
**BRNO UNIVERSITY OF TECHNOLOGY**
**VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ**

**FACULTY OF INFORMATION TECHNOLOGY**
**FAKULTA INFORMAČNÍCH TECHNOLOGIÍ**

**DEPARTMENT OF INFORMATION SYSTEMS**
**ÚSTAV INFORMAČNÍCH SYSTÉMŮ**


# GPG ENCRYPTED WEB PAGES

**ČÁSTI WEBOVÉ STRÁNKY ŠIFROVANÉ POMOCÍ GPG**

**TERM PROJECT**
**SEMESTRÁLNÍ PROJEKT**


**AUTHOR** — **Bc. JIŘÍ MATĚJKA**
**AUTOR PRÁCE**

**SUPERVISOR** — **Ing. LIBOR POLČÁK, Ph.D.**
**VEDOUCÍ PRÁCE**



**BRNO 2020**

**Vysoké učení technické v Brně**
Fakulta informačních technologií

Ústav informačních systémů (UIFS) Akademický rok 2019/2020

# Zadání diplomové práce


Student: **Matějka Jiří, Bc.**
Program: Informační technologie Obor: Bezpečnost informačních technologií
Název: **Části webové stránky šifrované pomocí GPG**
**GPG Encrypted Web Pages**
Kategorie: Web

Zadání:

1. Seznamte se s projektem GnuPG.
2. Nastudujte možnosti technologie WebExensions.
3. Navrhněte rozšíření pro prohlížeč Firefox implementující dešifrování části stránky. Vhodným způsobem navrhněte, jak bude aplikace tyto prvky rozpoznávat. Navrhněte řešení pro interaktivní úpravy stránky dat přijatých pomocí XHR API, Fetch API, či Push API.
4. Rozšíření implementujte alespoň pro systém GNU/Linux.
5. Implementaci otestujte.
6. Zveřejněte vytvořené rozšíření v rámci addons.mozilla.org, zhodnoťte dosažené výsledky a navrhněte možné pokračování projektu.

Literatura:

- Mueller: Security for Web Developers: Using JavaScript, HTML, and CSS. ISBN 978-1-491-92864-6, O'Reilly, 2016.
- The GNU Privacy Guard. The GnuPG Project. Dostupné online https://gnupg.org/.

Při obhajobě semestrální části projektu je požadováno:

- Body 1 až 3 zadání.

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce: **Polčák Libor, Ing., Ph.D.**
Vedoucí ústavu: Kolář Dušan, doc. Dr. Ing.
Datum zadání: 1. listopadu 2019
Datum odevzdání: 20. května 2020
Datum schválení: 21. října 2019

# Abstract

The aim of my term project was to get acquainted with *The GNU Privacy Guard Project* and a web browser extension development process. Then based on acquired knowledge, design a web browser extension for Firefox. The browser extension should be able to detect and decrypt encrypted elements of a web page using GPG. In the case of *The GNU Privacy Guard Project*, I focused on data encryption and decryption process, the Linux command–line application gpg and alternative software implementations of the *OpenPGP* standard that is suitable for developing web browser extensions. Then I described, how are developed web browsers extensions for Firefox and how web extensions can communicate with a native application installed on a user's computer. In conclusion, I defined the software development process of this thesis, described developed prototypes and designed, how the future development will proceed.



# Abstrakt

Cílem mého semestrálního projektu bylo seznámení se s projektem *The GNU Privacy Guard Project* a implementací webových rozšíření. Na základě nabytých znalostí posléze navrhnout rozšíření pro webový prohlížeč Firefox, které bude schopno detekovat a dešifrovat zašifrované části webové stránky s využitím GPG. U projektu *The GNU Privacy Guard Project* jsem se zaměřil na způsob, jakým se šifrují a podepisují data, na Linuxovou aplikaci gpg a na alternativní implementace *OpenPGP* standardu vhodných pro použití při vývoji webových rozšíření. Dále jsem popsal, jak se vyvíjí webová rozšíření pro webový prohlížeč Firefox a možnosti komunikace takového rozšíření s nativní aplikací v počítači. Na závěr jsem definoval postup, jakým způsobem bude probíhat implementace takového rozšíření, popsal dva implementované prototypy a navrhl, jak bude probíhat další vývoj webového rozšíření v rámci navazující diplomové práci.



# Keywords

GPG, GnuPG, OpenPGP, cryptography, WebExensions

# Klíčová slova

GPG, GnuPG, OpenPGP, kryptografie, WebExensions


# Reference


MATĚJKA, Jiří. *GPG Encrypted Web Pages*. Brno, 2020. Term project. Brno University of Technology, Faculty of Information Technology. Supervisor Ing. Libor Polčák, Ph.D.

# GPG Encrypted Web Pages

## Declaration

I declare that I have carried out the Master's thesis independently under supervision of *Ing. Libor Polčák, Ph.D.* from *Brno University of Technology*, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or processional qualification.

.....
Jiří Matějka
July 12, 2020


## Acknowledgements

I would like to express my gratitude to *Ing. Libor Polčák, Ph.D.* from *Brno University of Technology* who suggested me this topic and supported me with useful information and knowledge.

# Contents

# Chapter 1

# Indroduction

The most popular information resource today is undoubtedly the internet. One of its key advantages is data availability. Frequently, data are stored on remote devices (commonly servers) and users can connect to these devices and access data they require. These data can contain private or secret information such as family pictures, passwords, bills or other sensitive content that need to be protected. Access to a server with this kind of sensitive information can be protected by some kind of authorization (for example with a login and a password). But even the most secure kind of authorization is not sufficient enough to secure data from an unauthorized access. For a user, to obtain any type of content on a remote device, data must be transferred. In the case of the internet, data are transferred over multiple devices on which data can be accessed or even modified by a potential attacker (without the knowledge of either side of a communication). The figure 1.1 shows such possible unauthorised data modification.

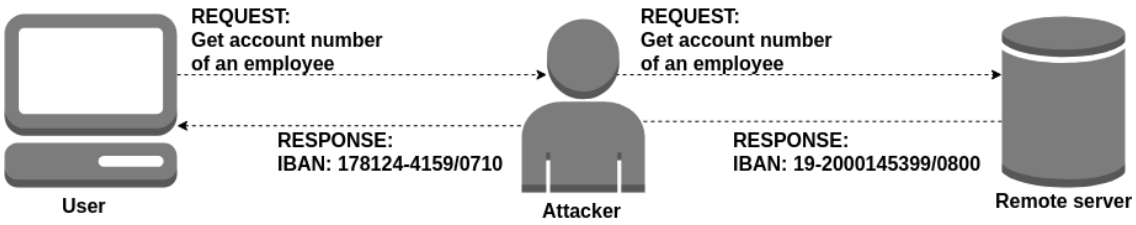


Figure 1.1: Example of unauthorised data access and modification during its transmission.

Usage of cryptography is the most frequently used solution for this problem. Data can be encrypted during the communication or encrypted data can be stored on servers and then transferred with or without further encryptions. This thesis will use the second approach, where data are already encrypted on a remote server. Under the term of data, you can imagine usual web page including not only a text, images or videos but JavaScript and CSS as well. Some elements of this page (or even whole page) can be encrypted using a symmetric cypher and different parts can be encrypted using a different key or some parts can be encrypted using multiple keys. These keys are encrypted with an asymmetric cypher and they are a part of encrypted content as well. The outcome of the thesis will be a web browser extension that will be able

to detect an encrypted content (images, videos, text, etc.) and decrypt it for a user using his available keys. With this encryption/decryption system, users can create web pages where different data can be accessible for different users without the need for authentication on a remote server.

The target platform will be GNU/Linux and browser extensions will be implemented for Firefox web browser. Data will be decrypted with the Linux command line application called gpg, that will be used not only for decryption but for key management as well.

# Chapter 2

# The GNU Privacy Guard Project

This chapter provides information about *The GNU Privacy Guard Project*. The text is divided into the following sections:

1. The GNU Privacy Guard Project,
2. OpenPGP standard,
3. GnuPG for Linux distributions,
4. Alternative Software.

Information provided by sections *The GNU Privacy Guard Project* and *GnuPG for Linux distributions* are a summarization of material published on *The GNU Privacy Guard Project* website [3]. The section dealing with the *OpenPGP standard* is a summarization of a large standard definition by RFC4880 [1].

## 2.1 The GNU Privacy Guard Project

*The GNU Privacy Guard Project*, also known as GnuPG or GPG, is a complete and free implementation of the *OpenPGP* standard as defined by RFC4880 [1]. The GnuPG offers encryption, decryption and signing both data and communication. It features a versatile key management system with access modules for many kinds of public key directories. Not only that the GnuPG is available for both Windows and Linux operating system, but also a wealth of applications and libraries are available.

The Linux implementation of the GnuPG is a command line tool with features for integration with other applications. The Windows version of the GnuPG is Gpg4win with a context menu tool, a crypto manager and an Outlook plugin to send and receive standard PGP/MIME mails.

## 2.2 OpenPGP standard

As mentioned earlier, the GnuPG is the implementation of the *OpenPGP* standard. The text of this section summarizes the *OpenPGP* standard definition provided by RFC4880 [1]. The *OpenPGP* combines symmetric–key encryption and public–key encryption to

provide confidentiality. First of all, the object is encrypted using a symmetric encryption algorithm. It is worth mentioning that each symmetric key is used only once for a single object. For each object, a new key is generated as a random number. This key is bound to the message and transmitted with it. Key is protected by encryption as well – the key is encrypted with the receiver's public key. The sequence is as follows (also described in the figure 2.1):

1. A message is created by the sender.
2. The sending *OpenPGP* generates a random number to be used as a session key for this message only.
3. The generated session key is encrypted using recipient's public key. This encrypted session key starts the message.
4. The sending *OpenPGP* encrypts the message using the session key, which forms the remainder of the message. Note that the message is also usually compressed.
5. The receiving *OpenPGP* decrypts the session key using the recipient's private key.
6. The receiving *OpenPGP* decrypts the message using the session key. If the message was compressed, it will be decompressed.

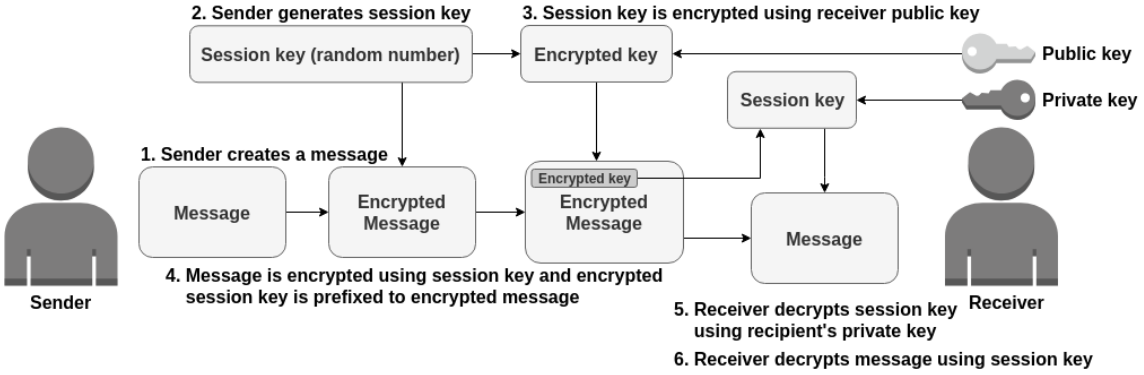


Figure 2.1: Schema of message encryption and decryption

A symmetric key, that is used for message encryption, can be derived from a passphrase (or different kind of shared secret), or a two–stage mechanism similar to the public–key method that was described above, in which a session key itself is encrypted with a symmetric algorithm keyed from a shared secret.

Authentication can be achieved using a digital signature. The digital signature uses a hash code or a message digest algorithm and a public–key signature algorithm. The sequence is as follows (also described in the figure 2.2):

1. A message is created by the sender.
2. The sending software generates a hash code of the message.

3. The sending software generates a signature by encrypting hash code of message using the sender's private key.
4. The binary signature is attached to the message.
5. The receiving software keeps a copy of the message signature.
6. The receiving software generates a new hash code for the received message and verifies it using the message's hash code obtained by decrypting signature with sender's private key.

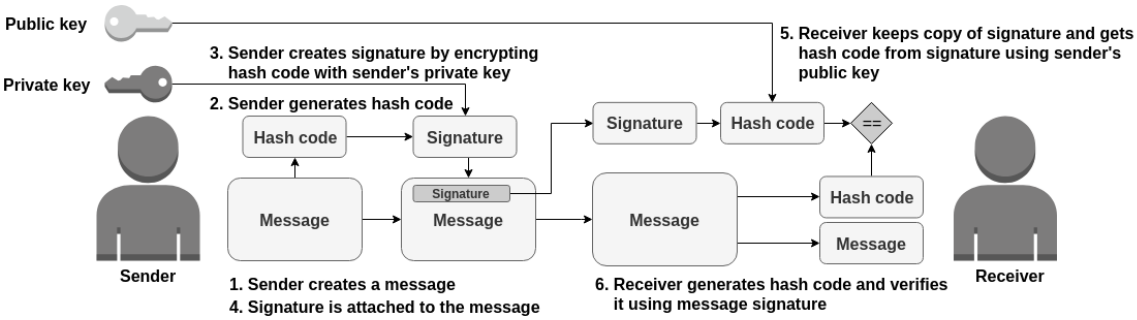


Figure 2.2: Schema of message signature

Both confidentiality and signature services may be applied to the same message. First, a signature is created and attached to the message. Then the message (including the signature) is encrypted using a symmetric session key. At last, the session key is encrypted using the receiver's public–key and prefixed to the encrypted message.

## 2.3 GnuPG for Linux distributions

As described above, there is a Linux application called gpg. It is important to have this application installed and configurated, otherwise implemented software in this thesis will not work correctly, or will not work at all. The reason for this is a fact, that GnuPG is not only used for decryption of encrypted elements but solves problems of the key management as well. Furthermore, the gpg application can be used for encryption of web pages.

Installation of the GnuPG may differ on different operating systems. Some GNU/Linux distribution may already come with directly installable packages. However, it is worth considering installation from the source code because the version of these packages may be old. The list of different GnuPG packages, libraries, required tools, optional software, legacy versions of GnuPG or manual can be found on *The GNU Privacy Guard Project* web page [3].

### 2.3.1 Basic Key Management

Since objects are signed using the receiver's public key and decrypted with the receiver's private key, it is clear that the receiver must have this keypair. The keypair

is not only needed to encrypt the messages but also to sign them. With gpg installed on user's machine, the user can generate its own private and public keys. Basic key management will be described in these steps:

1. Generating a new pair of private and public key
2. Importing and exporting of both private and public keys

### Generating a New Pair of Private and Public Key

To generate keypair with gpg, the user must complete several steps. The first step is to lunch gpg application with the argument *--gen-key*. To prevent problems with private key access permissions, the user can specify a directory, where keys will be stored with argument *--homedir*. Detailed instructions, how to manage user's access to private keys, are described in the article about GPG encrypted credentials by Fabian Lee [4]. Next, the user is asked for a name and an email address. From the name and the email address, gpg creates a User ID. The User ID is something like a name tag for the generated keypair and is also used to identify an owner of a public key.

```
xmatej52@merlin: ~$ gpg --gen-key
Real name: Jiří Matějka
E-mail address: xmatej52@stud.fit.vutbr.cz
You are using the 'utf-8' character set.
You selected this USER-ID:
    "Jiří Matějka <xmatej52@stud.fit.vutbr.cz>"

Change (N)ame, (E)mail, or (O)kay/(Q)uit?
```

Then, the user is asked to enter a passphrase. The passphrase is used to encrypt the private key so it is protected. If the passphrase is compromised, anyone who can access such private key will be able to decrypt owner's received messages and sign his messages as the owner of the private key. After the user enters the passphrase, gpg need to generate a lot of random bytes and ask the user to perform some other actions. After some time, a new keypair is finally generated.

```
Enter passphrase:
We need to generate a lot of random bytes. It is a good idea to perform
some other action (type on the keyboard, move the mouse, utilise the
disks) during the prime generation; this gives the random number
generator a better chance to gain enough entropy.

gpg: key 6C9359504F0C8F81 marked as ultimately trusted
gpg: revocation certificate stored as '/path/to/file'
public and secret key created and signed.

pub   rsa3072 2019-12-30 [SC] [expires: 2021-12-29]
      3252EA3A9A0F105E226BE7BF6C9359504F0C8F81
```

```
uid                      Jiří Matějka <xmatej52@stud.fit.vutbr.cz>
sub   rsa3072 2019-12-30 [E] [expires: 2021-12-29]
```

### Importing and Exporting of Both Private and Public Keys

Export of public keys is an essential feature but in some cases, it is required to export a private key as well. The GnuPG provides options for exporting both of the keys. While the export of a public key is not a problem, the export of a private key may require extra permissions (for example to run gpg via *sudo*), if the logged user does not have permissions to access the private key. To see a list of accessible keys, use command *gpg --list-public-keys* for public keys, and *gpg --list-secret-keys* for private keys.

```
xmatej52@merlin: ~$ sudo gpg --list-secret-keys
/path/to/dir/pubring.kbx
--------------------------------
sec   rsa3072 2019-08-11 [SC] [expires: 2021-08-10]
      346222BC9BEF6635994EF30DC7C029E805D6F30A
uid           [ultimate] test <test@test.cz>
ssb   rsa3072 2019-08-11 [E] [expires: 2021-08-10]

sec   rsa3072 2019-12-30 [SC] [expires: 2021-12-29]
      3252EA3A9A0F105E226BE7BF6C9359504F0C8F81
uid           [ultimate] Jiří Matějka <xmatej52@stud.fit.vutbr.cz>
ssb   rsa3072 2019-12-30 [E] [expires: 2021-12-29]
```

The user ID is shown as *uid* in the list. The user ID can be used to specify which key should be exported. The command *gpg --armor --export uid* will export the public key specified by *uid*. To export the private key, use arguments *--armor* and *--export-secret-keys uid*. Since private keys are encrypted, the user will be asked to enter the passphrase that was entered while generating the keypair.

```
xmatej52@merlin: ~$ gpg --armor --export xmatej52@stud.fit.vutbr.cz
-----BEGIN PGP PUBLIC KEY BLOCK-----

mQGNBF4J/jYBDAC2xruHMLrLn2SwV7n2N5dBkjVO4Jxu//BGjHNqbylcyOOaUT7l
.....
-----END PGP PUBLIC KEY BLOCK-----
xmatej52@merlin: ~$ sudo gpg --armor --export-secret-keys \
xmatej52@stud.fit.vutbr.cz
-----BEGIN PGP PRIVATE KEY BLOCK-----

nQWGBF00HzUBDADApYD+T6kx0Lb0h0MyB1CrWRmrINqHLQqQmzLiaABDLAYqvhgB
.....
-----END PGP PRIVATE KEY BLOCK-----
```

### 2.3.2 Encrypting, Decrypting, Signing, and Verifying Data

The GnuPG was created to provide cryptographic privacy and authentication for data communication. To achieve this goal, the GnuPG implements encryption, decryption and signatures algorithms. Instructions for basic usage of these algorithms will be described in the following steps:

1. Encrypting files and text
2. Decrypting files and text
3. Signing data
4. Verifying signed data

#### Encrypting Files and Text

The GnuPG do not distinguish between the text and binary files and both can be encrypted using the same arguments – *gpg --recipient uid --encrypt [filename]* (*uid* specifies which public key will be used for encryption of generated session key). Regardless of the source file type, the output will be binary content. Though it is not a problem to use the binary representation of encrypted data for images, videos or other files, binary content cannot be used on web pages. To create encrypted text that can be used on web pages, specify the argument *--armor* as well.

```
xmatej52@merlin: ~$ gpg --armor --recipient \
xmatej52@stud.fit.vutbr.cz --encrypt
Hello world
-----BEGIN PGP MESSAGE-----

hQGMA9uyf/0+dawwAQwAgUvSRPDKKtBcVUMOU4Wna/UCaVARIQwlfQUm7hFJa1xp
.....
-----END PGP MESSAGE-----
```

#### Decrypting Files and Text

Data are encrypted with a symmetric algorithm. The session key used for encryption is randomly generated and the session key is encrypted using the receiver's public key. To decrypt the session key, it is necessary to access the receiver's private key. Because of it, decryption may require extra permissions (for example to run gpg via *sudo*), if the logged user does not have permissions to access the private key. To decrypt data, run command *gpg --decrypt [filename]*. If the private key is encrypted, the user will be asked to enter a passphrase.

```
xmatej52@merlin: ~$ sudo gpg --decrypt
-----BEGIN PGP MESSAGE-----

hQGMA9uyf/0+dawwAQwAgUvSRPDKKtBcVUMOU4Wna/UCaVARIQwlfQUm7hFJa1xp
.....
```

```
-----END PGP MESSAGE-----
Enter passphrase:
gpg: encrypted with 3072-bit RSA key, ID DBB27FFD3E75A, created 2019...
      "Jiří Matějka <xmatej52@stud.fit.vutbr.cz>"
Hello world
```

### Signing Data

Data are signed using the sender's private key so it is guaranteed that the sender is the only one, who can create such data (unless the private key is compromised). As a result of using the private key, signing may require extra permissions, if the logged user is not authorized to access the private key. Data can be signed using the command *gpg --sign –local-user uid [filename]* (*uid* specifies which private key will be used for signing). To prevent binary content on output, the argument *--armor* can be specified as well. If the private key is encrypted, the user will be prompted for the passphrase he specified while generating the keypair.

```
xmatej52@merlin: ~$ gpg --armor --recipient \
xmatej52@stud.fit.vutbr.cz --encrypt
Hello world
Enter passphrase:
-----BEGIN PGP MESSAGE-----

owEB9QEK/pANAwAKAT3XKsKGzorWAcsSYgBeEPJkSGVsbG8gd29ybGQKiQHPBAAB
.....
-----END PGP MESSAGE-----
```

### Verifing Signed Data

Given signed data, the user can both verify the signature and recover original data. If user only wishes to check the signature, he can use *--verify* option. To verify the signature and extract the original data, the user can use *--decrypt* argument. To verify the signature, gpg need to access the sender's public key. Without the access, the original data can be still recovered but cannot be verified.

```
xmatej52@merlin: ~$ gpg --verify
-----BEGIN PGP MESSAGE-----

owEB9QEK/pANAwAKAT3XKsKGzorWAcsSYgBeEPJkSGVsbG8gd29ybGQKiQHPBAAB
.....
-----END PGP MESSAGE-----
gpg: Signature made Sat 04 Jan 2020 21:15:37 CET
gpg:            using RSA key 3252EA3A9A0F105E226BE7BF6C9359504F0C8F81
gpg:            issuer "xmatej52@stud.fit.vutbr.cz"
gpg: Good signature from "Jiří Matějka <xmatej52@stud.fit.vutbr.cz>"
[ultimate]
```

```
xmatej52@merlin: ~$ gpg --decrypt
xmatej52@merlin: ~$ gpg --verify
-----BEGIN PGP MESSAGE-----

owEB9QEK/pANAwAKAT3XKsKGzorWAcsSYgBeEPJkSGVsbG8gd29ybGQKiQHPBAAB
.....
-----END PGP MESSAGE-----
Hello world
gpg: Signature made Sat 04 Jan 2020 21:15:37 CET
gpg:            using RSA key 3252EA3A9A0F105E226BE7BF6C9359504F0C8F81
gpg:            issuer "xmatej52@stud.fit.vutbr.cz"
gpg: Good signature from "Jiří Matějka <xmatej52@stud.fit.vutbr.cz>"
```

### 2.3.3 User Interface

KGpg is KDE's application providing a simple interface for the GnuPG. The application can help to set up and manage keys, import and export keys, view key signatures, trust status, expiry dates or encrypt/decrypt text or files. The KGpg is a free and open source software available for Linux and similar operating systems. Since the KGpg provides user interface, KGpg makes it easy to work with the gpg command–line application so the user does not have to remember all the gpg's commands, particularly those for the key management and the data encryption/decryption.

## 2.4 Alternative Software

Although the GnuPG is a very popular software and one of the most used implementations of the *OpenPGP* standard implementation, there are some alternatives for the GnuPG that also implements the *OpenPGP* standard. *OpenPGP.js* is one of such alternatives and was even used in this thesis for a prototype development.

### OpenPGP.js

The *OpenPGP.js* is a project that aims to provide an open source *OpenPGP* JavaScript library so it can be used on most of the devices. While many other implementations of *OpenPGP* standard are aimed at using native code, the *OpenPGP.js* is meant to bypass this requirement so people are not forced to install gpg on their machines in order to use the library. The idea behind *OpenPGP.js* is to implement all the needed *OpenPGP* functionality in JavaScript library that can be reused in other projects that provides web browser extensions or server applications. The information provided about the *OpenPGP.js* is gained from *OpenPGP.js* project webpage [12].

The *OpenPGP.js* library was used for the implementation of the first prototype (Section 4.1). The implemented prototype was able to encrypt and decrypt elements using hardcoded private and public keys in the source code. Accessing the user's public and private keys was a serious problem. When a background script and a native applica-

tion were developed, this library was no longer necessary and was replaced with gpg (using gpg also solved the problem with private and public key access).

# Chapter 3

# Browser Extentions

This chapter summarizes basic knowledge about browser extensions that are needed to understand the software development process of the thesis. The emphasis will be on Firefox browser extensions because it is the target browser for the extension. The chapter is divided into the following sections:

1. WebExtensions API
2. Structure of a Firefox Extension
3. Native Applications

The *WebExtensions API* section provides an introduction into the browser extension development. The content of this section is a summarization of information provided by [2] and Firefox [6, 11] web pages. The following section, *Structure of a Firefox Extension*, describes the content of a browser extension's manifest file and the text of the section, and its subsections, summarizes the Firefox documentation [5, 8, 7, 10]. The last section of the chapter contains a brief description of native applications, and the content of the section is based on the Firefox documentation [9] as well.

## 3.1 WebExtensions API

The browser extensions are software programs that extend or modify the capabilities of a browser. They enable users to tailor web browser functionality and behaviour to individual needs or preferences. They are built on web technologies such as HTML, JavaScript, and CSS. The browser extension developed in this thesis is built using the *WebExtensions API*.

Extensions for the Firefox browser are built using the *WebExtensions API*. It is a cross–browser system for developing extensions. To a large extent, the system of the *WebExtensions API* is compatible with the extension API supported by Google Chrome, Opera and the W3C Draft Community Group.

Extensions cat take advantage of the same web APIs as JavaScript on a web page, but they can access to its own set of JavaScript APIs. Thanks to this, extensions can do a lot more than a developer can with code on a web page.

An extension must serve a single purpose that is narrowly defined and easy to understand. A single extension can include multiple components and a range of functionality, as long as everything contributes towards the defined purpose.

## 3.2 Structure of a Firefox Extension

An extension consists of a collection of files, libraries, packages and installation, yet there is the only file that must be present in every extension – *manifest.json*. The manifest contains basic metadata about the extension (name, version or the permissions it requires). It also provides a list of other files that are included in the extension.

The manifest can also include pointers to several other types of files:

1. Icons
2. Content scripts
3. Background scripts
4. Sidebars, popups, and option pages
5. Web–accessible resources

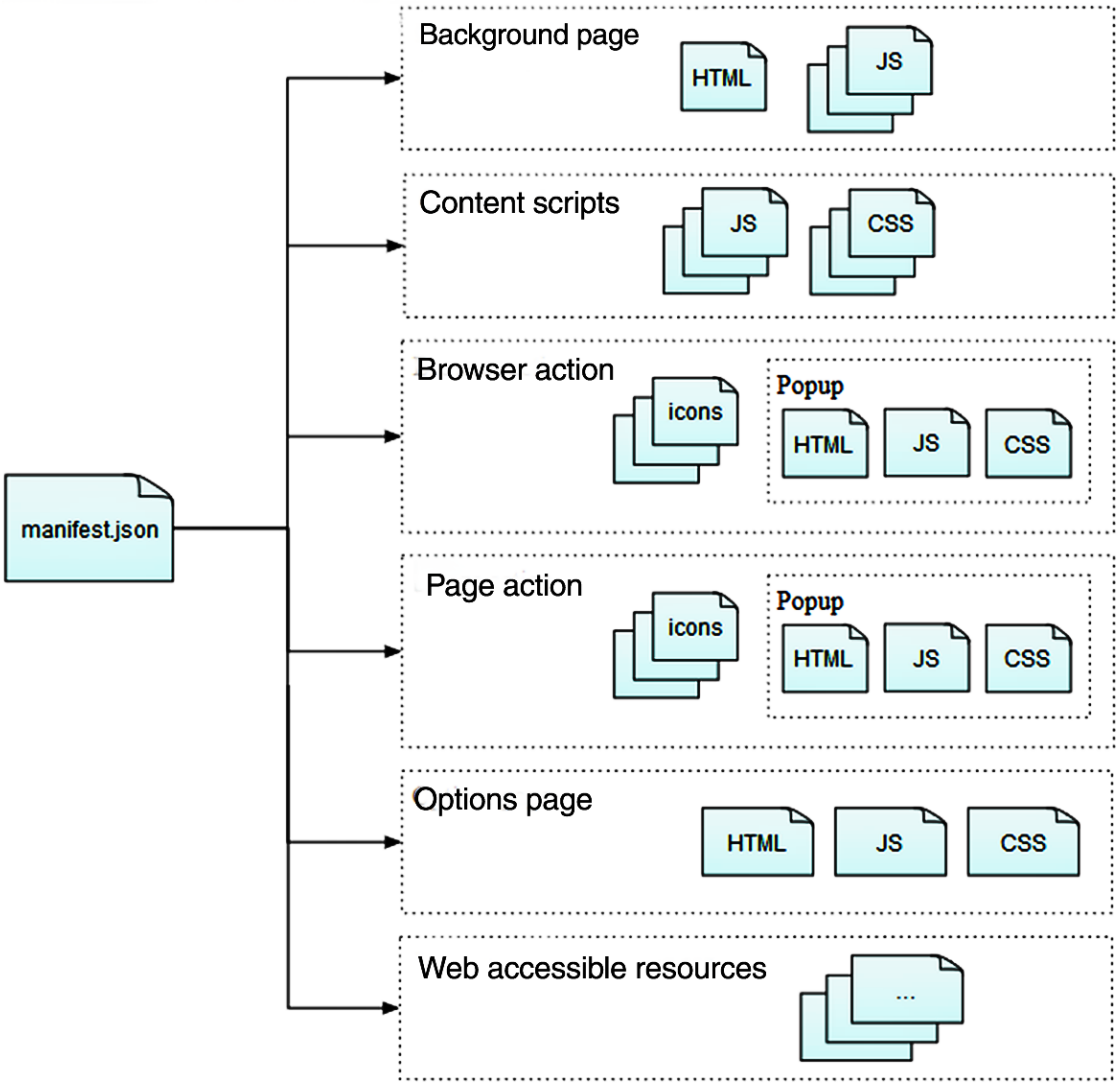


Figure 3.1: Anatomy of a broswer extension [5].

### 3.2.1 Icons

Icons are used to represent extension in components (for example in Add-ons Manager). Icons are represented in the manifest file as an object that consists of key–value pairs of image size (in px) and image path relative to the root directory of the extension.

It is recommended by Mozilla developers to supply at least the main extension icon, ideally in $48 \times 48$ px in size. However, it is possible to provide icons of any size and Firefox will attempt to select the best icon to display in different situations. Firefox also promises to consider screen resolution when choosing an icon so it is good practice to provide double–sized versions of all icons in order to deliver the best visual experience.

Firefox also supports the usage of icons in SVG format but it is necessary to specify *viewBox*. Nevertheless, a developer can use one file in the SVG format as an icon, he needs to specify various sizes of the icon in the manifest.

### 3.2.2 Content Scripts

Content scripts are used to access and manipulate a content of web pages. Content scripts are executed in the context of a particular web page and can access DOM APIs the same way, just like the scripts loaded by the web page.

Content scripts can only access a relatively small subset of the WebExtension API, but they can communicate with background scripts (Section 3.2.3) using a messaging system, thereby indirectly access the WebExtensions API.

#### Content Script Enviroment

As mentioned earlier, content scripts can access DOM APIs, therefore they are able to modify and to access DOM, just like normal page scripts can. Furthermore, they can detect and see any changes that were made to the DOM by page scripts. However, it is worth mentioning that content script does not share JavaScript variables defined by page scripts, and if page script redefines a built–in DOM property, the content script will be able only to access the original version of the modified property. The same is true in reverse – page scripts cannot see JavaScript property changes made by the content script.

Another feature content scrips have is the possibility of making requests using *window.XMLHttpRequest* and *window.fetch()* APIs. Content script shares the same cross–domain privileges with the rest of the extension. In case the extension has requested cross–domain access for domain using the permissions key in the manifest file, then its content script can get access to the same domain as well.

The possibility of communication with background scripts (Section 3.2.3) is another important feature of content scripts. There are two basic types of communication between content and background scripts – one–of messages and longer–lived connection. The One–off messaging is useful when none or only one response is expected to a message, and when only a small number of scripts listen to receive messages. On the other hand, it is recommended to use connection-based messaging, where multiple messages are exchanged, when the extension needs information about task progress, needs to be notified if a task is interrupted, or may want to interrupt a task that was initiated using messaging system.

### 3.2.3 Background Scripts

Background scripts are for maintaining long–term state or for performing long–term operations independently of the lifetime of any particular web page or browser window. Background scripts are loaded with extension installation and stay loaded until the extension is disabled or uninstalled. Developers can also profit from *WebExtension APIs* in the background scripts, although it is necessary to specify any needed permissions in the manifest file.

A background page is special page that provides a *window* global, along with all the standard DOM APIs provided by it. Since background scripts run in a context of the background page, they can access the *window* global.

The background scripts does not have direct access to web pages. But they are alowed to load content scripts into web pages and they can communicate with these scripts.

### 3.2.4 Sidebars, Popups, and Option Pages

Extensions can include various user interface components. Those components are defined with an HTML document. Components can be:

1. Sidebar – a pane displayed at the side of the browser window, next to the web page
2. Popup – a dialogue that is associated with a toolbar button or address bar button
3. Option page – a page that enables preferences that user can change

Every component is defined with its own HTML document and a point to it using specific property in the manifest file. Such HTML document can include both CSS and JavaScript files. JavaScript can use all privileged *WebExtension APIs* as background scripts (Section 3.2.3) and they can even directly access variables in the background page using a special method *runtime.getBackgroundPage()*.

### 3.2.5 Web–Accessible Resources

Sometimes it is needed to provide resources (images, HTML, CSS, JavaScript, etc.) and make them available to web pages. This problem is solved with web–accessible resources. Resources, which are made web–accessible, can be referenced by page scripts and content scripts using special URI scheme.

Note that if a page is made web–accessible, any website may link or redirect to that page. Therefore, web–accessible pages should treat any input as if it came from an untrusted source.

### 3.2.6 Extension Pages

Extension pages are HTML documents in broswer extention that are not attached to some predefined user interface components. Extension pages do not have entry in the manifest file unlike sidebars, popups or option pages. However, they also get access to the same privileged WebExtension APIs as the background script.

## 3.3 Native Applications

A native application is installed using the underlying operating system's installation machinery. A browser can communicate with the native application via a native messaging mechanism. This enables an extension to exchange messages with the native

application installed on the user's computer. Thanks to this mechanism, native applications can provide services to extensions without needing to be reachable over the web. Another feature provided by the native application is that they can enable to access resources that are not accessible through WebExtension APIs, such as hardware.

Along with the native application itself, the developer needs to provide another manifest file called *host manifest* or *app manifest* and install it in a defined location on the user's computer. The host manifest file describes how the browser can connect to the native application. Note that the extension must also request permission *nativeMessaging* in its manifest file and the native application must include the extension's ID in the *allowed_extensions* field of the host manifest.

Once both extension manifest and host manifest are correctly set, the extension can exchange messages in JSON format with the native application. The native application receives messages on its standard input and sends them using its standard output. The extension uses a set of functions of the runtime API.

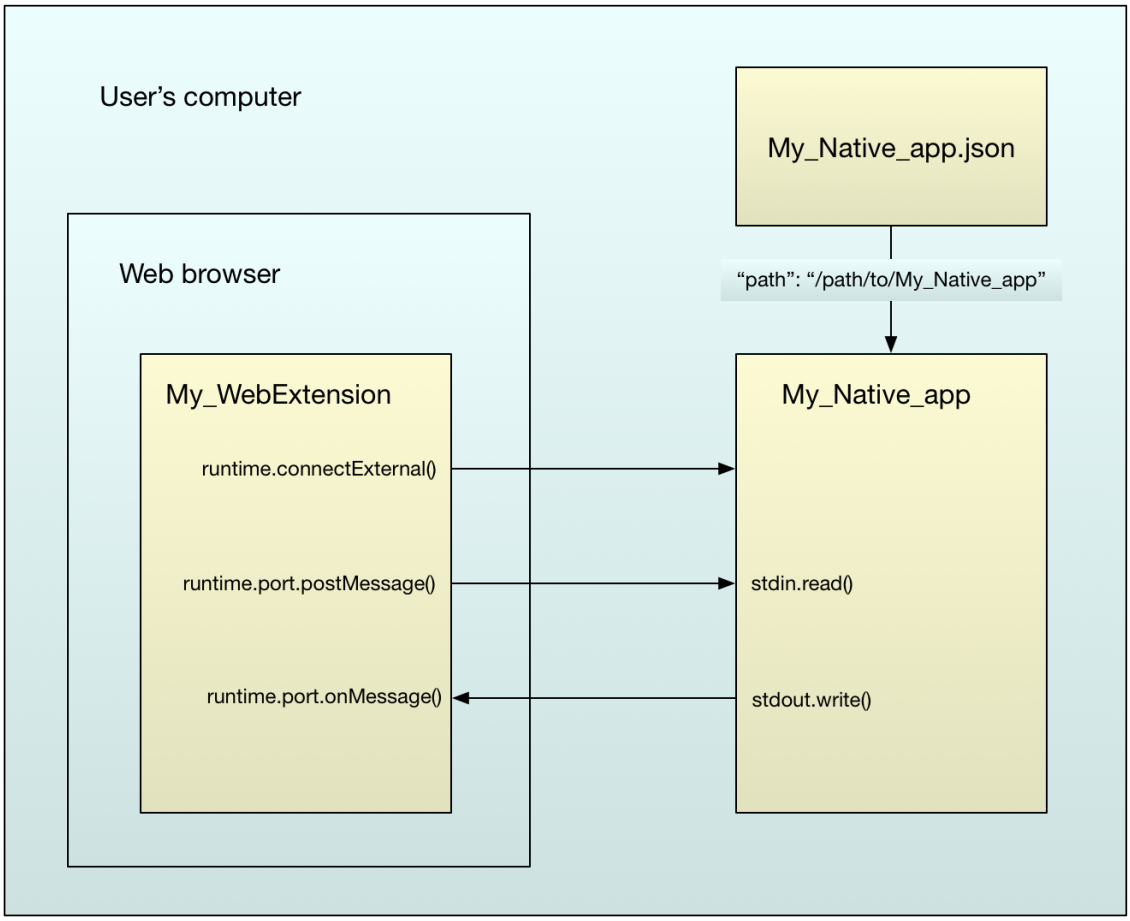


Figure 3.2: Native messaging scheme [9].

# Chapter 4

# Iterative Development

The goal of this thesis is to implement a browser extension that will detect encrypted elements on a web page and decrypt them. Developing a software is a complex and demanding process and it is not easy to deliver a project that correspondents with stakeholder's expectation. To fulfil all expectation, the bowser extention will be developed in several iterations.

Each iteration will consist of design, development, testing and review. The iteration will be usually 2 – 3 weeks long and will result in a prototype. The prototype will be then disgusted with thesis supervisor – Ing. Libor Polčák, Ph.D.

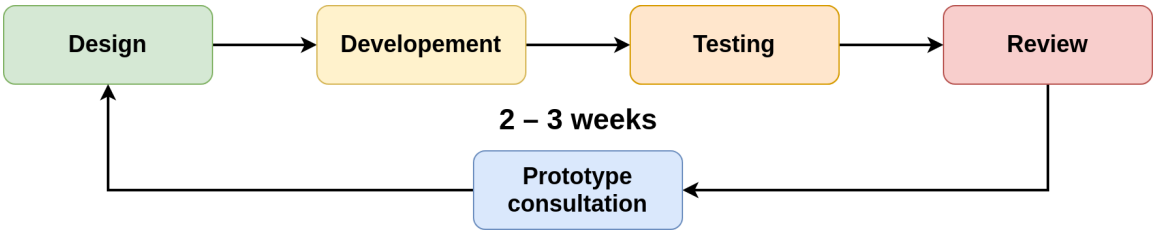


Figure 4.1: Software development model.

Each iteration will be described in the thesis from various reasons. During prototype development, new problems that were missed in the software design can occur. Also, some of the used technologies and methods can lead to a dead–end or lose their purpose in the future prototypes or the final product. Such methods or technologies might not be important for the final product of the thesis, but can be important for some readers.

## 4.1 OpenPGP.js Prototype

The purpose of the first prototype was to learn how to develop a browser extension. The goal was to implement a simple browser extension that will be able to decrypt images using a hardcoded private key. Ideal library for this task was *OpenPGP.js* (Section 2.4) and since the Firefox browser had troubles to load the *OpenPGP.js* library, this prototype was implemented for the Google Chrome browser.

### 4.1.1 Design

No background script or native application was needed to develop so new script – called *gnupg_decryptor.js*, and *OpenPGP.js* library could be loaded as content scripts. The designated schema of the web extention can be seen in the figure 4.2. Script *gnupg_decryptor.js* will detect encrypted image by file suffix in its source URL, downloads it, and tries to decrypt it using *OpenPGP.js* library. Once the content og the image is decrypted, *gnupg_decryptor.js* will create a new URL pointing to the decrypted image and replace the source URL of *img* element on the web page thus displaying the decrypted image to a user. The icon that represents the extension was taken from *The GNU Privacy Guard Project* web page [3].

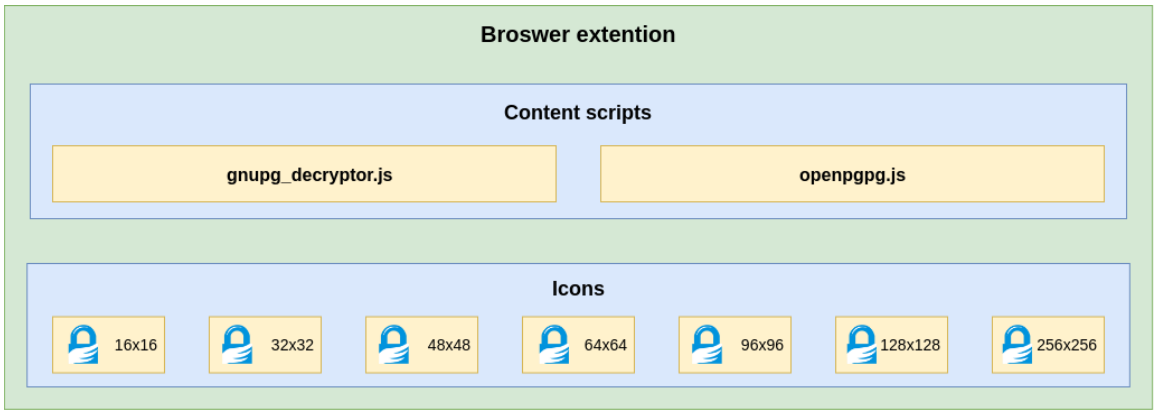


Figure 4.2: Structure of the *OpenPGP.js* prototype.

### 4.1.2 Implementation

Since some parts of this prototype will serve as a foundation stone for next development, special care was taken for reusability of useful functions as well as for preparing testing web pages, a different set of encrypted data and also to prepare special scenarios, specifically another web extension that changes DOM of a web page. Simplified schema of the implemented prototype can be seen on the figure 4.3 (Note that the schema does not show communication between objects, as usually a sequence diagram does, but it shows communication between the content scripts).

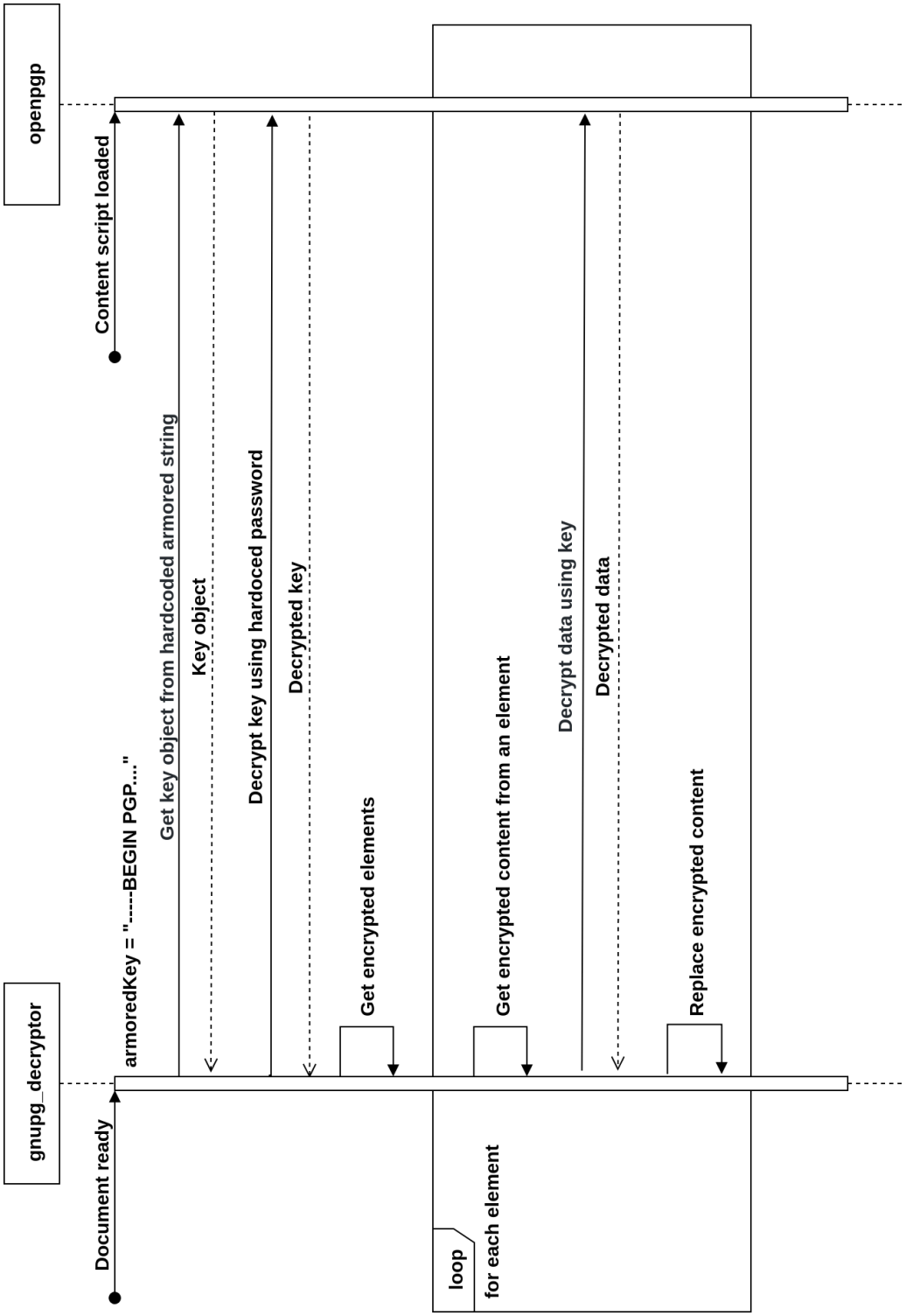


Figure 4.3: Communication between the content scripts of *OpenPGP.js* prototype.

### 4.1.3 Review

The implemented prototype was able to detect images on a web page by their source URL and then decrypt them using the *OpenPGP.js* library. The Prototype was able to use only hardcoded keys and could not access user's keyring on his computer. The prototype did not need any background scripts or native application. The only browser, where was implemented extention operational, was Google Chrome.

Some functions implemented in this prototype will be included in future prototypes and maybe in a final product of the thesis. The main benefits from the first iteration were implemented test pages, encrypted images, generated keypairs and gained experience with web extension development.

## 4.2 GnuPG_Decryptor Prototype

Hardcoded keys were one of the biggest problems of the previous prototype. A new prototype should have access to users keyring. In order to access it, it will be necessary to implement a native application that will be able to communicate with the gpg application. With the native application, the *OpenPGP.js* library will no longer be necessary and all decryption tasks will handle gpg itself.

### 4.2.1 Design

Content script (*gnupg_decryptor.js*) will start the communication with a background script. The background script will serve only as an intermediator and will resend messages from the content script to a native application. The native application will be implemented in Python language and will communicate with gpg using the *subprocess* library. Since a user must provide passphrase to a private key, the native application will have a user interface as well. The user interface will be implemented using the *tkinter* library. All used Python libraries are showned on the figure 4.4.

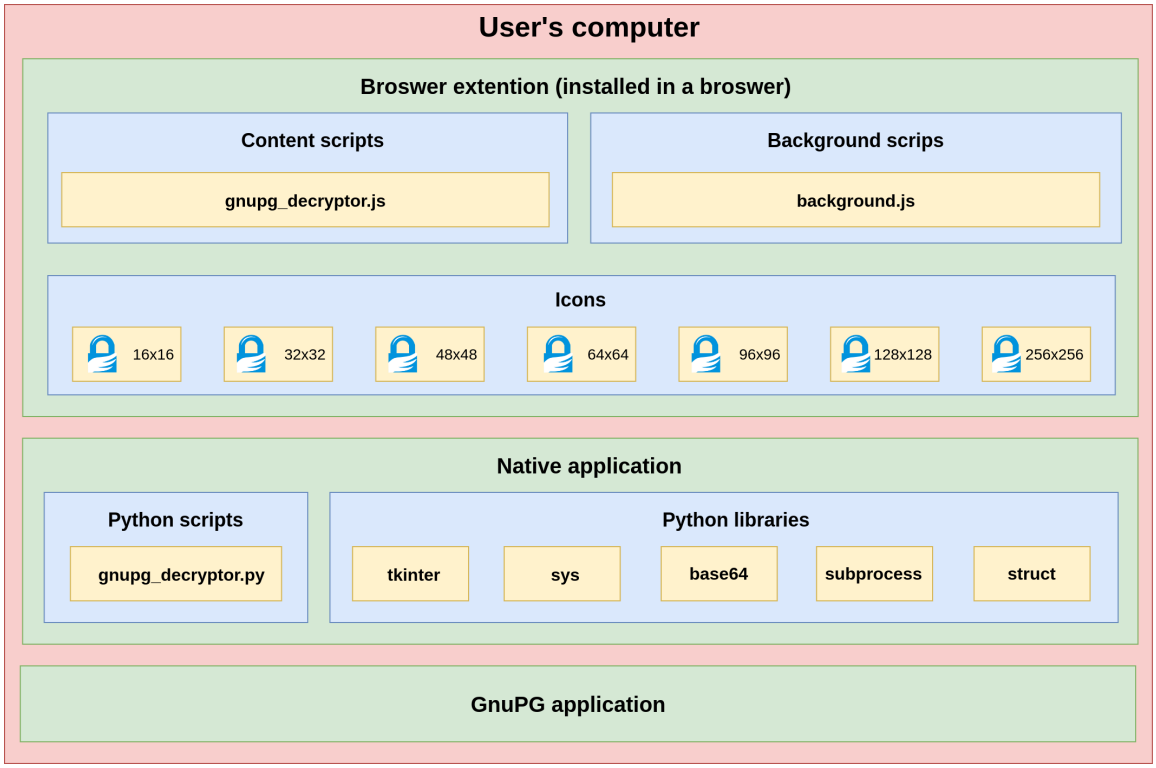


Figure 4.4: Anatomy of the GnuPG_Decryptor prototype.

Decoded content will be sent to the background script and it will resend it to the content script. The content script will then replace encrypted data with decrypted data the same way as in *OpenPGP.js* prototype (Section 4.1).

Messages must be in JSON format. All messages will have a mandatory field *type* that will describe the type of message. In this prototype, supported types will be *decryptResponse, decryptRequest, and debug*. A message with type *decryptRequest* will have mandatory fields *messageId* containing the ID of an encrypted element on a web page, *data* containing encrypted data, and *encoding* specifying which encoding was used to encode the content of field *data*. A message of type *decryptResponse* will have mandatory fields *success* specifing if the decryption was successful, *data* containing base64 string with decrypted content (or an empty string if failed), *message* containing an error message if decryption failed (or an empty string on success), and *messageId* containing the ID of *decryptRequest* message to which the response relates.

### 4.2.2 Implementation

Purpose of this prototype was the implementation of a messaging system that will be used in future development. The messaging system simplifies debugging of the native application by supporting debug messages and the messaging system is extensible and reusable although the messaging system is still incomplete.

The native application was implemented as simple as possible. Application has two dialogues to enter the passwords – password for sudo to access user's keyring and

for a passphrase to decrypt the private key. The application does support decryption with using only one encrypted key and does not resolve errors reported by sudo or gpg itself. A simplified schema of the implemented prototype can be seen on the figure 4.5 (Note that schema does not show communication between objects, as usually a sequence diagram does, but it shows communication between the content script, the background script, the native application and the GnuPG application).

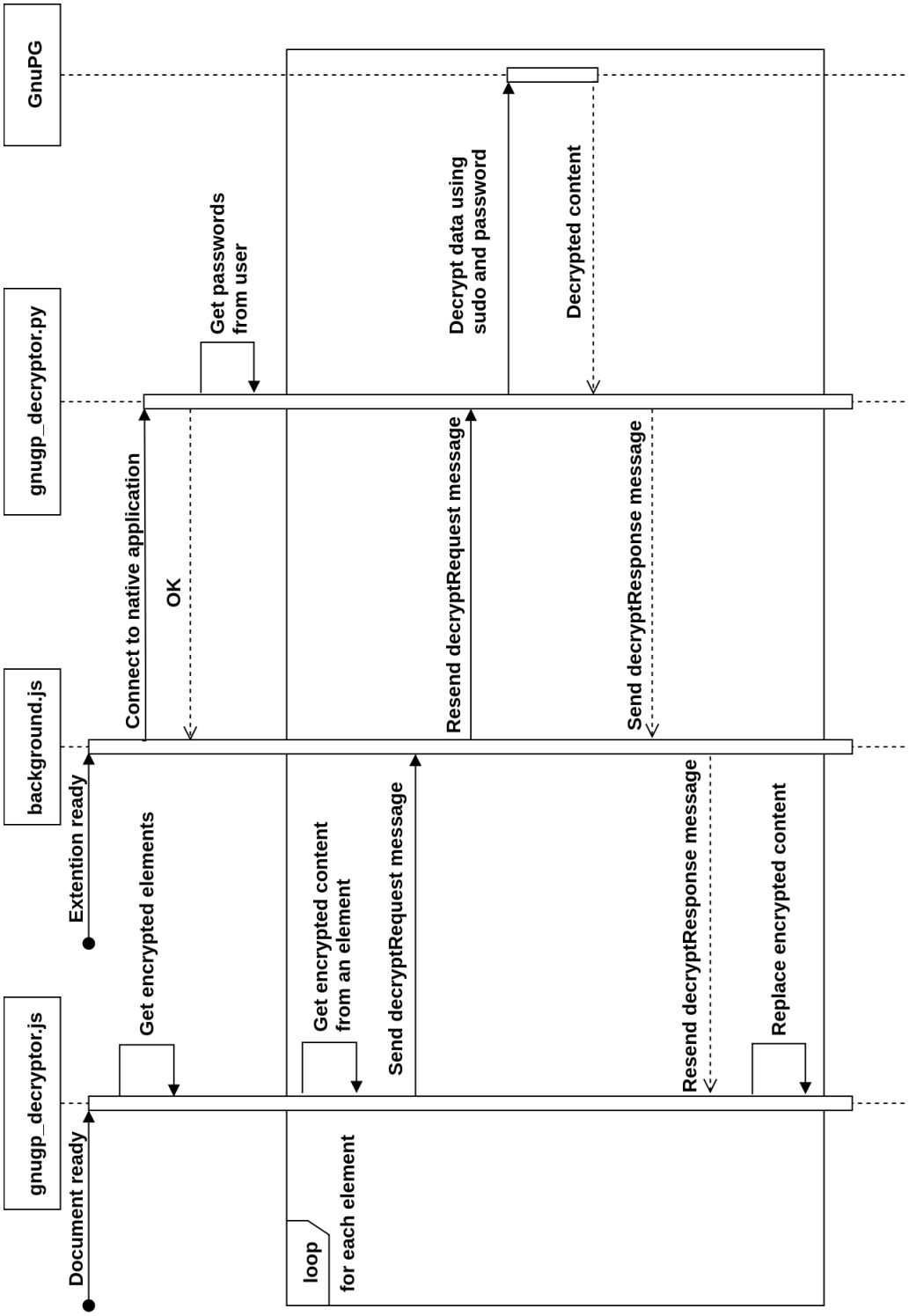


Figure 4.5: Communication between the web extension, the native application, and the GnuPG application.

### 4.2.3 Review

The implemented prototype can access to a certain extent user's keyring. Using the messaging system, the extension is able to exchange data between the gpg application installed on a user's computer and a web browser thus enabling usage of gpg for decryption. Through messaging system is incomplete, the GnuPG_Decryptor prototype offers sufficient basis for future development.

Future prototypes must complete messaging system – there is missing functionality dealing with the limited size of messages and maximum size of exchanged content. The user interface provided by the native application is also insufficient and must be improved in order to use multiple keys for decryption and do not require sudo password, if not necessary. The native application also needs to deal with errors occurs during the data decryption and somehow informs the user about the error.

## 4.3 Large Content Support

### 4.3.1 Design

### 4.3.2 Implementation

### 4.3.3 Review

## 4.4 Interactive Changes Support

### 4.4.1 Design

### 4.4.2 Implementation

### 4.4.3 Review

## 4.5 Graphic User Interface

### 4.5.1 Design

### 4.5.2 Implementation

### 4.5.3 Review

## 4.6 Final Iteration

### 4.6.1 Design

### 4.6.2 Implementation

### 4.6.3 Review

# Chapter 5

# Testing the Browser Extension

## 5.1 Basic Functions

### 5.1.1 Element Detection

### 5.1.2 Basic Decryption

### 5.1.3 Large Files

### 5.1.4 Interactive Changes

## 5.2 Advanced Functions

### 5.2.1 Duplicate Files

### 5.2.2 Multiple Receipments

### 5.2.3 Recursive Decryption

## 5.3 Performance

### 5.3.1 Pages with Encrypted Elements

### 5.3.2 Pages without Encrypted Elements

# Chapter 6

# Current State of Development

## 6.1 Browser Support

## 6.2 Future Development

# Chapter 7

# Conclusion

# Bibliography


[1] Atkins, D., Callas, J., Donnerhacke, L., Finney, H., Shaw, D. et al. *OpenPGP Message Format* [online]. IHTFP Consulting, Inc., november 2007 [cit. 2019-12-26]. Available at: https://www.ietf.org/rfc/rfc4880.txt.

[2] developer.chrome.com. *What are extensions?* [online]. chrome [cit. 2020-01-05]. Available at: https://developer.chrome.com/extensions.

[3] Koch, W., Ellmenreich, N., Ashley, M., Cappelletti, L., Shaw, D. et al. *The GNU Privacy Guard* [online]. The GnuPG Project, december 2019 [cit. 2019-12-25]. Available at: https://gnupg.org/.

[4] Lee, F. *Linux: Using GPG encrypted credentials for enhanced security* [online]. Fabian Lee, october 2018 [cit. 2020-01-04]. Available at: https://fabianlee.org/2018/10/30/linux-using-gpg-encrypted-credentials-for-enhanced-security/.

[5] Mozilla Contributors. *Anatomy of an extension* [online]. Mozilla, september 2019 [cit. 2020-01-05]. Available at: https://developer.mozilla.org/en-US/docs/Mozilla/Add-ons/WebExtensions/Anatomy_of_a_WebExtension.

[6] Mozilla Contributors. *Browser Extensions* [online]. Mozilla, october 2019 [cit. 2020-01-05]. Available at: https://developer.mozilla.org/en-US/docs/Mozilla/Add-ons/WebExtensions.

[7] Mozilla Contributors. *Content scripts* [online]. Mozilla, november 2019 [cit. 2020-01-05]. Available at: https://developer.mozilla.org/en-US/docs/Mozilla/Add-ons/WebExtensions/Content_scripts.

[8] Mozilla Contributors. *Icons* [online]. Mozilla, march 2019 [cit. 2020-01-05]. Available at: https://developer.mozilla.org/en-US/docs/Mozilla/Add-ons/WebExtensions/manifest.json/icons.

[9] Mozilla Contributors. *Native messaging* [online]. Mozilla, december 2019 [cit. 2020-01-06]. Available at: https://developer.mozilla.org/en-US/docs/Mozilla/Add-ons/WebExtensions/Native_messaging.

[10] Mozilla Contributors. *Web_accessible_resources* [online]. Mozilla, july 2019 [cit. 2020-01-05]. Available at: https://developer.mozilla.org/en-US/docs/Mozilla/Add-ons/WebExtensions/manifest.json/web_accessible_resources.

[11] Mozilla Contributors. *What are extensions?* [online]. Mozilla, november 2019 [cit. 2020-01-05]. Available at: https://developer.mozilla.org/en-US/docs/Mozilla/Add-ons/WebExtensions/What_are_WebExtensions.

[12] openpgpjs.org. *OpenPGP.js* [online]. ProtonMail, december 2019 [cit. 2019-12-29]. Available at: https://openpgpjs.org/.