FACULTY OF INFORMATION TECHNOLOGY CTU IN PRAGUE

Assignment of bachelor's thesis

Title:	KeePass Password Manager Secure Cloud Storage
Student:	Konstantin Filip Moisidis
Supervisor:	Ing. Jiří Dostál, Ph.D.
Study program:	Informatics
Branch / specialization:	Computer Security and Information technology
Department:	Department of Computer Systems
Validity:	until the end of summer semester 2022/2023

Instructions

Passwords and how to protect them is and forever will be a fundamental question. One of the ways how to protect passwords is by using an application called a password manager. The follow-up questions then are: How does the password manager protect stored passwords? Where are the passwords stored? Moreover, how is the storage protected? This thesis aims to implement a plugin for the KeePass password manager that provides means for storing the database in the cloud solution. In addition, a second authentication factor will be used to protect the database.

1. Get familiar with applicable KeePass plugins.
2. Examine public cloud storage services in terms of secure storage.
3. Based on the previous research, propose a solution for safe storing of the KeePass passwords database in the cloud.
4. Implement, test, and document your solution for the second factor secure storage.
5. Make a threat model and identify possible cybersecurity threats.

Electronically approved by prof. Ing. Pavel Tvrdík, CSc. on 31 January 2022 in Prague.

Bachelor's thesis

KEEPASS PASSWORD MANAGER SECURE CLOUD STORAGE

Konstantin Filip Moisidis

Faculty of Information Technology
Department of Information Security
Supervisor: Ing. Jiří Dostál, Ph.D.
May 11, 2022

Contents

List of Figures

List of Tables

List of code listings

Throughout the writing of this thesis, I have encountered several setbacks. However, thanks to a great deal of support and assistance, I have prevailed.

Firstly, I would like to thank my supervisor Ing. Jiří Dostál, Ph.D. for his guidance throughout this thesis and words of advice in times of greatest need.

Secondly, I would like to thank Ing. Josef Kokeš for the time spent on consultations, helping to improve found shortcomings.

Lastly and with equal importance, I would like to thank all my friends and family. Without their unimaginable support, this would have been an impossible feat.

Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis.

I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular that the Czech Technical University in Prague has the right to conclude a license agreement on the utilization of this thesis as a school work under the provisions of Article 60 (1) of the Act.

In Prague on May 11, 2022 .....

Abstract

Research on the KeePass plugin base and most common cloud storage providers showed that no combination of plugins or cloud providers creates a safe 2FA environment for the KeePass database. Therefore, several solutions were investigated. Researched technologies consist of OTP tokens, Microsofts crypto libraries, such as CNG or Windows Hello, and TPM 2.0. TPM was chosen for the solution proposal and later for implementation out of all these candidates. Using TPM and an arbitrary cloud provider, a cryptosystem was created and implemented into a working KeePass plugin. This plugin was tested and documented, as well as a creation of a threat model on this plugin, identifying possible cybersecurity threats.

Keywords password manager, KeePass plugin, cloud storage, multi-factor authentication, cryptography

Abstrakt

Výzkum základny zásuvných modulů KeePass a nejběžnějších poskytovatelů cloudových úložišť ukázal, že žádná kombinace zásuvných modulů a poskytovatelů cloudových úložišť nevytváří bezpečné, vícefaktorové prostředí pro databázi KeePass. Proto bylo zkoumáno několik řešení. Zkoumané technologie se skládají z tokenů OTP, kryptografických knihoven Microsoftu, jako je CNG nebo Windows Hello, a TPM 2.0. Ze všech těchto kandidátů byl pro návrh řešení a později pro implementaci vybrán čip TPM. Pomocí čipu TPM a námi zvoleného poskytovatele cloudových služeb byl vytvořen kryptosystém, který byl implementován do funkčního zásuvného modulu KeePass. Tento zásuvný modul byl testován, zdokumentován a podroben vytvoření modelu hrozeb, který identifikuje možné kybernetické bezpečnostní hrozby.

Klíčová slova správce hesel, KeePass plugin, cloudové úložiště, vícefázové ověření, kryptografie

Introduction

Password Managers

The world we live in today is a fast-moving, information-oriented roller coaster that will take by storm anyone who is not willing to adapt, improvise and eventually overcome its intricate inner workings. Average person is often required to have some piece of technology in their hands to operate normally. Chats with friends, phone calls with relatives, and even vaccination certificates are digitized. A lot of effort has to be made to make these things reliable, user-friendly, and, most importantly, secure. Cybersecurity is most certainly an infinite topic to unravel, so this thesis aims to cover only a narrow topic of password management.

Most of us concluded that keeping our PC password on a sticky note next to our computer is not the best way of protecting against unauthorized access. A better solution is to keep our passwords memorized, but we find it troublesome to remember complicated long strings of letters, numbers, and other symbols. We, therefore, tend to make those passwords memorable and predictable, which in turn makes them less secure due to dictionary attacks. One solution to this is to use software to keep passwords in one place. This software is called password manager, and there is a variety to choose from.

One of the most commonly used password managers is KeePass. KeePass has a vast community that is full of extensions and plugins. However, none of these plugins lets users use cloud storage with some other form of secure storing – e.g., two-factor, for instance. Users can upload KeePass database to the cloud manually or automatically and add additional layers of security to the password database. But not with one single extension.

Aim

This thesis aims to create a KeePass plugin which provides users with 2FA cloud storage. A research of the current KeePass plugin base and cloud storage providers will be conducted. Based on these findings, a solution will be proposed and implemented. Implementation will then be tested and documented. Lastly, a threat model will be made to identify possible cybersecurity threats.

Chapter 1
KeePass

This chapter provides a small introduction to KeePass password manager as well as a research on current cloud providers and the KeePass plugins related to a cloud communication and a second-factor authentication.

1.1 About

KeePass is a password manager designed mainly for Windows operating system. Its open-source platform allows more advanced users to inspect the source code and implementation of cryptography mechanisms [1]. As opposed to, for instance, 1Password, Dashline, KeePass stores passwords in a database placed in a file system [2]. It has many built-in features for importing and exporting [3], Auto-typing [4], and its Built-in password generator.

Thanks to its open-source license, KeePass has an extensive plugin base ranging from cosmetic changes to entirely new functionalities [3].

KeePass was created in 2003 by Dominik Reichl. This almost 24-years-old software receives updates to this day. Historically, there are two versions of KeePass. The first version (denoted as 1.x version) was written in C++. Around five years later, KeePass 2.00 released its alpha version this time; and, it was developed in C#. KeePass version 1 still receives occasional updates; however, based on edition comparison[1], KeePass version 2 offers more advantages.

[1] Full comparison can be found at: `https://keepass.info/compare.html`

1.2 Plugins

This chapter focuses on examining plugins related to OTP, TPM 2.0, or cloud synchronization. Providing examples for each category and discussing their potential uses and flaws.

1.2.1 Cloud storage plugins

KeePass plugins have a basic coverage for all the most popular cloud storage. Most prominent are KeeAnywhere [5] provide more than five cloud services, and KeepassSync [6], with four online storage providers. For a more conservative selection, KPGoogleSync [7] and KeePassOneDriveSync [8] are direct implementations for Google Drive API and Microsoft Graph API, respectively.

1.2.2 OTP plugins

Plugins allowing OTP can be divided into two categories. One category represents an additional layer of security with OTP protection of the database. The other category is a set of plugins that support storing OTP seeds within KeePass entry, ergo adding no new layer of security. The first category consists only of one plugin, and that is OtpKeyProv. Created by the KeePass creator [9], OtpKeyProv provides users with another factor of authentication; possession. This possession is in the form of a seed – more on OTP in the third chapter. Plugins from the second category, KeePassOTP, KeeOtp2, and KeeTrayTOTP, are only for a generation of OTP tokens. The user needs to supply a seed; plugins then generate the token. [10] [11] [12]

1.2.3 TPM plugins

Currently, KeePass plugin base does not have support for a TPM. The argument can be made that implementing TPM compatibility is somewhat of a reinventing the wheel[2] since KeePass has a safe mechanisms for storing its passwords. The only intriguing aspect of using TPM should include some other mechanism or principles for it to be a viable option. More on this topic in the Viable options chapter. The tenets of TPM are explained in the TPM chapter as well.

[2]KeePass already provides a solution for a safe password storing. Using TPM 2.0 can be only thought of as a complementary service.

Chapter 2

Cloud storage

This chapter focuses on examining current cloud service providers and their ability to offer a multi-factor authentication.

2.1 About

Cloud storage is a model of computer storage that allows storing digital data in, mostly, off-premise servers. Users do not need to know what servers their data is stored on or where the servers are physically located. Providers try to guarantee security and accessibility. However, not always are these guarantees upheld. Sadly, cloud leaks and security breaches are waiting to happen almost every day. Yahoo breach in 2013, Facebook in 2019, and, more recently, LinkedIn in 2021 all faced the same issue – sensitive data being exposed to the internet [13]. These issues were caused by the providers themselves. When it comes to end-users, the protection of their data is as strong as the password protecting their account. Many providers allow for two-factor authentication. However, some have additional features.

2.2 Providers

One of the newer additions to cloud storage is a term coined by Microsoft; Personal Vault. Microsoft's could storage OneDrive has developed an additional feature to their cloud, an obligatory second-factor authorization for accessing most sensitive data [14]. Upon logging in, the user gains access to cloud contents but to unlock the Personal Vault, the user must comply with the second factor, in this case, possession. This idea, however, is unparalleled in the competition. IDrive, Google Drive, and Dropbox are still missing similar feature of a Personal Vault. The major disadvantage persisting to the current day is a non-existent API for a Personal Vault[1]. This makes integration for KeePass plugin practically impossible.

[1]The current Microsoft Graph API, which provides all OneDrive API calls, does not mention of Personal Vault or any of its features.

Chapter 3
Viable options

To achieve second-factor protection for a KeePass database in the cloud, users may use the Personal Vault feature of OneDrive. However, this is not an ideal solution. This solution is bound to Microsoft's cloud service. This chapter will attempt to propose a solution for a general cloud service storage using a KeePass plugin.

3.1 One-time password

A one-time password, OTP for short, is a password meant to be used only once or in a specific timeframe. This kind of authorization is commonly used as a second factor due to its implementation. OTP can be divided into many categories. Main focus will be on two implementations. HOTP, short for HMAC-Based One-Time Password Algorithm, and TOTP Time-Based One-Time Password Algorithm. [15]

HOTP and TOTP both work on a pre-shared key premise. Meaning that both parties, be it a client and a server, need to share this key securely. With this assumption, a sequence of, most commonly, numbers is generated by the OTP client and verified by the server. [16] [17]

3.1.1 HOTP

HOTP works using HMAC; hashed message authentication code. Message authentication codes are short messages used for authenticating a message. With HMAC, messages sent over the internet are hashed with a derivation of a pre-shared private key, followed by another hashing of this hashed message with a different derivation of the pre-shared key. These derivations are named *opad* and *ipad*. The "i" and "o" are mnemonics for inner and outer. [16]

With the message as *text* and key as K, HOTP can be described with following expression:

$$H(K \oplus opad, H(K \oplus ipad, text))$$

Using *opad* and *ipad* in this manner is done to prevent a Length Extension Attack. [18]

For HOTP, HMAC takes in two parameters – a sequence counter and a secret key. A sequence counter is simply a counter starting at zero when the key exchange is completed. With each authentication, the client and server increment the sequence counter. [17]

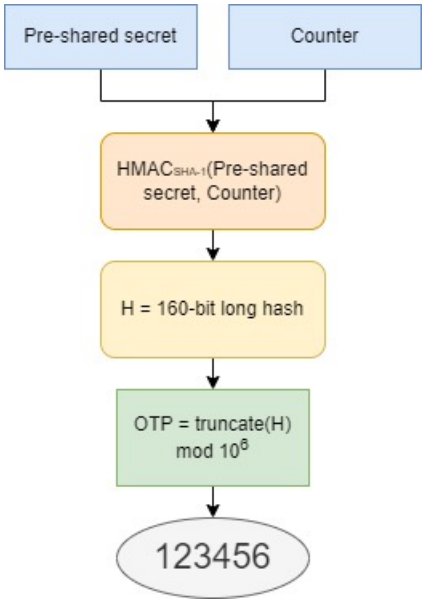

Figure 3.1 Chart flow diagram for HOTP implementation

3.1.2 TOTP

TOTP work very similarly with one difference. The sequence counter is replaced with the current time value. For this time value, Unix time is used.

Unix time is a signed integer value of 32-bits, representing the number of seconds that have passed since the Epoch – January 1st, 1970. Unix time and pre-shared key are then used as an input for HMAC and again truncated. Good practice on a server-side is to hold an error window for verification. This means that the server is precalculating a few hashes ahead and keeping a few hashes that have timed out. This helps users deal with shortly expired codes, making them still valid. [19]

3.1.3 Conclusion

A conclusion to this section would be an applicable OTP plugin for cloud storage; however, after considerable time and examinations, all solutions showed to be a pseudo-second factor as a standard cloud storage does not provide any computing power for calculating OTP token and evaluating it. OTP is, therefore, not within the realm of valid options.

3.2 Trusted Platform Module

A trusted Platform Module is a standard for a secure cryptoprocessor. It was standardized by a consortium called Trusted Computing Group in 2009 and, since then, it is developed and maintained. It represents an anchor point of trust in a system. It is primarily used to ensure the integrity of a platform. Integrity in this context means, for example, secure boot. [20]

TPM ensures that the boot process starts from a trusted combination of hardware and continues until all parts of an operating system are running. Other uses include disk encryption, a hardware random number generator, or Binding (encryption by a unique RSA key descended from the primary key). The implementation chapter highlights the Binding feature as a part of the solution. [21]

3.2.1 TPM Software Stack by Microsoft

TPM Software Stack from Microsoft Research is an API made by Microsoft. This API provides a much more user-friendly layer for using TPM devices as the raw documentation made by TCG is rather complicated.

TCG introduced Enhanced System API (ESAPI) and Feature API (FAPI), and TPM Command Transmission Interface (TCTI). This all combined constitutes a Trusted security stack.

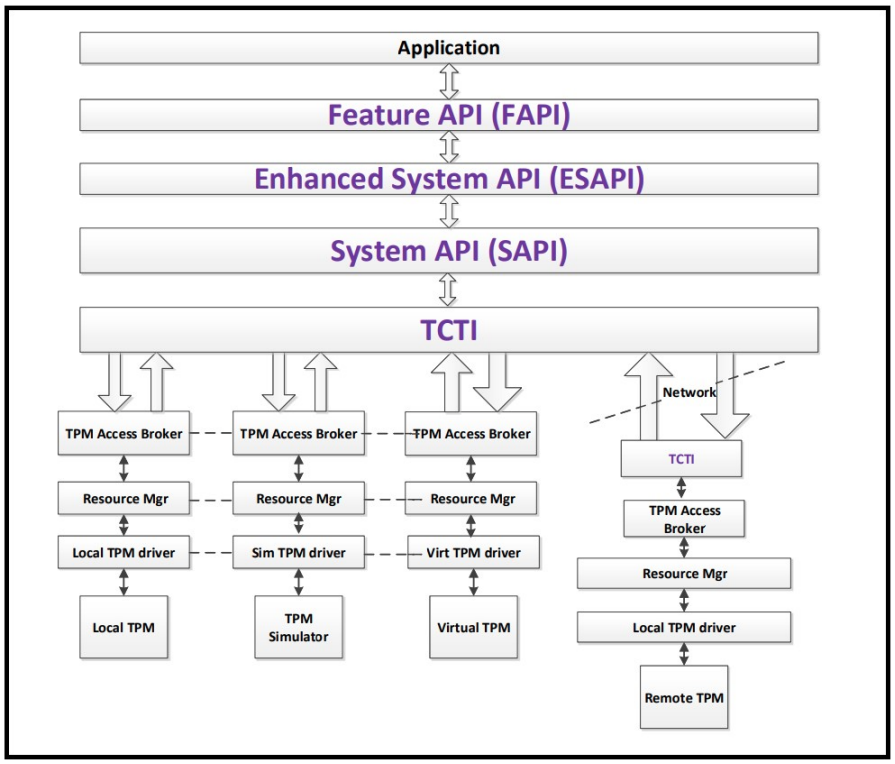

Figure 3.2 Description of TCGs TPM 2.0 Software Stack by TCG

The TPM Software Stack from Microsoft Research provides rich API wrappers encapsulating all the API layers with one-to-one mapping to TPM commands.

3.3 Microsoft

Moving away from lower levels of programming, Microsoft provides its developers with several security-related APIs for authorization, authentication, or general-purpose cryptography. APIs that could be used are introduced in the following subsections.

3.3.1 Windows Hello

Windows Hello is a biometric sign-in system provided by the Windows 10 operating system. Because it is built directly into the operating system, it allows users to login with, for example, face or fingerprint. This combines two vital elements of security; authentication and authorization. What is the difference? Authentication is a process of verifying a user's identity, whereas authorization is a process of verifying if the supplied credentials are allowed for the specified action – access rights. Windows Hello can makes use of both of these, making it essentially 2FA. Fingerprint (something a person is) and a PIN code (something a person knows).

Microsoft also made Windows Hello API available to developers, which means that any application can use Windows Hello 2FA. With a closer look into the Windows Hello documentation, TPM technology is used for public-private key generation. [22]

To conclude this section, Windows Hello is a suitable candidate for a KeePass multi-factor cloud storage providing a 2FA API.

3.3.2 CSP and CNG

The last thing worth mentioning is two Microsoft APIs; Cryptographic Service Provider (CSP) and Cryptography API: Next Generation (CNG). These are, as the name suggests, cryptography APIs made by Microsoft. They essentially represent the same capabilities. However, CNG is a renewed, updated version of the previous CryptoAPI with better API and newer algorithms. Using one of these APIs, KeePass plugin can use a pair of generated keys as a possession factor, encrypting KeePass database before uploading it to the cloud.

This would seem like a satisfactory solution; however, Windows as an operating system does not have an excellent way of storing private keys. The only exception may be CNGs Key Storage and Retrieval model, which mimics TPM with the hardware. [23]

To conclude this section, using only software to handle private keys, be it CSP or CNG, is not recommended and should be avoided if possible.

3.4 Research conclusion

The overall conclusion of this research chapter is a proposal to implement technologies suitable for KeePass plugin that can securely manage its database in the cloud.

The proposition is as follows: For demonstrative purposes, one cloud storage provider is to be chosen and, with its API, implement upload and download functionalities for the KeePass plugin. Using TPM, generate a pair of keys where the private key shall be stored within the TPM. Upon saving the database, encrypt it using the TPM key (representing a possession factor) and upload it to the cloud storage. Upon user request, download the database from the cloud, unlock it and let the user make their changes. Should the user be done or reopens the database at a later date, this process repeats.

Chapter 4
Implementation

This chapter focuses on the implementation details of the previously proposed solution. Starting with Google Drive API implementation for cloud communication and utilizing TPM for key handling. Lastly, putting all these components together with basic concepts of KeePass plugin creation.

4.1 Google Drive API

Google API is a collection of API calls allowing interaction with Google related services. The main emphasis of this plugin is on Google Drive communication, narrowing down what needs to be explained.

4.1.1 Google Cloud Platform

The first step in developing a Google Drive application is to create a project in the Google Cloud Platform. GCP shortly explained is a suite of cloud computing services. One of its features is a project that can be referenced with API tokens for inner Google authentication. This allows for a simple set of tokens to provide working Google Authorization. Desktop applications use these tokens to communicate with Google servers to authorize any existing user.

This is achieved with a hierarchy of classes constructing, in the end, an object representing a communication channel for an update, upload, and download.

4.1.2 Class hierarchy

The first class that needs to be constructed is `ClientSecrets`. `ClientSecrets` class represents API tokens gained from a previously created project on the Google cloud platform. This token consists of `ClientId` and `ClientSecrets`. These are somewhat self-explanatory. `ClientId` is a Cloud Platform project ID, and `ClientSecret` can be understood as a password authenticating an incoming connection (4.1).

Code listing 4.1 ClientSecret class

```
ClientSecrets = new ClientSecrets
    {
        ClientId = GoogleDriveClientId,
        ClientSecret = GoogleDriveClientSecret
    }
```

Once the `ClientSecrets` class is constructed[1], it is used as a building block for a `UserCredential` class. With the usage of `GoogleWebAuthorizationBroker`, a connection to Google Authorization servers is established, prompting users with browser login. With a few more arguments, like a scope of application, and cancellation tokens, the `GoogleWebAuthorizationBroker` then returns initialized `UserCredential` (4.2).

Code listing 4.2 UserCredential class

```
UserCredential = GoogleWebAuthorizationBroker.AuthorizeAsync(
    new GoogleAuthorizationCodeFlow.Initializer
        ClientSecrets,
        Scopes,
        name,
        cancellationToken,
        new NullDataStore()).Result;
```

As the last step, the `UserCredential` object is used to construct the final class – `DriveService`. `DriveService` is the object used for uploading, updating, or downloading files with the corresponding methods (4.3).

Code listing 4.3 DriveService class

```
DriveService service = new DriveService(
    new BaseClientService.Initializer()
        {
            HttpClientInitializer = UserCredential,
            ApplicationName = ApplicationName,
        });
```

This whole process is a description of a `CreateLogin` method. This method is called upon adding a new user to the KeePass plugin application. How users are managed is explained in the Profiling with Windows registry section.

To prevent users from recreating logins each time the KeePass application starts, a `LoginAs` method is used. This method uses the same API calls as `CreateLogin` however, `UserCredential` is supplied with a refresh token. This results in automatic login with no need for reauthorizing.

With `LoginAs` and `CreateLogin` methods explained, the only things remaining are upload and download functionalities. As mentioned before, `DriveService` is a handle for all of these operations. After Logging in with `LoginAs` method, `DriveService` object is created. When a user decides to save their work on a KeePass database or opens KeePass for accessing the database, methods `Upload` and `Download` are used. These are methods of mentioned `DriveService` object and are reasonably simple to use.

[1] Working token is included in the source code of this project; however, users should create this token for themselves. Firstly, it is not guaranteed that this service will be running in the long term; secondly, it is for their own security reasons.

Note: After a successful login, a refresh token is created. A refresh token is a client secret representing a login session. This token can have a time expiration but still represents a potential vulnerability. Should the token be stolen, a Google account with weak security measures can be logged in by an intruder.

To protect these tokens, a Microsoft DPAPI library was used. This library consists of `ProtectedData` class which provides a simple encryption service. The encryption uses a `DataProtectionScope`, which is either `CurrentUser` or `LocalMachine`. This class does not provide any keys; it encrypts messages with the current user session or with local machine credentials. In this case, use of the `CurrentUser` scope, protects tokens from unauthorized theft (4.4). [24]

Code listing 4.4 Data protection method

```
byte[] bytes = Encoding.ASCII.GetBytes
    (credential.Token.RefreshToken);
var encrypted = ProtectedData.Protect
    (bytes, null, DataProtectionScope.CurrentUser);
```

Uploading starts with a `Create` method. `Create` method needs a `FileStream` to create `FileResource`. `FileStream`, in this case, is a path to the password database. When this `FileResource` is created, an `Upload` method (method belonging `FileResource`) is called, uploading the file, with the current authorization session, to the user's Google Drive (4.5).

Code listing 4.5 Upload method

```
var driveFile = new Google.Apis.Drive.v3.Data.File();
driveFile.Name = "Personal Vault";
driveFile.Description = "Vault";
driveFile.MimeType = "application/octet-stream";

var request = driveService.Files.Create
            (driveFile, file, "application/octet-stream");
request.Fields = "id";

var response = request.Upload();
```

Updating file[2] method looks and works almost identically to Create method. The only difference is a `fileID` reference of the desired file. `FileID` is a unique ID referencing the Google Drive file (4.6).

Code listing 4.6 Update method

```
using (var stream = new FileStream(filePath,
                            FileMode.OpenOrCreate))
    {
        updateRequest = service.Files.Update(driveFile,
                                        fileID,
                                        stream,
                                "application/octet-stream");
        updateRequest.Upload();
        var file = updateRequest.ResponseBody;
        return file.Id;
    };
```

Downloading is made even more straightforward. Developers can use extensive filtering on users' Google Drive files. One specific filtering is by ID. This ID is stored together with user credentials when `CreateLogin` is called. Meaning it is persistent and can be referenced later. So to download a specific file, `FileID` is looked up and handed to the download method (4.7).

Code listing 4.7 Download method

```
var request = service.Files.Get(fileID);

string FileName = request.Execute().Name;
string FilePath = Path.Combine(FolderPath, FileName);
MemoryStream stream = new MemoryStream();

request.MediaDownloader.ProgressChanged +=
    (IDownloadProgress progress) =>
    {
        switch (progress.Status){
            case DownloadStatus.Downloading:
            { break; }
            case DownloadStatus.Completed:
            {
                SaveStream(stream, FilePath);
                break;
            }
            case DownloadStatus.Failed:
            { throw new Exception(); }
        }
    };
request.Download(stream);
```

To summarize this section, Google Drive API establishes a connection to the GCP project with `ClientId` and `ClientSecret`. This connection prompts the user with a Google login, and if successful, a refresh token is saved. This token can be reused to re-login if it is not expired. The functionality described in this section is implemented in `GooglAuth.cs` class.

[2]Meaning that file already exists and does not to be created.

4.2 TSS.MSR

TSS.MSR is a Windows TPM Software Stack developed by Microsoft. This seems to be one of few applicable implementations due to the fact that KeePass is written in C# and TSS.MSR has a .NET variant.

4.2.1 TSS.MSR and TPM 2.0

To fully understand what TSS is, we first need to understand what TCG has standardized. As mentioned in the TCG guide, TPM is a standard for a secure cryptoprocessor [21]. This means that developers who are bound to an operating system can not directly interact with the TPM chip. First, the operating system needs to know how to communicate with the TPM. This, therefore, implies that communication with TPM is different with each operating system. For the Windows system, one of the solutions is mentioned TSS.MSR.

TPM and its architecture offer hundreds of commands. However, building a secure cryptosystem does not require hundreds of commands. With this plugin's implementation, only a handful is needed.

4.2.2 Basic concepts

The first thing to understand is that TPM comes with an endorsement key. The endorsement key is embedded into TPM and can not be changed. This endorsement key is then used to derive a hierarchy of keys for the user. [25]

The second thing to understand is an authorization session. If a developer decides to communicate with TPM, they must do so with a secure channel. This authorization is achieved with HMAC, parameter decryption, and then response encryption[3]. In more detail, this authorization session starts with a nonce[4] generated by the user side and a nonce generated by TPM. These two are then used to derive a session key. [26]

4.2.3 Usage

To start using the TPM and to start creating keys for users, a hierarchy needs to be created as well. A key always needs a parent authority. A primary key is a key that uses the endorsement key or storage root key as its parent. The primary key is the first key of the hierarchy and is used as a parent to all other user-created keys. When the application is run, this key is created each time. [27]

This key remains the same based on the fact that the endorsement key is persistent within the TPM. This, however, creates a slight complication. This key generation is time-consuming, and only recently, the TCG created a Provisioning Guidance which states that these primary keys are stored in the persistent storage at NV address 8100001 [28]. This, however, was not implemented. The Key is, in this case, always recreated (4.8) [29]. [30]

[3]These examples all are principles used for an authorization session. All three should be present while communicating with the TPM.

[4]In cryptography, a nonce (number once) is an arbitrary number that can be used just once in a cryptographic communication.

Code listing 4.8 Create primary key method

```
// Construction of necessary parameters
var sensCreate = new SensitiveCreate
    (new byte[] { 0xa, 0xb, 0xc }, null);
byte[] outsideInfo = new byte[] { 0, 1, 2 };
var creationPcr = new PcrSelection
    (TpmAlgId.Sha1, new uint[] { 0, 1, 2 });

TpmPublic parms = new TpmPublic( ... )
TpmPublic pubCreated;
CreationData creationData;
TkCreation creationTicket;
byte[] creationHash;

// Creation of primary key
TpmHandle h = tpm.CreatePrimary(TpmRh.Owner, sensCreate, parms,
                        outsideInfo,
                        new PcrSelection[] { creationPcr },
                        out pubCreated, out creationData,
                        out creationHash, out creationTicket);
return h;
```

Note: Values for `SensitiveCreate` and `creationPcr` are chosen arbitrarily. Code for `TpmPublic` is purposely hidden and can be inspected in `TSS.cs` class for further detail.

With the primary key created, a standard[5] key can be created, and this time for encrypting user data. When creating a key, TPM does not load it automatically. Creating (4.9) and loading (4.10) a key are two separate operations .

Code listing 4.9 Create key method

```
TpmPublic keyInPublic = new TpmPublic( ... )

SensitiveCreate sensCreate = new SensitiveCreate
                                (new byte[] { 2, 2, 3 }, null);
CreationData keyCreationData;
TkCreation creationTicket;
byte[] creationHash;

TpmPrivate keyPrivate = tpm.Create(primHandle, sensCreate,
                                   keyInPublic, null,
                                   new PcrSelection[0],
                                   out keyPublic,
                                   out keyCreationData,
                                   out creationHash,
                                   out creationTicket);

return keyPrivate;
```

[5]Naming the key standard is done only to distinguish it from the primary key.

Note: `SensitiveCreate` represents authentication key. It needs to be remembered for later reconstruction and authorization sessions.

Note: Despite its class name `TpmPrivate` is not a readable private key. This structure is encrypted by the TPM and represents the private part of the key. Sometimes referred to as the private blob. [31]

Code listing 4.10 Load key method

```
TpmHandle keyHandle = null;

tpm._Behavior.Strict = true;

// No auth session is added automatically when
// TPM object is in strict mode.
tpm._ExpectError(TpmRc.AuthMissing)
    .Load(primHandle, keyPrivate, keyPublic);

// Now explicitly request an auth session of a desired type.
// Actual auth value will be supplied by TSS.Net implicitly.
keyHandle = tpm[Auth.Default].Load(primHandle,
                                   keyPrivate,
                                   keyPublic);

// Switch TPM object back to the normal mode.
tpm._Behavior.Strict = false;

return keyHandle;
```

Note: `tpm._Behavior.Strict` is a TSS.Net specific piece of functionality, not a part of TPM 2.0 specification [32]

This kind of key is, unfortunately, not persistent. This can be prevented by making the key persistent. Making key persistent is done by the `EvictControl` function. `EvictControl` internalizes the `keyhandle` making it a valid reference even after a computer restart (4.11). [33]

Code listing 4.11 Make key persistent method

```
hPers = NextPersistentHandle(TpmRh.Owner);
tpm._ExpectResponses(TpmRc.Success,
            TpmRc.NvSpace,
            TpmRc.NvDefined)
   .EvictControl(TpmRh.Owner, handle, hPers);
```

Note: `NextPersistentHandle` is a method returning a persistent handle. `EvictControl` then uses the newly loaded `handle` and assigns it to `hPers`

This keyhandle needs to be exported to a file and kept by the user. In this case, it is the user's responsibility to protect this file from corruption as it represents the only means of gaining control over this key.

With a persistent key, developers can start encrypting and decrypting messages with TPM methods (4.12). However, in this case, the KeePass database can grow in size. This means that asymmetric encryption is not an efficient approach. What is used here is a hybrid encryption. The TPMs `GetRandom` method generates a symmetric key for AES-256.

Note: *`GetRandom` is a True Random Number Generator (TRNG) that can be used for various applications including cryptographic purposes. This RNG module serves as the source for randomness for the TPM 2.0 Chip.* [34]

Code listing 4.12 Database encryption method

```
// Create Symmetric Key
byte[] aesKey = tpm.GetRandom(3);

// Encrypt the db with Block Cypher
AESEncryptFile(dbPath, aesKey, false);

// Encrypt the key with TPM
encryptedKey = tpm.RsaEncrypt(hPers, aesKey, scheme, null);

// Write the key to file
WriteToBinaryFile(pathAESKey, encryptedKey);
```

This symmetric key is then encrypted with a private key previously Evicted. When the user decides to save their work on the KeePass database, the AES key is used to encrypt the database, which is then uploaded to the cloud. The other way around, if the user downloads the encrypted database, the AES key is decrypted and used for database decryption. The functionality described in this section is implemented in `TSS.cs` class.

The summary of this section is as follows. TPM creates keys in a hierarchy. Each key is encrypted with its parent, and on top of this hierarchy stands the endorsement key[6], which never changes. To securely communicate with TPM, authorization sessions are used to encrypt the communication. Using the endorsement key, a primary key is created. This primary key is always the same since it is based on the endorsement key. A standard key is created with the primary key as a parent and made persistent. The database itself is encrypted with an AES key, which is encrypted by the TPM key. User's requests for uploading and downloading the database are bound to these encrypting and decrypting operations.

[6] or a storage root key

4.3 Profiling with Windows registry

A use Windows registry is implemented to create a profiling system for managing more than one database and or more users.

Windows registry has a specification for what is meant to be used by the system, users, or anything else.

The root key structure for users is `HKEY_CURRENT_USER`. SubKey `SOFTWARE` is then used for program-specific data. The plugin creates a new key named `KeePassVault` inside the `SOFTWARE` key, and stores information about created profiles.

One key consists of `Name` and `Value`. The `Name`, in this case, is the profile name that is entered in profile creation. `Value` is a serialized JSON string containing boolean `isDefault` and `fileID` (Database ID on Google Cloud). Example of what a registry may look like in the following figure (4.1).

Table 4.1 Registry entry

Name	Type	Data
`Filip`	`REG_SZ`	`{"isDefault":false,"fileID":"abcdef123456"}`
`Kostas`	`REG_SZ`	`{"isDefault":true,"fileID":"ghijkl678901"}`

`IsDefault` shows which profile is meant to be loaded automatically when the KeePass starts. This string is serialized and deserialized when it needs to be read or changed. The functionality described in this section is implemented in `ProfileManagerForm.cs` class.

4.4 KeePass plugin creation

As the KeePass is open-source, some documentation on plugin creation can be found on the product's website. [35]

The first thing to understand is how plugins are imported. A plugin can be either DLL or PLGX. These are only file extensions, so the difference lies within the files. DLL is already compiled and compacted, and all KeePass does is include this DLL. PLGX is a packed file with source code that needs to be compiled by the KeePass binary with specified utility commands. Both these approaches have pros and cons. For instance, PLGX can not use cache while DLL can. DLL plugins are rarely supported on third-party KeePass implementations for Linux systems. The development will be made for Windows operating systems, so the DLL variant is more suitable. Complete comparison in the following table (4.2). [35]

Table 4.2 Comparison table

	DLL	PLGX
`Compatibility check`	`Weak only`	`Strong`
`Compatibility with custom builds (Linux)`	`Partial`	`Strong`
`Authenticode signing support`	`Yes`	`no`
`No compilation on the user's system`	`Yes`	`no`
`No plugin cache`	`Yes`	`no`

Importing a plugin starts with placing a plugin in the "Plugins" folder in the KeePass installation folder. The DLL, however, must have a Project Assembly information set to "KeePass Plugin". Only then is it loaded.

The second thing is to understand how plugins are written. A KeePass plugin must be a Visual Studio C# Class Library project. This project must have a KeePass binary referenced in the "Reference tab" to access the KeePass namespace.

With access to the KeePass namespace, developers can access the most important function, `Initialize`, which needs to be overridden. This function represents the startup of the KeePass application, even before the main window is displayed. In this method, all the main components of the plugin are called.

Code listing 4.13 Plugin insertion method

```
public override bool Initialize(IPluginHost host)
{
    if (host == null) return false;
    m_host = host;

    // If profile exists, open the database
    if (profile != null)
    {
        // Grab stored refreshtoken fomr datastore
        var token = gAuth.GetTokenFromDataStore(profile.name);

        // Login using the token
        service = gAuth.LoginAs(token);

        // Download the file
        filePath = gAuth.DownloadFile(service,profile.fileID);

        // Decrypt the file
        TSS tss = new TSS();
        tss.Decrypt(filePath);
        tss.Disconnect();

        // Open the database
        var ci = IOConnectionInfo.FromPath(filePath + ".dec");
        ci.CredSaveMode = IOCredSaveMode.SaveCred;
        m_host.MainWindow.OpenDatabase(ci, null, false);
        m_host.MainWindow.FileSaved += SaveBackToDrive;
    }

    return true;
}
```

In the case of this plugin, this method, if a default profile is present, calls for `GoogleAuth` class, which handles downloading and uploading to Google Drive. Should no profile be marked as default, nothing is downloaded.

The last thing worth mentioning is an event action provided by `MainWindow` class named `FileSaved`. This event can be subscribed to and will call a specified action when the user saves the database. In this case, it encrypts the database and uploads it to the cloud. Functionality described in this section is implemented in `KeePassVaultExt.cs` class.

4.5 Testing

This section provides insight into the preparation of the testing environment as well as an example of an installation. The installation part is described in a way that the user can follow with the figures and instructions along.

4.5.1 Preparation

To properly test the proposed solution and its implementation, it would be desirable to use actual TPM 2.0. This, however, was not possible due to the limitations of the implementation environment. TPM chip was not present on the system used for implementation; therefore, another solution was found.

TSS.MSR also offers a testing binary called TPM 2.0 simulator. [36] This binary simulates TPM 2.0 according to the TCG standard and is reliable for software development. This simulator binary uses a TCP connection and communicates with applications the same way a real TPM chip would[7]. [37]

To make use of this simulator, changes in `TSS.cs` class need to be made. In the TSS constructor, either `TcpTpmDevice` or `TbsDevice` is used to construct the TPM object. `TcpTpmDevice` takes in two arguments, `IP` and `PORT`. These credentials are used to connect to the simulator binary. `TbsDevice` takes no arguments and establishes the connection to TPM with system calls.

Code listing 4.14 Simulator or Device method

```
public TSS()
{
    // Choose Simulator or Chip
    tpmDevice = new TcpTpmDevice(DefaultSimulatorName,
                                 DefaultSimulatorPort);
    tpmDeviceHW = new TbsDevice();
    ...
}
```

In theory, these two commands can be swapped, and no loss of functionality shall arise. This, however, remains to be tested due to the limitations of the current hardware.

[7] This binary can be downloaded from `https://www.microsoft.com/en-us/download/details.aspx?id=52507`

4.5.2 Installation and example

To begin testing, compiled library KeePassVault.dll needs to be imported into the plugins folder of the KeePass directory. KeePass then should be launched. Following the launch of KeePass, the plugin should be loaded. A ”Personal Vault options” item should be visible in the ”Tools” section of the navigation menu to confirm if the loading process worked correctly (4.1).

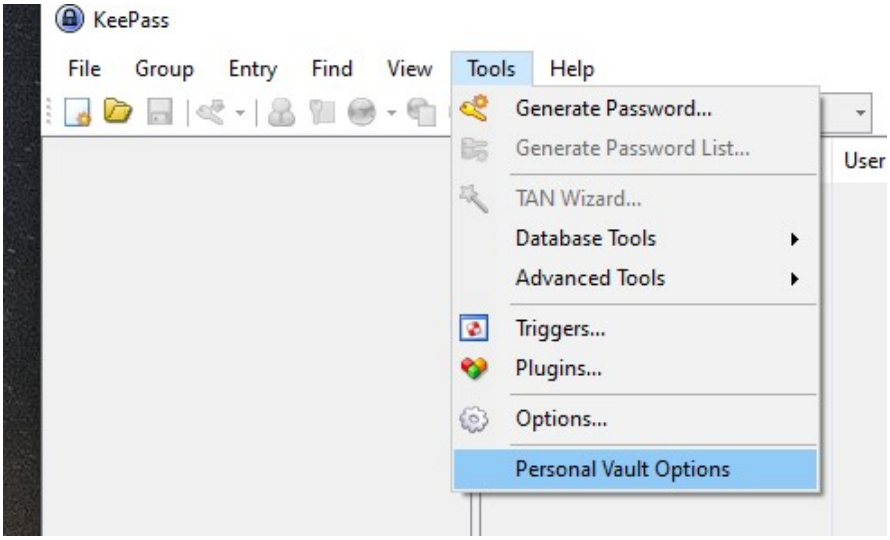

Figure 4.1 Successful installation and where to find settings for Personal Vault

Opening these options, the user is shown a profile manager window. Now empty, the list box presents added profiles (4.2).

Figure 4.2 Profile manager – used for adding and removing profiles

The user should begin by adding a profile by clicking the ”Add profile” button. This opens a window for the selection of database and Google account authorization. The order of these two does not matter (4.3).

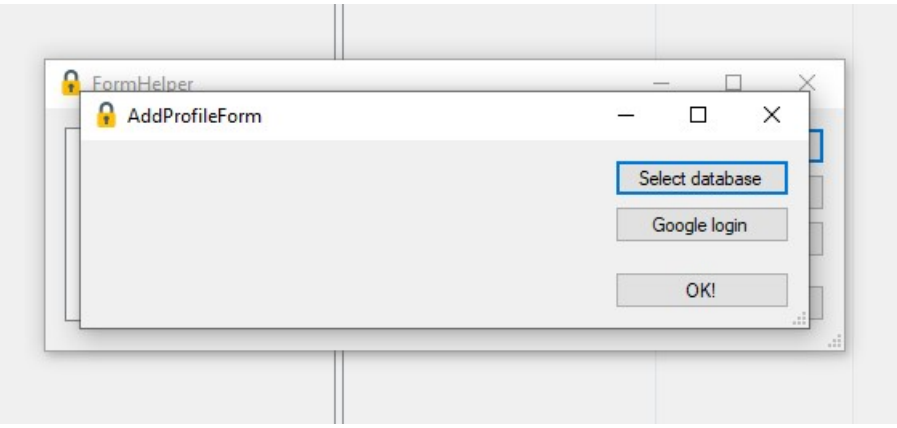

Figure 4.3 Adding profile – selecting database and a Google account

Proceeding with ”Select database”, the user selects a database to encrypt and upload to the cloud. This needs to be followed by the ”Google login” button, which prompts the user with a profile name selection. This is purely a quality of life feature allowing easier management of databases and profiles for users. Any name or nickname can be chosen. After confirming the name, a browser window is opened automatically, and the user is prompted with the Google authorization page (4.4).

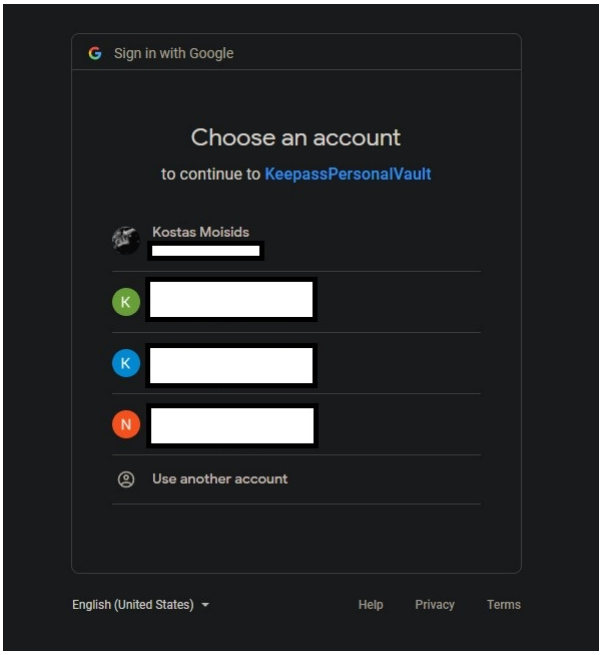

Figure 4.4 Google login – OAuth Browser Window

At this moment, settings in the GCP project need to be adjusted. A newly created project in GCP is automatically in testing mode. Testing mode warns users when authorizing of an unverified application hosted by GCP. If the user is not in the "test users" list, which can be found on the "OAuth consent screen" page in project management, they will not be allowed to use this application. So GCP project either needs to be published, or test users need to be added. In this case, the application has been published and should be accessible to any Google user.

Should the authorization be successful, the profile's name with the full name of the logged-in user is displayed next to the Google login button (4.5).

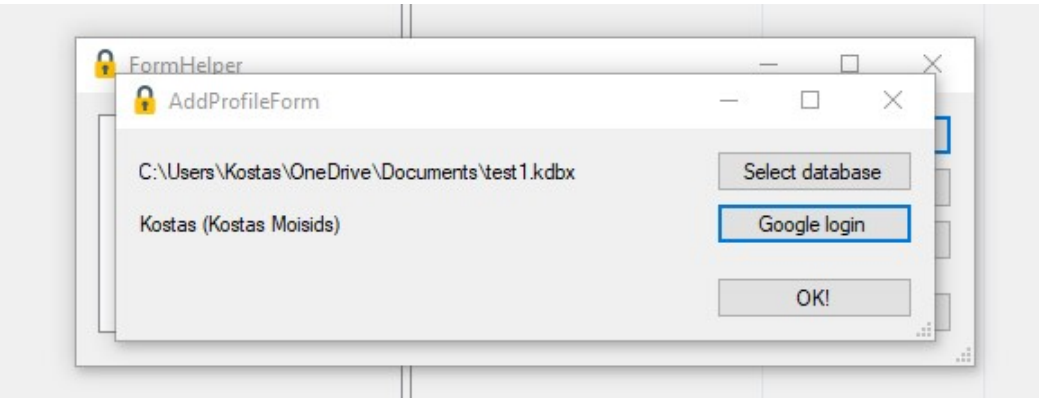

Figure 4.5 Adding profile – completed selection of database and a Google account

Note: 20-second cancellation token is used in this section giving the user a 20-second window. Should the user take too long with the login, the connection is dropped.

Clicking the "OK!" button initiates first-time database encryption and uploads it to the cloud. More information on how TPM performs these actions in TSS.MSR section. Should the encryption and upload be successful, the profile is added to the registry. The profile will now appear in the listbox, and by selecting it and pressing the "Make Default" button, the user will select this profile as default. The default profile is only one, and the default profiles database is automatically downloaded and decrypted upon KeePass startup (4.6).

Figure 4.6 Profile manager – profile creation completed

Should the user be done, they can save the database manually, force encryption immediately, and re-upload or exit. KeePass will prompt the user to save should the user exit with an unsaved database. This will trigger encryption and file update as well. This process can be repeated indefinitely as long as the simulator binary is running.

Note: The simulator binary needs to be kept running between KeePass reruns because re-running the simulator binary regenerates its endorsement keys resulting in a different primary key. This invalidates every key derived for the endorsement key, and decryption will no longer be possible. [38]

Note: Due to the nature of testing, one protection mechanism of the TPM is manually disabled. This protection mechanism is `DictionaryAttackProteciton`. This mechanism locks the TPM communication down should some requests be sent and fail too frequently. [39] Disabling this mechanism is done in the constructor for the TPM object in `TSS.cs` class.

Lastly, if the user decides to disable the TPM encryption, they can do so by removing the profile. Removing the profile downloads the database from the cloud, decrypts it and places it into the current user profile folder. The registry entry is then removed.

Chapter 5
Security analysis

This chapter focuses on identifying possible threats and creating a threat model. A threat model represents and evaluates how the application is implemented through a lens of security.

5.1 Threat model

There are many approaches to threat modeling. One of those approaches is The Process for Attack Simulation and Threat Analysis, PASTA. PASTA is a seven-step, risk-centric methodology. [40] [41]

The seven steps or stages are as follows:

Stage 1 Define the objectives

Stage 2 Define the technical scope of assets and components

Stage 3 Decompose the application

Stage 4 Threat analysis

Stage 5 Vulnerability detection

Stage 6 Attack Analysis

Stage 7 Risk or impact analysis and development of countermeasures

5.1.1 Define the objectives

Objectives, in this case, are things that are important with the application. Understanding what is the objective of the application is understanding its purpose.

The main objective of the KeePass is to protect passwords. Following this idea, this plugin's objective is to store the KeePass database in the cloud securely. This objective can be expanded to handling the database outside the cloud. This will be taken into account.

So the main objectives are:

- Database protection in the cloud
- Database handling off the cloud

5.1.2 Define the technical scope

The second stage should understand the attack surface by defining assets and components. Defining the attack surface, in this case, includes consideration of the dependencies of this plugin. However, this needs to be under-scoped because the primary focus should be placed on the application.

Using this, the display of considerable technologies is as follows:

- KeePass Password Manager
- Google Drive and its API
- TSS.MSR
- Overall implementation

5.1.3 Decompose the application

Decomposing the application should map the relations between mentioned components. How they communicate and what is being communicated.

A simplified decomposition for a specific section of KeePass startup using data flow diagrams in the following figure (5.1).

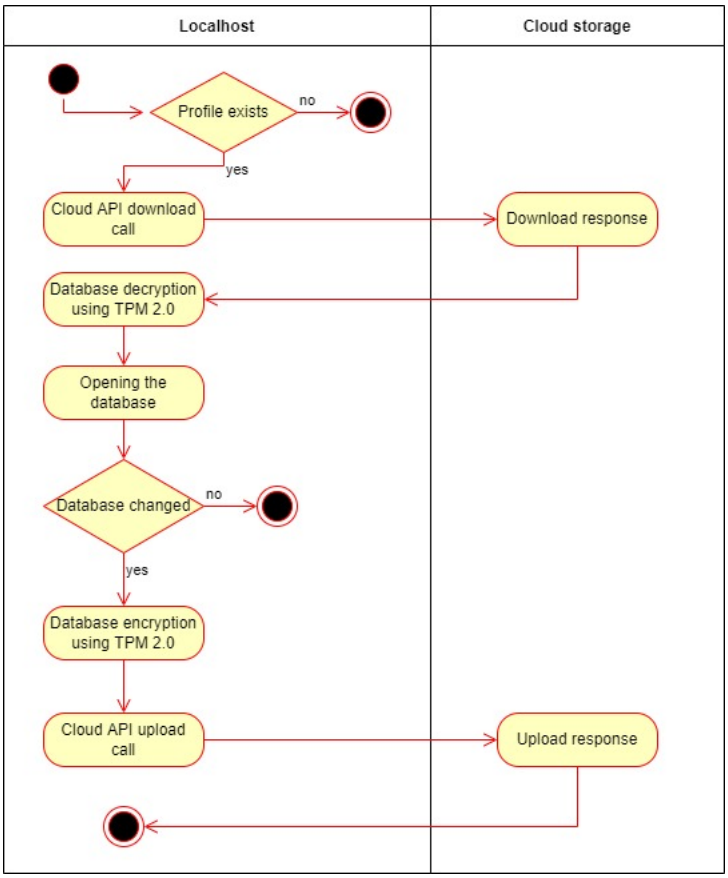

Figure 5.1 Demonstrative dataflow diagram for KeePass Startup

5.1.4 Threat analysis

Threat analysis is focused on the scope of the KeePass plugin. Considerable attack vectors, in this case, can be the following:

- Compromised credentials of the Google Drive storage
- Weak credentials of the Google Drive storage
- Ransomware
- Brute force

The last two attack vectors can be used if the user loses control over their Google Drive. Ransomware – extortion where data is deleted or encrypted unless a ransom is paid. Or a brute force attack on the encrypted database.

5.1.5 Vulnerability detection

Considering the first two stages, vulnerability detection can be done on two levels. The first level only considers the security of the cloud storage. The second level assumes compromisation of localhost.

With the first level, hardly any significant attack vectors can be found. The database is under two-factor authentication protection - possession factor (SHA-256 key) and knowledge factor (master password). However, some threats can still be pointed out. The first attack vector is credential compromisation of the Google Drive account. This would give an attacker access to the encrypted database. The second attack vector is a brute force attack on the AES key and the master password.

With the second level, more attack vectors arise. Limiting the scope of this analysis, assumption about the security of the KeePass application will be made. Proclaiming it secure, the analysis will treat KeePass component as a safe one. The same assumption will be made about TPM 2.0. Thanks to TGC, private keys can not be extracted from TPM, nor are they exposed to the user while using them [42].

However, the first attack vector can be found in the moment of only master password protection. The following figure highlights the time window of the decrypted database (5.2).

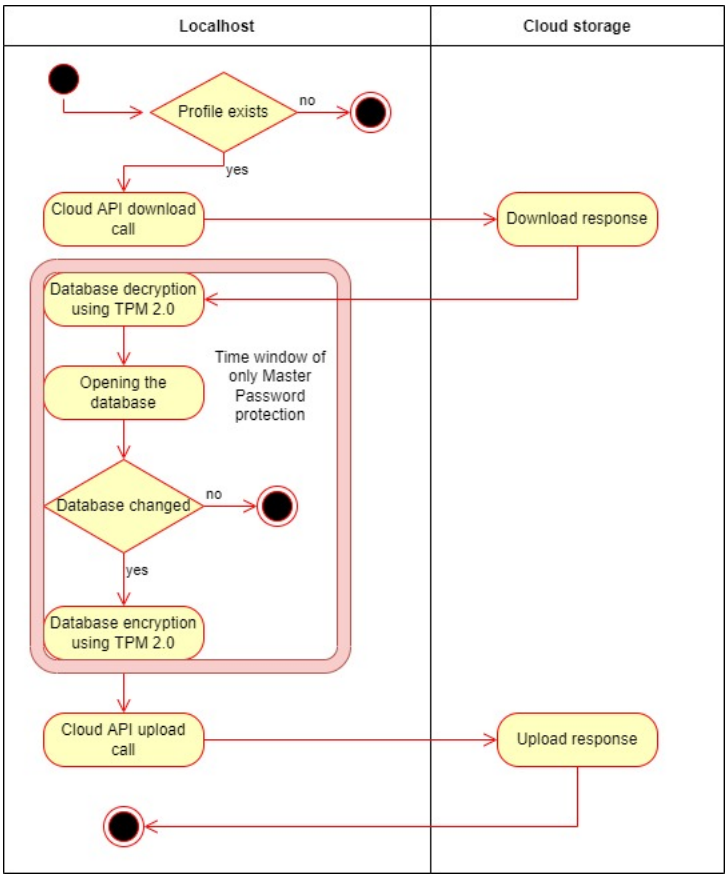

Figure 5.2 Vulnerability window in dataflow

This window represents the database's vulnerability to theft. Theft can then be followed by brute force attack on the master password, eliminating the second factor.

The next attack vector deals with localhost storage. Assuming that database is handled in a secure environment, the unencrypted database (unencrypted by the TPM) needs to be securely disposed of. Depending on the hardware, difficulty changes.

5.1.6 Attack Analysis

This stage focuses on the attacker's side of the analysis. As previous stages revealed attack surfaces, this stage simulates poetical attacks.

The first scenario that will be simulated is considering only a credential theft of the Google Drive account followed by the brute force attack on the AES key and master key. Credential theft can be done in various ways. One of the more common is a phishing attack. *"Phishing is a type of social engineering attack often used to steal user data, including login credentials and credit card numbers. It occurs when an attacker, masquerading as a trusted entity, dupes a victim into opening an email, instant message, or text message."* [43]

With access to the encrypted database, the attacker can begin brute-forcing AES-256. This already had become an almost impossible task. This kind of attack would require to go through 2^{256} combinations. The fastest computer in the world is currently the Fugaku supercomputer [44]. The Fugaku supercomputer has been benchmarked by LINPACK to 415.5 PFLOPS [45]. Petaflop is 10^{15} floating-point operations per second. In terms of base 2, this can be rewritten to 2^{50} with some inaccuracy.

Putting this all together Fugaku supercomputer can do 415.5×2^{50} operations per second. The following equation can then express the estimate for the final time needed for brute-forcing AES-256:

$$\frac{2^{256}}{415.5 \times 2^{50} \times 365 \times 24 \times 60 \times 60} \approx 8.047 \times 10^{51} \text{ years.}$$

However, time spent brute-forcing AES-256 only gets the attacker halfway there. Should the attacker be astronomically and unimaginably lucky and crack the encryption in their lifetime, the attacker needs to crack the master password.

Cracking the KeePass database can prove to be a rather lengthy task as well. The attacker, in essence, needs to go through all password combinations. What helps in favor of the attacker is the fact that the master password will most probably be a word or a phrase made by the user since it should be memorable.

Smart tools using password lists or carefully crafted rules for password generation can be used. To name a few, Keepass2john and Hashcat are both tools designed for brute force attacks. Keepass2john is a sub-module for John the Ripper tool. [46]

Assuming the credential theft, brute force of AES-256 and KeePass database, the attacker can, in theory, gain access to the victim's passwords.

The second scenario will assume an attack on the unsecured time window and compromisation of the operating system. There are numerous ways of compromising the Windows system, ranging from the most recent Log4Shell RCE [47] to an older EthernalBlue RCE [48]. With the assumption of a compromised system, the attacker can create an application that waits for a download request followed by TPM decryption. Once the decryption is done, the malicious application can copy the database to a different location entirely. The database can then be subjected to Keepass2john or Hashcat cracking, significantly reducing the estimated time.

The last issue with the second scenario is a secure deletion of the decrypted database. This eliminates the need for a time window when considering that the database is being worked within an insecure environment. An argument can be made that assuming the working environment is unsecured makes even a basic use of KeePass dangerous. As Dominik Reichl, creator of KeePass, said: *"Neither KeePass nor any other password manager can magically run securely in a spyware-infected, insecure environment."* [49]

Deleting the database with simple system calls can be easily reverted as there are many software projects for retrieving deleted files, such as EaseUS Data Recovery or Disk Drill.

5.1.7 Risk and Impact Analysis

A proposal of countermeasures that mitigate the threats will be made to finalize the threat modeling.

The main objective, secure cloud storage, fails to protect the database only under credential theft conditions. Be it a phishing attack or weak credentials, this risk is in the hands of the user. It is paramount for users not to use weak passwords or, in case of phishing, click suspicious links or reveal any personal information to an unverified end.

For the second objective – compromisation of the operating system or local storage, a secure deletion of the database should be implemented. One solution would be the utilization of Sysinternals applications. Sysinternals is a set of utilities to help users manage, troubleshoot, and diagnose Windows systems. One of those utilities is SDelete. *"Using the defragmentation API, SDelete can determine precisely which clusters on a disk are occupied by data belonging to compressed, sparse, and encrypted files."* [50] [51]

This, however, becomes complicated when users use SSDs. SSDs make the process of secure erasure much harder. The TRIM commands can be used; however, SSDs essentially keep a record of the data due to wear-leveling process buffering. This makes SSDs susceptible to many data recovery attacks. [52]

Chapter 6
Discussion and results

This chapter discusses and elaborates on found results of research and implementation.

6.1 Cloud findings

With the research done on the cloud platforms, it would be desirable for Microsoft to release Personal Vault API. That way, users can use Microsoft 2FA services with cloud storage.

As for other cloud providers, any implementation of Personal Vault would seem like the next step in cloud security. Developing a secure folder for users with sensitive documents is, in my opinion, time well invested.

6.2 Two-factor authentication

In terms of 2FA research, OTP tokens were initially focused on for their simplicity in theory and implementation. It was only later discovered that a form of computing power is imperatively needed for the server-side to use OTP properly. Ideas were discussed on how to implement OTP without computing power.

For instance, the cloud storage could hold a part of the key (the second-factor key). The user then must login into the cloud and combine this partial key with the part they have, completing the key for 2FA. This seemed like a satisfactory solution; however, TPM was chosen over the OTP.

6.3 TPM as 2FA

At the beginning of implementation, TCGs documentation on TPM 2.0 was harder to understand as it is made in an abstract manner. The direct implementation of TCGs TSS – TSS.MSR, even though adequately developed following the TCGs standard, introduced yet another new layer of abstraction that needed to be understood. Their implementation, however, is accompanied by a set of basic examples that created a starting ground for the development.

The most significant milestone to overcome was to export the keyhandle correctly in compliance with authorization sessions and the key hierarchy.

Chapter 7
Conclusion

Although the KeePass plugin base is not small, its support for the second factor is lacking. This plugin created a secure way of providing a safe cloud environment, resilient to simple credential loss in combination with secure cloud storing.

First attempts to involve OTP tokens proved unsuccessful due to the fundamental concepts of cloud storage. File systems alone can not provide any computing power for validation. Any other cryptosystems created for a key exchange of sorts resulted in either complication beyond the scopes of this thesis or loss of security altogether. TPM technology was researched and later implemented as a solution to this issue.

As for the objectives of this thesis, the examination of the current state of the KeePass plugin community was conducted and confirmed the lack of 2FA plugins. Analysis of cloud storage providers revealed that only one provider (Onedrive) considers the usage of 2FA within the cloud storage. The rest of examined providers rely solely on password protection. The concept of secure cloud storage was constructed with TPM 2.0 as a second authentication factor, encrypting the database and using one of the cloud providers as storage. This proposal of the solution was later implemented, tested, and documented. Lastly, a threat model was created, revealing possible attack surfaces, and a proposition of the solution to these threats was also made. With the last step, all the objectives of the thesis were successfully fulfilled.

Future work

Although the implementation and threat analysis proved to be successful endeavors, much work needs to be done to meet the potential fully. Mainly in the three following components.

Firstly, implementation of other cloud storage providers like Onedrive, Dropbox, and others. Their APIs are available and should not need any wide changes in the application.

Secondly, implementation of countermeasures described in threat analysis would also be desirable, making the plugin less susceptible to user mistakes and other threats.

And lastly, as stated in previous chapters, the plugin remains to be tested on actual TPM 2.0 rather than simulating it with, even though adequately developed according to TCG standard, simulator binary provided by Microsoft Research Team.

Bibliography

1. REICHL, Dominik. *KeePass* [online]. 2003 [visited on 2022-04-19]. Available from: `https://keepass.info/`.
2. ZUKERMAN, Erez. Tools for the paranoid: 5 free security tools to protect your data [online]. 2013 [visited on 2022-04-21]. Available from: `https://www.pcworld.com/article/456636/tools-for-the-paranoid-5-free-security-tools-to-protect-your-data.html`.
3. REICHLN, Dominik. *KeePass plugins* [online]. 2003 [visited on 2022-04-20]. Available from: `https://keepass.info/plugins.html`.
4. REICHLN, Dominik. *KeePass Auto-typing and its protection* [online]. 2003 [visited on 2022-04-20]. Available from: `https://keepass.info/help/v2/autotype_obfuscation.html`.
5. BÖLTS, Daniel. *KeePass plugin keeanywhere* [online]. 2015 [visited on 2022-04-20]. Available from: `https://keeanywhere.de/`.
6. REICHLN, Dominik. *KeePass plugin keepasssync* [online]. 2012 [visited on 2022-04-20]. Available from: `https://sourceforge.net/projects/keepasssync/`.
7. SHAWN, Casey; MITCH, Capper. *KeePass plugin kpgsync* [online]. 2013 [visited on 2022-04-20]. Available from: `https://sourceforge.net/projects/kp-googlesync/`.
8. DANYAL. *KeePass plugin kpodsync* [online]. 2013 [visited on 2022-04-20]. Available from: `https://keepass.info/plugins.html#kpodsync`.
9. ZOMERS, Koen. *KeePass plugin otpkeyprov* [online]. 2014 [visited on 2022-04-20]. Available from: `https://github.com/KoenZomers/KeePassOneDriveSync`.
10. ROOKIESTYLE. *KeePass plugin kpotp* [online]. 2013 [visited on 2022-04-20]. Available from: `https://keepass.info/plugins.html#kpotp`.
11. TIUUB. *KeePass plugin keeotp* [online]. 2015 [visited on 2022-04-20]. Available from: `https://keepass.info/plugins.html#keeotp`.
12. TIME, Crash; VICTOR, Rezende. *KeePass plugin keetraytotp* [online]. 2010 [visited on 2022-04-20]. Available from: `https://keepass.info/plugins.html#keetraytotp`.
13. STORAGECRAFT. *Cloud security breaches* [online]. [N.d.] [visited on 2022-04-20]. Available from: `https://blog.storagecraft.com/7-infamous-cloud-security-breaches/`.

14. MICROSOFT. *Personal Vault* [online]. 2022 [visited on 2022-04-15]. Available from: https://www.microsoft.com/en-us/microsoft-365/blog/2019/06/25/onedrive-personal-vault-added-security-onedrive-additional-storage/.
15. ONELOGIN. *What's the Difference Between OTP, TOTP and HOTP?* [Online]. 2021 [visited on 2022-05-09]. Available from: https://www.onelogin.com/learn/otp-totp-hotp#:~:text=There%20are%20two%20types%20of%20OTP%3A%20HOTP%20and%20TOTP..
16. KRAWCZYK, H.; BELLARE, M.; CANETTI, R. *HMAC: Keyed-Hashing for Message Authentication* [online]. 1997 [visited on 2022-04-16]. Available from: https://datatracker.ietf.org/doc/html/rfc2104.
17. M'RAIHI, D.; BELLARE, M.; HOORNAERT, F.; NACCACHE, D.; RANEN, O. *HOTP: An HMAC-Based One-Time Password Algorithm* [online]. 2005 [visited on 2022-04-15]. Available from: https://datatracker.ietf.org/doc/html/rfc4226.
18. CLEM, Douglass. *Hash Length Extension Attacks* [online]. 2012 [visited on 2022-05-09]. Available from: https://www.whitehatsec.com/blog/hash-length-extension-attacks/.
19. M'RAIHI, D.; MACHANI, S.; PEI, M.; RYDELL, J. *TOTP: Time-Based One-Time Password Algorithm* [online]. 2011 [visited on 2022-04-21]. Available from: https://datatracker.ietf.org/doc/html/rfc6238.
20. MICROSOFT. *Trusted Platform Module (TPM)* [online]. 2021 [visited on 2022-04-26]. Available from: https://www.microsoft.com/en-us/research/project/the-trusted-platform-module-tpm/.
21. TRUSTEDCOMPUTINGGROUP. *TPM 2.0 A Brief Introduction* [online]. 2022 [visited on 2022-04-22]. Available from: https://www.trustedcomputinggroup.org/wp-content/uploads/TPM-2.0-A-Brief-Introduction.pdf.
22. MICROSOFT. *Microsoft Hello* [online]. 2020 [visited on 2022-04-25]. Available from: https://docs.microsoft.com/en-us/windows/uwp/security/microsoft-passport.
23. CODEGURU. *Windows Cryptography API* [online]. 2007 [visited on 2022-04-24]. Available from: https://www.codeguru.com/windows/windows-cryptography-api-next-generation-cng/#:~:text=The%5C%20Cryptography%5C%20API%5C%3A%5C%20Next%5C%20Generation,part%5C%20of%5C%20the%5C%20National%5C%20Security.
24. MICROSOFT. *Data Protection* [online]. 2021 [visited on 2022-05-07]. Available from: https://docs.microsoft.com/en-us/dotnet/standard/security/how-to-use-data-protection.
25. ARTHUR, Will; KENNETH GOLDMAN, David Challener with. A Practival Guide to TPM 2.0. In: River Edge, NJ: APRESS OPEN, 2015, pp. 21–22. ISBN 978-1-4302-6583-2.
26. ARTHUR, Will; KENNETH GOLDMAN, David Challener with. A Practival Guide to TPM 2.0. In: River Edge, NJ: APRESS OPEN, 2015, p. 99. ISBN 978-1-4302-6583-2.
27. ARTHUR, Will; KENNETH GOLDMAN, David Challener with. A Practival Guide to TPM 2.0. In: River Edge, NJ: APRESS OPEN, 2015, p. 105. ISBN 978-1-4302-6583-2.

28. BOTTOMLEY, James. *Security and Trust* [online]. 2017 [visited on 2022-04-22]. Available from: https://www.hansenpartnership.com/Impress-Slides/LinuxCon-Japan-2018/#/step-85.
29. WININTRO. *Tpm Key Creatrion* [online]. 2015 [visited on 2022-04-22]. Available from: http://winintro.ru/tpmadmin.en/.
30. TOMLINSON, Allan. *Introduction to the TPM* [online]. 2012 [visited on 2022-04-29]. Available from: https://courses.cs.vt.edu/cs5204/fall10-kafura-BB/Papers/TPM/Intro-TPM-2.pdf.
31. BOTTOMLEY, James. *Security and Trust* [online]. 2017 [visited on 2022-04-22]. Available from: https://www.hansenpartnership.com/Impress-Slides/LinuxCon-Japan-2018/#/step-68.
32. MICROSOFT. *Switch TPM object to the strict mode* [online]. 2014 [visited on 2022-04-25]. Available from: https://github.com/microsoft/TSS.MSR/blob/d365231fe799024f8194ba0182b0b7cf3f327dcb/TSS.NET/Samples/Authorization/Program.cs#L189.
33. EBRARY. *Persistence of Keys* [online]. 2014 [visited on 2022-04-25]. Available from: https://ebrary.net/24767/computer_science/persistence_keys.
34. MANDAL, Ritwik. *Random Number Generator (TPM2)* [online]. 2021 [visited on 2022-04-21]. Available from: https://developers.tpm.dev/posts/random-number-generator-tpm2-12528972.
35. REICHL, Dominik. *Plugin Development (2.x)* [online]. 2016 [visited on 2022-04-22]. Available from: https://keepass.info/help/v2_dev/plg_index.html.
36. MICROSOFT. *TPM Simulator* [online]. 2021 [visited on 2022-04-21]. Available from: https://github.com/microsoft/ms-tpm-20-ref/blob/d638536d0fe01acd5e39ffa1bd100b3da82d92c7/TPMCmd/Simulator/src/TPMCmdp.c#L78.
37. MICROSOFT. *Official TPM 2.0 Reference Implementation* [online]. 2021 [visited on 2022-04-25]. Available from: https://github.com/microsoft/ms-tpm-20-ref.
38. MICROSOFT. *TPM Simulator* [online]. 2021 [visited on 2022-04-20]. Available from: https://github.com/microsoft/ms-tpm-20-ref/tree/master/TPMCmd/Simulator/src.
39. MICROSOFT. *ResetAuthLockOut method of the Win32_Tpm class* [online]. 2021 [visited on 2022-04-22]. Available from: https://docs.microsoft.com/en-us/windows/win32/secprov/resetauthlockout-win32-tpm.
40. UCEDAVÉLEZ, Tony. *PASTA Threat Modeling* [online]. 2021 [visited on 2022-04-30]. Available from: https://versprite.com/blog/what-is-pasta-threat-modeling/.
41. TONY, UcedaVelez; MARCO, Morana. *Risk Centric Threat Modeling: Process for Attack Simulation and Threat Analysis*. wiley, 2015. ISBN 978-0-470-50096-5.
42. MICROSOFT. *TPM fundamentals* [online]. 2021 [visited on 2022-04-21]. Available from: https://docs.microsoft.com/en-us/windows/security/information-protection/tpm/tpm-fundamentals.
43. IMPERVA. *Phishing attacks* [online]. 2021 [visited on 2022-04-30]. Available from: https://www.imperva.com/learn/application-security/phishing-attack-scam/#:~:text=What%5C%20is%5C%20a%5C%20phishing%5C%20attack,instant%5C%20message%5C%2C%5C%20or%5C%20text%5C%20message..

44. HUSSEIN, Mohammed. *Visualising the race to build the world's fastest supercomputers* [online]. 2022 [visited on 2022-04-30]. Available from: https://www.aljazeera.com/news/2022/1/14/infographic-visualising-race-build-world-fastest-supercomputers-interactive#:~:text=According%5C%20to%5C%20Top500%5C%2C%5C%20which%5C%20ranks,is%5C%20the%5C%20world's%5C%20fastest%5C%20supercomputer..

45. TOP500. *Japan Captures TOP500 Crown with Arm-Powered Supercomputer* [online]. 2020 [visited on 2022-04-30]. Available from: https://top500.org/news/japan-captures-top500-crown-arm-powered-supercomputer/.

46. ROY. *Can You Crack a KeePass Database if You Forgot Your Password?* [Online]. 2021 [visited on 2022-04-22]. Available from: https://davistechmedia.com/can-you-crack-a-keepass-database-if-you-forgot-your-password/.

47. KENDRA, Cyber. *Worst Apache Log4j RCE Zero day Dropped on Internet* [online]. 2021 [visited on 2022-04-22]. Available from: https://www.cyberkendra.com/2021/12/worst-log4j-rce-zeroday-dropped-on.html.

48. GOODIN, DAN. *NSA-leaking Shadow Brokers just dumped its most damaging release yet* [online]. 2017 [visited on 2022-04-20]. Available from: https://arstechnica.com/information-technology/2017/04/nsa-leaking-shadow-brokers-just-dumped-its-most-damaging-release-yet/.

49. REICHLN, Dominik. *Security Issues* [online]. 2003 [visited on 2022-04-20]. Available from: https://keepass.info/help/kb/sec_issues.html#keefarce.

50. MICROSOFT. *Defragmenting Files* [online]. 2021 [visited on 2022-04-30]. Available from: https://docs.microsoft.com/en-us/windows/win32/fileio/defragmenting-files?redirectedfrom=MSDN#defragmenting_a_file.

51. MICROSOFT. *SDelete v2.04* [online]. 2021 [visited on 2022-05-01]. Available from: https://docs.microsoft.com/en-us/sysinternals/downloads/sdelete.

52. PHILLIPS, GAVIN. *Can SSDs Really Securely Delete Your Data?* [Online]. 2020 [visited on 2022-04-30]. Available from: https://www.makeuseof.com/tag/ssd-secure-delete-data/.

Appendix A

Acronyms

OTP	One-time password
TOTP	Time-based one-time password
HOTP	HMAC-based one-time password
TPM	Trusted Platform Module
HMAC	Hash-based message authentication code
API	Application programming interface
FAPI	Feature API
ESAPI	Enhanced System API
DPAPI	Data Protection API
TCTI	TPM Command Transmission Interface
TSS	TPM2 Software Stack
2FA	Two-factor authentication
CNG	Cryptography API: Next Generation
CSP	Cryptographic Service Provider
GCP	Google Cloud Platform
TSS.MSR	The TPM Software Stack from Microsoft Research
TCG	Trusted Computing Group
NV	Non-volatile
TRNG	True random number generator
DLL	Dynamic-link library
PASTA	The Process for Attack Simulation and Threat Analysis
FLOPS	Floating-point operations per second
RCE	Remote code execution

Contents of enclosed SD Card

- `dll`
  - `KeePassVault.dll` ..... Plugin library
  - `readme-dll.md` ..... Read-me describing installation of the plugin
- `sim`
  - `tss-sim.zip` ..... Zip file containing Simulator binary
- `src`
  - `code` ..... Directory with source code of the project
    - `KeePassVault` ..... Directory with source code for Visual Studio
    - `readme-code.md` ..... Read-me describing project source code
  - `thesis` ..... Directory with source code of the thesis
    - `moisikon-bachelor-thesis.zip` ..... Source code the the thesis
    - `readme-thesis.md` ..... Read-me describing thesis source code
- `text`
  - `moisikon-thesis.pdf` ..... Thesis text in PDF format
- `readme.md` ..... Short guide to contents of enclosed SD Card