Hochschule für Technik und Wirtschaft Berlin

University of Applied Sciences

BACHELORARBEIT

FACHBEREICH 4: INTERNATIONALE MEDIENINFORMATIK

Thunderbird: One Time Password

Student
Esteban LICEA
Matr. Nr. 536206

Primary Mentor
Prof. Dr. Debora WEBER-WULFF

Secondary Mentor
Prof. Dr Kai Uwe BARTHEL

June 20, 2022

0.1 Abstract

The developer describes the steps in researching and developing a Thunderbird Add-on that offers E2EE, without the need for key changes, i.e. without PGP. The developed software will allow one user to exchange a keyword/password with another user, encipher a message with that keyword/password, and the other user will be able to decipher to message with that password.

Contents

Chapter 1

Introduction

The digital age has fully absorbed our societies. We do everything in some form or another of digital media: create art, science, communicate, create and share memories, play games, and write thesis reports with our computers. There is basically no limit to what people do with their computers.

Proportional to this growth, the internet's influence on our lives has also ballooned. Our activities have been pushed more and more online, onto the cloud. Originally, few bothered to think about privacy. Most damaging, perhaps, was the erroneous expectation of private communication. Edward Snowden's revelations about the "Five Eyes" intelligence alliance, and cooperation in the collection of all online communication, social media, and phone data. No online communication has been considered safe ever since.

1.1 Problem

Mozilla has tried to support end-to-end encryption (E2EE? for a long time, it has been faced with a major obstacles:

- Setting the PGP add-on Enigmail was too technical
- Generating keys was too technical
- Even if conditions 1. & 2. were fulfilled, it was especially uncommon that anyone else you would want to converse with would have gone through the trouble to setup a client or keys for themselves

- Mozilla is in the process of using OpenPGP build-in to the client, but that also has problems, most obviously, you again need new keys (granted easier to setup this time)
- and, again, both people must have generated keys (again

Thus, the problem: How can Alice send an encrypted email to someone that does not have any type of public key available?

1.2 Context

While PGP has existed for years, it is predicated on the exchange of public keys. In clear text, there is a technical requirement to create and exchange keys, and installation of any additional required client software that most average users do not have the patience to complete. Originally, Thunderbird relied on an add-on, Enigmail, to create, manage, and exchange keys.

Starting with Thunderbird 78, Mozilla implemented OpenPGP as part of it's core client software, and dropped support for all add-ons not using MailExtensions (which includes Enigmail). However, the feature is disabled by default, and is still considered a work in progress. All other add-ons found on Thunderbird's extensions page or searching through Github were considered to be in a testing or experimental phase.

1.3 My solution to the problem

This project will implement of an Email Add-on that will allow end-to-end encrypted (E2EE) communication. More specifically, it will focus on the Mozilla Thunderbird client, for the simple fact that I have personally used it for over ten years, it's free, open-source, and cross platform. While I grant that not everyone uses Thunderbird, at least there should be no shortage of users, and theoretically anyone can get it easily, for free.

Ultimately, this project aims to offer a simple, albeit *not* perfect solution for those interested in privacy, that don't have the technical expertise to engage in key creation, exchanges or have zero knowledge about encryption. The will demonstrate the advantages and disadvantages

of various implementations strategies, and implement a solution that offers, hopefully, a viable encryption option that will fulfill some use cases.

1.4 Methods applied

The methods and tools used to solve this research inquiry will include:

1. Literature either in the form of online or paper publications, i.e. books
2. Online learning resources
3. Thunderbird and JS Encryption APIs
4. Guidance from Mentors
5. Visual Studio Code for code production
6. Github for Source Code and Thesis code management
7. Latex for writing the Thesis
8. Jira for project management, i.e. Kanban board, sprints, and road maps

After the research has been completed, all coding will proceed using a test driven development approach. Thunderbird Add-ons are based on MailExtension technology, which are created using the follow standard languages:

1. HTML
2. CSS
3. Javascript

Chapter 2

Cryptography

2.1 Algorithm selection overview

2.1.1 Symmetric key encryption

Selecting an algorithm, among so many, was pretty straightforward given my use case, but I wanted to show my thought processes. Firstly, there are two fundamental paths for selecting an encryption algorithm. The selection between *asymmetric* and *symmetric* key encryption is the initial decision.

1. Asymmetric-key cryptography: A public and private key are created by both people wanting to exchange encrypted emails. This is the most secure and most commonly implementated encryption available, popularly known as "public-key encryption." Examples encryption key algorithms used include RSA and Diffe-Hellman-Merkle. There are challenges though:[Shirey, 2007]

   (a) Both parties need to create their own keys

   (b) Keys need to be exchanged, i.e. a person has to be acute enough to search for the other person's public key – assuming one even exists

   (c) Additional client software is also often required

2. Symmetric-key Encryption: use the same key for both encryption and decryption[Delfs and Knebl, 2002, p. 155]

   (a) The primary drawback is that both parties will need to exchange that key, often times in the form of a password.

The goal of this project is *ease of use* (at the cost of security), so our choice is clear: symmetric-key encryption.

2.1.2 Block vs. Stream cipher encryption

Next, we need to decide between a block cipher or a stream cipher. As Bruce Schneier defines the two in his book ″Applied Cryptography: Protocols, Algorithms in C″ as:

> There are two basic types of symmetric algorithms: block ciphers and stream ciphers. Block ciphers operate on blocks of plaintext and ciphertext—usually of 64 bits but sometimes longer. Stream ciphers operate on streams of plaintext and ciphertext one bit or byte (sometimes even one 32-bit word) at a time. With a block cipher, the same plaintext block will always encrypt to the same ciphertext block, using the same key. With a stream cipher, the same plaintext bit or byte will encrypt to a different bit or byte every time it is encrypted.[Schneier, 2015, p. 12]

The advantages of a stream ciphers: [1]

- bit or byte at a time encryption
- speed of encryption/decryption

are more appropriate for hardware implementations.

According to Bruce Schneier, block ciphers are more suitable for software implementation as they are easier to implement, avoid time-consuming bit manipulations, and operate on computer sized blocks. [Schneier, 2015, p. 172]

2.1.3 Block cipher selection

There are many block ciphers to choose from, these are just some of the most popular: [Nirula, 2022]

[1]https://crashtest-security.com/block-cipher-vs-stream-cipher/

1. Digital Encryption Standard(DES): DES is a symmetric key block cipher that uses 64-bit blocks, but it has been found vulnerable to powerful attacks. This is the reason the use of DES is on a decline.

2. Triple DES:This symmetric key cipher uses three keys to perform encryption-decryption-encryption. It is more secure than the original DES cipher but as compared to other modern algorithms, triple DES is quite slow and inefficient.

3. Advanced Encryption Standard(AES): AES has superseded the DES algorithm and has been adopted by the U.S. government. It is a symmetric key cipher and uses blocks in multiple 32 bits with minimum length fixed at 128 bits and maximum at 256 bits. The algorithm used for AES, was originally named Rijndael.[2]

4. Blowfish: Blowfish is a symmetric key block cipher with a block size of 64 and a key length varying from 32 bits to 448 bits. It is unpatented, and the algorithm is available in the public domain.

5. Twofish: Twofish is also a symmetric key block cipher with a block size of 128 bits and key sizes up to 256 bits. It is slower than AES for 128 bits but faster for 256 bits. It is also unpatented and the algorithm is freely available in the public domain.

But, after an overall account of the block enciphers available, the author decided there is really only one reasonable choice: the Advanced Encryption Standard (AES). This decision is based

2.2 Advanced Encryption Standard (AES)

AES is basically the only choice for a block cipher scheme. It has been the industry standard for the past 20 years, even used by the U.S. government. [Aumasson, 2017]

The foundation upon which the AES standard is built is based on Field Theory. More precisely, the theory of *Galois Fields*. Explain basically what a Galois field is here.

[2]The winners of the AES competition were two Belgians: Vincent Rijmen, Joan Daemen, thus the algorithm's name: "**Rinjdae**l"

Modulo arithmetic is commonly used in Computer science, and needs little explanation. It is the amount left over by the division of two integers, commonly known as the *remainder*. It is written as: $11 MOD 4 = 3$.

We'll also need to remember polynomials from basic algebra. More precisely, the mathematics upon them, with special interest on the coefficients. Next, we will recall the division of polynomials, and the special set of polynomial divisors name *irreducible polynomials*. These are polynomials that ???

Now, that we have an reviewed all the required mathematics,

1. Galois Fields
2. Polynomials, and their division
3. Irreducible polynomials
4. Modulo arithmetic

we can begin with the genius of the algorithm.

2.2.1 Step One:

Adding the key to the mix, the longer the key the better!

2.2.2 "SubBytes" or byte substitution

What happens here is the that bytes get shifted?

2.2.3 ShiftRows or the rows are shifted

The rows are shifted..

2.2.4 MixColums or the columns are mixed

Here, the columns are mixed in "this" manner.

2.2.5 AddRoundKey or the key (which key, partial) is re-added

"This" key is re-added, xor'd? Which key, back to the field.

2.2.6 The process is repeated x number of times.

2.2.7 AES algorithm summary

The goal of the algorithm is to insert confusion and diffusion into the field, over and over. And, the algorithm is just reversed to retrieve the plain text. The algorithm was fast in 2001, when it was introduced, but now, 20 years later, it is built into all modern desktop CPUs (at least Intel and AMD), so it's blazingly fast.

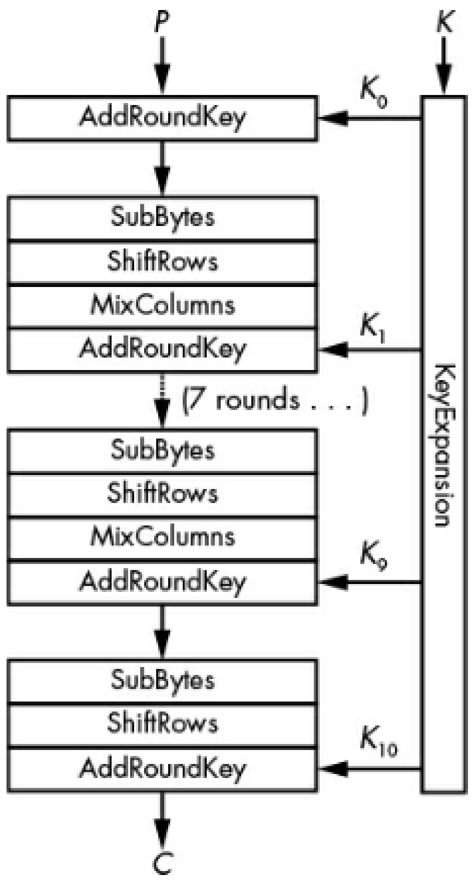

Figure 4-4: The internal operations of AES

Chapter 3

Implementation

3.1 WebExtensions

Write a blah blah paragraph, can be short, about the basic structure of WebExtension. The following are Manifest Keys that will be required for my implementation.

1. *manifest_version*: Always 2
2. *name*: Name of the implementation
3. *description*: Description of the extension
4. *version*: The version of the software
5. *author*: The name of the developer
6. *applications*: Also known as the *brower_specific_settings* contains keys that are specific to the particular host application.

```
"background": {
    "page": "background.html"
},
```

In Thunderbird, all WebExtension API can be accessed through the browser.* namespace, as with Firefox, but also through the messenger.* namespace, which is a better fit for Thunderbird. [1]

[1]From website: https://developer.thunderbird.net/add-ons/mailextensions/hello-world-add-on/using-webextension-apis

messenger.tabs.query

The tabs API provides access to Thunderbird's tabs. In some cases, since the call is executed from where it is called from, it may be necessary to execute tabs *message_display_action* and not from inside the tab we are looking for.

messenger.messageDisplay.getDisplayedMessage()

the *getDisplayedMessage* method of the *messageDisplay API* provides access to the currently viewed message in a given tab. It returns a promise for a *MessageHeader* object from the *message API* with basic imformation about the message. [2]

messenger.messages.getFull()

The *getFull()* method returns a Promise for a MessagePart object, which relates to messages containing multiple MIME parts. The headers member of the part returned by getFull includes the headers of the message.

Steps to create an addon in Thunderbird.

3.1.1 Using a background page

First, we need to add "backgrounds" to our manifest.

```
"background": {
    "page": "background.html"
},
```

[2]The getDisplayMessage method requires the messageRead permission, which needs to be added ot the permissions key of the manifest.json file.

Additionally, we’ll need an html and javascript document, as well as keep the manifest updated.

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8"/>
    <script src="background.js" type="module"></script>
</head>
</html>
```

1. Create manifest.json
2. Have it reference a folder (that it is outside of), like ”MyAddon”, which contains
    - myaddon.html
    - myaddon.css
    - myaddon.js
3. Then, create another folder named ”images”, for images.
4. To install, open thunderbird
5. click on the hamburger button on the right side of tb
6. select ”Debug Add-ons”
7. blick on ”Load Temporary Add-on...” button
8. navigate to the previously created manifest.json file.

The Add-on should be fully functional now.

3.2 Crypto JS

Chapter 4

Security Considerations

4.1 Attack Vectors

4.2 Attack Mitigation

Chapter 5

Summary

Chapter 6

Appendix

6.1 Specifications details

6.1.1 Use Cases

The Use Cases used in this project will be defined, and or be restricted to the following items:

Use Case ID

The Use Case ID will be a unique, numeric identifier for the use case.

Actor(s)

An actor is a person or other entity external to the system who interacts with it, and performs use cases to complete task. Included in this designation, will be additional actors who participate in the use case.

Description

This section should describe at a high level the purpose of the use case, what it aims to achieve, and any other relevant outcomes.

Preconditions

The preconditions are all those conditions that must exist prior to the execution of the use case.

Basic Flow

These are the basic, ordered steps and the description required for the completion of the use case. The steps will be numbers, and should be executed in this exact order. Completing the steps, in this order, should lead to the completion of the use case without error.

Exceptions

Describes any anticipated errors that could occur during the execution of the use case, and how the system will handle these errors. The exceptions systems will not describe unanticipated errors, or error that are not included in the basic flow.

Postconditions

Describes the state of all relevant parties, including the system, *after* the execution of the use case.

Use Case ID:	0
Actor(s):	Alice
Description:	Alice will encrypt an email to Bob
Preconditions: 1. Thunderbird Email client installed. 2. Thunderbird Email client configured to send and receive emails. 3. Super-duper Addon installed. 4. Email written	
Basic Flow: 1. Alice writes an email in Thunderbird. 2. Alice locates and click on the add-on button. 3. Observe: Alive sees a popup screen encrypt the email. 4. Alice is prompted to enter a password.	
Exceptions: 1. N/A	
Postconditions: 1. The email is enciphered. 2. The addon window closes. 3. Alice is returned to the Thunderbird client.	

Use Case ID:	1
Actor(s):	Bob (or any other intended recipient) decrypts an Email from Alice
Description:	Alice will encrypt an email to another actor, then share a password with them, that they will then be able to decrypt
Preconditions: 1. Thunderbird Email client installed. 2. Thunderbird Email client configured to send and receive emails. 3. Super-duper Add-on installed. 4. Email written	
Basic Flow: 1. Alice writes an email in Thunderbird. 2. Alice locates and click on the addon button. 3. Observe: Alive sees a popup screen encrypt the email. 4. Alice is prompted to enter a password. 5. Alice enters a password. 4. Alice shares this password with said actor *offline*.	

Exceptions: 1. N/A

Postconditions: 1. The email is enciphered. 2. The addon window closes. 3. Alice is returned to the Thunderbird client.

Use Case ID:	2
Actor(s):	Mallory
Description:	Mallory will decipher an email
Preconditions: 1. Thunderbird Email client installed. 2. Thunderbird Email client configured to send and receive emails. 3. Super-duper Addon installed. 4. Email written	
Basic Flow: 1. Alice writes an email in Thunderbird. 2. Alice locates and click on the addon button. 3. Observe: Alive sees a popup screen encrypt the email. 4. Alice is prompted to enter a password. 5. TBD.	

Exceptions: 1. None allowed. =)

Postconditions: 1. The email is still enciphered. 2. The add-on window closes. 3. Mallory is returned to the Thunderbird client.

6.2 Software Requirements

Chapter 7

Software Requirments Specifications

7.1 Introduction

7.1.1 Purpose

This document will describe the entire software development process, including use cases, personas, diagrams, and the end goals of the system. The audience for this document will be any persons interested in the software engineering process used for this project, but more specifically, those responsible for overseeing and rating this project.

7.1.2 Scope

The name for this product will be "Thunderbird: One Time Password." This product will be a Thunderbird add-on, that will encipher plain text into cipher text, which will be delivered by the Thunderbird client to another Thunderbird recipient, that also has the add-on installed. Finally, the second person will be able to decipher the cipher text back to plain text, and read the message.

7.1.3 Definitions, acronyms, abbreviations

The following definitions, acronyms, and abbreviations may be used with in the software development process:

client Refers to an email client, more specifically Mozilla's Thunderbird email client.

E2EE End-to-end encrypted, in this case, an end-to-end encrypted email.

JS JavaScript.

AES Advanced Encryption Standard.

IEEE Institute of Electrical and Electronics Engineers.

asymmetric encryption Encryption that only uses one key for encryption.

symmetric encryption Encryption that requires two keys, one on each side of the private message exchange.

API application programming interface.

extensions An extension adds features and functions to a browser.

plain text The text that we wish to encrypt.

cipher text The encrypted text.

ECB Electronic Codebook, a AES encryption mode.

CBC Cipher Block Chaining, a AES encryption mode.

CFB Cipher Feedback Mode, a AES encryption mode.

OFB Output Feedback Mode, a AES encryption mode.

CTR Counter Mode, a AES encryption mode.

SRS Software Requirements Specification.

7.1.4 References

Author used the IEEE document:

1. IEEE Std 803-1998

the IEEE Recommended Practice for Software Requirements Specifications. [1]

[1]https://cse.msu.edu/ cse870/IEEEXplore-SRS-template.pdf

7.1.5 Overview

7.2 Overall Description

The following subsections will describe the general factors that will influence the product requirements, including any background information.

7.2.1 Product perspective

The developed software product, *Thunderbird: One Time Password*, has not current rival. It current alternatives would be Mozilla's own implementation of OpenPGP. The previous option was PgP through the add-on Enigmail. However, at the writing of this document, the add-on is no longer supported.

The two alternatives do have the advantage that they used symmetric key exchange to encrypt emails, which is more secure, and recommended for encoded email exchange. The *Thunderbird: One Time Password* add-on will have the feature that it is easy to use, at the expense of security.

System interfaces

The required, and assumed interfaces required for the product include the following:

1. A modern system, running one of three operating systems:
   (a) Windows 10 or later
   (b) Apple running Big Sur or later
   (c) Linux variant, running a modern system
2. an Internet connection

User interfaces

There are no special user interface requirements.

Hardware interfaces

There are no special hardware interfaces required for this product to function.

Software interfaces

The required software interfaces are:

1. Mozilla's free, open source email client, Thunderbird, to be installed on the system.

2. The client should be configured to send and receive emails.[2]

3. The client should be updated to the latest current software version.

4. The client can be installed on any current (or recent) Windows, Linux, or Apple OS.[3]

Communications interfaces

No special communication interfaces will be required, than would already be prerequisites for Email communication, i.e. network capable computer.

Memory constraints

Not applicable

Operations

Not applicable

Site adaptation requirements

Not applicable

[2]Thus, an email account on an email server is assumed.
[3]No other OS will be tested.

7.2.2 Product functions

7.2.3 User characteristics

7.2.4 Constraints

There will be various constraints within this project listed below:

- Security: It will not be possible to account for all attack vectors. Thus, only known, common attack vectors will be discussed.
- Security: How Mallory comes into possession of an encrypted email may not be fully explored. Related to 1. above, but we'll at least give an examination to this possibility – however she came into possess the Email.

7.2.5 Assumptions and dependencies

7.3 Specific Requirements

7.4 Appendix

Bibliography

[Shirey, 2007] Shirey, R., *"RFC 4949 - Internet Security Glossary, Version 2."*, Document Search and Retrieval Page, Aug. 2007, https://datatracker.ietf.org/doc/html/rfc4949.

[Delfs and Knebl, 2002] Delfs, Hans, and Helmut Knebl, *Introduction to Cryptography: Principles and Applications*, p. 12, Springer Verlag, Berlin, 2002.

[Schneier, 2015] Schneier, Bruce, *Applied Cryptography: Protocols, Algorithms, and Source Code in C*, p. 155, Wiley, Indianapolis, IN, 2015.

[Nirula, 2022] Nirula, Urvashi, *Block Cipher — Purpose, Applications & Examples*, Document Search and Retrieval Page, https://study.com/learn/lesson/block-cipher-purpose-applications.html

[Aumasson, 2017] Aumasson, Jean-Philippe, *Serious cryptography: A practical introduction to modern encryption*, p. 107, No Starch Press Inc, San Francisco, CA, 2017

[Delfs & Knebl, 2007] Delfs, H., & Knebl, H., *Introduction to cryptography: Principles and applications*, Springer. Berlin, 2007

[Paar & Pelzl, 2009] Paar, Christof., & Pelzl, J., *Understanding cryptography: A textbook for students and practitioners.*, Springer Science & Business Media, 2009

[Dooley, 2008] Dooley, J. F., *History of cryptography and cryptanalysis: Codes, ciphers, and their algorithms*, Springer, 2008

[Kahate, 2008] Kahate, A., *Cryptography and network security*, Tata McGraw-Hill Education, 2008

[Katz & Lindell, 2007] Katz, J., & Lindell, Y., *Introduction to modern cryptography: Principles and protocols*, CRC Press, 2007

[Martin, 2017] Martin, K., *Everyday cryptography: Fundamental principles and applications*, Oxford University Press, 2017