDFA system’s steps to prepare both server and client for operation (bootstrapping), to protect files, and access protected files is provided with figure 4.2. The general steps are as follows:

1. **Server Bootstrapping**: The server creates or loads the database (see figure 4.6) and the Server Certificate (SCert) (see figure 4.9).

2. **Client Bootstrapping**: The client creates or loads the User Certificate (UCert) (see figure 4.9), authenticates at the server, fetches the public part of the SCert, and registers the new UCert at the server.

3. **Protection**: The client loads an existing file. If the file is not a container, internally, a container of status 3 is created (see figure 4.4). The client then verifies that the user has the right to protect the file. If that is the case, the Posting License (PL) is compiled, signed with the UCert, and encrypted with the SCert.

4. **File Access**: The client extracts the encrypted PL from the container, encrypt it, and sends it to the server. The server decrypts and verifies the PL. After that, the server evaluates the rights and compiles a List of Rights (LoR) that concerns the user. If the user has any rights, the content key is added. If the user has the rights to change the protection of the file, the unencrypted PL is added. After selecting the right UCert (using user and device information), the server responds with the respective rights. The client then interprets the response and applies the rights and restrictions to the file, allowing or disallowing the file specific user operations. These can be *view*, *edit*, *protect*, or *decrypt*.

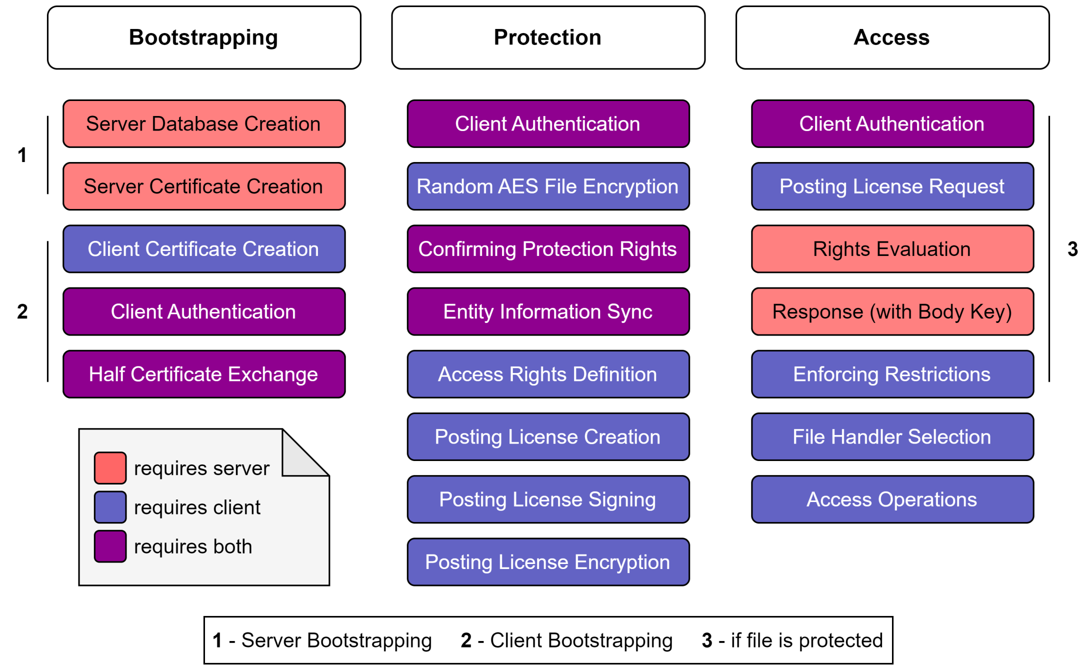
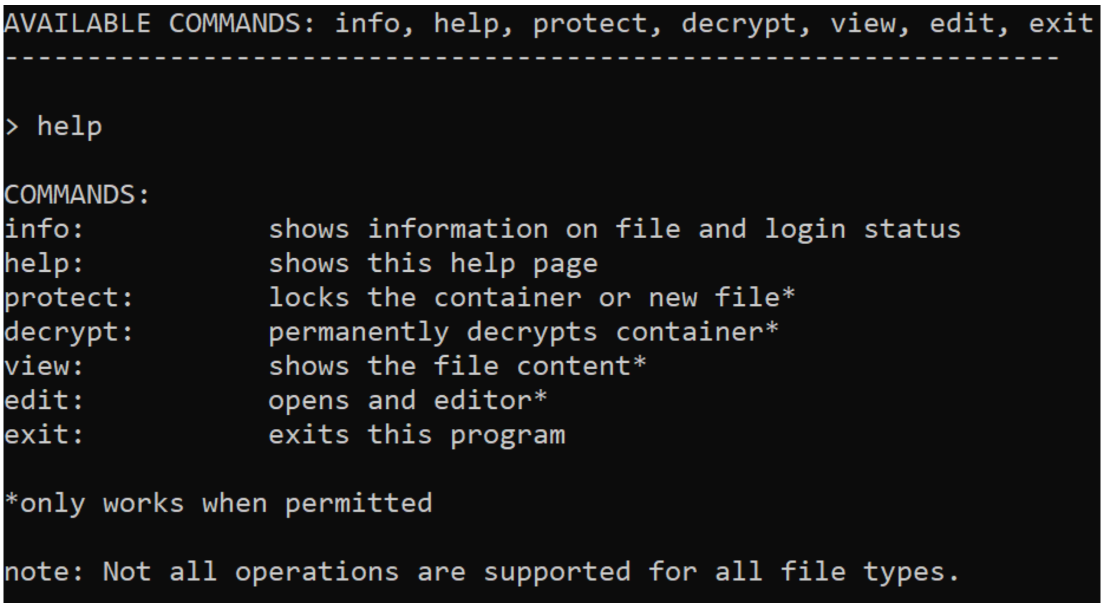


Figure 4.2: Overview of the necessary steps of the bootstrapping process, protecting containers, and accessing them.

4.3.3 Program Flow

First of all, the server must be started. Upon first launch, the server generates a dummy database with two test users and a few example groups, representing sensitivity classes. Furthermore, the server certificate is generated and saved. After that, the HTTP-server itself is launched on the local machine (localhost) on port 11235. Subsequently, the client can be initialized by running the CLI on a machine for the first time, generating and registering the client certificate at the server. This only works if the user already exists on the server and can authenticate with valid credentials (user name and password). By default, the client is set to run on the same machine as the server. However, the server location can be defined by changing the standard configuration in the client configuration file handler, respectively the server location in the client configuration file. Also, to try out several users, the project main folder (in its uninitialized state) can be copied and bootstrapped separately to simulate several users.
After bootstrapping, the CLI can be used by calling it with the parameter of the file that should be operated on. The client determines the protection level of the file, fetches information about the current users and groups from the server, and provides a list of supported commands to the user (see figure 4.3), including the main commands *info*, *help*, *protect*, *decrypt*, and *exit* as well as further commands enabled by local or server permissions and executed by the respective file handler (*view* and *edit*). The command *protect* opens up a sub-menu that allows the generation or modification of the Posting License which contains all user and group rights on the file.
With this set of operations available, users known to the server can register on devices



```
AVAILABLE COMMANDS: info, help, protect, decrypt, view, edit, exit
-----------------------------------------------------------------

> help

COMMANDS:
info:           shows information on file and login status
help:           shows this help page
protect:        locks the container or new file*
decrypt:        permanently decrypts container*
view:           shows the file content*
edit:           opens and editor*
exit:           exits this program

*only works when permitted

note: Not all operations are supported for all file types.
```

Figure 4.3: Screenshot of the commands available in the provided command line interface client.

and protect files. The client certificates are local, but files created on different devices can be accessed nevertheless as the verification step is executed by the server (who knows all active public user certificates) and the body key is encrypted with the respective user certificate during transmission. Launch instructions for testing the model implementation are provided in section 4.3.8.

4.3.4 File States

The file format with which SDFA protection can be achieved is called container. A container file has the extension ".container" in addition to its original file extension and contains both the body of the file as well as the Posting License (PL), owner ID (no effect on rights as of now), container version number, and the original file location. The state of encryption for the body and the PL determines the file's inner state, shown in figure 4.4. State 1 represents a plain (non-container) file, state 2 is a container file with unencrypted body and PL, state 3 represents a container with an encrypted body but a plain PL, and state 4 is a fully protected container with encrypted body and encrypted PL. The states 3 and 4 are the only desirable ones, so status 1 files are converted into status 3 files internally. When a file is saved on the disk, it is either dumped into status 1 (unprotected plain file) or exported as status 4 (protected container). Status 5 and further could be reserved for future container versions supporting extended features,

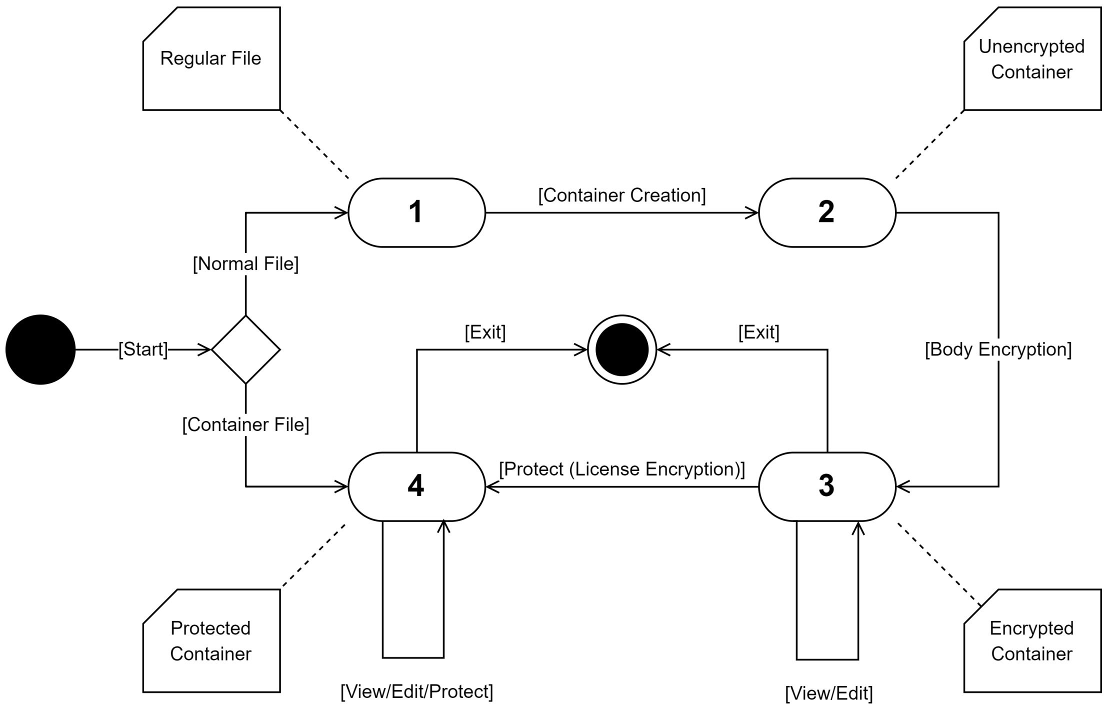


Figure 4.4: State machine diagram of the internal file states.

such as a protection with several keys for preview thumbnails. However, this is not yet included in the prototype implementation.

4.3.5 Communication

The project consists of a client-server structure. The server is a HTTP-server implemented with Flask [41]. Since Flask works mainly with decorators and doesn't naturally support server classes, the project's object orientation is lifted for the server application. However, in the project context, the server application can be seen as a static class with a static main method, using objects from other modules during runtime. While this is not consistent for all requests and responses exchanged between client and server, a double-encryption scheme is used to encrypt requests and responses concerned with the evaluation of PLs, respectively granted access rights (see figure 4.5) and represents layer 2 in the overall encryption strategy (see figure 4.7).

$$k_{symmetric} \mid \text{randomly generated}$$
$$c_1 = E_{AES}(m, k_{symmetric})$$
$$c_2 = E_{RSA}(k_{symmetric}, k_{certificate\_public\_key})$$
$$c = (c_1, c_2)$$

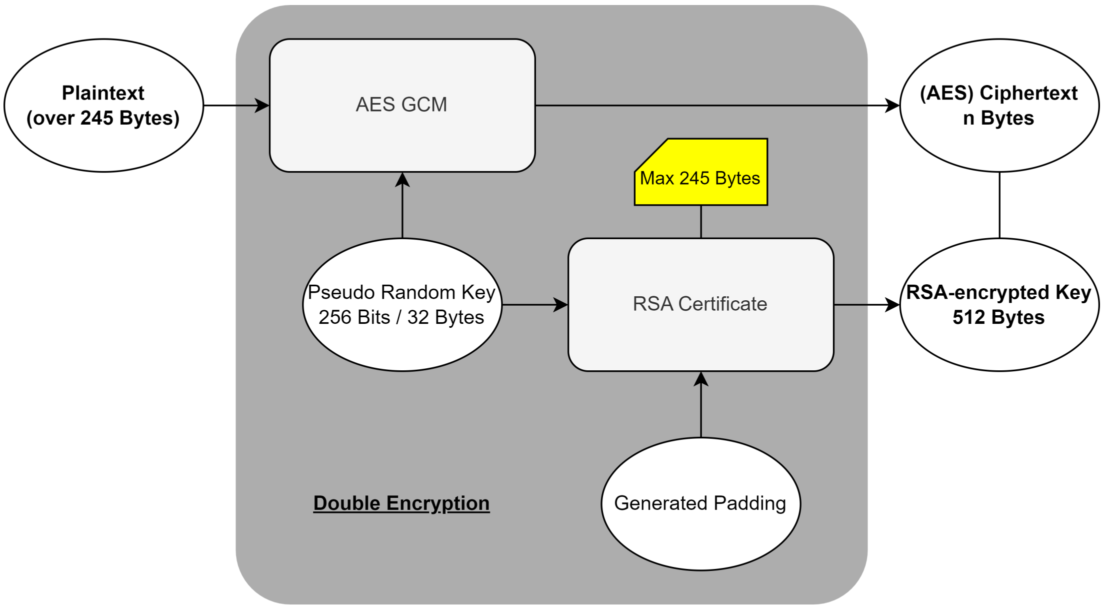


Figure 4.5: Diagram of the double-encryption scheme used to encrypt the requests and responses concerned with the evaluation of Posting Licenses.

On the receiving end, the double-decryption can be performed as follows:

$$\begin{gathered} c_1, c_2 = c \\ k_{symmetric} = D_{RSA}(c_2, k_{certificate\_private\_key}) \\ m = D_{AES}(c_1, k_{symmetric}) \end{gathered}$$

In general, this double encryption strategy allows it to encrypt plaintexts (indirectly) using an RSA certificate, but avoiding the maximum plaintext length. This is done by adding a layer of symmetric (AES) encryption and only encrypting the randomly generated symmetric key with an asymmetric (RSA) cipher.

4.3.6 Server Database

The server database is implemented using Python's native sqlite library. The structure of the database follows a simple approach with three tables (see figure 4.6), treating both users and groups as entities, defining group membership with a separate table, and storing the public part of user certificates in the third. While this is not part of the database, the identification of both groups and users that can have rights in the SDFA environment is defined with the randomly chosen entity ID stored in the entities table. Subsequently, a List of Rights (LoR) that is received by the user contains the entity ID of the authorized user or group and the respective access rights.

4.3.7 Encryption Layers

The project implements several layers of encryption, best shown with the client's main asset, the Posting License (PL) (the equivalent to a Publishing License in Microsoft RMS). The encryption layers of a PL are shown in figure 4.7. The first and most

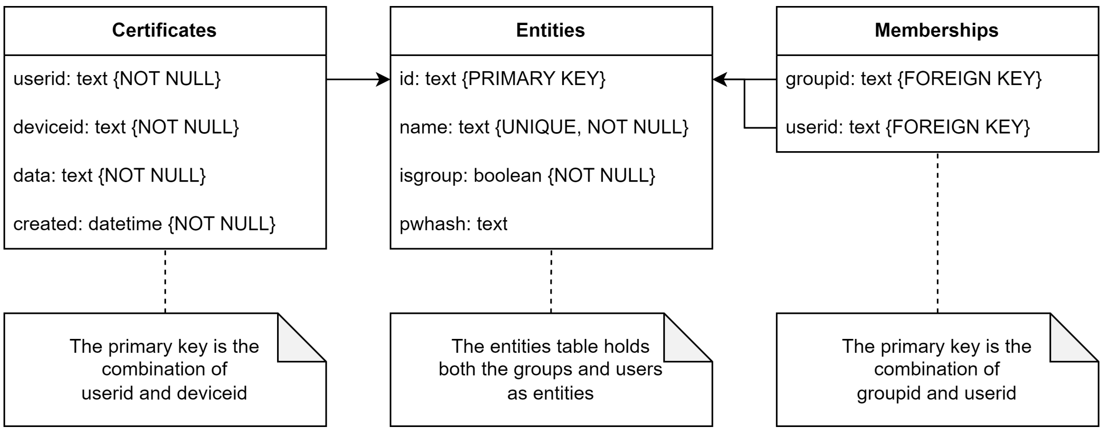

Certificates
userid: text {NOT NULL}
deviceid: text {NOT NULL}
data: text {NOT NULL}
created: datetime {NOT NULL}
Entities
id: text {PRIMARY KEY}
name: text {UNIQUE, NOT NULL}
isgroup: boolean {NOT NULL}
pwhash: text
Memberships
groupid: text {FOREIGN KEY}
userid: text {FOREIGN KEY}
The primary key is the combination of userid and deviceid
The entities table holds both the groups and users as entities
The primary key is the combination of groupid and userid

Figure 4.6: Database layout of the model implementation.

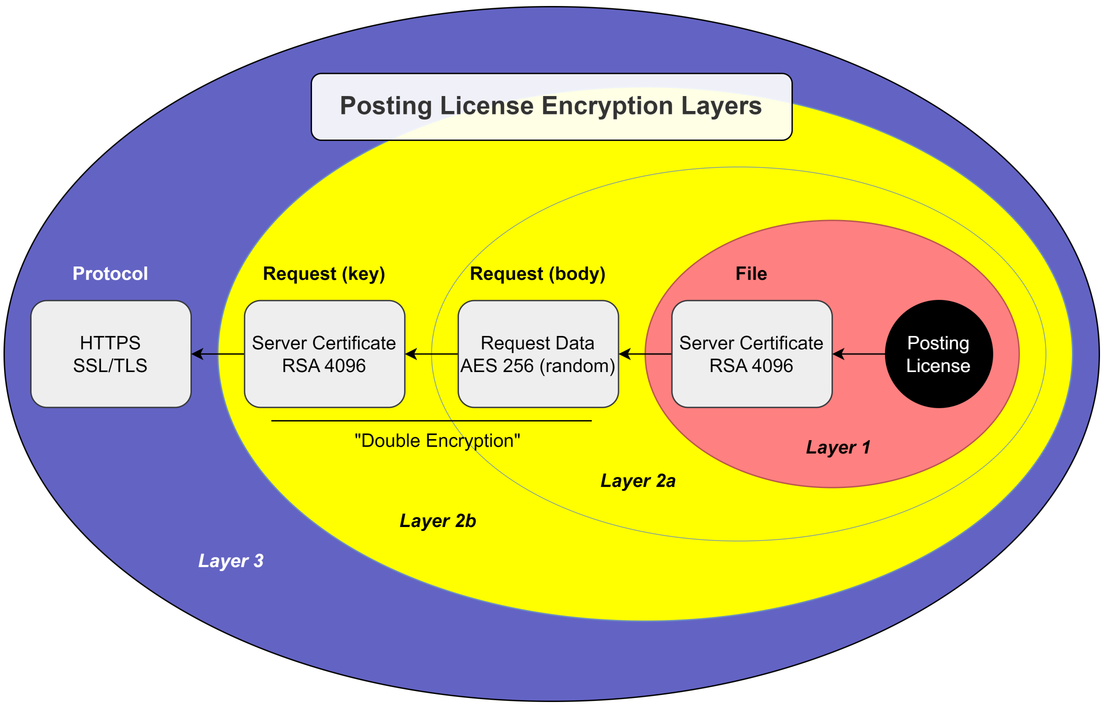
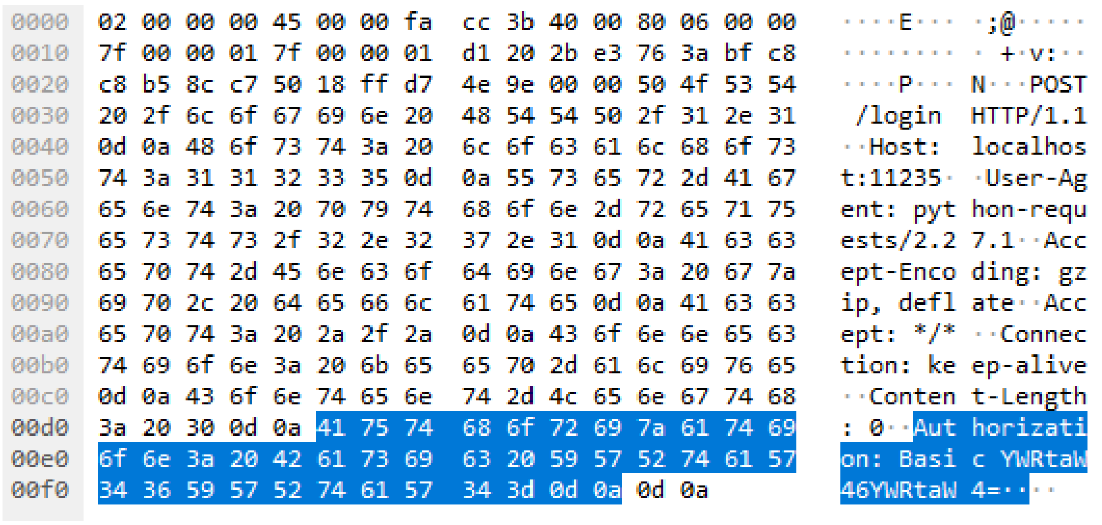


Figure 4.7: Encryption layers of Posting Licenses between client and server.

important protection layer is layer 1, consisting of the SCerts PL-encryption. With the PL decrypted, the content of the file can be retrieved (based on the encrypted body and decrypted body key) and all security goals are endangered. While being stored on the client's side, the layer 1 protection is sufficient to ensure the file's confidentiality. However, during data transmission, additional layers are necessary to protect the security goals regarding both the PL and the user. To ensure confidentiality and integrity, the data are encrypted in a two-stage cryptosystem, including a random AES key that is used to encrypt the request data in GCM and the client certificate, signing the content and encrypting the random AES key. For authentication, Argon2 and short-lived JSON Web Tokens are used. The authentication process is managed by the client server handler in the background. However, especially during the authentication, the user credentials, respectively an active JWT, are still exposed in the HTTP-request in plain text (see figure 4.8). Therefore, it is essential to add another layer of encryption on the communication level to protect the user's identity from Man-in-the-Middle (MitM) attackers. Subsequently, the usage of the latest SSL/TLS encryption version for the communication between client and server is crucial, even though this is not implemented in the model.



```
0000  02 00 00 00 45 00 00 fa  cc 3b 40 00 80 06 00 00   ····E··· ·;@·····
0010  7f 00 00 01 7f 00 00 01  d1 20 2b e3 76 3a bf c8   ········ · +·v:··
0020  c8 b5 8c c7 50 18 ff d7  4e 9e 00 00 50 4f 53 54   ····P··· N···POST
0030  20 2f 6c 6f 67 69 6e 20  48 54 54 50 2f 31 2e 31    /login  HTTP/1.1
0040  0d 0a 48 6f 73 74 3a 20  6c 6f 63 61 6c 68 6f 73   ··Host:  localhos
0050  74 3a 31 31 32 33 35 0d  0a 55 73 65 72 2d 41 67   t:11235· ·User-Ag
0060  65 6e 74 3a 20 70 79 74  68 6f 6e 2d 72 65 71 75   ent: pyt hon-requ
0070  65 73 74 73 2f 32 2e 32  37 2e 31 0d 0a 41 63 63   ests/2.2 7.1··Acc
0080  65 70 74 2d 45 6e 63 6f  64 69 6e 67 3a 20 67 7a   ept-Enco ding: gz
0090  69 70 2c 20 64 65 66 6c  61 74 65 0d 0a 41 63 63   ip, defl ate··Acc
00a0  65 70 74 3a 20 2a 2f 2a  0d 0a 43 6f 6e 6e 65 63   ept: */* ··Connec
00b0  74 69 6f 6e 3a 20 6b 65  65 70 2d 61 6c 69 76 65   tion: ke ep-alive
00c0  0d 0a 43 6f 6e 74 65 6e  74 2d 4c 65 6e 67 74 68   ··Conten t-Length
00d0  3a 20 30 0d 0a 41 75 74  68 6f 72 69 7a 61 74 69   : 0··Aut horizati
00e0  6f 6e 3a 20 42 61 73 69  63 20 59 57 52 74 61 57   on: Basi c YWRtaW
00f0  34 36 59 57 52 74 61 57  34 3d 0d 0a 0d 0a         46YWRtaW 4=····
```

Figure 4.8: Wireshark capture of a HTTP login-request, exposing the login credentials "YWRtaW46YWRtaW4=" (admin:admin) encoded in plain base64.

4.3.8 Launch Instructions

Please note that the author explicitly rejects any claims of liability for lost data caused by any usage of the model implementation or its code. The code is run at your own risk! The model implementation can be run on operating systems with Python 3 installed and added to the OS path. Furthermore, the project code and pip are required for the installation. Moreover, the Tkinter must be installed if it is not automatically included in the Python installation. On systems supporting the package manager aptitude, this can be done with the command `apt install python3-tk`.
The first step of the installation is to put the project files in their desired program directory. To install the required external modules, the requirements.txt file can be used with the command `pip install -r requirements.txt` to batch install the required modules. The usage of a virtual Python environment is recommended. Note that it might be necessary to use `pip3` instead of `pip` in the command. After that, the server can be started by running the server app with `python server_app.py`. The server will perform the bootstrapping by generating a dummy database as well as its certificate and then start its main functionality as web API server. After that, the client CLI can be called by using the command `python <path to client_cli.py> <path to file>`. Note that the command `python` might change, depending on the operating system type, installing location, and whether a virtual environment is used. The client will bootstrap itself by generating its certificate (UCert), its configuration file (config.json), and authenticating at the server to receive the public part of the server certificate (SCert). However, the username and password of the user must be entered during the bootstrapping process. The test users admin (with the password admin) and the user jondoe (with the password password) are included in the server's default database. After

the bootstrapping process, the client CLI will provide its main functionality, consisting of the commands available for the file that has been provided as an argument.

4.4 Evaluation

4.4.1 Development Process Evaluation

The goals for the implementation process defined in section 4.2 have been achieved. An unpublished prototype has been developed, providing the author with useful experience for the main implementation regarding many aspects, such as the database layout, the Liskov Substitution Principle (LSP) regarding the cryptosystem classes, a variant of the Singleton pattern for the client configuration file handler, object-reference issues, and the usage of decorators. UML-like diagrams have been used throughout the development, along with a waterfall-like approach. Static code checking has been used with mypy. The pep-8 convention has been checked by the IDE PyCharm. Typing has been used throughout the program. Common Python libraries have been used, such as Flask, requests, or PyCryptoDome. The project is almost completely object-oriented with the exception of the server application, which can be viewed as a static main class.

4.4.2 Validation

The prototype implementation has been successfully tested on Windows 10 as well as (Linux) Ubuntu Desktop 22. The testing has confirmed that the installation process, the bootstrapping process (for both client and server), the server operation, and the client protection features work for the tested files. Furthermore, it has been confirmed that an unauthorized user is not able to access a file due to client restrictions, respectively the server's denial to grant access with a LoR and the content key. Therefore, the prototype can be considered functional and a working example [22] for an SDFA system.

4.4.3 Result Evaluation

The goals for the implementation outcome defined in section 4.1.2 have been achieved. A working model implementation of a general-purpose SDFA system has been provided, along with structural resources that might help programmers of future SDFA systems to plan and implement their projects. The provided implementation can be considered easily-understandable and is extendable in several ways. Potential ideas for further development are (more) type-specific file handler implementations, an improved server that supports CRUD-complete user management, or better frontend clients that support a larger number of features like a Graphical User Interface (GUI) or predefined sensitivity classes. However, the model implementation can be used to explain the basic strengths and weaknesses of SDFA systems. The implementation provides the file operations *view*, *edit*, *protect*, and *decrypt* for all file types. It is once more important to note that while this is intended, the model implementation should not be used to actually protect sensitive data, as each safety and security of the implementation lack in many major regards.

4.5 Comparison With Microsoft RMS

4.5.1 Context

While the Microsoft RMS applications and plugins aim to provide production-ready, large-scale enterprise solutions, the provided implementation aims at being an easily-understandable model which is specifically not intended for production use. While Microsoft RMS and all related services are developed and maintained by Microsoft as one of the largest technology companies in the world, the provided model was developed in a relatively short amount of time, with different intentions in mind, and (apart from the used external libraries and the cited code references) developed by only one person. This is the context in which all of the following comparisons have to be seen.

4.5.2 File Support

With RMS, Microsoft primarily aims to support its own native family of office files. Therefore, naturally, the support for Microsoft Word, Microsoft PowerPoint, Microsoft Excel, and other Microsoft files has priority. Several other file formats are supported as well as some types of data assets. However, some file formats are also explicitly excluded from being supported by RMS.
On the other hand, the provided model implementation has some focus on specific file types (like utf-8-compatible text-like files) with the file handler class structure, but it is generally compatible with all file types. Also, files that end with the file extension “.container” are interpreted as protected container files and therefore cannot be protected again without changing the file extension.

4.5.3 Features

Compared to Microsoft RMS, the provided model implementation has a greatly reduced feature set. While viewing, editing, protecting, and decrypting functions are provided in both approaches, Microsoft RMS supports a larger range of permissions with finer granularity [51]. These range from the general ability to copy content from an opened file to very specific permissions, describing who can forward a protected email. More notably, data classification is supported by having predefined labels with associated protection functionality. Furthermore, in Microsoft RMS, more features are supported concerning the nature of the file use licenses (such as access expiration or temporary local licenses).

4.5.4 Scale

Azure RMS is part of the Microsoft Software-as-a-Service (SaaS) ecosystem that spans many services in different domains. It is run on data centers in many different countries while many local instances of AD and AD RMS exist. The scale and scope of Microsoft's RMS approach is almost not comparable to the provided model implementation. While a largely improved version of the model implementation could, in theory, be used by a small group of private users, it is most likely not capable to be a commercial product on its own due to a number of different factors, including but not limited to the choice of programming language, security issues, and lack of user-friendliness.

4.5.5 Performance

While the author conducted no speed comparison of RMS services and the model implementation, the latter has no ambitions to be a fast or seamless application. Python is commonly known as a relatively slow programming language, which is the price for dynamic typing, being interpreted, and other amenities. While Python might not be the best choice for a productive SDFA system implementation, it is known to be a good choice for explaining all kinds of concepts and algorithms, which is much closer to the goal of the model implementation.

4.5.6 Safety

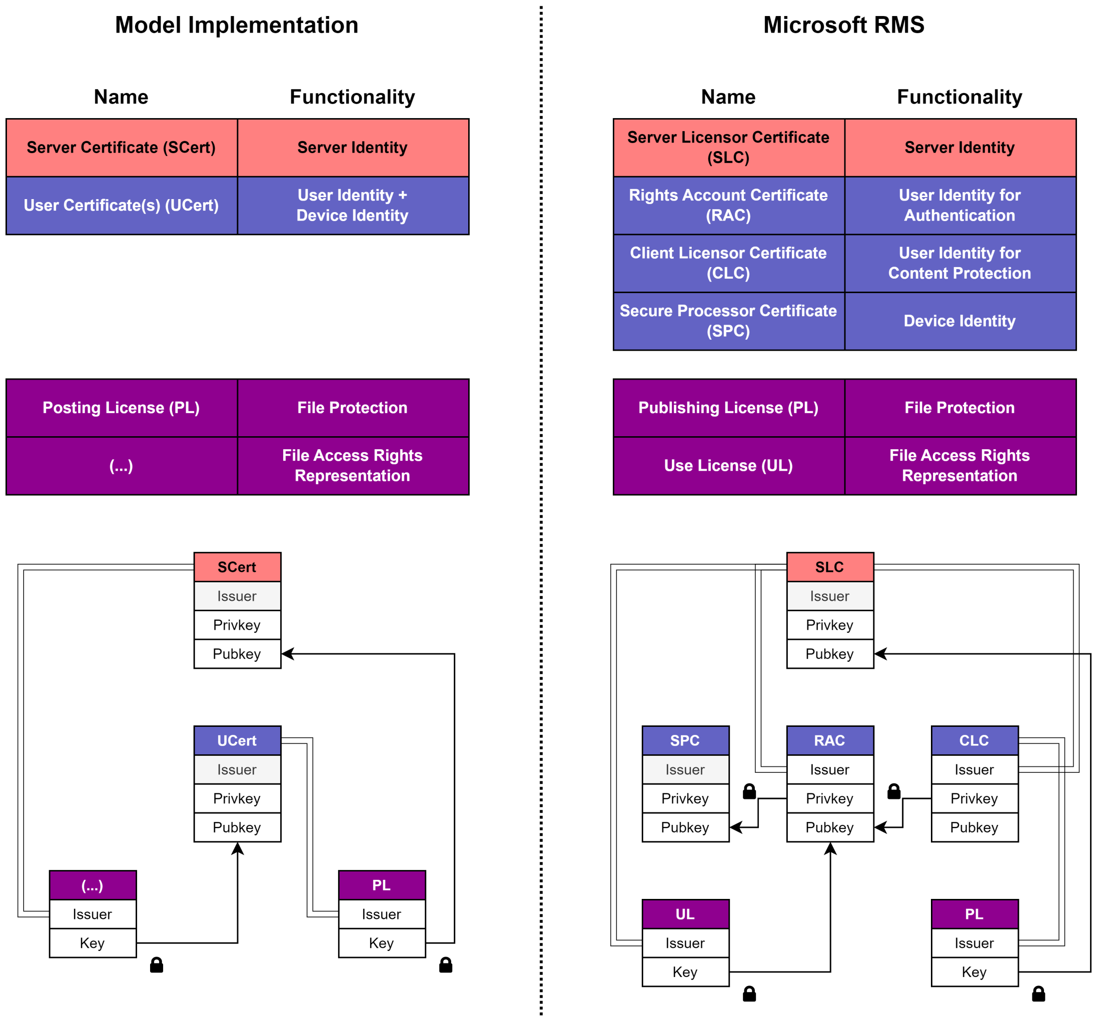
While the provided model implementation follows some basic good practices regarding safety, the model implementation in its current state is not safe to use for any data that is not backed up. The author explicitly rejects any claims of liability for lost data caused by the usage of the model implementation; the project code is used at own risk.

4.5.7 Security

While the provided model implementation follows some basic good practices regarding security (like the usage of injection-safe SQL statements), the model implementation cannot be considered secure for many reasons. To name only a few:

- The model implementation was never intended to be used in a productive environment. The main abstract reason for the general insecurity of the implementation is that simplicity and readability have been chosen over security on many occasions. One outstanding example for the simplified approach followed by the model implementation compared to Microsoft RMS is the license structure as demonstrated in figure 4.9. While the simplified version of the model implementation provides a better understanding, in a real-world implementation, a finer and PKI-compatible certificate structure is highly recommended.
- The model implementation has been developed by a single author over the course of a few weeks. It has not been reviewed by other people.
- During development, no strategies have been followed to make the implementation safe or secure in a provable matter.
- Several external libraries are used that - if compromised - would also undermine the safety of the model implementation (inherited vulnerabilities).
- The author is not qualified to professionally assess the security of cryptographic systems or code implementations, including the implementation and usage of the libraries used in the project.
- As pointed out by the PyAesCrypt documentation, “there is no low-level memory management in Python, hence it is not possible to wipe memory areas were sensitive information was stored” [44].
- Unlike Microsoft RMS or other ERP solutions, the model implementation does not provide the usage of advanced authentication methods like the one offered by an AD environment, but only provides a relatively unsafe authentication with HTTP (and short-lived JSON Web Tokens).
- Some security issues of the existing state of the implementation are known to the author, such as:
  - the openly exposed client configuration file which contains unencrypted user credentials
  - the potentially unsafe usage of binary files for object storage

  - the unencrypted transmission of the user credentials with login requests
  - the unencrypted transmission of the user ID and the device ID in requests (no level 2 protection)
  - the unencrypted temporary files created by the provided generic file handler



Model Implementation

Name	Functionality
Server Certificate (SCert)	Server Identity
User Certificate(s) (UCert)	User Identity + Device Identity
Posting License (PL)	File Protection
(...)	File Access Rights Representation

Microsoft RMS

Name	Functionality
Server Licensor Certificate (SLC)	Server Identity
Rights Account Certificate (RAC)	User Identity for Authentication
Client Licensor Certificate (CLC)	User Identity for Content Protection
Secure Processor Certificate (SPC)	Device Identity
Publishing License (PL)	File Protection
Use License (UL)	File Access Rights Representation

Figure 4.9: Comparison of the used certificates and licenses in the provided model implementation and Microsoft (Azure) RMS; based on Goldbergs' article on Azure RMS functionality [20].

5 Threat Scenarios

5.1 Specific SDFA Security Concepts

5.1.1 All-Or-Nothing Protection

One of the most remarkable conceptual weaknesses of SDFA systems derives from the concept of all-or-nothing protection. In the context of DLP, it is assumed that an organization has to deal with internal attackers who have access to sensitive resources. The protection against these attackers is an important goal of information security. However, on a conceptual level, an attacker can freely choose what to do with all information available. For the concept of SDFA systems, if a client has the (means to obtain the) body key of a file, there is nothing stopping an internal attacker from using this key to act on the protected assets in a malicious way, and therefore ignoring existing restrictions. Apart from the client's implementation, further technical restrictions an organization might have in place and the good will of people entrusted with sensitive assets, there is nothing that stops an internal attacker from a privilege escalation by abusing an obtained content key. In the current SDFA approaches (including the model implementation presented, but also Microsoft RMS), the only real way to stop a yet unidentified internal attacker from performing this kind of privilege escalation is to deny all access. This means that no local access exists and the SDFA server refuses to provide the client with the content key. To put the situation in a nutshell on a conceptual level, the attacker either has any permission, resulting in a potential content key abuse and all permissions during an attack, or no permissions, which results in the attacker not having the means to perform a privilege escalation. This binary concept is defined as all-or-nothing protection [2].

5.1.2 Service Entity Division

The security of an SDFA system hugely depends on the strict separation of the protected assets and the means to access them. The SDFA server provides the unlocking mechanism, turning the protecting metadata of an asset into the means to unlock the asset itself. The server can also decide to instantly revoke or deny any future access rights (for reasons determined by the server), respectively the party controlling it. However, combining control over both the asset and the ability to operate on it results in immediate loss of all protective features provided by the SDFA system. While the availability can be disturbed more easily, all CIA and related security goals collapse immediately, as the entity controlling both asset and access can freely view and manipulate the content and access rights of the asset. Therefore, it is crucial that the SDFA service and any kind of hosting service are separated and are not combined in one entity (that could turn malicious). While Microsoft promises to strictly separate the domains of file hosting and SDFA functionality (RMS), there is no guarantee that in a no-trust scenario, an entity or organization, combining both asset and access, could not turn malicious and instantly overcome all security goals of an organization, leading to a devastating security breach. Therefore, for the security of SDFA systems, it is crucial to achieve an absolute separation of SDFA and hosting services. Therefore, the provided definition of SDFA systems includes that the server never obtains the file's content (see section 3.2).

5.2 External Attacks

5.2.1 Denial of Service

Denial of Service (DoS) attacks are attacks against the availability of a system, which is mostly a server. A classic approach to DoS attacks is to flood a server with more requests than the server can process, which leads to the unavailability of the server to users who send legitimate requests with a legitimate interest for the service. While in most cases, single attacking devices have to be significantly more powerful than the server to make a DoS attack succeed, there are a few strategies that can increase the chance of success. One of those strategies is to use a great number of clients instead of a single one that all attack one target at the same time, increasing incoming data traffic for the target and the chance of success. This is called a distributed denial of service Distributed Denial of Service (DDoS) attack. Modern DDoS attacks use botnets with a Command and Control (C&C) server to attack targets with huge amounts of data. However, other techniques can further increase the amount of data that is directed at a target, such as DNS-amplification attacks. While DoS attacks do not endanger the existence of assets, they can deny the access (and therefore affect the availability) temporarily. While being a very simple attack, it can have major consequences for the entity using an SDFA system. If a company for instance decides to protect critical assets with an SDFA system, the temporary unavailability caused by a DoS attack can greatly impair the entire workflow of the respective organization. It is not unrealistic that well-timed DoS attacks on the SDFA system can cause significant interruptions that disable the organization's functionality. That way, an attacker could cause great financial or reputational damage to a company or government entity. From the defending perspective, one good aspect of DoS attacks is that they are well-known, relatively simple, and not limited to a certain kind of systems. Therefore, many systems, strategies, and providers exist to detect and prevent DDoS attacks, such as the Cloudflare DDoS protection service or just a physical firewall that implements an IP-address based DDoS detection.

5.2.2 Man-In-The-Middle

Man-in-the-Middle (MitM) attacks target the communication between two parties and can affect all CIA goals. For a MitM attack, the attacker convinces both parties that the attacker is their communication partner. That way, the attacker can read and potentially manipulate all traffic between the legitimate communication partners. MitM attacks can be passive or active. During a passive attack, the attacker listens to a communication between two parties without manipulating it, in which case the confidentiality of the communication is endangered. During an active attack, the attacker also manipulated the communication, endangering availability and integrity as well. Often, MitM attacks start out passively and then turn into active attacks in a second step. MitM attacks propose a number of different threats to SDFA systems. Assuming that a passive MitM attack is running, the following assets are endangered:

- **User Credentials**: Depending on the used encryption, the confidentiality of the communication can be attacked. Assuming that for instance user credentials are only protected by a layer of SSL/TLS encryption, the user credentials can be extracted if this layer of encryption is stripped. In the case of the model implementation, SSL/TLS encryption is not used at all, for which the credentials are directly exposed (see figure 4.8). This is also true for temporary access tokens, such as the issues JWTs, allowing the attacker to impersonate the client at least temporarily.

- **Program Flow Information**: By observing the request-response exchange between a client and a server, the attacker can gain information about how access is requested and granted regarding a file. The attacker can derive different information from the attributes of these messages, such as the lifetime of tokens, the complexity of the rights associated with the requested source, or standard denial messages. All of this might be useful information for further attacks. Furthermore, if the attacker manages to observe the bootstrapping process, the attacker could try to mimic this process in a further attack stage to register a malicious client device.

- **Usage Metadata**: By observing the traffic, a MitM attacker might gain valuable information about the usage of the SDFA service, even when cryptographic attacks are not attempted or successful. Among a plethora of other information about the user, such as working hours, habits, and estimated importance, also information about the server is revealed, such as response times, exact routes for different services, token expiration time, and the general program flow. This can propose valuable information for further attacks, including but not limited to active MitM approaches.

Assuming that an active MitM attack is in progress, the following assets are endangered in addition:

- **Service Availability**: With an active MitM attack, the attacker can actively stall the communication between the client and the server, in one or both directions.
- **Server Integrity**: Apart from just denying communication, an attacker could also redirect the traffic to an internal or external malicious server. With a malicious SDFA server (or a dummy) in place, the attacker could obtain a wide variety of information, such as the user's credentials, the files that are about to be accessed. Depending on the information known to the attacker, the malicious server could perfectly mimic the actions of the legitimate server and therefore be used, giving the attacker the sole control over the files "protected." This would for instance allow the attacker to have exclusive access to the allegedly protected files and for instance ask for a ransom to decrypt the files with the malicious server certificate.
- **User Integrity**: While the attacker can pretend to be the server for the user, the attacker can also pretend to be the user for the server. Depending on the level of encryption used and available to the attacker, the attacker could pretend to register a new device for the user at the server and therefore gain theoretical access rights to all protected documents. However, to make use of this right, the attacker would also have to obtain a copy of the protected file, for instance by capturing an email containing the file or by extracting it from a file upload to a cloud service. While this is not a part of the level of Posting Licenses, the attacker can cause a much greater amount of damage if the user credentials are also obtained. The most devastating consequences would arise when the same user login state was also used for services able to obtain the encrypted file, such as a related cloud service. In this case, the attacker could use the obtained credentials (user name and password or a temporary token) to log into the cloud service and obtain the required copy, which the attacker could unlock subsequently with the stolen SDFA identity.
- **Rights**: While PLs and the rights are encrypted with at least one layer of asymmetric encryption (see figure 4.7) including integrity verification, the data in motion could theoretically be modified by an attacker. Most likely, this would

lead back to a service interruption as both server and client would have to verify the integrity and authenticity of each other's message using their public certificate. However, if for instance the MitM attacker managed to inject his own malicious certificate into the communication or break one of the real ones, the rights over a file could also be manipulated more subtly (in both directions), depending on which certificate was compromised.

It is important to note that through the sole usage of an SDFA system, the file content is **not** endangered or exposed by a MitM attack at any point in time.

5.2.3 Cryptographic Attacks

The cryptographic methods used in the model implementation, namely RSA, AES, SHA-256, and Argon2 are all considered safe at this point in time. However, there are many potential attack vectors that arise from the usage of cryptosytems. While attacking a cryptosystem, every known piece of information can be useful to the attacker. One example is the padding oracle attack in which the sole (side-channel) information of whether a certain ciphertext has valid padding (or not) is enough to break the entire encryption efficiently [52]. A different example are known-plaintext attacks where knowledge over the corresponding plaintext of a given ciphertext can be used to reconstruct its encryption and decryption process. A third example includes timing attacks, where a bad implementation of the cryptosystem can result in usable information about the key by measuring the time for a series of decryption attempts.
Other attack types concern not the encryption, but the hashing process. A primary attribute of a safe hash function is that no collisions can be created. When collisions can be found or even crafted, the integrity and authenticity of hashes and signatures are severely endangered. This can lead to identity theft, integrity breaches for protected data, or the general collapse of non-repudiation, which is hard to detect and highly relevant in forensic contexts.

5.3 Internal Attacks

5.3.1 Ripping Attack

Ripping is the process of circumventing existing protections by viewing and recording a protected piece of media [2]. Classically, ripping refers to programs on a system that "rip" the contents of storage device like DVDs. However, for the scope of this paper, ripping attacks also include the usage of external devices to create unauthorized copies of protected documents. The most obvious and intuitive way to breach the confidentiality of a protected document is to access (*view*) a document with minimal rights and then "rip" its content. This is probably the most effective attack against the confidentiality of SDFA-protected assets and a good example of the all-or-nothing protection principle.

5.3.2 Memory Dumping Attacks

When a program is executed, the necessary data must be loaded into the main memory, also referred to as Random Access Memory (RAM). When a program loads a file from a hard drive to process its contents for instance, the program reads the data from the hard drive (respectively the API provided by the OS) and stores the contents into the main memory where it can be accessed by the program itself. This is a crucial point when discussing the security of protected content in several ways. At some point, data that must be processed or even presented in an unencrypted format must be decrypted. When a user has access to an encrypted file stored on a hard drive and wants to view the file, somewhere in between the hard drive and the displaying device, the file must be decrypted. The same is true for any other operation requiring the data in a plain format, such as editing or re-encrypting. In most cases, this results in the data being decrypted in the main memory. Therefore, it is likely to find a fully unencrypted version of a file in the main memory. While main memory is volatile and therefore automatically deleted when the device loses power, for the time of using the data, the data are stored in the main memory in an unencrypted format. If an attacker (both internal or external) obtains a snapshot of the current main memory contents, the unencrypted data can be extracted and stripped from their protection. This proposes a major threat to any protected data, yet is a very common problem. In 2020, Ján Mojžiš and Štefan Balogh demonstrated that the SDFA client Microsoft Azure Information Protection Viewer, a program used to view content protected with Azure Information Protection, was prone to such attacks [1]. While suggesting several methods to obfuscate the unencrypted file content in the main memory (like hiding file headers), they noted that it was "disturbing [...] how easy such an attack can be realized" and that they "cannot give a final recommendation as to how to avoid such an attack" [1]. However, the potential consequences of memory attacks can reach even further. To use encryption and decryption functionality of the used certificates for instance, the necessary keys must also be loaded into the main memory. Subsequently, when obtaining a memory dump, not only the contents of an opened file could be exposed, but also the secret keys used to protect and access content. One of the examples mentioned by Mojžiš is the extraction of TrueCrypt keys while the encrypted container files are mounted [53]. Transferring the same concept to SDFA systems, the user certificate's private key and other secret information crucial to the user can be exposed in memory dumps. When a user's certificate secrets are obtained by an attacker, this results in a complete loss of confidentiality and authenticity for the user, as the attacker can most likely imitate the user perfectly.

5.3.3 Malicious Client

When a user is granted rights over a file, the SDFA server sends the body decryption key to the client. It is the SDFA client's responsibility to interpret the rights granted and allow or disallow certain user operations. Attempts to perform operations that are not allowed should be left unrequited. However, from a technical point of view, the client is not bound to realize any of the restrictions that are requested by the server. Subsequently, even with minimum rights assigned (for instance *view* rights), the body key is provided by the server either in a direct way, or by sending the unencrypted PL for modification and re-encryption, which contains the body key. This results in a binary (all-or-nothing) protection scheme that can be abused by the client ignoring all intermediate states as discussed in section 5.1.1. Grothe et al. demonstrated in 2016, that this all-or-nothing situation, arising from the client being responsible for the correct interpretation of granted rights, can be broken for both AD RMS as well as Azure RMS and its integration into the customer line of Microsoft 365 [2] (formally known as Microsoft Office 365). Furthermore, they showed that not only the confidentiality can be broken as suggested, but also the integrity of a file can be broken by manipulating its contents in a subtle way. The only case where these kinds of malicious client attack are not possible is when the client is not entitled to any rights regarding a file and has no self-signed local access rights (a feature supported by RMS).

5.3.4 Malicious Server

When an attacker obtains control over a server, this proposes a major threat to the SDFA system. By design, the service (the server and the connection to it) is a Single Point of Failure (SPOF) as it is the only entity controlling access and rights to files for the users. This is true even though the users technically have the file content (and permission information) stored on their local device. By refusing service to clients, the server can actively and effectively attack the availability of the entire system. Moreover, this also happens when the server - for any reason - goes down or refuses services. Possible reasons for that are human error, programming mistakes, maintenance breaks, but also reasons that are not related to the server itself, like an interrupted internet connection between client and server. This leads to immediate and severe problems, especially if clients have no local temporary access rights (a feature implemented in M365 but for instance not the model implementation). To prevent service downtime, redundant clusters, backups, 24/7/365 support, and alternative routes can be used to ensure the availability of the server functionality at all times. Nevertheless, the server is the most critical SDFA asset. The ways in which a server can impact the availability are plenty, and can be permanent. Deleting its certificate's private key (without backups) results in immediate and permanent loss of access to all files that have been protected with the SDFA system. The only way to retrieve their contents would be unprotected backups, locally saved permissions (allowing access with the body key), or breaking the server certificate's RSA cryptosystem (by trying to restore the private key from its public key with cryptoanalytic attacks). Single files could also be decrypted by breaking the AES encryption of the body key. Breaking the RSA system would be preferable as it would allow the decryption of all protected AES content keys, not just a single one. However, the confidentiality and integrity of all server assets is also at risk, which includes the user certificates, server certificates, user credentials and data, group memberships, and log data. As an example, an attacker could easily create a user that is used to decrypt protected data assets in further stages of the attack. The only asset that is not directly endangered by a malicious server is the confidentiality of protected containers unknown to the attacker and server.

5.3.5 Malicious Server With Insufficient Service Entity Division

Modern enterprises often use cloud solutions to store their data if there are economic or other reasons not to host them on-premise. Combined solutions like Microsoft Azure offer both storage space on remote servers and Rights Management Services. However, not dividing these two domains clearly can propose enormous risks to the confidentiality of the data (as discussed in section 5.1.2). While the services might be independent from each other in theory, it is possible for a vendor of both the cloud and the SDFA service to combine the data and have full control over the data and overthrow all security goals of the data owner. Grothe et al. [15] presented in a 2016 paper that Microsoft (as the vendor of Azure Rights Management Service) and Tresorit (as the vendor of Tresorit RMS) could easily obtain full control over all data managed by their services as malicious attackers.
To obtain full access over a file protected with an SDFA system, the only asset required is the private key of the SDFA service. An attack breaking all goals of the CIA Triad could follow these steps:

1. The Posting License (PL) of the file is received, encrypted with the SDFA server's public key.
2. The malicious SDFA server decrypts the PL, obtaining the content key. The verification step is optional as the rights in the PL are irrelevant for the attack. The availability can be broken by simply refusing service to the client. Also, the integrity can be partially broken by changing the rights of the returned rights.

3. The SDFA fetches the encrypted file body from the related service, such as a cloud storage.

4. The content key is used to decrypt the body. The confidentiality is broken by viewing or exporting the decrypted content. Moreover, the integrity can be fully broken by manipulating the content and/or rights and arbitrarily granting or refusing access permissions.

5.4 Human-Error-Based Attacks

5.4.1 Unintentional Misconfiguration (Availability)

When an attacker attacks a system, the weakest link is the one most commonly exploited. Oftentimes, the weakest link in a security system is human, which results in social engineering attacks being so effective. However, apart from intentional attacks that include manipulating humans to act as an attacker, humans can also be unintentional attackers by simple mistake. The following example demonstrates a scenario in which human error results in the loss of availability on a file:

1. An employee creates a file containing the latest secrets blueprints, crucial to the company.

2. The employee classifies the file as highly confidential and uses an SDFA system to protect the file.

3. The employee wants the maximum security level for the file and therefore disables all the rights for all employees and groups.

4. The SDFA system allows the employee to create a policy in which no one (including the employee) has any rights on the file anymore.

5. The file is protected and no local license is created.

6. Subsequently, the access to the file is lost and the availability is greatly impaired.

This example shows that mistakes and the non-consideration of such scenarios by the developer of an SDFA system can be dangerous to the availability. To retrieve the file in the presented scenario, there has to be a way to overwrite the non-existing rights in the policy with at least minimal rights on the server side for a specific user. However, this option opens up another attack vector on the option of granting more rights than defined in the Posting License, respectively the user with these special privileges.

5.4.2 Unintentional Data Leaks

However, not only the availability can be impaired by unintentional (or intentional) misconfiguration. One of the major threats SDFA systems are supposed to prevent are mistakes that threaten the confidentiality. Correctly classified and configured documents can - in theory - be distributed to unauthorized users, and these users will not be able to access the file. This can especially be tempting in a scenario where shared directories are used. However, combining access and an incomplete or falsely configured configuration can have severe consequences for the confidentiality of documents. The severity of this increases with further attacks that can be performed with minimal rights (as discussed in section 5.3.3). Another dangerous situation arises from the false assumption that a file is already protected according to its classification and then transferred or published (without double-checking that the correct protection is in place). This can also affect emails or other supported message formats themselves.

5.5 Future Challenges

5.5.1 Cryptographic Durability

Cryptosystems lose their cryptographic abilities over time, and eventually become obsolete. After this happens to a cryptosystem, it is dangerous to use that cryptosystem in a project as they open up new and potentially severe attack vectors. However, the way and time interval in which cryptographic systems become obsolete can differ. The main three causes of cryptographic strength degeneration discussed are advances in computing power, newly discovered attack vectors, and new technological approaches.

Advances in Computing Power

Moore's Law is generally known as the rule of thumb, that the number of transistors on a chip will double every one or two years [54]. This results in a theoretical general computational linear speed up. While Moore's Law is now faced with its physical limits (after holding up for several decades), the trend towards increasing computational performance continues [55]. While being a strong block cipher, in the early 1990s, the Data Encryption Standard (DES) was cracked in about 3.5 hours, and in 1998 again with a computer costing less than $250,000 at that time [56]. This resulted in Triple DES (3DES) being invented; a cryptosystem that uses the DES in three operational stages with an increased key length. This had the advantage of not having to come up with a completely new block cipher until the Advanced Encryption Standard (AES) was eventually standardized [57]. However, with increasing computational power, even strong cryptographic systems are subject to attacks like brute force, endangering their security in the long run. It is up for speculation whether this, over time, leads to developments like Triple AES until a stronger algorithm is standardized. In any case, advances in computational power endanger all cryptosystems in the long run. This also concerns hashing methods. The search for collisions is ongoing and mathematically supported by the so-called Birthday Problem or Birthday Paradox [58]. This model shows that forging collisions of a specific hash is a task much more complex than finding any hash collision. However, finding any hash collision is enough to deem a hash algorithm broken. The still very popular hash algorithm MD5 is broken since several collisions have been found. However, one of its popular successors, Secure Hash Algorithm 1 (SHA-1), is also compromised by now [40]. This shows the evolutionary degeneration of both cryptosystems and hashing algorithms. It is to assume that in some years or decades, the currently used cryptosystems will be broken one by one, and that the breach of a cryptosystem will open up a multitude of attacks against systems using them, including but not limited to SDFA systems.

Newly Discovered Attack Vectors

New attacks on systems can be found at any time. One of the most popular targets are not the concepts of the systems, but its implementations. Conceptual or semantic mistakes in implementations can result both in side-channel attacks, like the padding oracle attack [6], but also direct or inherited vulnerabilities. While a large community of developers and high usage increases the chance that vulnerabilities are discovered, many severe vulnerabilities remain unknown to the public for several years. “Heartbleed” for instance, a vulnerability in OpenSSL that has been described as “catastrophic” [59] in scientific literature, remained undiscovered for about two years [60]. This shows that dangerous vulnerabilities can remain undiscovered for several years until they take the owner or manufacturer of a system by surprise. But also, new ways to break cryptographic algorithms can be found at any time. In 2022, Wouter Castryck and Thomas Decru presented a paper [61] demonstrating a successful and efficient attack on the Supersingular Isogeny Key Exchange (SIKE) algorithm, a fourth-round candidate of an effort by the National Institute of Standards and Technology (NIST) to standardize quantum-safe cryptographic algorithms.

New Technological Approaches

Quantum Computers are programmable machines that solve computational tasks by using quanta (and their quantum effects) as so-called qubits (also qbits or q-bits), bits with non-digital and complex intermediate states. While the field of quantum computers is fairly new and many use cases are still about to be discovered, one particularly interesting one is the potential ability to solve specific problems faster than classical digital computers. One of these very interesting new capabilities is the ability to factor numbers. In the late 1990s, Peter Shor implied that factorization could be performed faster by (a specific kind of) quantum computers and that it "could even make breaking RSA on a quantum computer asymptotically faster than encrypting with RSA on a classical computer" [62]. Since then, much research has been conducted, both on the matter, and the algorithm suggested by Shore. While access to quantum computers is now available to the general public using remote access, modern research on the topic is generally limited by the number of (stable) qubits available [63]. Nevertheless, many are convinced that with quantum computers "it will be possible to break an RSA key in a reasonable amount of time when combined with a supercomputer" [64]. A recent paper estimated the number of required noisy qubits to break RSA-2048 with 20 million [65]. This is not only a danger to the safety of SDFA systems, but all systems that (as of now) rely on RSA. In fact, in a recent summary of post-quantum cryptography [66], Bavdekar et al. wrote that "quantum algorithms exist for cracking all the major public key cryptosystems. It is only a matter of time before they are broken completely." The matter also concerns hashing functions as quantum computers "allow [a] faster search for collisions for most of the existing algorithms" [67]. This threat on the horizon, the search for quantum-safe cryptographic algorithms is currently ongoing. However, even though the algorithm McEliece currently seems to be a promising candidate for replacing RSA as the main asymmetric cryptosystem in the post-quantum age [68], the search for quantum safe algorithms remains a complicated matter as shown by the previously mentioned example of the recently broken SIKE algorithm.

6 Discussion

6.1 Strengths of SDFA Systems

Server-Dependent File Access (SDFA) systems can greatly support an entity's efforts to restrict file access to certain users or groups and can provide a number of useful features that exceed the ones presented in the model implementation. Furthermore, a granular and unified access system can be implemented. This can both include expressing real-world classes in a technical form (such as a group representing all members that have signed an NDA) as well as directing certain rights dynamically to single users (such as providing access to project resources to external employees or granting a streaming service customer a 7-day access to a video stream). When applied consistently, SDFA systems help to defend against internal attacks and can provide some level of protection against external attacks on different ways of communication, such as file transfer via email attachment. SDFA systems can greatly support DLP as the usage of such a system can make accidental leakage of unencrypted data much more unlikely. In general, the usage of an SDFA system can result in major improvements for the confidentiality of sensitive data.

6.2 Weaknesses of SDFA Systems

However, the security of SDFA systems heavily depends on the compliance and good will of the environment's members, respectively on the consistent use of further technical measures to limit potential attacker's options. While well-designed and implemented SDFA systems can be assumed to be effective against several internal and external attacks, a major threat is proposed by internal attackers, especially ones with malicious intent. The problem of all-or-nothing protection and ripping attacks are major threats to the confidentiality of the data assets an SDFA system is supposed to protect. On any SDFA systems's level of abstraction, there is no protection against attacks with external recording devices that capture the content of a file, such as cameras or mobile phones. Furthermore, the SDFA server (or the server-representing cluster) and an SDFA client's connection to it are SPOFs that can make the system deny access to protected assets. When a company decides to protect its most valuable data with an SDFA system, a malicious or otherwise incapable server can have devastating consequences for the company. In case of a non-recoverable data loss (in the sense of permanent unavailability), the company's survival itself is at stake. Therefore, using an SDFA system also opens up new attack vectors that have to be defended against with measures. These measures must not rely on the functionality of the SDFA system. Examples for these measures would be external backups or technical restrictions to the client's ability to execute foreign software, such as malicious SDFA client or other malware.

6.3 Impact on (Offline) Forensic Investigations

When looking at a classically encrypted file, investigators cannot look into a file's content unless they obtain the key or find a copy of the content in metadata [18]. With SDFA systems, the situation is a more interesting one since in theory, the decryption key for the file content is already stored on the device. During a live acquisition, if a file is currently used and a memory dump is created, the content and body key can most likely be extracted (as shown in section 5.3.2). However, assuming that no offline data of the decrypted content key or the content itself exists, the only way for an investigator to obtain the content in a static acquisition is to decrypt the content key. While it may be possible to break the used cryptosystem (in this case RSA), the easiest way is to obtain the certificate private key of the server. However, depending on how many Posting Licenses are encrypted with the server certificate, this endangers not only the confidentiality of one, but of all files protected with the server certificate. Nevertheless, an organization could be forced to reveal the private key to state investigators, leading to a security risk for all files encrypted with the same certificate. Therefore it makes sense to think about layered certificates when implementing an SDFA system so that revealing the private key of a sub-certificate is not a universal threat to the SDFA hosting institution and all the assets it protects. Switching perspective, for suspects of an investigation who use an SDFA system, a “panic button” or “kill switch” implemented to instantly revoke all access to sensitive files could be a powerful anti-forensic measure, as file encryption can greatly disrupt forensic investigations [18].

6.4 Research Questions Review

At the beginning of this work, research questions have been defined in section 1.4. The answers to them have been provided as follows:

1. A definition of Server-Dependent File Access (SDFA) systems has been provided in section 3.2.

2. The general matter of SDFA systems has been discussed in chapter 3. Furthermore, a prototype model implementation has been provided and discussed in chapter 4.

3. A comparison of the provided model and Microsoft RMS has been drawn in section 4.5.

4. Strengths of SDFA systems have been discussed in section 6.1.

5. Weaknesses of SDFA systems have been discussed in section 6.2.

6. The general use case of SDFA systems has been discussed in section 3.3. Potential dangers of doing so have been discussed in chapter 5.

7. The emerging threat of quantum computers has been discussed in section 5.5.1.

6.5 Future Work

Alongside the answer to the research questions asked in section 1.4, this work provided a model implementation of an SDFA system. However, even though the model implementation works, it cannot yet be considered complete. The model implementation succeeds in demonstrating the general functionality and implementation concepts of a general-purpose SDFA. However, the implementation lacks completeness in regards to documentation, RESTful CRUD functionality, non-ad-hoc classification features, backup functionality, guaranteed handling of raised exceptions, specific file handler implementations (for several file types), an interactive Graphical User Interface (GUI) for users, a non-interactive Command Line Interface (CLI) or developer API, support for a PKI, and fixes for known issues (like the ones pointed out in section 4.5.7). Based on this, future work is suggested to provide an improved version of the existing implementation. This would not only provide a better user experience, but also improved stability, safety, security, and feature support.

Apart from technical improvements regarding the presented prototype implementation, there are also different aspects yet to be covered. Although in this work, the general-purpose and use cases of SDFA systems have been discussed from a hypothetical point of view, it would also be interesting to conduct research on sociological and psychological matters. For instance, further research could include studies on the general acceptance and usage of SDFA systems in business environments, the development of intuitive employee training programs, user experience research, the phenomena of over- and under-classification, and the occurrence and prevention of bad practices.

7 Conclusion

This work provided a definition of Server-Dependent File Access (SDFA) systems. Furthermore, a prototype model implementation of a general-purpose SDFA system has been provided. The implementation, its features, its implementation, as well as its limitations have been discussed. Different threat scenarios have been applied to SDFA systems and were partially simulated using the model implementation. Potential attack vectors have been shown and discussed. Furthermore, the threat of quantum computer driven factorization for asymmetric cryptosystems has been considered. Further research and technical improvements to the presented implementation have been suggested.

The author concludes that the biggest threat to the security goals SDFA systems are trying to achieve are internal attackers with malicious intent. While it is assumed that well-implemented SDFA systems offer relatively good protection against unintentional data leakage and outside threats - in part due to the clean separation of asset and access control - it is easy to circumvent client-controlled partial protections. This can be done by either extracting key information that can be used to lift the remaining restrictions, or by leaving the realm of computing and using external devices to break the confidentiality. The author concludes that SDFA systems for multiple users are most effective when paired up with other security measures that restrict the user's possible actions, like (malicious) program execution on a client device, accessing SDFA-protected resources outside the company, or bringing external devices with cameras. Furthermore, SDFA systems can also be useful to single users in certain situations to perform anti-forensic or protective functionality. Another interesting finding is that all security goals are greatly endangered by an entity controlling both the SDFA system and the storage service of the SDFA-concerning data.

Bibliography

[1] J. Mojžiš and Š. Balogh, "Breaking microsoft azure information protection viewer using memory dump," in *Software Engineering Perspectives in Intelligent Systems*. Springer International Publishing, 2020, pp. 913–920.

[2] M. Grothe, C. Mainka, P. Rösler, and J. Schwenk, "How to break microsoft rights management services." in *WOOT*, 2016.

[3] K. Kaur, I. Gupta, and A. K. Singh, "A comparative study of the approach provided for preventing the data leakage," *International Journal of Network Security & Its Applications*, vol. 9, no. 5, pp. 21–33, 2017.

[4] J. Mahn, "Sabotage bei der bahn: Viele vertrauliche infos sind offen zugänglich," Oct. 2022, https://www.heise.de/news/Sabotage-bei-der-Bahn-Viele-vertrauliche-Infos-sind-offen-zugaenglich-7307277.html. [Online]. Available: https://web.archive.org/web/20230315095929/https://www.heise.de/news/Sabotage-bei-der-Bahn-Viele-vertrauliche-Infos-sind-offen-zugaenglich-7307277.html

[5] M. E. Whitman and H. J. Mattord, *Principles of information security*. Cengage learning, 2021.

[6] R. Merget, J. Somorovsky, N. Aviram, C. Young, J. Fliegenschmidt, J. Schwenk, and Y. Shavitt, "Scalable scanning and automatic classification of tls padding oracle vulnerabilities." in *USENIX Security Symposium*, 2019, pp. 1029–1046.

[7] P. Rogaway, "Evaluation of some blockcipher modes of operation," *Cryptography Research and Evaluation Committees (CRYPTREC) for the Government of Japan*, vol. 630, 2011.

[8] G. Firican, "What is the difference between data classification and data categorization?" Aug. 2021. [Online]. Available: https://web.archive.org/web/20230305114252/https://www.lightsondata.com/what-is-the-difference-between-data-classification-and-data-categorization/

[9] N. Syynimaa, "Exploring azure active directory attack surface: Enumerating authentication methods with open-source intelligence tools." in *ICEIS (2)*, 2022, pp. 142–147.

[10] R. Allen and A. Lowe-Norris, *Active directory*. " O'Reilly Media, Inc.", 2003.

[11] Microsoft, "Office is now microsoft 365," 2023. [Online]. Available: https://web.archive.org/web/20230301193852/https://www.microsoft.com/en-us/microsoft-365

[12] ——, “Looking back at 10 years of microsoft 365 making history,” Jan. 2023. [Online]. Available: https://web.archive.org/web/20230314095245/http://www.microsoft.com/en-us/microsoft-365-life-hacks/stories/looking-back-ten-years-microsoft-365

[13] ——, “Azure information protection - also known as ...” Feb. 2022. [Online]. Available: https://web.archive.org/web/20230408151614/https://learn.microsoft.com/en-us/azure/information-protection/aka

[14] M. Kratzenberg, “Drm – was ist das eigentlich?” *GIGA*, Apr. 2022.

[15] M. Grothe, C. Mainka, P. Rösler, J. Jupke, J. Kaiser, and J. Schwenk, “Your cloud in my company: Modern rights management services revisited,” in *2016 11th International Conference on Availability, Reliability and Security (ARES)*. IEEE, 2016, pp. 217–222.

[16] A. Biryukov, G. Leurent, and A. Roy, “Cryptanalysis of the “kindle” cipher,” in *Selected Areas in Cryptography: 19th International Conference, SAC 2012, Windsor, ON, Canada, August 15-16, 2012, Revised Selected Papers 19*. Springer, 2013, pp. 86–103.

[17] Microsoft, “File types supported by the azure information protection (aip) unified labeling client,” Mar. 2023. [Online]. Available: https://web.archive.org/web/20230405075232/https://learn.microsoft.com/en-us/azure/information-protection/rms-client/clientv2-admin-guide-file-types

[18] L. Schmitt and G. Kul, “Anti forensic measures and their impact on forensic investigations,” 2023.

[19] Microsoft, “How does azure rms work? under the hood,” 2023. [Online]. Available: https://web.archive.org/web/20230314095704/https://learn.microsoft.com/en-us/azure/information-protection/how-does-it-work

[20] A. Goldbergs, “Azure rms under the hood,” article on medium.com, Mar. 2019. [Online]. Available: https://web.archive.org/web/20230314100315/https://medium.com/@agoldbergs/azure-rms-under-the-hood-7ea736135d95

[21] K. Jendrian and C. Schäfer, “Verschlüsseln in der cloud: Visualisiert am beispiel von microsoft azure rms,” *Datenschutz und Datensicherheit-DuD*, vol. 39, no. 8, pp. 548–552, 2015.

[22] M. Shaw, “Writing good software engineering research papers,” in *25th International Conference on Software Engineering, 2003. Proceedings.* IEEE, 2003, pp. 726–736.

[23] G. Van Rossum, B. Warsaw, and N. Coghlan, “Pep 8–style guide for python code,” *Python. org*, vol. 1565, p. 28, 2001.

[24] “Python programming language,” https://www.python.org. [Online]. Available: https://web.archive.org/web/20230306184755/https://www.python.org/

[25] “pip,” https://pip.pypa.io/en/stable. [Online]. Available: https://web.archive.org/web/20230306184732/https://pip.pypa.io/en/stable/

[26] JetBrains, “Pycharm,” https://www.jetbrains.com/pycharm. [Online]. Available: https://web.archive.org/web/20230306184759/https://www.jetbrains.com/pycharm/

[27] “Vscodium,” https://vscodium.com/. [Online]. Available: https://web.archive.org/web/20230308135311/https://vscodium.com/

[28] “diagrams.net,” https://github.com/jgraph/drawio-desktop. [Online]. Available: https://web.archive.org/web/20230306184716/https://github.com/jgraph/drawio-desktop

[29] “mypy,” https://mypy-lang.org. [Online]. Available: https://web.archive.org/web/20230306184718/https://mypy-lang.org/

[30] “Db browser for sqlite,” https://sqlitebrowser.org/. [Online]. Available: https://web.archive.org/web/20230308150959/https://sqlitebrowser.org/

[31] “Wireshark,” https://www.wireshark.org/. [Online]. Available: https://web.archive.org/web/20230308115156/https://www.wireshark.org/

[32] R. L. Rivest, A. Shamir, and L. Adleman, “A method for obtaining digital signatures and public-key cryptosystems,” *Communications of the ACM*, vol. 21, no. 2, pp. 120–126, 1978.

[33] R. T. Fielding and J. Reschke, “Hypertext Transfer Protocol (HTTP/1.1): Authentication,” RFC 7235, Jun. 2014. [Online]. Available: https://www.rfc-editor.org/info/rfc7235

[34] N. Nurseitov, M. Paulson, R. Reynolds, and C. Izurieta, “Comparison of json and xml data interchange formats: a case study.” *Caine*, vol. 9, pp. 157–162, 2009.

[35] M. Jones, J. Bradley, and N. Sakimura, “Json web token (jwt). rfc 7519,” *Internet Engineering Task Force*, 2015.

[36] jwt.io, “Libraries for token signing/verification,” 2023. [Online]. Available: https://web.archive.org/web/20230306000949/https://jwt.io/libraries

[37] A. Biryukov, D. Dinu, and D. Khovratovich, “Argon2: new generation of memory-hard functions for password hashing and other applications,” in *2016 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE, 2016, pp. 292–302.

[38] “Password hashing competitionand our recommendation for hashing passwords: Argon2,” 2019. [Online]. Available: https://web.archive.org/web/20230312114650/https://www.password-hashing.net/

[39] *Secure Hash Standard (SHS)*, Federal Information Processing Standards publication, U.S. Department of Commerce Std. 180-4.

[40] S. D. Almotiri, “Forensic hash value guidelines: Why md5 and sha1 should no longer be used and a recommendation for their replacement.” Mar. 2022.

[41] “Flask,” https://flask.palletsprojects.com. [Online]. Available: https://web.archive.org/web/20230303154526/https://flask.palletsprojects.com/en/2.2.x/

[42] “requests,” https://requests.readthedocs.io. [Online]. Available: https://web.archive.org/web/20230306184818/https://requests.readthedocs.io/en/latest/

[43] "Python-rsa," https://stuvel.eu/software/rsa/. [Online]. Available: https://web.archive.org/web/20230306184820/https://stuvel.eu/software/rsa/

[44] "pyaescrypt," https://github.com/marcobellaccini/pyAesCrypt. [Online]. Available: https://web.archive.org/web/20230306184750/https://github.com/marcobellaccini/pyAesCrypt

[45] "Pycryptodome," https://www.pycryptodome.org. [Online]. Available: https://web.archive.org/web/20230306184800/https://www.pycryptodome.org/

[46] "argon2-cffi," https://github.com/hynek/argon2-cffi. [Online]. Available: https://web.archive.org/web/20230306184457/https://github.com/hynek/argon2-cffi

[47] "Pyjwt," https://github.com/jpadilla/pyjwt. [Online]. Available: https://web.archive.org/web/20230306184753/https://github.com/jpadilla/pyjwt

[48] "psutil," https://github.com/giampaolo/psutil. [Online]. Available: https://web.archive.org/web/20230306184733/https://github.com/giampaolo/psutil

[49] "universal-startfile," https://github.com/jacebrowning/universal-startfile. [Online]. Available: web.archive.org/web/20230412084936/https://github.com/jacebrowning/universal-startfile

[50] martineau and wolf, "Answer to "how to get the text out of a scrolledtext widget?"," stackoverflow, Dec. 2018, stackoverflow.com/a/53938684. [Online]. Available: web.archive.org/web/20230306141620/https://stackoverflow.com/questions/53937400/how-to-get-the-text-out-of-a-scrolledtext-widget/53938684

[51] Microsoft, "Configure usage rights for azure information protection," Sep. 2022. [Online]. Available: https://web.archive.org/web/20230327112201/https://learn.microsoft.com/en-us/azure/information-protection/configure-usage-rights

[52] J. Rizzo and T. Duong, "Practical padding oracle attacks." in *WOOT*, 2010.

[53] Š. Balogh and M. Pondelik, "Capturing encryption keys for digital analysis," in *Proceedings of the 6th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems*, vol. 2. IEEE, 2011, pp. 759–763.

[54] E. Mollick, "Establishing moore's law," *IEEE Annals of the History of Computing*, vol. 28, no. 3, pp. 62–75, 2006.

[55] C. E. Leiserson, N. C. Thompson, J. S. Emer, B. C. Kuszmaul, B. W. Lampson, D. Sanchez, and T. B. Schardl, "There's plenty of room at the top: What will drive computer performance after moore's law?" *Science*, vol. 368, no. 6495, p. eaam9744, 2020.

[56] S. Landau, "Standing the test of time: The data encryption standard," *Notices of the AMS*, vol. 47, no. 3, pp. 341–349, 2000.

[57] P. Patil, P. Narayankar, D. Narayan, and S. M. Meena, "A comprehensive evaluation of cryptographic algorithms: Des, 3des, aes, rsa and blowfish," *Procedia Computer Science*, vol. 78, pp. 617–624, 2016.

[58] E. Thompson, "Md5 collisions and the impact on computer forensics," *Digital investigation*, vol. 2, no. 1, pp. 36–40, 2005.

[59] Z. Durumeric, F. Li, J. Kasten, J. Amann, J. Beekman, M. Payer, N. Weaver, D. Adrian, V. Paxson, M. Bailey *et al.*, "The matter of heartbleed," in *Proceedings of the 2014 conference on internet measurement conference*, 2014, pp. 475–488.

[60] M. Carvalho, J. DeMott, R. Ford, and D. A. Wheeler, "Heartbleed 101," *IEEE security & privacy*, vol. 12, no. 4, pp. 63–67, 2014.

[61] W. Castryck and T. Decru, "An efficient key recovery attack on sidh (preliminary version)," *Cryptology ePrint Archive*, 2022.

[62] P. W. Shor, "Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer," *SIAM review*, vol. 41, no. 2, pp. 303–332, 1999.

[63] A. Joshi, R. Kumbhar, A. Mehta, V. Kosamkar, and H. Shetty, "Breaking rsa encryption using quantum computer," 2022.

[64] A. Albuainain, J. Alansari, S. Alrashidi, W. Alqahtani, J. Alshaya, and N. Nagy, "Experimental implementation of shor's quantum algorithm to break rsa," in *2022 14th International Conference on Computational Intelligence and Communication Networks (CICN)*. IEEE, 2022, pp. 748–752.

[65] C. Gidney and M. Ekerå, "How to factor 2048 bit rsa integers in 8 hours using 20 million noisy qubits," *Quantum*, vol. 5, p. 433, 2021.

[66] R. Bavdekar, E. J. Chopde, A. Agrawal, A. Bhatia, and K. Tiwari, "Post quantum cryptography: A review of techniques, challenges and standardizations," in *2023 International Conference on Information Networking (ICOIN)*. IEEE, 2023, pp. 146–151.

[67] O. Safaryan, L. Cherckesova, N. Lyashenko, P. Razumov, V. Chumakov, B. Akishin, and A. Lobodenko, "Modern hash collision cyberattacks and methods of their detection and neutralization," in *Journal of Physics: Conference Series*, vol. 2131, no. 2. IOP Publishing, 2021, p. 022099.

[68] I. P. A. E. Pratama and I. G. N. A. K. Adhitya, "Post quantum cryptography: Comparison between rsa and mceliece," in *2022 International Conference on ICT for Smart Society (ICISS)*. IEEE, 2022, pp. 01–05.

Acronyms

3DES	Triple DES
AD	Active Directory
AD RMS	Active Directory Rights Management Service
AEAD	Authenticated Encryption with Associated Data
AES	Advanced Encryption Standard
AIP	Azure Information Protection
API	Application Programming Interface
Azure AD	Azure Active Directory
Azure RMS	Azure Rights Management Service

C&C	Command and Control
CA	Certification Authority
CBC	Cipher Block Chaining
CIA	confidentiality, integrity, and availability
CIA Triad	CIA security goal triad
CLC	Client Licensor Certificate
CLI	Command Line Interface
CPU	Central Processing Unit
CRUD	create, read, update, and delete
CSPRNGs	cryptographically safe random number generators
DDoS	Distributed Denial of Service
DES	Data Encryption Standard

DLP

Data Loss/Leakage Prevention

DNS

Domain Name System

DoS

Denial of Service

DRM

Digital Rights Management

ECB

Electronic Code Book

ECC

Elliptic Curve Cryptography

ERM

Enterprise Resource Management

ERP

Enterprise Resource Planning

GCM

Galois/Counter mode

GUI

Graphical User Interface

HTTP

Hypertext Transfer Protocol

HTTPS

Hypertext Transfer Protocol Secure

IDE

Integrated Developing Environment

IRM

Information Rights Management

JSON

JavaScript Object Notation

JWT

JSON Web Token

LoR

List of Rights

LSP

Liskov Substitution Principle

M365

Microsoft 365

MitM

Man-in-the-Middle

NDA

Non-Disclosure Agreement

NIST

National Institute of Standards and Technology

O365

Microsoft Office 365

OS

operating system

PKI

Public Key Infrastructure

PL

Posting License

RAC

Rights Account Certificate

RAM

Random Access Memory

REST

Representational State Transfer

RMS

Rights Management Service

RSA

Rivest-Shamir-Adleman

SaaS

Software-as-a-Service

SCert

Server Certificate

SDFA

Server-Dependent File Access

SHA

Secure Hash Algorithm

SHS	Secure Hashing Standard
SIKE	Supersingular Isogeny Key Exchange
SLC	Server Licensor Certificate
SPC	Secure Processor Certificate
SPOF	Single Point of Failure
SQL	Structured Query Language
SSL/TLS	Secure Socket Layer / Transport Layer Security
UCert	User Certificate
UL	Use License
UML	Unified Modeling Language
XML	Extensible Markup Language

List of Figures

A Code of the Model Implementation

A.1 client_backend.py

```python
# typing imports
from typing import Tuple, Optional
# standard library imports
import sys
import os
import secrets
import pickle
from datetime import datetime as dt
# external imports
import requests  # requests
# project level imports
from package_dataconversion import *
from client_config_file_handler import ClientConfigFileHandler
from client_server_handler import ClientServerHandler
from client_certificates import ClientCertificate, ServerCertificateCopy
from package_cryptography import CryptoSystemAESGCM
from package_licenses import PostingLicenseUnencrypted,
    PostingLicenseEncrypted, PermissionObject
from client_container_file import ContainerFile


class ClientBackendException(Exception):
    pass


def client_backend_exception(function: Callable) -> Callable:
    def wrapper(*args, **kwargs):
        try:
            return function(*args, **kwargs)
        except:  # noqa
            raise ClientBackendException(f"ERROR IN CLIENT BACKEND! ({
                function.__name__})")
    return wrapper


class ClientBackend:
    program_path: str
    cch: ClientConfigFileHandler
    csh: ClientServerHandler
    client_certificate: ClientCertificate
    server_certificate: ServerCertificateCopy

    def __init__(self, with_bootstrapping: bool = True, username: str = Optional[str], password: Optional[str] = ""):
        self.program_path = os.path.dirname(os.path.abspath(sys.argv[0]))
        self.cch = ClientConfigFileHandler(self.program_path, "config.json", True)
        self.csh = ClientServerHandler()
        if with_bootstrapping:
            self.cch.enter("userid", self.csh.get_user_id())
            self.bootstrapping(username, password)
        else:
            self.csh.uname = self.cch.get("username")
            self.csh.password = self.cch.get("password")

    @client_backend_exception
    def bootstrapping(self, username: str, password: str) -> bool:  # raises error if bootstrapping is not possible
        pickle_configuration_object: Callable = lambda something: pickle.dumps(something).hex()
        if not self.is_bootstrapped():
            self.client_certificate = ClientCertificate()
            self.csh.uname = username
            self.csh.password = password
            self.cch.generate_user_configuration(self.csh.uname, self.csh.password)
            self.cch.enter("userid", self.csh.get_user_id())
            self.client_certificate.user_name = self.cch.get("username")
            self.client_certificate.user_id = self.cch.get("userid")
            self.client_certificate.device_id = self.cch.get("deviceid")
            self.cch.enter("certificate", pickle_configuration_object(self.client_certificate))
            server_public_key: Dict = json.loads(self.csh.get_public_server_key())
            self.server_certificate = ServerCertificateCopy(server_public_key)
            self.cch.enter("servercertificate", pickle_configuration_object(self.server_certificate))
            encrypted_user_certificate, encrypted_symmetric_key = self.encrypt_own_public_certificate()
            if self.csh.register_client_certificate(encrypted_user_certificate, encrypted_symmetric_key):
                self.cch.enter("bootstrapped", str(int(dt.utcnow().timestamp())))
                return True
            else:
                raise ClientBackendException("BOOTSTRAPPING FAILED.")
        return False

    @client_backend_exception
    def load_user_configuration(self) -> None:
        unpickle_configuration_object: Callable = lambda key: pickle.
            loads(bytes.fromhex(self.cch.get(key)))
        self.client_certificate = unpickle_configuration_object("
            certificate")
        self.server_certificate = unpickle_configuration_object("
            servercertificate")

    @client_backend_exception
    def is_bootstrapped(self) -> bool:
        if not self.cch:
            return False
        return self.cch.is_bootstrapped()

    @client_backend_exception
    def encrypt_own_public_certificate(self) -> Tuple[str, str]:
        assert self.server_certificate
        plain_user_certificate: str = self.client_certificate.
            get_compiled_json_version()
        plain_message: bytes = con_utf_bytes(plain_user_certificate)
        ciphertext, plain_symmetric_key = self.
            encrypt_data_randomly_symmetrically(plain_message)
        encrypted_symmetric_key: bytes = self.server_certificate.encrypt(
            con_utf_bytes(plain_symmetric_key))
        return ciphertext, con_bytes_hex(encrypted_symmetric_key)

    @client_backend_exception
    def encrypt_data_randomly_symmetrically(self, data: bytes) -> Tuple[
        str, str]:  # noqa
        crypto_system: CryptoSystemAESGCM = CryptoSystemAESGCM()
        crypto_system.generate_random_key()
        ciphertext: str = con_bytes_hex(crypto_system.encrypt(data))
        plain_symmetric_key: str = con_dict_json(crypto_system.
            export_full_private_key())
        return ciphertext, plain_symmetric_key

    @client_backend_exception
    def decrypt_data_symmetrically(self, data: str, key: str) -> bytes:
        # noqa
        crypto_system: CryptoSystemAESGCM = CryptoSystemAESGCM()
        crypto_system.import_full_private_key(con_json_dict(key))
        restored_plaintext = crypto_system.decrypt(con_hex_bytes(data))
        return restored_plaintext

    @client_backend_exception
    def double_decryption(self, double_encrypted_data: str, encrypted_key
        : str) -> bytes:
        assert double_encrypted_data and encrypted_key
        assert self.client_certificate
        decrypted_key: bytes = self.client_certificate.decrypt(
            con_hex_bytes(encrypted_key))
        restored_aes_key: Dict = con_json_dict(con_bytes_utf(
            decrypted_key))
        return self.decrypt_data_symmetrically(double_encrypted_data,
            con_dict_json(restored_aes_key))

    @client_backend_exception
    def double_encryption(self, plain_data: bytes) -> Tuple[str, str]:
        assert self.server_certificate
        ciphertext, plain_aes_key = self.
            encrypt_data_randomly_symmetrically(plain_data)
        encrypted_aes_key: str = con_bytes_hex(self.server_certificate.
            encrypt(con_utf_bytes(plain_aes_key)))
        return ciphertext, encrypted_aes_key

    @client_backend_exception
    def check_certificate_functionality(self) -> bool:
        solution: bytes = secrets.token_bytes(500_000)
        ciphertext, encrypted_aes_key = self.double_encryption(solution)
        payload: Dict = {"challenge": ciphertext, "key":
            encrypted_aes_key, "deviceid": self.cch.get("deviceid")}
        response: requests.Response = self.csh.authenticated_post("/
            testcertificates", con_dict_json(payload), 10)
        response_data: Dict = response.json()
        encrypted_challenge: str = response_data["data"]
        server_encrypted_aes_key: str = response_data["key"]
        restored_plaintext: bytes = self.double_decryption(
            encrypted_challenge, server_encrypted_aes_key)
        return restored_plaintext == solution

    @client_backend_exception
    def retrieve_group_information(self) -> Dict:
        payload: Dict = {"deviceid": self.cch.get("deviceid")}
        response_data: Dict = self.csh.authenticated_get("/groups", 10,
            con_dict_json(payload))
        encrypted_information: str = response_data["data"]
        server_encrypted_aes_key: str = response_data["key"]
        restored_plaintext: bytes = self.double_decryption(
            encrypted_information, server_encrypted_aes_key)
        groups: Dict = con_json_dict(con_bytes_utf(restored_plaintext))
        return deepcopy(groups)

    @client_backend_exception
    def retrieve_user_information(self) -> Dict:
        payload: Dict = {"deviceid": self.cch.get("deviceid")}
        response_data: Dict = self.csh.authenticated_get("/users", 10,
            con_dict_json(payload))
        encrypted_information: str = response_data["data"]
```

```python
        server_encrypted_aes_key: str = response_data["key"]
        restored_plaintext: bytes = self.double_decryption(encrypted_information, server_encrypted_aes_key)
        users: Dict = con_json_dict(con_bytes_utf(restored_plaintext))
        return deepcopy(users)

    @client_backend_exception
    def encrypt_and_sign_posting_license(self, posting_license: PostingLicenseUnencrypted) -> PostingLicenseEncrypted:
        assert not posting_license.encrypted
        assert posting_license.version
        assert posting_license.content_key
        assert posting_license.rights
        byte_representation: bytes = con_utf_bytes(posting_license.json_version())  # json to bytes
        encrypted_license_text, encrypted_key = self.double_encryption(byte_representation)  # double encryption
        signature: str = con_bytes_hex(self.client_certificate.sign(byte_representation))  # signs unencrypted
        new_license: PostingLicenseEncrypted = PostingLicenseEncrypted(encrypted_license_text, encrypted_key, signature)
        return deepcopy(new_license)

    @client_backend_exception
    def lock_container_posting_license(self, container: ContainerFile) -> ContainerFile:
        assert container.version and container.owner_id and \
               container.container_file_path and container.container_file_extension \
               and container.full_original_file_path and container.body and container.posting_license
        assert container.body_is_encrypted
        assert not container.posting_license.encrypted
        assert container.posting_license.content_key # noqa
        temporary_casted_license: PostingLicenseUnencrypted = container.posting_license  # noqa
        container.posting_license = self.encrypt_and_sign_posting_license(temporary_casted_license)
        return deepcopy(container)

    @client_backend_exception
    def get_protected_file_rights(self, protected_file: ContainerFile) -> \
            Tuple[PermissionObject, str, PostingLicenseUnencrypted]:
        assert protected_file.posting_license.encrypted
        payload: Dict = con_json_dict(protected_file.posting_license.json_version())
        payload["deviceid"] = con_bytes_utf(con_bytes_b64(con_utf_bytes(self.cch.get("deviceid"))))
        response: requests.Response = self.csh.authenticated_put("/evaluate", payload)
        response_data: Dict = response.json()
        decrypted_text: str = con_bytes_utf(self.double_decryption(response_data["data"], response_data["key"]))
        decrypted_payload: Dict = con_json_dict(decrypted_text)
        permissions: PermissionObject = PermissionObject(decrypted_payload["permissions"])
        body_key: str = ""
        recast_posting_license: Optional[PostingLicenseUnencrypted] = None
        if permissions.anything():
            body_key = decrypted_payload["bodykey"]
        if permissions.owner or permissions.repost:
            full_posting_license: str = decrypted_payload["fulllicense"]
            recast_posting_license = PostingLicenseUnencrypted(full_posting_license)
        return permissions, body_key, recast_posting_license

    @client_backend_exception
    def load_file(self, file_path: str) -> ContainerFile:
        assert os.path.isfile(file_path)
        assert self.cch
        return ContainerFile(file_path, self.cch.get("userid"))
```

A.2 client_certificates.py

```
# typing imports
# standard library imports
# external imports
# project level imports
from package_certificates import FullCertificate, HalfCertificate
from package_dataconversion import *


class ClientCertificate(FullCertificate):
    user_id: str
    user_name: str
    device_id: str

    def __init__(self):
        super().__init__(2048)
        self.generate_keys()

    def get_compiled_json_version(self) -> str:
        assert self.user_name and self.user_id and self.device_id
        assert self.export_public_key()
        payload: Dict = {
            "username": self.user_name,
            "userid": self.user_id,
            "deviceid": self.device_id,
            "publickey": con_dict_json(self.export_public_key())
        }
        return json.dumps(payload)


class ServerCertificateCopy(HalfCertificate):
    pass
```

A.3 client_cli.py

```
# typing imports
# standard library imports
# external imports
# project level imports
import client_config_file_handler  # required to stick to client config
    handler singleton pattern when restarting
from client_backend import *
from client_container_file import ContainerFileException
from file_handler import *
from file_handler_factory import FileHandlerFactory
from package_licenses import PostingLicense, RightsObject


class InsufficientPermissionsException(Exception):
    pass


class PolicyIntegrityException(Exception):
    pass


class CommandLineInterface:
    backend: ClientBackend  # backend performing operations
    current_file: str  # path to container
    current_container: ContainerFile  # container object
    informational_prints: bool  # information printing flag
    file_handler: FileHandler  # file handler based on file type
    file_handler_name: str  # name of the used file handler
    file_status: int = 0
    permission_cache: Tuple[PermissionObject, Optional[str], Optional[
        PostingLicenseUnencrypted]]
    group_cache: Dict
    user_cache: Dict

    def __init__(self, target_file: str, informational_prints: bool =
        True):
        self.informational_prints = informational_prints
        self.determine_current_file(target_file)
        self.info("FRONTEND STARTING...")
        self.startup()
        self.info("LOADING FILE...")
        self.load_argument_file()
        self.info("FRONTEND READY!")
        self.run()

    def restart(self, target_file: str) -> None:
        self.info("destroying objects...")
        del self.backend.cch
        client_config_file_handler.CONFIG_FILE_HANDLER_INSTANCES -= 1
        del self.backend
        del self.current_container
        del self.file_handler
        del self.permission_cache
        del self.group_cache
        del self.user_cache
        self.println("")
        self.println("#")
        self.determine_current_file(target_file)
        self.info("RESTARTING FRONTEND...")
        self.startup()
        self.info("LOADING FILE...")
        self.load_argument_file()
        self.info("FRONTEND READY!")
        self.run()

    def startup(self) -> None:
        self.info("resetting permission cache...")
        self.permission_cache = (None, "", None)  # noqa
        self.info("resetting group cache...")
        self.group_cache = {}
        self.user_cache = {}
        self.info("loading backend...")
        self.backend = ClientBackend(False)
        self.info("loading client configuration...")
        try:
            if self.backend.is_bootstrapped():
                self.info("user is already bootstrapped")
                self.info("loading configuration...")
                self.backend.load_user_configuration()
            else:
                self.info("attempting to bootstrap user...")
                username: str = self.input_request("ENTER USERNAME > ",
                    False)
                password: str = self.input_request("ENTER PASSWORD > ", True)
                if self.backend.bootstrapping(username, password):
                    self.info("bootstrapping complete")
                else:
                    raise ClientBackendException("bootstrapping failed for unknown reason")
        except ClientBackendException:
            self.error("error occurred during bootstrapping")
            self.info("make sure you have an internet connection and entered the credentials correctly")
            self.critical("BOOTSTRAPPING COULD NOT BE COMPLETED!")
        self.update_user_and_group_cache()

    def update_user_and_group_cache(self, silent: Optional[bool] = False) -> bool:
        success: bool = True
        if not silent:
            self.info("fetching group information from server...")
        try:
            self.group_cache = self.backend.retrieve_group_information()
            if not silent:
                self.info("user groups received.")
        except ClientBackendException:
            success = False
            if not silent:
                self.error("user groups could not be fetched!")
        if not silent:
            self.info("fetching user information from server...")
        try:
            self.user_cache = self.backend.retrieve_user_information()
            if not silent:
                self.info("users received.")
        except ClientBackendException:
            success = False
            if not silent:
                self.error("users could not be fetched!")
        return success

    def determine_current_file(self, target_file: str) -> None:
        self.info("checking target file...")
        if not os.path.exists(target_file):
            self.error(f"TARGET FILE DOES NOT EXIST! ({target_file})")
            if target_file.endswith(".container"):
                self.info("trying to find matching plain file...")
                target_file = target_file[:-10]
            else:
                self.info("trying to find matching container...")
                target_file += ".container"
            if not os.path.exists(target_file):
                self.error("TARGET FILE NOT FOUND!")
            else:
                self.info("corresponding file found.")
        raw_content: bytes
        with open(target_file, "rb") as opened_file:
            raw_content = opened_file.read()
        if not raw_content:
            self.critical("FILE APPEARS TO BE EMPTY!")
        self.current_file = os.path.abspath(target_file)

    def load_argument_file(self) -> None:
        self.info("loading file handler factory...")
        handler_factory = FileHandlerFactory()
        self.info("loading file...")
        self.load_file()
        self.info("loading file permissions...")
        self.__reconstruct_permission_cache()
        permissions, _, _ = self.permission_cache
        self.info("creating file handler...")
        self.file_handler, matching_file_handler = handler_factory.create
            (self.current_container, permissions)
        self.file_handler_name = str(type(self.file_handler).__name__)
        original_file_type: str = self.current_container.
            full_original_file_path.strip().split(".")[-1].lower()
        self.info(f"original file extension identified as {
            original_file_type}")
        annotation: str = "(considering upgrade)" if not
            matching_file_handler else ""
        self.info(f"using file handler {self.file_handler_name} {
            annotation}.")
        if self.__cached_body_key():
            self.info("injecting key into file handler...")
            self.file_handler.secret_key = self.__cached_body_key()
            if not matching_file_handler:
                self.info("checking file handler upgrade...")
                alternative_file_handler, success = \
                    handler_factory.create(self.current_container,
                        permissions, self.file_handler.secret_key)
                if success:
                    self.file_handler = alternative_file_handler
                    self.file_handler_name = str(type(self.file_handler).
                        __name__)
                    self.info(f"file handler has been upgraded to {self.
                        file_handler_name}")
                else:
                    alternative_handler_name = str(type(
                        alternative_file_handler).__name__)
                    self.info(f"no better file handler identified,
                        alternative was {alternative_handler_name}")
        self.file_status = self.get_file_status()

    def info(self, *args, **kwargs) -> None:
        if self.informational_prints:
            print("info:        ", *args, **kwargs)

    @staticmethod
    def input_request(text: str, hidden_input: bool = "False") -> str:
        result: str
        print("request:      " + text, end="", flush=True)
        if hidden_input:
            result = getpass.getpass("")
        else:
            result = input("")
        return result

    @staticmethod
    def error(*args, **kwargs) -> None:
        print("\nerror:       ", *args, **kwargs)

    @staticmethod
    def critical(*args, **kwargs) -> None:
        print("critical:    ", *args, **kwargs)
        exit(1)

    @staticmethod
    def println(character: str = "-", length: int = 64) -> None:
        print(character * length)

    def run(self) -> None:
        try:
            self.println()
            self.command_info()
        except Exception as e:
            print(e)
            self.error("serious problem in information displaying...")
        try:
            self.commands_available()
        except Exception as e:
            print(e)
            self.error("serious problem in displaying available commands
                ...")
        while True:
            command = input("\n> ")
            command = command.strip().lower()
            try:
                if command == "":
                    continue
                elif command == "info":
                    self.command_info()
                elif command == "help":
```

```python
                    self.command_help()
                elif command == "view":
                    self.command_view()
                elif command == "edit":
                    self.command_edit()
                elif command == "protect":
                    self.command_protect()
                elif command == "decrypt":
                    self.command_decrypt()
                elif command == "backendtest":
                    self.command_test_backend()
                elif command == "exit":
                    exit(0)
                else:
                    self.commands_available()
            except Exception as e:
                print(e)
                self.error("serious problem during command loop execution ...")
                self.info(f"execution of command {command} aborted.")

    def commands_available(self) -> None:
        print(f"\nAVAILABLE COMMANDS: info, help, {self.file_handler.get_supported_command_text()}, exit")
        self.println()

    def command_help(self) -> None:
        buffer: str = ""
        buffer += f"\nCOMMANDS:\n"
        buffer += f"info:           shows information on file and login status\n"
        buffer += f"help:           shows this help page\n"
        buffer += f"protect:        locks the container or new file*\n"
        buffer += f"decrypt:        permanently decrypts container*\n"
        buffer += f"view:           shows the file content*\n"
        buffer += f"edit:           opens and editor*\n"
        buffer += f"exit:           exits this program\n\n"
        buffer += f"*only works when permitted\n\n"
        buffer += f"note: Not all operations are supported for all file types.\n"
        print(buffer)
        self.println()

    def command_info(self) -> None:
        if not (self.group_cache and self.user_cache):
            self.update_user_and_group_cache(True)
        get: Callable = lambda key: self.backend.cch.get(key)
        protection_status: str = 'unprotected' if not self.is_protected() else 'protected'
        self.println("")
        print(f"FILE:           {os.path.split(self.current_file)[1]} ({protection_status}, status {self.file_status})")
        print(f"FILE HANDLER:   {self.file_handler_name}")
        self.println("")
        print(f"USER:           {get('username')}@{get('serverdomain')}")
        print(f"USER ID:        {get('userid')}")
        print(f"DEVICE ID:      {get('deviceid')}")
        print(f"PERMISSIONS:    ", end="")
        permissions = self.__cached_personal_permissions()
        inverted: List[str] = [key for key in permissions.keys if permissions.get(key)]
        print(", ".join(inverted))
        self.println("")
        print(f"SERVER GROUPS:  ", end="")
        groups = self.group_cache
        group_text = "\n                ".join([f"{key} ({groups.get(key)})" for key in groups.keys()])
        print(group_text)
        self.println("")
        print(f"SERVER USERS:   ", end="")
        users = self.user_cache
        users_text = "\n                ".join([f"{key} ({users.get(key)})" for key in users.keys()])
        print(users_text)

    def __reconstruct_permission_cache(self) -> None:
        try:
            del self.permission_cache
        except NameError:
            pass
        cc = self.current_container
        personal_permissions: PermissionObject = None # noqa
        body_key: str
        full_posting_license: PostingLicenseUnencrypted
        file_status: int = self.get_file_status()
        if file_status <= 2:
            personal_permissions = PermissionObject()
            personal_permissions.set_all()
            body_key = ""
            full_posting_license = None # noqa
        elif file_status == 4:
            personal_permissions, body_key, full_posting_license = deepcopy(self.backend.get_protected_file_rights(cc))
        else:  # status 3
            casted_version: PostingLicenseUnencrypted = cc.posting_license  # noqa
            own_id: str = self.backend.cch.get("userid")
            for right in casted_version.rights:
                if right.subject == own_id:
                    personal_permissions = deepcopy(right.permissions)
            body_key = casted_version.content_key
            full_posting_license = deepcopy(casted_version)
        self.permission_cache = personal_permissions, body_key,
            full_posting_license

    def __cached_personal_permissions(self) -> PermissionObject:
        permissions, _, _ = self.permission_cache
        return permissions

    def __cached_body_key(self) -> str:
        _, key, _ = self.permission_cache
        return key

    def __cached_posting_license(self) -> PostingLicenseUnencrypted:
        _, _, posting_license = self.permission_cache
        return posting_license

    def posting_license_editor(self, pre_existing_license: PostingLicense
        , key: str = "") \
            -> Optional[PostingLicenseUnencrypted]:
        assert self.backend.cch.get("userid")
        new_license: PostingLicenseUnencrypted
        if isinstance(pre_existing_license, PostingLicenseUnencrypted):
            assert pre_existing_license.content_key
            new_license = deepcopy(pre_existing_license)
        else:
            if not key:
                raise Exception("no key received for new posting license!
                    ")
            new_license = PostingLicenseUnencrypted()
            new_license.content_key = key
        self.command_editor_show(new_license)
        self.command_editor_help()
        while True:
            command = input("\n> ")
            command = command.strip().lower()
            if command == "show":
                self.command_editor_show(new_license)
            elif command == "users":
                self.command_editor_users()
            elif command == "groups":
                self.command_editor_groups()
            elif command == "add":
                new_license = self.command_editor_add(new_license)
            elif command == "remove":
                new_license = self.command_editor_remove(new_license)
            elif command == "abort":
                return None
            elif command == "done":
                try:
                    new_license = self.editor_integrity_check(new_license)
                except PolicyIntegrityException:
                    self.error("the newly configured permission was not allowed")
                    return None
                break
            else:
                self.command_editor_help()
        return new_license

    def command_editor_help(self) -> None:
        buffer: str = ""
        buffer += f"\nCOMMANDS:\n"
        buffer += f"help:           shows this help page\n"
        buffer += f"show:           shows the currently set permissions\n"
        buffer += f"groups:         shows the groups\n"
        buffer += f"users:          shows the groups\n"
        buffer += f"add:            adds a group/user to whitelist\n"
        buffer += f"remove:         removes a group/user from whitelist\n"
        buffer += f"abort:          discards changes and exits the editor\n"
        buffer += f"done:           saves changes and exits the editor"
        print(buffer)
        self.println("")

    @staticmethod
    def restore_key(dictionary: Dict, value: Any) -> str:
        for key in dictionary:
            if dictionary.get(key, "") == value:
                return key
        return ""

    def command_editor_show(self, current_license: PostingLicenseUnencrypted) -> None:
        print(f"CURRENT LICENSE:")
        for ro in current_license.rights:
            subject_id: str = ro.subject
            subject_name: str = self.restore_key(self.group_cache, subject_id)
            if not subject_name:
                subject_name: str = self.restore_key(self.user_cache, subject_id)
            print(f"{subject_name} --> {ro.permissions}")

    def get_id_from_cache(self, name: str) -> str:
        entity_id = self.group_cache.get(name, "")
        if not entity_id:
            entity_id = self.user_cache.get(name, "")
        return entity_id

    def command_editor_add(self, current_license: PostingLicenseUnencrypted) -> PostingLicenseUnencrypted:
        name: str = input("enter name of subject (user/group) to add > ")
        entity_id = self.get_id_from_cache(name)
        if not entity_id:
            self.error("subject name not found")
            return current_license
        for rights_object in current_license.rights:
            if rights_object.subject == entity_id:
                self.error("subject already in license!")
                return current_license
        new_rights = RightsObject()
        new_rights.subject = entity_id
        possible_permissions: List[str] = new_rights.permissions.keys
        not_okay: bool = True
        while not_okay:
            print("POSSIBLE PERMISSIONS:\n" + ", ".join(possible_permissions))
            numbers: str = input("please enter a 1 (yes) or 0 (no) for every permission > ")
            if len(numbers) != len(possible_permissions):
                continue
            emptiness: str = numbers.strip().replace("0", "").replace("1", "")
            if emptiness:
                continue
            for i, permission in enumerate(possible_permissions):
                new_rights.permissions.set(permission, (False if numbers[i] == '0' else True))
            not_okay = False
        current_license.rights.append(deepcopy(new_rights))
        self.command_editor_show(current_license)
        return current_license

    def command_editor_remove(self, current_license: PostingLicenseUnencrypted) -> PostingLicenseUnencrypted:
        while True:
            subject_ids: List[str] = current_license.get_subjects()
            subject_names: List[str] = [self.restore_key(self.user_cache, entity) for entity in subject_ids]
            subject_names += [self.restore_key(self.group_cache, entity) for entity in subject_ids]
            subject_names = [x for x in subject_names if x]
            print(f"the following entities have rights: {', '.join(subject_names)}")
            name: str = input("enter name of subject (user/group) to delete > ")
            if not name:
                break
            entity_id = self.get_id_from_cache(name)
            skip: bool = False
            for rights_object in current_license.rights:
                if rights_object.subject == entity_id:
                    current_license.rights.remove(rights_object)
                    skip = True
                    break
            if not skip:
                print(f"{name} was not found in license")
        return deepcopy(current_license)

    def editor_integrity_check(self, current_license: PostingLicenseUnencrypted) -> PostingLicenseUnencrypted:
        # checking if own user will (still) have reposting rights
        updated_subjects: List[str] = current_license.get_subjects()
        own_id: str = self.backend.cch.get("userid")
        if own_id not in updated_subjects:
            self.error("own id not found in policy!")
            self.info("adding reposting rights...")
            emergency_rights = RightsObject()
            emergency_rights.subject = own_id
            emergency_permission = PermissionObject()
            emergency_permission.set("repost", True)
            emergency_rights.permissions = emergency_permission
            current_license.rights.append(emergency_rights)
        at_least_one_entity_has_reposting_rights: bool = False
        # checking if anyone can overwrite (/repost/protect) the file's permissions
        for rights_object in current_license.rights:
            if rights_object.permissions.get("owner") or rights_object.permissions.repost:
                at_least_one_entity_has_reposting_rights = True
        if not at_least_one_entity_has_reposting_rights:
            self.error("NO ENTITY CAN CHANGE THE RIGHTS ANYMORE IN THIS CONFIGURATION!")
            raise PolicyIntegrityException()
        return deepcopy(current_license)

    def command_editor_groups(self) -> None:
        print(f"SERVER GROUPS:  ", end="")
        groups = self.group_cache
        group_text = "\n                ".join([f"{key} ({groups.get(key)})" for key in groups.keys()])
        print(group_text)

    def command_editor_users(self) -> None:
        print(f"SERVER USERS:   ", end="")
        users = self.user_cache
```

```python
        users_text = "\n                ".join([f"{key} ({users.get(key)})" for key in users.keys()])
        print(users_text)

    def full_server_updates(self) -> None:
        self.info("updating server-provided information...")
        self.info("reconstructing permission cache...")
        self.__reconstruct_permission_cache()
        self.info("updating group information...")
        self.group_cache = self.backend.retrieve_group_information()
        self.info("updating user information...")
        self.user_cache = self.backend.retrieve_user_information()

    def command_protect(self) -> None:
        try:
            self.full_server_updates()
            permissions = self.file_handler.permissions
            posting_license: PostingLicenseUnencrypted
            try:
                assert permissions.owner or permissions.repost
            except AssertionError:
                raise InsufficientPermissionsException
            bypass_key: str  # necessary if file status is 2 (unencrypted container) or 4 (protected container)
            if not self.current_container.body_is_encrypted:
                assert self.file_status <= 2
                self.info(f"file body is not encrypted (status {self.get_file_status()})")
                bypass_key = self.current_container.encrypt_body()
                posting_license = self.posting_license_editor(self.current_container.posting_license,
                                                              bypass_key)  # None, key
            elif self.get_file_status() >= 4:
                self.info("file is already protected, asserting permission change.")
                existing_license: PostingLicenseUnencrypted = self.__cached_posting_license()
                posting_license = self.posting_license_editor(existing_license, self.__cached_body_key())
            else:  # status 3
                self.info("file is not protected yet.")
                self.println("")
                posting_license = self.posting_license_editor(self.current_container.posting_license)
            self.println("")
            if posting_license:  # if editor was successful
                self.current_container.posting_license = posting_license
                self.current_container = self.backend.lock_container_posting_license(
                    self.current_container)  # expects status 3
                self.info("posting license published")
                self.current_container.save_container_file()
                self.info("container saved")
                if os.path.isfile(self.current_container.full_original_file_path):
                    try:
                        self.current_container.delete_original_file()
                        self.info("original file removed")
                    except ContainerFileException:
                        self.error("original file could not be removed...")
                self.info("closing session")
                self.restart(self.current_container.container_file_path)
            else:
                self.error("posting license editing was aborted.")
                self.commands_available()
        except InsufficientPermissionsException:
            self.error("you are not allowed to protect this file")

    def command_decrypt(self) -> None:
        if self.get_file_status() < 4:
            self.error("file is not protected")
            raise Exception("file is not protected")
        try:
            permissions = self.file_handler.permissions
            try:
                assert permissions.owner or permissions.decrypt
            except AssertionError:
                raise InsufficientPermissionsException
            file_path: str = deepcopy(self.current_container.container_file_path)
            if file_path.endswith(self.current_container.container_file_extension):
                file_path = file_path[:-len(self.current_container.container_file_extension)]  # removes extension
            if os.path.isfile(file_path):
                self.error("TARGET FILE ALREADY EXISTS!")
                raise Exception("TARGET FILE ALREADY EXISTS!")
            with open(file_path, "wb") as opened_file:
                opened_file.write(self.file_handler._get_decrypted_body())  # noqa
            os.remove(os.path.join(file_path + self.current_container.container_file_extension))
            self.info("file decrypted")
            self.info("closing session")
            self.restart(file_path)
        except InsufficientPermissionsException:
```

```python
            self.error("you are not allowed to decrypt this file")

    def command_view(self) -> None:
        try:
            self.file_handler.view()
        except FileHandlerPermissionException:
            self.error("you are not allowed to view this file")
        except FileHandlerSupportException:
            self.error("viewing this file type is not supported at this point")

    def command_edit(self) -> None:
        preserved_file_status: int = self.get_file_status()
        assert preserved_file_status > 2
        preserved_content_key: str
        if preserved_file_status < 4:  # file status 3
            preserved_content_key = self.current_container.posting_license.content_key  # noqa
        else:  # file status 4
            preserved_content_key = self.__cached_body_key()
            assert self.file_handler.secret_key == preserved_content_key
        try:
            self.current_container = self.file_handler.edit()
            if preserved_file_status < 4:  # status 3 --> export into status 1 permanent file
                assert preserved_content_key == self.current_container.posting_license.content_key  # noqa
                with open(self.current_container.full_original_file_path, "wb") as opened_file:
                    temp_license: PostingLicenseUnencrypted = self.current_container.posting_license  # noqa
                    opened_file.write(self.current_container.get_decrypted_body(temp_license.content_key))
            else:  # status 4 --> body and PL are encrypted already
                self.current_container.save_container_file()
        except FileHandlerPermissionException:
            self.error("you are not allowed to edit this file")
        except FileHandlerSupportException:
            self.error("editing this file type is not supported at this point")

    def command_test_backend(self) -> bool:
        result = False
        self.info("testing backend certificates with random payload...")
        try:
            result: bool = self.backend.check_certificate_functionality()
        except ClientBackendException as e:
            self.error(e)
        if result:
            self.info("backend roundtrip certificate test successful")
        else:
            self.error("backend roundtrip certificate test failed")
        return result

    def is_protected(self) -> bool:
        return self.get_file_status() >= 4

    def is_encrypted(self) -> bool:
        return self.get_file_status() >= 3

    def get_file_status(self) -> int:
        status: int
        if self.current_container.posting_license is None:
            status = 1
        else:
            if not self.current_container.posting_license.encrypted:
                if not self.current_container.body_is_encrypted:
                    status = 2
                else:
                    status = 3
            else:
                status = 4
        assert 5 > status > 0
        self.file_status = status
        return self.file_status

    def load_file(self) -> None:
        self.current_container = self.backend.load_file(self.current_file
            )


if __name__ == "__main__":
    if not len(sys.argv) > 1:
        print(f"CRITICAL: NO FILE PATH HAS BEEN SUPPLIED AS ARGUMENT!")
        exit(1)
    cli: CommandLineInterface = CommandLineInterface(sys.argv[1].strip())
```

A.4 client_config_file_handler.py

```
# typing imports
from typing import Callable, List, Dict
# standard library imports
import os
import json
import platform
from hashlib import sha512
# external imports
# project level imports
```

```
from package_dataconversion import deepcopy, con_utf_bytes


class ClientConfigFileException(Exception):
    pass


def client_config_file_exception(function: Callable) -> Callable:
    def wrapper(*args, **kwargs):
        try:
            return function(*args, **kwargs)
        except:  # noqa
            raise ClientConfigFileException(f"ERROR IN CLIENT CONFIG FILE
                 HANDLING! ({function.__name__})")
    return wrapper


CONFIG_FILE_HANDLER_INSTANCES: int = 0


class ClientConfigFileHandler:
    target_file: str
    __current_dict: Dict
    __keys_mandatory: List

    def __new__(cls, *args, **kwargs):  # Singleton implementation
        global CONFIG_FILE_HANDLER_INSTANCES
        if CONFIG_FILE_HANDLER_INSTANCES > 0:
            raise ClientConfigFileException("ONLY ONE INSTANCE OF
                ClientConfigFileHandler ALLOWED!")
        CONFIG_FILE_HANDLER_INSTANCES += 1
        return super(ClientConfigFileHandler, cls).__new__(cls)

    def __init__(self, program_directory: str, file_name: str, create_new
        : bool = False):
        self.__keys_mandatory = "username userid password serverdomain
            deviceid certificate " \
                                "servercertificate bootstrapped".split()
        if not file_name.strip():
            raise ClientConfigFileException("NO FILE SPECIFIED!")
        self.target_file = os.path.join(program_directory, file_name)
        if create_new and not os.path.isfile(self.target_file):
            self.__current_dict = {}
            self.save_file()
        self.load_file()

    @client_config_file_exception
    def load_file(self) -> None:
        assert os.path.isfile(self.target_file)
        with open(self.target_file, "r") as file:
```

```python
            self.__current_dict = json.load(file)

    @client_config_file_exception
    def save_file(self) -> None:
        with open(self.target_file, "w") as file:
            json.dump(self.__current_dict, file)

    def is_bootstrapped(self) -> bool:
        self.load_file()
        checked_data: Dict = deepcopy(self.__current_dict)
        for key in self.__keys_mandatory:
            if checked_data.get(key, "") == "":  # key = potentially missing attribute
                return False
        return True

    def enter(self, key: str, value: str) -> None:
        self.load_file()
        self.__current_dict[key] = value
        self.save_file()

    @deepcopy
    def get(self, key: str) -> str:
        self.load_file()
        return self.__current_dict.get(key, "")

    @deepcopy
    def get_mandatory_key_list(self) -> List:
        return self.__keys_mandatory

    def generate_user_configuration(self, username: str, password: str) -> None:
        self.enter("username", username)
        self.enter("password", password)
        self.enter("deviceid", self.get_device_id())
        self.enter("serverdomain", "http://localhost:11235")

    @staticmethod
    def get_device_id() -> str:
        components: List[str] = [platform.node(), platform.system(), platform.processor()]
        # for testing, os.getcwd() can be added to the components to simulate different devices in different folders
        uncooked: bytes = con_utf_bytes(":".join(components))
        return sha512(uncooked).hexdigest()
```

A.5 client_container_file.py

```
# typing imports
from typing import Optional, Callable, Dict
# standard library imports
import os
import pickle
# external imports
# project level imports
from package_dataconversion import deepcopy, con_dict_json, con_json_dict
from package_cryptography import CryptoSystemAES
from package_licenses import PostingLicense, PostingLicenseUnencrypted


class ContainerFileException(Exception):
    pass


def container_file_exception(function: Callable) -> Callable:
    def wrapper(*args, **kwargs):
        try:
            return function(*args, **kwargs)
        except:  # noqa
            raise ContainerFileException(f"ERROR WITH CONTAINER FILE! ({
                function.__name__})")
    return wrapper


class ContainerFile:
    version: int = 1
    owner_id: str
    container_file_path: str
    full_original_file_path: str
    body: bytes
    body_is_encrypted: bool
    posting_license: PostingLicense
    container_file_extension: str

    def __init__(self, source_file: str, owner_id: Optional[str] = ""):
        self.container_file_extension = ".container"
        assert source_file
        if source_file.endswith(self.container_file_extension):
            self.load_existing_container_file(source_file)
        else:
            self.create_new_container_file(source_file, owner_id)

    @container_file_exception
    def load_existing_container_file(self, path: str) -> None:
        assert os.path.isfile(path) and path.endswith(self.
            container_file_extension)
        loaded_container: ContainerFile
        with open(path, "rb") as opened_file:
            loaded_container = pickle.load(opened_file)
        self.version = loaded_container.version
        self.owner_id = loaded_container.owner_id
        self.container_file_path = os.path.abspath(path)
        self.full_original_file_path = loaded_container.
            full_original_file_path
        self.body = loaded_container.body
        self.body_is_encrypted = loaded_container.body_is_encrypted
        self.posting_license = loaded_container.posting_license
        assert self.posting_license.encrypted

    @container_file_exception
    def create_new_container_file(self, source_file: str, owner_id: str)
        -> None:
        assert source_file and owner_id
        assert os.path.isfile(source_file)
        self.owner_id = owner_id
        self.full_original_file_path = os.path.abspath(source_file)
        self.container_file_path = self.full_original_file_path + self.
            container_file_extension
        self.body_is_encrypted = False
        with open(self.full_original_file_path, "rb") as opened_file:
            self.body = opened_file.read()
        self.posting_license = PostingLicenseUnencrypted()
        self.posting_license.content_key = self.encrypt_body()
        self.posting_license.add_owner(owner_id)  # noqa

    @container_file_exception
    def encrypt_body(self, known_json_key: str = "") -> str:
        assert self.body
        assert not self.body_is_encrypted
        crypto_system: CryptoSystemAES = deepcopy(CryptoSystemAES())  #
            is local to prevent accidentally reusing a key
        if known_json_key:
            crypto_system.import_full_private_key(con_json_dict(
                known_json_key))  # uses existing key
        else:
            crypto_system.generate_random_key()  # creates random key
        self.body = crypto_system.encrypt(self.body)
        assert self.body
        self.body_is_encrypted = True
        return con_dict_json(crypto_system.export_full_private_key())

    @deepcopy
    @container_file_exception
    def get_decrypted_body(self, json_key: str) -> bytes:
        assert json_key
        if not self.body_is_encrypted:
            return self.body
        key: Dict = con_json_dict(json_key)
        crypto_system: CryptoSystemAES = CryptoSystemAES()
        crypto_system.import_full_private_key(key)
        return crypto_system.decrypt(self.body)

    @container_file_exception
    def save_container_file(self) -> None:
        assert self.version and self.owner_id and self.
            container_file_path and self.full_original_file_path
        assert self.body and self.body_is_encrypted and self.
            posting_license.encrypted and self.container_file_extension
        with open(self.container_file_path, "wb") as opened_file:
            pickle.dump(self, opened_file)

    @container_file_exception
    def delete_original_file(self) -> None:
        assert os.path.isfile(self.container_file_path)
        assert os.path.isfile(self.full_original_file_path)
        os.remove(self.full_original_file_path)
```

A.6 client_server_handler.py

```
# typing imports
from typing import Optional, Callable, Dict
# standard library imports
# external imports
import requests  # requests
# project level imports
from package_dataconversion import con_dict_json


class ServerHandlerException(Exception):
    pass


def server_handler_exception(function: Callable) -> Callable:
    def wrapper(*args, **kwargs):
        try:
            return function(*args, **kwargs)
        except:  # noqa
            raise ServerHandlerException(f"ERROR IN SERVER HANDLER! ({
                function.__name__})")
    return wrapper


class ClientServerHandler:
    server_domain: str
    uname: str
    password: str
    jwt_token: str

    def __init__(self, user_name: Optional[str] = None, password:
        Optional[str] = None) -> None:
        self.server_domain = "http://localhost:11235"
        self.uname = user_name if user_name else ""
        self.password = password if password else ""
        self.jwt_token = ""

    @server_handler_exception
    def authenticate(self) -> None:
        self.jwt_token = ""
        response = requests.post(f"{self.server_domain}/login", auth=(
            self.uname, self.password), timeout=5)
        self.jwt_token = response.json()['data']

    @server_handler_exception
    def get(self, path: str, timeout: int = 3) -> Dict:
        response = requests.get(self.server_domain + path, timeout=
            timeout)
        return response.json()

    @server_handler_exception
    def authenticated_get(self, path: str, timeout: int = 3, json_data:
        str = "") -> Dict:
        if not self.jwt_token:
            assert self.uname and self.password
            self.authenticate()
        headers = {'Authorization': ("Bearer " + self.jwt_token)}
        try:
            if json_data:
                response = requests.get(self.server_domain + path,
                    headers=headers, timeout=timeout, json=json_data)
            else:
                response = requests.get(self.server_domain + path,
                    headers=headers, timeout=timeout)
        except Exception as e:
            print(e)
            response = requests.Response()
            response.status_code = 450
        if 500 > response.status_code >= 400:
            self.authenticate()
            headers = {'Authorization': ("Bearer " + self.jwt_token)} if
                len(self.jwt_token) > 0 else {}
            if json_data:
                response = requests.get(self.server_domain + path,
                    headers=headers, timeout=timeout, json=json_data)
            else:
                response = requests.get(self.server_domain + path,
                    headers=headers, timeout=timeout)
        return response.json()

    @server_handler_exception
    def authenticated_post(self, path: str, json_data: str, timeout: int
        = 3) -> requests.Response:
        if not self.jwt_token:
            self.authenticate()
        headers = {'Authorization': ("Bearer " + self.jwt_token)} if len(
            self.jwt_token) > 0 else {}
        try:
            response = requests.post(self.server_domain + path, json=
                json_data, headers=headers, timeout=timeout)
        except Exception as e:
            print(e)
            response = requests.Response()
            response.status_code = 450
        if 500 > response.status_code >= 400:
            self.authenticate()
            headers = {'Authorization': ("Bearer " + self.jwt_token)} if
                len(self.jwt_token) > 0 else {}
            response = requests.post(self.server_domain + path, json=
                json_data, headers=headers, timeout=timeout)
        return response

    @server_handler_exception
    def authenticated_put(self, path: str, json_data: str, timeout: int =
         3) -> requests.Response:
        if not self.jwt_token:
            self.authenticate()
        headers = {'Authorization': ("Bearer " + self.jwt_token)} if len(
            self.jwt_token) > 0 else {}
        try:
            response = requests.put(self.server_domain + path, json=
                json_data, headers=headers, timeout=timeout)
        except Exception as e:
            print(e)
            response = requests.Response()
            response.status_code = 450
        if 500 > response.status_code >= 400:
            self.authenticate()
            headers = {'Authorization': ("Bearer " + self.jwt_token)} if
                len(self.jwt_token) > 0 else {}
            response = requests.put(self.server_domain + path, json=
                json_data, headers=headers, timeout=timeout)
        return response

    @server_handler_exception
    def get_public_server_key(self) -> Dict:
        return self.authenticated_get("/certificate")["data"]

    @server_handler_exception
    def register_client_certificate(self, encrypted_payload: str,
        server_encrypted_meta_key: str) -> bool:
        json_data: str = con_dict_json({"payload": encrypted_payload, "
            key": server_encrypted_meta_key})
        response = self.authenticated_post("/newdevice", json_data)
        return response.status_code == 201

    @server_handler_exception
    def get_user_id(self) -> str:
        return self.authenticated_get("/profile")["data"]
```

A.7 file_handler.py

```
# typing imports
from typing import Callable, List
# standard library imports
import os
import secrets
import getpass
import tempfile
# external imports
# project level imports
from client_container_file import ContainerFile
from package_dataconversion import deepcopy
from package_licenses import PermissionObject, PostingLicenseUnencrypted


class FileHandlerException(Exception):
    pass


class FileHandlerMissingKeyException(FileHandlerException):
    pass


class FileHandlerPermissionException(FileHandlerException):
    pass


class FileHandlerSupportException(FileHandlerException):
    pass


def file_handler_exception(function: Callable) -> Callable:
    def wrapper(*args, **kwargs):
        try:
            return function(*args, **kwargs)
        except:  # noqa
            raise FileHandlerException(f"ERROR IN FILE HANDLER! ({
                function.__name__})")

    return wrapper


class FileHandler:
    secret_key: str
    supported_commands: List
    container: ContainerFile
    permissions: PermissionObject

    def __init__(self, container: ContainerFile, permissions:
        PermissionObject, json_key: str = ""):
        assert container and permissions
        self.container = container
        self.secret_key = json_key
        self.permissions = permissions
        self.supported_commands = deepcopy(["protect, decrypt"])
        if type(self).__name__ == "FileHandler":
            raise NotImplementedError

    @file_handler_exception
    def view(self) -> None:
        raise FileHandlerSupportException

    @file_handler_exception
    def edit(self) -> ContainerFile:
        raise FileHandlerSupportException

    def _get_decrypted_body(self) -> bytes:
        decrypted_body: bytes
        if not self.container.posting_license.encrypted:
            source: PostingLicenseUnencrypted = deepcopy(self.container.
                posting_license)
            self.secret_key = source.get_content_key()
        else:
            if not self.secret_key:
                raise FileHandlerMissingKeyException()
        decrypted_body = deepcopy(self.container.get_decrypted_body(self.
            secret_key))
        return decrypted_body

    def get_supported_command_text(self) -> str:
        return deepcopy(", ".join(self.supported_commands))

    def _generate_temp_decrypted_file(self) -> str:
        temp_file_name: str = secrets.token_urlsafe(32) + "." + self.
            container.full_original_file_path.split(".")[-1]
        temp_directory: str = tempfile.gettempdir()
        full_temp_path: str = os.path.abspath(os.path.join(temp_directory
            , temp_file_name))
        content: bytes = self._get_decrypted_body()
        with open(full_temp_path, "wb") as opened_file:
            opened_file.write(content)
        return full_temp_path

    @staticmethod
    def _remove_temp_file(temp_file: str) -> None:
        while os.path.isfile(temp_file):
            try:
                os.remove(temp_file)
            except PermissionError:
                print("\n<you must close all programs and processes using
                     the file before continuing>")
                getpass.getpass("<press enter when you closed all
                    processes and programs using the file>")

    def _set_new_body_content(self, new_content: bytes) -> None:
        self.container.body_is_encrypted = False
        self.container.body = deepcopy(new_content)

    def _secure_new_body_content(self, new_body: bytes) -> ContainerFile:
        status_4_plus: bool = deepcopy(self.container.posting_license.
            encrypted)
        assert self.container.posting_license
        if not status_4_plus:  # status 3
            assert isinstance(self.container.posting_license,
                PostingLicenseUnencrypted)
            self.secret_key = self.container.posting_license.content_key
        self._set_new_body_content_protected(new_body, self.secret_key)
            # for status 4, the key is already in secret_key
        return self.get_container()  # returns container of status 3 or 4

    def get_container(self) -> ContainerFile:
        return deepcopy(self.container)

    def _set_new_body_content_protected(self, new_content: bytes,
        existing_body_key: str) -> None:
        self._set_new_body_content(new_content)
        self.container.encrypt_body(existing_body_key)
```

A.8 file_handler_factory.py

```
# typing imports
from typing import Tuple
# standard library imports
# external imports
# project level imports
from file_handler import *
from file_handler_text import FileHandlerText
from file_handler_generic import FileHandlerGeneric


class FileHandlerFactory:

    def create(self, container: ContainerFile, permissions: PermissionObject, key: str = "")\
            -> Tuple[FileHandler, bool]:
        result: FileHandler
        success: bool
        if self.__check_if_content_utf_compatible(container, permissions, key):
            result, success = FileHandlerText(container, permissions), True
        else:
            result, success = FileHandlerGeneric(container, permissions), False
        if key:
            result.secret_key = key
        return result, success

    @staticmethod
    def __extract_file_extension(container: ContainerFile) -> str:
        extension: str = container.full_original_file_path.strip().split(".")[-1].lower()
        return extension

    @staticmethod
    def __check_if_content_utf_compatible(container: ContainerFile, permissions: PermissionObject, key: str) -> bool:
        try:
            temporary_generic: FileHandlerGeneric = FileHandlerGeneric(container, permissions, key)
            if container.body_is_encrypted:
                if not container.posting_license.encrypted:  # not status 4 --> status 3
                    temporary_generic.secret_key = container.posting_license.content_key # noqa
            decrypted_body: bytes = temporary_generic._get_decrypted_body() # noqa
            decrypted_body.decode("utf-8")
            return True
        except (UnicodeDecodeError, FileHandlerMissingKeyException):
            return False
```

A.9 file_handler_generic.py

```
# typing imports
from typing import Optional
# standard library imports
# external imports
import startfile as os_independent  #universal-startfile # noqa
# project level imports
from file_handler import *


class FileHandlerGeneric(FileHandler):

    def __init__(self, container: ContainerFile, permissions: PermissionObject, json_key: Optional[str] = ""):
        super().__init__(container, permissions, json_key)
        self.supported_commands.append("view")
        self.supported_commands.append("edit")

    def view(self) -> None:
        try:
            assert self.permissions.owner or self.permissions.view
        except AssertionError:
            raise FileHandlerPermissionException()
        temp_file: str = self._generate_temp_decrypted_file()
        os.chmod(temp_file, 0o777)  # fixes file permission issue on linux
        os_independent.startfile(temp_file)
        getpass.getpass("<press enter when you finished viewing the file>")
        FileHandlerGeneric._remove_temp_file(temp_file)

    def edit(self) -> ContainerFile:  # receives containers of status 3 and 4
        try:
            assert self.permissions.owner or self.permissions.modify
        except AssertionError:
            raise FileHandlerPermissionException()

        temp_file: str = self._generate_temp_decrypted_file()
        os.chmod(temp_file, 0o777)  # fixes file permission issue on linux
        os_independent.startfile(temp_file)
        getpass.getpass("<press enter when you saved the changes to the file>")
        new_body: bytes
        with open(temp_file, "rb") as opened_file:
            new_body = opened_file.read()
        FileHandlerGeneric._remove_temp_file(temp_file)
        return self._secure_new_body_content(new_body)
```

A.10 file_handler_text.py

```python
# typing imports
# standard library imports
# external imports
# project level imports
from file_handler import *
from extended_code_library.text_editor import TextEditor


class FileHandlerText(FileHandler):

    def __init__(self, container: ContainerFile, permissions:
        PermissionObject, json_key: str = ""):
        super().__init__(container, permissions, json_key)
        self.supported_commands.append("view")
        self.supported_commands.append("edit")

    def __get_content(self) -> str:
        content: str
        try:
            content = self._get_decrypted_body().decode("utf-8")
        except UnicodeDecodeError:
            content = str(self._get_decrypted_body())
        return content

    def view(self) -> None:
        try:
            assert self.permissions.owner or self.permissions.view
        except AssertionError:
            raise FileHandlerPermissionException()
        content: str = self.__get_content()
        print(content)

    def edit(self) -> ContainerFile:
        try:
            assert self.permissions.owner or self.permissions.modify
        except AssertionError:
            raise FileHandlerPermissionException()
        content: str = self.__get_content()
        content = TextEditor.edit_text(content)
        return self._secure_new_body_content(content.encode("utf-8"))
```

A.11 package_certificates.py

```python
# typing imports
```

```python
from typing import Callable, Dict, Optional
# standard library imports
# external imports
# project level imports
from package_dataconversion import deepcopy
from package_cryptography import CryptographyException, CryptoSystemRSA


class CertificateException(CryptographyException):
    pass


def certificate_exception(function: Callable) -> Callable:
    def wrapper(*args, **kwargs):
        try:
            return function(*args, **kwargs)
        except:  # noqa
            raise CryptographyException(f"ERROR IN CERTIFICATE PACKAGE!
                ({function.__name__})")
    return wrapper


class HalfCertificate:
    __csys: CryptoSystemRSA

    @certificate_exception
    def __init__(self, public_key: Optional[Dict] = None) -> None:
        self.__csys = CryptoSystemRSA()
        if public_key:
            self.import_public_key(public_key)

    @certificate_exception
    def encrypt(self, message: bytes) -> bytes:
        return self.__csys.encrypt(message)

    @certificate_exception
    def verify(self, message: bytes, signature: bytes) -> str:
        return self.__csys.verify(message, signature)

    @certificate_exception
    def import_public_key(self, key: Dict) -> None:
        self.__csys.import_public_key(key)


class FullCertificate:
    __csys: CryptoSystemRSA

    def __init__(self, key_size: int = 0):
        self.__csys = CryptoSystemRSA()
        if key_size > 0:
            self.__csys.set_key_size(key_size)

    @certificate_exception
    def set_key_size(self, size: int) -> None:
        return self.__csys.set_key_size(size)

    @deepcopy
    def get_msg_max_size(self) -> int:
        return self.__csys.get_msg_max_size()

    @certificate_exception
    def generate_keys(self) -> None:
        self.__csys.set_threads_maximum()
        return self.__csys.generate_keys()

    @certificate_exception
    def encrypt(self, message: bytes) -> bytes:
        return self.__csys.encrypt(message)

    @certificate_exception
    def decrypt(self, ciphertext: bytes) -> bytes:
        return self.__csys.decrypt(ciphertext)

    @certificate_exception
    def sign(self, message: bytes) -> bytes:
        return self.__csys.sign(message, "SHA-256")

    @certificate_exception
    def verify(self, message: bytes, signature: bytes) -> bool:
        return self.__csys.verify(message, signature) == "SHA-256"

    @certificate_exception
    @deepcopy
    def export_public_key(self) -> Dict:
        return self.__csys.export_public_key()

    @certificate_exception
    def import_public_key(self, key: Dict) -> None:
        self.__csys.import_public_key(key)
```

A.12 package_cryptography.py

```
# typing imports
# standard library imports
import secrets
import io
# external imports
import rsa  # rsa
import pyAesCrypt  # pyAesCrypt # type: ignore
from Crypto.Cipher import AES as AES4GCM  # pycryptodome # noqa
from psutil import cpu_count  # psutil
# project level imports
from package_dataconversion import *


class CryptographyException(Exception):
    pass


def cryptography_exception(function: Callable) -> Callable:
    def wrapper(*args, **kwargs):
        try:
            return function(*args, **kwargs)
        except:  # noqa
            raise CryptographyException(f"ERROR IN CRYPTOGRAPHIC PACKAGE!
                 ({function.__name__})")
    return wrapper


class CryptoSystem:
    def encrypt(self, data: bytes) -> bytes:
        raise NotImplementedError()

    def decrypt(self, data: bytes) -> bytes:
        raise NotImplementedError()

    def export_full_private_key(self) -> Dict:
        raise NotImplementedError()

    def import_full_private_key(self, key: Dict) -> None:
        raise NotImplementedError()


class CryptoSystemAES(CryptoSystem):
    __key: bytes = b""
    buffer_size: int = 64 * 1024

    @cryptography_exception
    def generate_random_key(self) -> None:
        assert not self.__key
        self.__key = secrets.token_bytes(256)

    @deepcopy
    @cryptography_exception
    def export_full_private_key(self) -> Dict:
        return {"key": con_bytes_utf(con_bytes_b64(self.__key))}

    @cryptography_exception
    def import_full_private_key(self, key: Dict) -> None:
        self.__key = con_b64_bytes(con_utf_bytes(key["key"]))

    @deepcopy
    @cryptography_exception
    def encrypt(self, data: bytes) -> bytes:
        assert self.__key
        pbs = deepcopy(io.BytesIO(data))  # plaintext binary stream
        cbs = deepcopy(io.BytesIO())  # ciphertext binary stream
        pyAesCrypt.encryptStream(pbs, cbs, str(self.__key), self.
            buffer_size)
        return cbs.getvalue()

    @deepcopy
    @cryptography_exception
    def decrypt(self, data: bytes) -> bytes:
        assert self.__key
        cbs = deepcopy(io.BytesIO(data))  # ciphertext binary stream
        dbs = deepcopy(io.BytesIO())  # decrypted binary stream
        pyAesCrypt.decryptStream(cbs, dbs, str(self.__key), self.
            buffer_size, len(data))
        return dbs.getvalue()


class CryptoSystemRSA(CryptoSystem):
    __key_size: int
    threads: int
    __msg_max_size: int
    __pubkey: rsa.key.PublicKey
    __privkey: rsa.key.PrivateKey

    def __init__(self, key_size: int = -1):
        self.__key_size = key_size
        self.threads = 1
        self.__msg_max_size = -1

    @cryptography_exception
    def set_key_size(self, size: int) -> None:
        assert size in [128, 256, 384, 512, 1024, 2048, 3072, 4096]
        self.__key_size = size

    def update_msg_max_size(self) -> None:
        self.__msg_max_size = int(self.__key_size / 8 - 11)  # always 11
            byte overhead with this RSA implementation

    @deepcopy
    def get_msg_max_size(self) -> int:
        return self.__msg_max_size

    @cryptography_exception
    def generate_keys(self) -> None:
        (pubkey, privkey) = rsa.newkeys(self.__key_size, poolsize=self.
            threads, accurate=True)
        self.__pubkey = pubkey
        self.__privkey = privkey
        self.update_msg_max_size()

    @cryptography_exception
    def encrypt(self, data: bytes) -> bytes:
        assert self.__pubkey
        return rsa.encrypt(data, self.__pubkey)

    @cryptography_exception
    def decrypt(self, data: bytes) -> bytes:
        assert self.__privkey
        return rsa.decrypt(data, self.__privkey)

    @cryptography_exception
    def sign(self, message: bytes, hash_algorithm: str = "SHA-256") ->
        bytes:
        assert self.__privkey
        assert self.__msg_max_size > 51
        return rsa.sign(message, self.__privkey, hash_method=
            hash_algorithm)

    @cryptography_exception
    def verify(self, message: bytes, signature: bytes) -> str:
        assert self.__pubkey
        try:
            return rsa.verify(message, signature, self.__pubkey)
        except rsa.VerificationError:
            return ""

    @cryptography_exception
    @deepcopy
    def export_public_key(self) -> Dict:
        return {"n": self.__pubkey.n, "e": self.__pubkey.e}

    @cryptography_exception
    def import_public_key(self, key: Dict) -> None:
        n = key["n"]
        e = key["e"]
        self.__pubkey = rsa.PublicKey(n, e)

    @cryptography_exception
    def set_threads(self, threads: int) -> bool:
        if not (1 <= threads <= cpu_count()):
            return False
        self.threads = threads
        return True

    @cryptography_exception
    def set_threads_maximum(self) -> bool:
        return self.set_threads(cpu_count())

    @cryptography_exception
    def set_threads_minimum(self) -> bool:
        return self.set_threads(1)

    @deepcopy
    @cryptography_exception
    def export_full_private_key(self) -> Dict:
        return {"n": self.__privkey.n,
                "e": self.__privkey.e,
                "d": self.__privkey.d,
                "p": self.__privkey.p,
                "q": self.__privkey.q}

    @cryptography_exception
    def import_full_private_key(self, key: Dict) -> None:
        key_n: int = key["n"]
        key_e: int = key["e"]
        key_d: int = key["d"]
        key_p: int = key["p"]
        key_q: int = key["q"]
        self.__privkey = rsa.PrivateKey(key_n, key_e, key_d, key_p, key_q
            )


class CryptoSystemAESGCM(CryptoSystem):
    __key: bytes = b""
    __header: bytes = b""

    def set_header(self, header_text: bytes) -> None:
        self.__header = header_text

    @cryptography_exception
    def generate_random_key(self) -> None:
        assert not self.__key
        self.__key = secrets.token_bytes(32)

    @deepcopy
    @cryptography_exception
    def encrypt(self, data: bytes) -> bytes:
        cipher = AES4GCM.new(self.__key, AES4GCM.MODE_GCM)
        cipher.update(self.__header)
        new_ciphertext, tag = cipher.encrypt_and_digest(data)
        outbound: Callable = lambda x: con_bytes_utf(con_bytes_b64(x))
        cipher_data: Dict = {
            "nonce": outbound(cipher.nonce),
            "header": outbound(self.__header),
            "ciphertext": outbound(new_ciphertext),
            "tag": outbound(tag)
        }
        return con_utf_bytes(con_dict_json(cipher_data))

    @deepcopy
    @cryptography_exception
    def decrypt(self, data: bytes) -> bytes:
        cipher_data: Dict = con_json_dict(con_bytes_utf(data))
        fin: Callable = lambda x: con_b64_bytes(con_utf_bytes(x))
        old_ciphertext = fin(cipher_data["ciphertext"])
        nonce = fin(cipher_data["nonce"])
        header = fin(cipher_data["header"])
        tag = fin(cipher_data["tag"])
        cipher = AES4GCM.new(self.__key, AES4GCM.MODE_GCM, nonce=nonce)
        cipher.update(header)
        plaintext = cipher.decrypt_and_verify(old_ciphertext, tag)
        return plaintext

    @deepcopy
    @cryptography_exception
    def export_full_private_key(self) -> Dict:
        return {"key": con_bytes_utf(con_bytes_b64(self.__key))}

    @cryptography_exception
    def import_full_private_key(self, key: Dict) -> None:
        self.__key = con_b64_bytes(con_utf_bytes(key["key"]))
```

A.13 package_dataconversion.py

```
# typing imports
from typing import Any, Dict, Callable
# standard library imports
import copy
import json
import base64
# external imports
# project level imports


class ConversionException(Exception):
    pass


def conversion_exception(function: Callable) -> Callable:
    def wrapper(*args, **kwargs):
        try:
            return function(*args, **kwargs)
        except:  # noqa
            raise ConversionException(f"ERROR IN DATA CONVERSION PACKAGE!
                 ({function.__name__})")
    return wrapper


def deepcopy(thing: Any) -> Any:
    if callable(thing):
        def wrapper(*args, **kwargs):
            x: Any = thing(*args, **kwargs)
            return copy.deepcopy(x)

        return wrapper
    else:
        return copy.deepcopy(thing)


@deepcopy
@conversion_exception
def con_bytes_hex(data: bytes) -> str:
    return data.hex()


@deepcopy
@conversion_exception
def con_hex_bytes(data: str) -> bytes:
    return bytes.fromhex(data)


@deepcopy
@conversion_exception
def con_dict_json(data: Dict) -> str:
    return json.dumps(data)


@deepcopy
@conversion_exception
def con_json_dict(data: str) -> Dict:
    return json.loads(data)


@deepcopy
@conversion_exception
def con_bytes_b64(data: bytes) -> bytes:
    return base64.urlsafe_b64encode(data)


@deepcopy
@conversion_exception
```

```
def con_b64_bytes(data: bytes) -> bytes:
    return base64.urlsafe_b64decode(data)


@deepcopy
@conversion_exception
def con_bytes_utf(data: bytes) -> str:
    return data.decode("utf-8")


@deepcopy
@conversion_exception
def con_utf_bytes(data: str) -> bytes:
    return data.encode("utf-8")


@deepcopy
@conversion_exception
def con_b64_hex(data: bytes) -> str:
    return con_bytes_hex(con_b64_bytes(data))


@deepcopy
@conversion_exception
def con_hex_b64(data: str) -> bytes:
    return con_bytes_b64(con_hex_bytes(data))
```

A.14 package_debugging.py

```
# typing imports
from typing import Callable, Any
# standard library imports
from time import perf_counter
# external imports
# project level imports


def printing_timer(function: Callable) -> Callable:
    def wrapper(*args, **kwargs) -> Callable:
        start: float = perf_counter()
        x: Any = function(*args, **kwargs)
        end: float = perf_counter()
        print(f"{str(function)[1:-1]} terminated in {end - start} seconds
            ...")
        return x
    return wrapper
```

A.15 package_licenses.py

```python
# typing imports
from typing import List, Optional
# standard library imports
# external imports
# project level imports
from package_dataconversion import *


class LicenseException(Exception):
    pass


def license_exception(function: Callable) -> Callable:
    def wrapper(*args, **kwargs):
        try:
            return function(*args, **kwargs)
        except:  # noqa
            raise LicenseException(f"ERROR IN LICENSE! ({function.
                __name__})")

    return wrapper


class PermissionObject:
    version: int
    keys: List[str]
    # Permission Flags
    owner: bool  # all permissions below (overwriting)
    view: bool  # can view document
    decrypt: bool  # can restore unencrypted version
    modify: bool  # can change file content (without changing the PL)
    repost: bool  # can issue new posting license and send to server

    def __init__(self, permission_object_string: Optional[str] = ""):
        self.version = 1
        self.keys = ["owner", "view", "decrypt", "modify", "repost"]
        if not permission_object_string:
            permission_object_string = "{}"
        data: Dict = json.loads(permission_object_string)
        self.version = data.get("version", self.version)
        self.owner = data.get("owner", False)
        self.view = data.get("view", False)
        self.decrypt = data.get("decrypt", False)
        self.modify = data.get("modify", False)
        self.repost = data.get("repost", False)

    def __repr__(self):
        return con_dict_json({
            "version": self.version,
            "owner": self.owner,
            "view": self.view,
            "decrypt": self.decrypt,
            "modify": self.modify,
            "repost": self.repost
        })

    def anything(self) -> True:
        rights: List[bool] = [self.owner, self.view, self.decrypt, self.
           modify, self.repost]
        return bool(max(rights))

    def set(self, key: str, value: bool) -> None:
        assert key in self.keys
        assert isinstance(value, bool)
        setattr(self, key, value)

    def set_all(self, value: bool = True) -> None:
        for key in self.keys:
            self.set(key, value)

    def get(self, key: str) -> bool:
        assert key in self.keys
        return deepcopy(getattr(self, key))

    @deepcopy
    def json_version(self) -> str:
        return str(self)


class PermissionMerger:
    @staticmethod
    @deepcopy
    def merge(permissions: List[PermissionObject]) -> PermissionObject:
        base_object: PermissionObject = PermissionObject()
        for additional_permission in permissions:
            base_object.owner = base_object.owner or
                additional_permission.owner
            base_object.view = base_object.view or additional_permission.
                view
            base_object.decrypt = base_object.decrypt or
                additional_permission.decrypt
            base_object.modify = base_object.modify or
                additional_permission.modify
            base_object.repost = base_object.repost or
                additional_permission.repost
        return base_object


```

```
class RightsObject:
    subject: str
    permissions: PermissionObject

    def __init__(self, rights_object_string: Optional[str] = ""):
        if not rights_object_string:
            rights_object_string = "{}"
        data: Dict = json.loads(rights_object_string)
        self.version = data.get("version", 1)
        self.subject = data.get("subject", "")
        self.permissions = PermissionObject(data.get("permissions", ""))

    def __repr__(self) -> str:
        return con_dict_json({
            "version": self.version,
            "subject": self.subject,
            "permissions": self.permissions.json_version()
        })

    def generate_owner(self, owner: str) -> None:
        assert owner
        self.subject = owner
        owner_permissions: PermissionObject = PermissionObject()
        owner_permissions.owner = True
        self.permissions = owner_permissions

    @deepcopy
    def json_version(self) -> str:
        return str(self)


class License:
    encrypted: bool

    def json_version(self) -> str:
        raise NotImplementedError
        return ""  # noqa


class PostingLicense(License):
    pass


class PostingLicenseEncrypted(PostingLicense):
    __encrypted: bool
    __data: bytes
    __encrypted_key: bytes
    __signature: bytes

    def __init__(self, encrypted_license: str, encrypted_key: str,
        signature: str):
        assert encrypted_license and encrypted_key and signature
        self.encrypted = True
        self.__data = con_hex_bytes(encrypted_license)
        self.__encrypted_key = con_hex_bytes(encrypted_key)
        self.__signature = con_hex_bytes(signature)

    @deepcopy
    def json_version(self) -> str:
        assert self.encrypted and self.__data and self.__encrypted_key
            and self.__signature
        return con_dict_json({
            "data": con_bytes_utf(con_bytes_b64(self.__data)),
            "key": con_bytes_utf(con_bytes_b64(self.__encrypted_key)),
            "signature": con_bytes_utf(con_bytes_b64(self.__signature))
        })


class PostingLicenseUnencrypted(PostingLicense):
    encrypted = False
    version: int = 1
    content_key: str = ""
    rights: List[RightsObject] = []

    def __init__(self, posting_license_string: Optional[str] = ""):
        self.content_key = ""
        self.rights = []
        if posting_license_string:
            data: Dict = json.loads(posting_license_string)
            self.version = data["version"]
            self.content_key = data["content_key"]
            rights_object_string_list: List[str] = json.loads(data["
                rights"])
            self.rights = [RightsObject(x) for x in
                rights_object_string_list]

    def __repr__(self) -> str:
        return con_dict_json({
            "version": self.version,
            "content_key": self.content_key if self.content_key else "",
            "rights": json.dumps([x.json_version() for x in self.rights])
                 if self.rights else "[]"
        })

    def add_owner(self, owner: str) -> None:
        assert owner
        assert len(self.rights) < 1
        owner_rights_object = RightsObject()
        owner_rights_object.generate_owner(owner)
```

```python
        self.rights.append(deepcopy(owner_rights_object))

    def get_content_key(self) -> str:
        return deepcopy(self.content_key)

    @deepcopy
    def json_version(self) -> str:
        assert self.content_key and self.rights
        return str(self)

    @deepcopy
    def get_subjects(self) -> List[str]:
        return [x.subject for x in self.rights]
```

A.16 server_app.py

```python
# typing imports
from typing import Tuple, Union
# standard library imports
import os
import pickle
import traceback
import secrets
import datetime as dt
# external imports
import jwt  # PyJWT # noqa
from flask import Flask, request, abort  # flask # noqa
# project level imports
from server_database import DatabaseHandler, ServerDatabaseException
from server_certificates import ServerCertificate, ClientCertificateCopy
from package_cryptography import CryptoSystemAESGCM,
    CryptographyException
from package_licenses import *

# EXTERNAL
certificate_file: str = "server_certificate.bin"
db_file: str = "server_database.sqlite"
port: int = 11235
jwt_lifetime: float = 0.5
# INTERNAL
certificate: ServerCertificate
db = DatabaseHandler
app: Flask = Flask(__name__)
app.config['SECRET_KEY'] = secrets.token_urlsafe(512)
next_endpoint_counter: int = 0


def generate_endpoint() -> str:
    global next_endpoint_counter
    next_endpoint_counter += 1
    return "ep" + str(next_endpoint_counter)


def generate_token(data: Dict, minutes: float = 30) -> str:
    assert data.get("exp", "") == ""
    data["exp"] = (dt.datetime.utcnow() + dt.timedelta(minutes=minutes))
    return jwt.encode(data, app.config['SECRET_KEY'], algorithm='HS256')


def needs_authentication(function: Callable):
    def wrapper(*args, **kwargs):
        if "Authorization" not in request.headers:
            abort(401)  # "authorization required"
        user_id_from_jwt: str = ""  # assignment for pep8 compliance
        try:
            token = str(request.headers["Authorization"]).split(" ")[1]
            data = jwt.decode(token, app.config['SECRET_KEY'], algorithms
                =['HS256'])
            user_name_from_jwt = data['username']
            user_id_from_jwt = data['id']
            assert db.id_is_user(user_id_from_jwt)
            assert db.retrieve_name_from_id(user_id_from_jwt) ==
                user_name_from_jwt
        except Exception as e:
            # print("-->", e)
            traceback_text: str = traceback.format_exc().strip()
            if not isinstance(e, jwt.exceptions.ExpiredSignatureError):
                print(traceback_text)
            abort(403)  # "forbidden"
        try:
            return function(user_id_from_jwt, *args, **kwargs)
        except Exception as e:
            print("-->", e)
            traceback_text: str = traceback.format_exc().strip()  # noqa
            print(traceback_text)
            abort(500)
    return wrapper


def encrypt_data_randomly_symmetrically(data: bytes) -> Tuple[str, str]:
    # noqa
    crypto_system: CryptoSystemAESGCM = CryptoSystemAESGCM()
    crypto_system.generate_random_key()
    ciphertext: str = con_bytes_hex(crypto_system.encrypt(data))
    plain_symmetric_key: str = con_dict_json(crypto_system.
        export_full_private_key())
    return ciphertext, plain_symmetric_key

```

```python

def decrypt_data_symmetrically(data: str, key: str) -> bytes:  # noqa
    crypto_system: CryptoSystemAESGCM = CryptoSystemAESGCM()
    crypto_system.import_full_private_key(con_json_dict(key))
    restored_plaintext = crypto_system.decrypt(con_hex_bytes(data))
    return restored_plaintext


def respond(message: str, data: str = "", key: str = "") -> str:
    return json.dumps({"msg": message, "data": data, "key": key})


def double_decryption(double_encrypted_data: str, encrypted_key: str) ->
    bytes:
    assert double_encrypted_data and encrypted_key
    decrypted_key: bytes = certificate.decrypt(con_hex_bytes(
        encrypted_key))
    restored_aes_key: Dict = con_json_dict(con_bytes_utf(decrypted_key))
    return decrypt_data_symmetrically(double_encrypted_data,
        con_dict_json(restored_aes_key))


def double_encryption(plain_data: bytes, user_certificate_text: str) ->
    Tuple[str, str]:
    assert plain_data and user_certificate_text
    re_encrypted_data, plain_aes_key =
        encrypt_data_randomly_symmetrically(plain_data)
    temporary_user_certificate: ClientCertificateCopy =
        ClientCertificateCopy(con_json_dict(user_certificate_text))
    re_encrypted_aes_key: str = con_bytes_hex(temporary_user_certificate.
        encrypt(con_utf_bytes(plain_aes_key)))
    return re_encrypted_data, re_encrypted_aes_key


def verify_data_with_any_specific_user_signature(user_id: str, data:
    bytes, signature: bytes) -> bool:
    user_certificate_strings: List = db.retrieve_user_certificates(
        user_id)
    if not len(user_certificate_strings) > 0:
        return False
    signature_verified: bool = False
    for certificate_option in user_certificate_strings:
        current_certificate: ClientCertificateCopy =
            ClientCertificateCopy(con_json_dict(certificate_option[0]))
        try:
            if current_certificate.verify(data, signature):
                signature_verified = True
                break
        except CryptographyException as e:
            print(e)
            pass
    return signature_verified


def verify_data_with_any_user_signature(data: bytes, signature: bytes) -> bool:
    users: Dict = db.retrieve_all_users()
    for username in users.keys():
        userid = users[username]
        if verify_data_with_any_specific_user_signature(userid, data, signature):
            return True
    return False


# ###################################################################################

@app.get("/", endpoint=generate_endpoint())
def hello_world() -> str:
    return "Hello World"


@app.post("/login")
def login() -> Union[str, Any]:
    auth = request.authorization
    if not auth or not auth.username or not auth.password:
        abort(400)  # "bad request"
    uid: str
    try:
        uid = db.retrieve_id_from_name(auth.username)
        assert db.id_is_user(uid)
        assert db.check_password(uid, auth.password)
        token: str = generate_token({"username": auth.username, "id": uid}, jwt_lifetime)
        return respond(uid, token)
    except (AssertionError, ServerDatabaseException):
        abort(401)  # "unauthorized"


@app.get("/certificate", endpoint=generate_endpoint())
@needs_authentication
def send_certificate(user_id: str) -> Union[str, Any]:  # noqa
    public_key_dict: str = json.dumps(certificate.export_public_key())
    return respond("certificate", public_key_dict), 200


@app.get("/profile", endpoint=generate_endpoint())
@needs_authentication
```

```
def send_userid(user_id: str) -> Union[str, Any]:
    return respond(f"You are authenticated as {user_id}", user_id), 200


@app.get("/groups", endpoint=generate_endpoint())
@needs_authentication
def send_groups(user_id: str) -> Union[str, Any]:
    device_id: str = json.loads(request.json)['deviceid']
    all_groups: Dict = dict(db.retrieve_all_groups())
    payload: bytes = con_utf_bytes(con_dict_json(all_groups))
    key_text: str = db.retrieve_user_device_certificate(user_id,
        device_id)
    assert key_text
    group_information, decryption_key = double_encryption(payload,
        key_text)
    return respond(f"group information available", group_information,
        decryption_key), 200


@app.get("/users", endpoint=generate_endpoint())
@needs_authentication
def send_users(user_id: str) -> Union[str, Any]:
    device_id: str = json.loads(request.json)['deviceid']
    all_users: Dict = dict(db.retrieve_all_users())
    payload: bytes = con_utf_bytes(con_dict_json(all_users))
    key_text: str = db.retrieve_user_device_certificate(user_id,
        device_id)
    assert key_text
    user_information, decryption_key = double_encryption(payload,
        key_text)
    return respond(f"user information available", user_information,
        decryption_key), 200


@app.post("/newdevice", endpoint=generate_endpoint())
@needs_authentication
def save_user_certificate(user_id: str) -> Union[str, Any]:  # noqa
    encrypted_payload: bytes = con_hex_bytes(json.loads(request.json)['
        payload'])
    encrypted_meta_key: bytes = con_hex_bytes(json.loads(request.json)['
        key'])
    decrypted_meta_key: Dict = con_json_dict(con_bytes_utf(certificate.
        decrypt(encrypted_meta_key)))
    aes_system: CryptoSystemAESGCM = CryptoSystemAESGCM()
    aes_system.import_full_private_key(decrypted_meta_key)
    decrypted_payload: Dict = con_json_dict(con_bytes_utf(aes_system.
        decrypt(encrypted_payload)))
    user_id = decrypted_payload["userid"]
    user_name = decrypted_payload["username"]
    device_id = decrypted_payload["deviceid"]
    assert user_id and user_name and device_id
    assert user_name == db.retrieve_name_from_id(user_id)
    public_key: Dict = json.loads(decrypted_payload["publickey"])
    assert public_key
    db_formatted_client_pubkey: str = con_dict_json(public_key)
    db.enter_user_certificate(user_id, device_id, db_formatted_client_pubkey)
    return respond("certificate registered"), 201


@app.post("/testcertificates", endpoint=generate_endpoint())
@needs_authentication
def reencrypt_dummy_data(user_id: str) -> Union[str, Any]:
    device_id: str = json.loads(request.json)['deviceid']
    double_encrypted_challenge: str = json.loads(request.json)['challenge']
    encrypted_key: str = json.loads(request.json)['key']
    plain_challenge: bytes = double_decryption(double_encrypted_challenge, encrypted_key)
    key_text: str = db.retrieve_user_device_certificate(user_id, device_id)
    assert key_text
    re_encrypted_challenge: str
    new_symmetric_key: str
    re_encrypted_challenge, new_symmetric_key = double_encryption(plain_challenge, key_text)
    return respond("challenge accepted", re_encrypted_challenge, new_symmetric_key), 202


@app.put("/evaluate", endpoint=generate_endpoint())
@needs_authentication
def evaluate_publishing_license(user_id: str) -> Union[str, Any]:
    pl_data_encrypted: str = con_b64_hex(request.json['data'])
    pl_key_encrypted: str = con_b64_hex(request.json['key'])
    pl_signature: bytes = con_b64_bytes(request.json['signature'])
    device_id: str = con_bytes_utf(con_b64_bytes(request.json['deviceid']))
    assert pl_data_encrypted and pl_key_encrypted and pl_signature and device_id
    pl_data_unencrypted: bytes = double_decryption(pl_data_encrypted, pl_key_encrypted)
    assert verify_data_with_any_user_signature(pl_data_unencrypted, pl_signature)
    posting_license: PostingLicenseUnencrypted = PostingLicenseUnencrypted(con_bytes_utf(pl_data_unencrypted))
    file_body_key: str = deepcopy(posting_license.content_key)
    users_identities: List[str] = db.retrieve_all_identities(user_id)
    permitted_rights: List[PermissionObject] = []
    for rights_object in posting_license.rights:
        if rights_object.subject in users_identities:
            permitted_rights.append(deepcopy(rights_object.permissions))
    concrete_permissions: PermissionObject = PermissionMerger.merge(
        permitted_rights)
    response_certificate_key_text: str = db.
        retrieve_user_device_certificate(user_id, device_id)
    assert response_certificate_key_text
    payload: Dict = {"permissions": concrete_permissions.json_version()}
    if concrete_permissions.anything:
        payload["bodykey"] = file_body_key
    if concrete_permissions.owner or concrete_permissions.repost:
        payload["fulllicense"] = posting_license.json_version()
    unencrypted_byte_payload: bytes = con_utf_bytes(con_dict_json(payload
        ))
    ciphertext, new_aes_key = double_encryption(unencrypted_byte_payload,
         response_certificate_key_text)
    return respond("okay", ciphertext, new_aes_key), 200


if __name__ == "__main__":
    db = DatabaseHandler(db_file)
    if os.path.isfile(certificate_file):
        with open(certificate_file, "rb") as opened_file:
            certificate = pickle.load(opened_file)
    else:
        certificate = ServerCertificate()
        with open(certificate_file, "wb") as opened_file:
            pickle.dump(certificate, opened_file)
    exit(app.run(port=port))
```

A.17 server_certificates.py

```
# typing imports
# standard library imports
# external imports
# project level imports
from package_certificates import FullCertificate, HalfCertificate


class ServerCertificate(FullCertificate):
    def __init__(self):
        super().__init__(4096)
        self.generate_keys()


class ClientCertificateCopy(HalfCertificate):
    pass
```

A.18 server_database.py

```
# typing imports
from typing import Callable, List, Tuple, Dict
# standard library imports
import os
import sqlite3
import secrets
# external imports
import argon2.exceptions  # argon2-cffi
from argon2 import PasswordHasher  # argon2-cffi # successor of bcrypt,
    less complicated than scrypt # noqa
# project level imports
from package_dataconversion import deepcopy


class ServerDatabaseException(Exception):
    pass


def server_database_exception(function: Callable) -> Callable:
    def wrapper(*args, **kwargs):
        try:
            return function(*args, **kwargs)
        except:  # noqa
            raise ServerDatabaseException(f"ERROR IN SERVER DATABASE! ({
                function.__name__})")

    return wrapper


class DBCursor:
    database_file: str
    connection: sqlite3.Connection
    cursor: sqlite3.Cursor

    def __init__(self, file_name):
        self.database_file = file_name

    def __enter__(self):
        self.connection = sqlite3.connect(self.database_file)
        self.cursor = self.connection.cursor()
        return self.cursor

    def __exit__(self, exit_type, exit_value, traceback):
        self.connection.commit()
        self.connection.close()


class DatabaseHandler:
```

```python
    database_file: str

    def __init__(self, database_file: str) -> None:
        self.database_file = os.path.abspath(database_file)
        if not os.path.isfile(database_file):
            self.create_new()

    def create_new(self) -> None:  # defines database standard configuration
        self.reset()
        self.generate_standard_tables()
        group_id_01: str = self.enter_group("01_external")  # noqa
        group_id_02: str = self.enter_group("02_internal")  # noqa
        group_id_03: str = self.enter_group("03_nda")  # noqa
        group_id_04: str = self.enter_group("04_scl1")  # noqa
        group_id_05: str = self.enter_group("05_scl2")  # noqa
        user_id_admin: str = self.enter_user("admin", "admin")
        self.enter_group_membership(group_id_05, user_id_admin)
        user_id_jondoe = self.enter_user("jondoe", "password")
        self.enter_group_membership(group_id_01, user_id_jondoe)

    @server_database_exception
    def reset(self) -> None:
        try:
            os.remove(self.database_file)
        except FileNotFoundError:
            pass

    @server_database_exception
    def __generate_table(self, table_name: str, attribute_definitions: List) -> None:
        with DBCursor(self.database_file) as cursor:
            cursor.execute(f"DROP TABLE IF EXISTS {table_name}")
            table = f"CREATE TABLE {table_name} ({str(chr(10)).join(attribute_definitions)});"
            cursor.execute(table)

    @server_database_exception
    def generate_standard_tables(self) -> None:
        generate: Callable = lambda name, attribs: self.__generate_table(name, attribs.strip().split())
        generate("entities", """
                id TEXT PRIMARY KEY,
                name TEXT UNIQUE NOT NULL,
                isgroup BOOLEAN NOT NULL,
                pwhash TEXT
                """)
        generate("certificates", """
                userid TEXT NOT NULL,
                deviceid TEXT NOT NULL,
                data TEXT NOT NULL,
                created DATETIME DEFAULT CURRENT_TIMESTAMP,
                FOREIGN KEY(userid) REFERENCES entities(id),
                PRIMARY KEY (userid, deviceid)
                """)
        generate("memberships", """
                groupid TEXT NOT NULL,
                userid TEXT NOT NULL,
                FOREIGN KEY(groupid) REFERENCES entities(id),
                FOREIGN KEY(userid) REFERENCES entities(id),
                PRIMARY KEY (userid, groupid)
                """)
        return None

    @server_database_exception
    def hash_password(self, password: str) -> str:
        return str(PasswordHasher().hash(password))

    def __retrieve_password_hash(self, user_id: str) -> str:
        assert user_id
        result: List
        with DBCursor(self.database_file) as cursor:
            cursor.execute(f"SELECT pwhash FROM entities WHERE id=? AND
                isgroup=?;", (user_id, 0))
            result = cursor.fetchone()
        return result[0] if result is not None else ""

    def __update_password(self, user_id: str, password: str) -> None:
        assert user_id
        assert self.id_is_user(user_id)
        password_hash = str(self.hash_password(password))
        with DBCursor(self.database_file) as cursor:
            cursor.execute(f"UPDATE entities SET pwhash=? WHERE id=? AND
                isgroup=?;", (password_hash, user_id, 0))

    @server_database_exception
    def check_password(self, user_id: str, password: str) -> bool:
        if (not user_id) or (not self.id_is_user):
            return False
        pwh = PasswordHasher()
        try:
            stored_hash = self.__retrieve_password_hash(user_id)
            if pwh.verify(stored_hash, password):
                if pwh.check_needs_rehash(stored_hash):
                    self.__update_password(user_id, password)
                return True
        except argon2.exceptions.VerifyMismatchError:
            pass
        return False

    @server_database_exception
    def retrieve_id_from_name(self, name: str) -> str:
        assert name
        result: str
        with DBCursor(self.database_file) as cursor:
            cursor.execute(f"SELECT id FROM entities WHERE name=?", (name
                ,))
            result = cursor.fetchone()
        return result[0] if result is not None else ""

    @server_database_exception
    def retrieve_name_from_id(self, entity_id: str) -> str:
        assert entity_id
        result: str
        with DBCursor(self.database_file) as cursor:
            cursor.execute(f"SELECT name FROM entities WHERE id=?", (
                entity_id,))
            result = cursor.fetchone()
        return result[0] if result is not None else ""

    @server_database_exception
    def __get_unused_id(self) -> str:
        new_id: Callable = lambda: secrets.token_hex(8)
        random_id: str = new_id()
        while self.id_is_user(random_id) or self.id_is_group(random_id):
            random_id = new_id()
        return random_id

    @server_database_exception
    def enter_user(self, name: str, plain_password: str) -> str:
        name = name.lower().strip()
        assert name and plain_password
        password_hash = self.hash_password(plain_password)
        random_id = self.__get_unused_id()
        with DBCursor(self.database_file) as cursor:
            cursor.execute(f"INSERT INTO entities (id, name, pwhash,
                isgroup) VALUES (?, ?, ?, ?);",
                           (random_id, name, password_hash, 0))
        return self.retrieve_id_from_name(name)

    @server_database_exception
    def enter_group(self, name: str) -> str:
        name = name.lower().strip()
        assert name
        random_id = self.__get_unused_id()
        with DBCursor(self.database_file) as cursor:
            cursor.execute(f"INSERT INTO entities (id, name, isgroup)
                VALUES (?, ?, ?);", (random_id, name, 1))
        return self.retrieve_id_from_name(name)

```

```python
    @server_database_exception
    def id_is_user(self, identity: str) -> bool:
        if not identity:
            return False
        with DBCursor(self.database_file) as cursor:
            cursor.execute(f"SELECT id FROM entities WHERE id=? AND isgroup=0 LIMIT 1;", (identity,))
            result = cursor.fetchone()
        return False if result is None else True

    @server_database_exception
    def id_is_group(self, identity: str) -> bool:
        if not identity:
            return False
        with DBCursor(self.database_file) as cursor:
            cursor.execute(f"SELECT id FROM entities WHERE id=? AND isgroup=1 LIMIT 1;", (identity,))
            result = cursor.fetchone()
        return False if result is None else True

    @server_database_exception
    def enter_group_membership(self, group_id: str, user_id: str) -> None:
        assert user_id and group_id
        assert self.id_is_user(user_id)
        assert self.id_is_group(group_id)
        with DBCursor(self.database_file) as cursor:
            cursor.execute(f"INSERT INTO memberships (groupid, userid) VALUES (?, ?);", (group_id, user_id))

    @server_database_exception
    def is_group_member(self, user_id: str, group_id) -> bool:
        if not self.id_is_user(user_id):
            print("user is no user")
            return False
        with DBCursor(self.database_file) as cursor:
            cursor.execute(f"SELECT groupid FROM memberships WHERE groupid=? AND userid=? LIMIT 1;",
                           (group_id, user_id))
            result = cursor.fetchone()
        return False if result is None else True

    @server_database_exception
    def retrieve_all_identities(self, user_id: str) -> List[str]:
        assert user_id
        assert self.id_is_user(user_id)
        identities: List[str]
        with DBCursor(self.database_file) as cursor:
            cursor.execute(f"SELECT groupid FROM memberships WHERE userid=?;", (user_id,))
            meta_list: List[List[str]] = [[user_id]] + cursor.fetchall()
        identities = [str(identity[0]) for identity in meta_list]
        return identities

    @server_database_exception
    def enter_user_certificate(self, user_id: str, device_id, data: str) -> None:
        assert user_id and device_id and data
        assert self.id_is_user(user_id)
        with DBCursor(self.database_file) as cursor:
            cursor.execute(f"INSERT INTO certificates (userid, deviceid, data) VALUES (?, ?, ?);",
                           (user_id, device_id, data))

    @server_database_exception
    def retrieve_user_certificates(self, user_id: str) -> List[Tuple[str, str]]:
        assert user_id
        assert self.id_is_user(user_id)
        certificates: List[Tuple[str, str]]
        with DBCursor(self.database_file) as cursor:
            cursor.execute(f"SELECT data, deviceid FROM certificates WHERE userid=?;", (user_id,))
            certificates = [(str(certificate[0]), str(certificate[1])) for certificate in cursor.fetchall()]
        return deepcopy(certificates)

    @server_database_exception
    def retrieve_user_device_certificate(self, user_id: str, device_id: str) -> str:
        assert user_id and device_id
        assert self.id_is_user(user_id)
        certificate: str
        with DBCursor(self.database_file) as cursor:
            cursor.execute(f"SELECT data FROM certificates WHERE userid=? and deviceid=?;", (user_id, device_id))
            certificate = [certificate[0] for certificate in cursor.fetchall()][0]
        return certificate

    @server_database_exception
    def __retrieve_all_entity_type_members(self, isgroup: int) -> List[Tuple[str, str]]:
        assert isgroup == 0 or isgroup == 1
        entries: List[Tuple[str, str]]
        with DBCursor(self.database_file) as cursor:
            cursor.execute(f"SELECT name, id FROM entities WHERE isgroup=?;", (isgroup,))
            entries = [(str(entry[0]), str(entry[1])) for entry in cursor.fetchall()]
        return deepcopy(entries)

    @server_database_exception
    def retrieve_all_groups(self) -> Dict:
        return deepcopy(dict(self.__retrieve_all_entity_type_members(1)))

    @server_database_exception
    def retrieve_all_users(self) -> Dict:
        return deepcopy(dict(self.__retrieve_all_entity_type_members(0)))
```

A.19 text_editor.py

```
# CODE ROUGHLY BASED ON:    stackoverflow.com/a/53938684
# SNAPSHOT AVAILABLE AT:    web.archive.org/web/20230220083931/
    stackoverflow.com/a/53938684

# typing imports
# standard library imports
import tkinter
from tkinter.scrolledtext import ScrolledText
# external imports
# project level imports


class TextEditor:
    root_frame: tkinter.Tk
    text_frame: ScrolledText
    content: str

    def __init__(self, root_window: tkinter.Tk, initial_content: str):
        self.root_frame = root_window
        self.root_frame.title("CHANGES ARE SAVED ON CLOSING THIS EDITOR")
        self.content = initial_content
        self.text_frame = ScrolledText(self.root_frame)
        self.text_frame.insert(1.0, self.content)
        self.text_frame.pack(fill="both", expand=True)

    def synchronize(self) -> str:
        self.content = self.text_frame.get('1.0', tkinter.END)
        return self.content

    def closing(self) -> None:
        self.synchronize()
        self.root_frame.destroy()

    @staticmethod
    def edit_text(previous_text: str) -> str:
        root = tkinter.Tk()
        t: TextEditor = TextEditor(root, previous_text)
        root.protocol("WM_DELETE_WINDOW", t.closing)
        root.mainloop()
        return t.content
```

A.20 _burn.bat

```
call _burn_light.bat
del /s /q *.bin
```

A.21 _burn_light.bat

```
rmdir /s /q __pycache__
rmdir /s /q .mypy_cache

del /s /q *.json
del /s /q *.config
del /s /q *.sqlite
del /s /q *.container
```

A.22 _burn.sh

```
#!/bin/bash

./_burn_light.sh
rm -f -r -d *.bin
```

A.23 _burn_light.sh

```
#!/bin/bash

rm -f -r -d __pycache__
rm -f -r -d .mypy_cache

rm -f -r -d *.json
rm -f -r -d *.config
rm -f -r -d *.sqlite
rm -f -r -d *.container
```

A.24 requirements.txt

```
# CRYPTO
pyAesCrypt~=6.0.0            # AES CBC (files)
pycryptodome~=3.17           # AES GCM (communication)
rsa~=4.9                     # RSA (certificates)
psutil~=5.9.4                # multithreaded certificate generation

# SERVER
Flask~=2.2.3                 # HTTP-server
argon2-cffi~=21.3.0          # password hashing
PyJWT~=2.6.0                 # JSON Web Token

# CLIENT
requests~=2.28.2             # HTTP-requests
universal-startfile~=0.2     # OS-dependent file opening
```