



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

July 1, 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

0.1	Abstract	2
1	Introduction	5
1.1	Background	5
1.2	Problem	6
1.3	Solution	7
1.4	Methods applied	7
2	Cryptography	9
2.1	Algorithm selection overview	9
2.1.1	Symmetric key encryption	9
2.1.2	Block vs. Stream cipher encryption	10
2.1.3	Block cipher selection	10
2.2	Advanced Encryption Standard (AES)	11
2.2.1	Mathematics: Overview	11
2.2.2	Step One:	13
2.2.3	"SubBytes" or byte substitution	13
2.2.4	ShiftRows or the rows are shifted	13
2.2.5	MixColumns or the columns are mixed	13
2.2.6	AddRoundKey or the key (which key, partial) is re-added	13
2.2.7	The process is repeated x number of times.	14
2.2.8	AES algorithm summary	14
2.2.9	Block	14
3	Implementation	15
3.1	WebExtensions	15
3.1.1	Using a background page	16
3.2	Crypto JS	17
4	Challenges Encountered	19

5	Summary	21
6	Software Requirments Specifications	23
6.1	Introduction	23
6.1.1	Purpose	23
6.1.2	Scope	23
6.1.3	Definitions, acronyms, abbreviations	23
6.1.4	References	24
6.1.5	Overview	25
6.2	Overall Description	25
6.2.1	Product perspective	25
6.2.2	Product functions	27
6.2.3	User characteristics	27
6.2.4	Constraints	27
6.2.5	Assumptions and dependencies	27
6.3	Specific Requirements	27
6.4	Appendix	27

Chapter 1

Introduction

The digital age has fully absorbed our societies. We do everything in some form or another on 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, and 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, between the United States, the United Kingdom, Canada, Australia, and New Zealand and their the collection of all online communication, social media, and phone data removed any doubt about expectations to individual privacy. No online communication, or online activities in general, has been considered safe ever since.

1.1 Background

The the sphere of email communication, PGP has existed for decades. It is predicated on the exchange of user created public (and private) keys. However, it has a few inherent problems. First, there is a technical requirement to create and exchange keys. In order to facilitate this, additional client software must be installed. Additionally, several challenging steps beyond the scope of the average end user will need to be completed, like selecting encryption algorithm, size of key, etcetra.

Originally, Thunderbird relied on an add-on, Enigmail, to create, manage, and exchange keys. The author used this add-on for many years. And, while it was satisfactory, it was plain to see that it was not without its technical requirements.

Starting with Thunderbird 78, Mozilla implemented OpenPGP as part of its 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.

1.2 Problem

Mozilla has tried to support end-to-end encryption (E2EE) for a long time, it has been faced with a major obstacles, mostly mentioned above, including:

- Setting up 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 set up a client or keys for themselves
- Mozilla is in the process of using OpenPGP, a build-in component to the Thunderbird client, but that also has problems, most obviously, you again need new keys (granted, easier to set up this time)
- and, again, both people must have generated keys (again, easier this time)
- Lastly, this OpenPGP built-in component, is still in its early developmental stages.

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

1.3 Solution

The bachelor thesis candidate intends to research and develop a Thunderbird Add-on, that will allow Alice to send an encrypted message to Bob. Bob is not a tech savvy person, and is clueless about encryption technology. So, the idea of learning, installing, and setting up any types of keys, for him, is overwhelming. However, they do communicate regularly, so Alice can just whisper a one time password to him. Subsequently, Alice could then use developed add-on to encipher an email message for Bob. Which, he then could decipher using the same add-on, and the analog agreed upon passphrase.¹

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 author 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.

Research will dictate the best implementation strategy.

1.4 Methods applied

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

1. Literature
2. Online learning resources
3. Thunderbird and JS Encryption APIs

¹Alice and Bob are fictional characters commonly used as placeholders in discussions about cryptographic systems and protocols.

4. Guidance from Mentors, and fellow Thunderbird add-on developers
5. Visual Studio Code
6. Github
7. Latex
8. Jira

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 implemented encryption available, popularly known as "public-key encryption." Examples encryption key algorithms used include RSA and Diffie-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[?, 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:

- 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. 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.¹
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 . It has been the industry standard for the past 20 years, even used by the U.S. government. [Aumasson, 2017]

For that reason, the algorithm is built into modern CPU microcode, by both AMD and Intel, for even faster processing.

2.2.1 Mathematics: Overview

Since a full background is beyond the scope of this project (it could entail it's own thesis), the author will gloss over it quickly.

¹The winners of the AES competition were two Belgians: Vincent Rijmen, Joan Daemen, thus the algorithm's name: "Rijndael"

The foundation of AES is grounded in Abstract mathematics, more specifically, *set theory*. Within set theory, is the study of groups. A group is a set of elements upon which an operation (and its inverse) can be executed. [Paar & Pelzl, 2009, p. 92]

In short, basic mathematical operations

1. addition
2. subtraction
3. multiplication
4. division

are operations that can be applied to a group of elements.

A field is an extension of a group, in that all four basic arithmetic operations are included in a single structure. [Paar & Pelzl, 2009, p. 92]

However, as cryptographers, we are not yet satisfied. We need a working set that is finite, or as they are commonly known *Galois fields*, or *unendliche Körpe*. In short, the purpose of these fields is that the set of elements, regardless of the four principle operations performed on them, the result will remain *within* the set of elements. This is profound, necessary and brilliant for cryptographic usage. But, how does this work?

Essentially, the operations are carried out with the aid of the *modulo operator*, and it ensures our result remains in the set. But, what set?

We extend beyond a finite field, to focus on a extension field, in particular: $GF(2^8)$

As this conveniently translates to 256 elements, which is perfect for a computer byte.

The last thing we will want to have in mind as we proceed is polynomial division. It's not any different than grammar school algebra, but is carried out with one of a very special type of polynomial, known as a *irreducible polynomial*. An irreducible polynomial is similar to a prime number, only it is a polynomial. In other words, it cannot be broken down, divided down, into smaller components. For our cryptographic purposes, the polynomial we'll use is the following: $X^8 + X^4 + X^3 + X + 1$ [Delfs and Knebl, 2002, p.21]

Now, that we have reviewed all the required mathematics,

1. Galois Fields/Finite Fields, or *endliche Körper*
2. Polynomial division, and irreducible polynomials
3. Modulo arithmetic

we continue with the algorithm.

Key expansion - how?

2.2.2 Step One:

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

2.2.3 "SubBytes" or byte substitution

What happens here is that bytes get shifted?

2.2.4 ShiftRows or the rows are shifted

The rows are shifted..

2.2.5 MixColumns or the columns are mixed

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

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

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

2.2.7 The process is repeated x number of times.

2.2.8 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.

2.2.9 Block

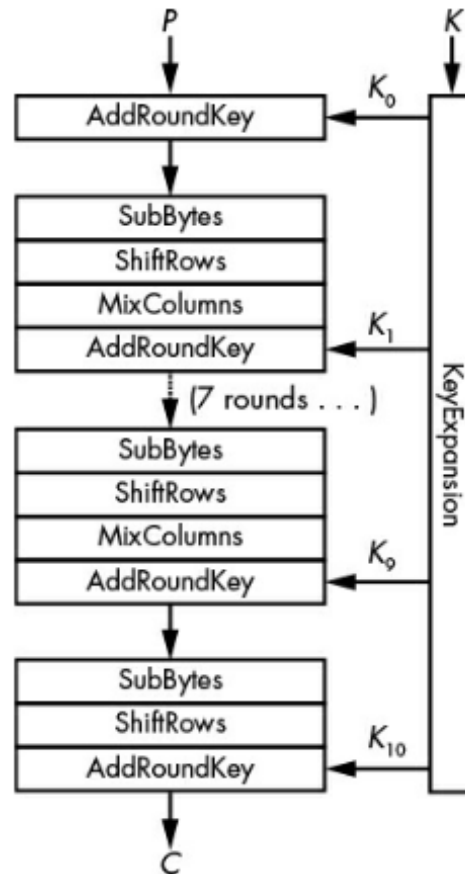


Figure 4-4: The internal operations of AES

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 *browser_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

¹From 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 information 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"  
},
```

²The *getDisplayMessage* method requires the *messageRead* permission, which needs to be added to 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. click 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

Challenges Encountered

Chapter 5

Summary

Chapter 6

Software Requirments Specifications

6.1 Introduction

6.1.1 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.

6.1.2 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.

6.1.3 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.

6.1.4 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/IEEEExplore-SRS-template.pdf>

6.1.5 Overview

6.2 Overall Description

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

6.2.1 Product perspective

The developed software product, *Thunderbird: One Time Password*, has not current rival. Its 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

²Thus, an email account on an email server is assumed.

³No other OS will be tested.

6.2.2 Product functions

6.2.3 User characteristics

6.2.4 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.

6.2.5 Assumptions and dependencies

6.3 Specific Requirements

6.4 Appendix

References

- [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