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

September 1, 2022

0.1 Abstract

The developer describes the steps in researching and developing a Thunderbird add-on that offers End-to-End Encryption (E2EE), without the need for key changes, i.e. without PGP and its asymmetric key exchange requirement. The developed software will allow one user to exchange a password with another user, encrypt a message with that password, and the other user will be able to decrypt the message with that password.

0.2 Acknowledgements

First off, I need to give credit to my partner and child, who did their best to limit distractions and allow me to complete this project. Without their continued support and love, this project would have been a great deal more complicated.

Secondly, I need to give special mention to John Bieling, an adminitrative Thunderbird Developer, who gave invaluable guidance to many roadblocks and issues I had along the way. He was always able to point me in the right direction, and without his help this project would have not been completed.

Contents

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

In the realm of email communication, Pretty Good Privacy (PGP) has existed for decades. It is predicated on the exchange of user created public keys (in combination with private, non-exchanged keys). However, it has a few inherent obstacles. 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, etc. 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 it's technical requirements.

Starting with Thunderbird 78 (2020), Mozilla implemented OpenPGP as part of its core client software, and dropped support for all add-ons not using MailExtensions (which includes Enigmail). There were a few reasons for this, including Mozilla's desire to simplify the process. But, also the desire to move away from the PGP trademarked software. Nevertheless, the OpenPGP feature is disabled by default, and is still considered a work

in progress. All other encryption 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, but it has been faced with major obstacles, partly mentioned above, including:

- Setting up the Enigmail add-on was too technical
- Generating keys was too technical
- Even if conditions 1. & 2. were fulfilled, its uncommon that others have created their own public keys
- Mozilla is in the process of using OpenPGP, a built-in component to the Thunderbird client, but that also has problems: most obviously, you 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 the developed add-on to encrypt an email message for Bob. Which, he then could decrypt using the same add-on, and the previously agreed upon password. [1]

This thesis will focus on the Mozilla Thunderbird client, for the simple fact that 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.

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

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. Research literature and books
2. Online learning resources
3. Mozilla's WebExtensions API and JS Encryption APIs
4. Guidance from Mentors, and fellow Thunderbird add-on developers
5. Visual Studio Code coding
6. Github for version control
    (a) Project source code: https://github.com/j-autuick/thunderbird-addon-jscrypto/settings
7. Latex for composing text
    (a) Latex source code: https://github.com/j-autuick/ba-latex-code
8. Jira for project management

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. 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 Diffe-Hellman-Merkle. [Shirey, 2007, Website]
2. There are challenges though:
   (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
3. Symmetric-key encryption: Use the same key for both encryption and decryption [Delfs and Knebl, 2007, p. 155]
   (a) The primary drawback is both parties need to exchange that single 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 of 32 bits with the minimum length fixed at 128 bits and the 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.

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

After an overall account of the available block enciphers, the author decided there is really only one choice: the Advanced Encryption Standard (AES) as it's the industry standard.

2.2 Advanced Encryption Standard (AES)

For reasons that will soon become apparent, AES has been the industry standard for the past 20 years, even used as a secure standard by the U.S. government. [Aumasson, 2017, p. 107]

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.

The foundation of AES is grounded in Abstract mathematics, more specifically, *set theory*. Within set theory exists 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,

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 *endliche Körpe*. In short, the beauty of the Galois fields is that 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? [Paar, 2014, Website]

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

The Galois field principles are focused on one particular sub-field, namely the field defined by: $GF(2^8)$

This conveniently translates to 256 elements, which fits perfectly within 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 that it is a polynomial. In other words, it cannot be factored into smaller components. The AES algorithm uses the following polynomial: $x^8 + x^4 + x^3 + x + 1$ [Delfs and Knebl, 2007, p.21]

2.2.2 The AES algorithm

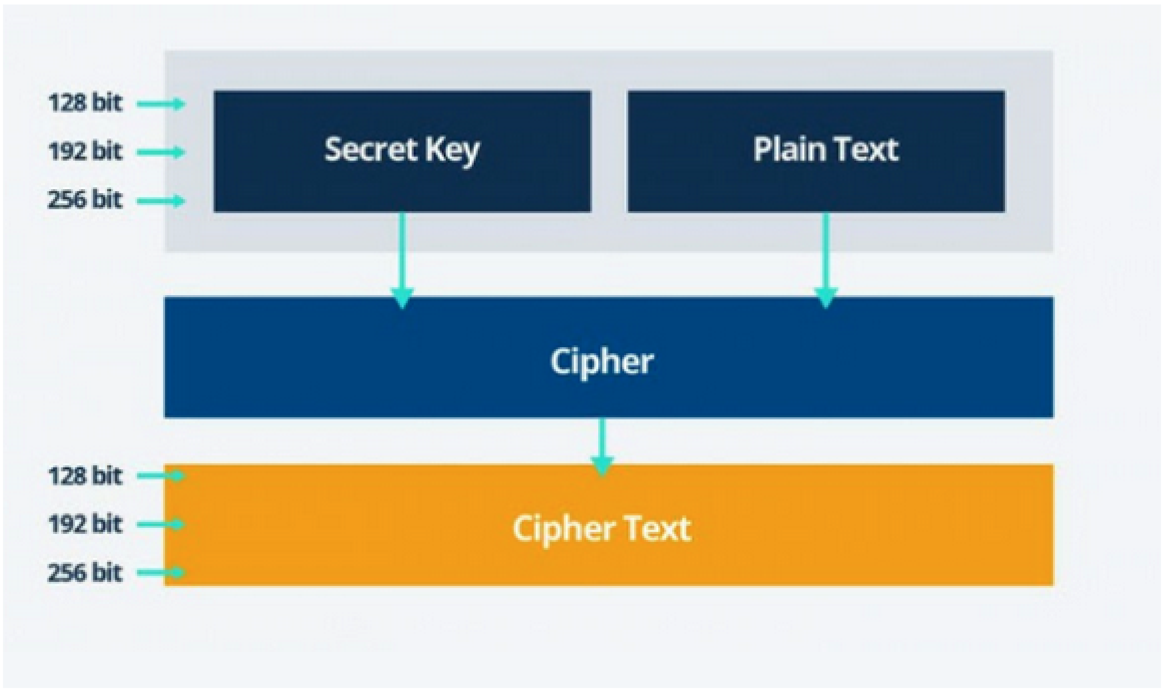


Figure 2.1: Simplified AES Overview
[Crawford, 2019, Webpage]

With the mathemathical theories out of the way, we can look at how all the theory get implemented into our computers. Here is a simplifed overview: (See: 2.1).

The first step in the AES algorithm is the creation of a 4-by-4 array of bytes called the state, that is modified in a series of rounds. The state is initially set equal to the input of the cipher (notice: a 128 bit minimum is exactly 16 bytes, perfect for execution on a computer). Then, the following four operations are applied. [Katz & Lindell, 2008, p. 186]

2.2.3 Step one: "AddRound" key

In every round of the AES, a 128-bit sub-key is derived from the master key, and it is interpreted as a 4-by-4 array of bytes. The state array is updated by XORing it with the sub-key (See: 2.2). [Katz & Lindell, 2008, p. 186]

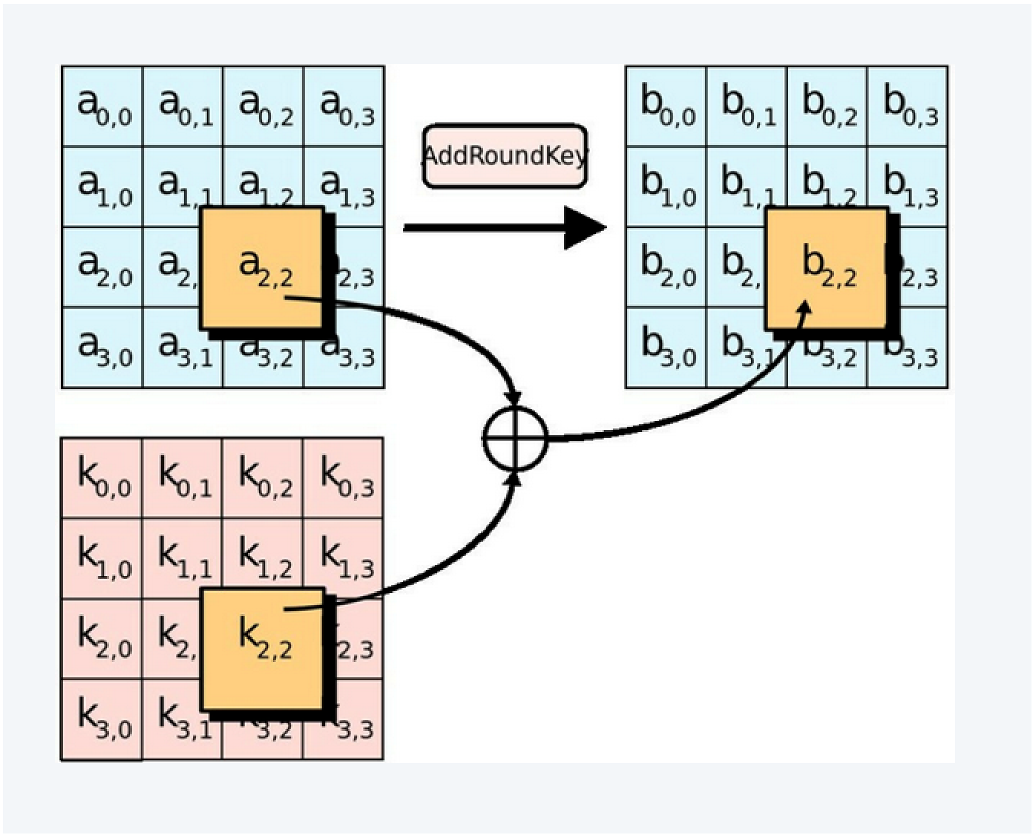


Figure 2.2: AddRound Key Round
[Crawford, 2019, Webpage]

2.2.4 Step two: ”SubBytes” or byte substitution

In this step, each byte of the state array is replaced by another byte according to a single fixed table S. This substitution table (or S-box) is a bijection over $\{0,1\}^8$. There is only one S-box that is substituting all the bytes in the state array, every round (See: 2.3). [Katz & Lindell, 2008, p. 186]

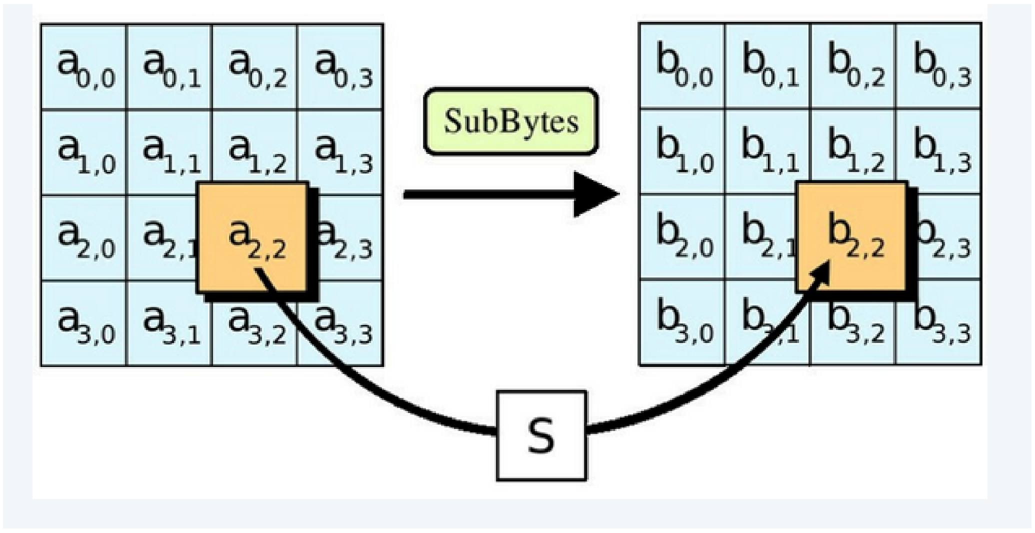


Figure 2.3: SubBytes Round
[Crawford, 2019, Webpage]

2.2.5 Step three: ”ShiftRows” or the rows are shifted

In this step, the bytes in each row of the state array are cyclically shifted to the left as follows: the first row of the array is untouched, and the second row is shifted one place to the left, the third row is shifted two places to the left, and the fourth row is shifted three places to the left (See: 2.4). [Katz & Lindell, 2008, p. 186]

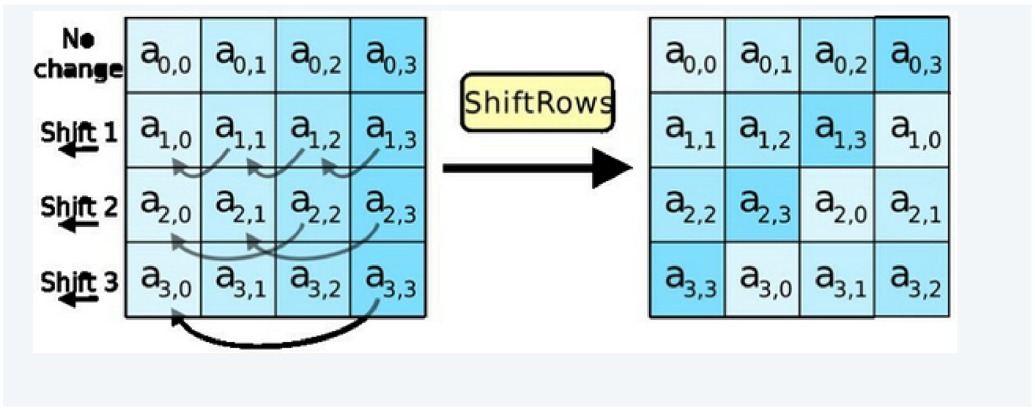


Figure 2.4: ShiftRows Round
[Crawford, 2019, Webpage]

2.2.6 Step four: ”MixColumns” or the columns are mixed

In this step, an invertible linear transformation is applied to each column. One can think of this as a matrix multiplication (See: 2.5). [Katz & Lindell, 2008, p. 186]

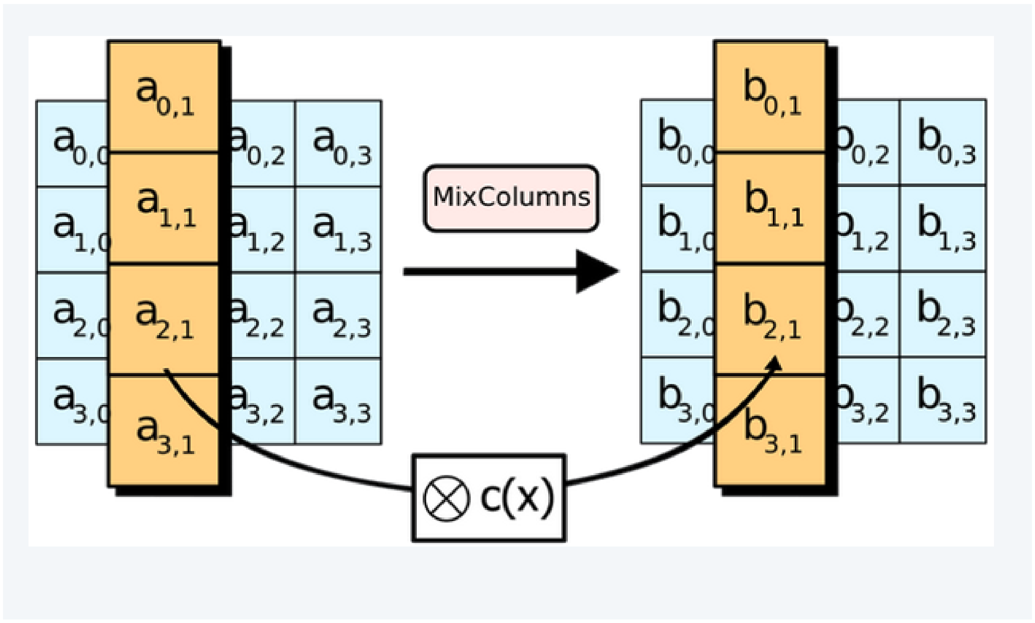

$a_{0,0}$ $a_{0,1}$ $a_{0,2}$ $a_{0,3}$
$a_{1,0}$ $a_{1,1}$ $a_{1,2}$ $a_{1,3}$
$a_{2,0}$ $a_{2,1}$ $a_{2,2}$ $a_{2,3}$
$a_{3,0}$ $a_{3,1}$ $a_{3,2}$ $a_{3,3}$
MixColumns
$\otimes c(x)$
$b_{0,0}$ $b_{0,1}$ $b_{0,2}$ $b_{0,3}$
$b_{1,0}$ $b_{1,1}$ $b_{1,2}$ $b_{1,3}$
$b_{2,0}$ $b_{2,1}$ $b_{2,2}$ $b_{2,3}$
$b_{3,0}$ $b_{3,1}$ $b_{3,2}$ $b_{3,3}$

Figure 2.5: MixColumns Round
[Crawford, 2019, Webpage]

2.2.7 The process is repeated x number of times

The AES algorithm supports bit sizes of 128 (minimum requirement), 192, and 256. Depending on the bit size specified, the algorithm will be repeated either 10, 12, or 14 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 CPUs, so it's blazingly fast.

Although the algorithm is derived from a combination of complex mathematical theories, its algorithmic implementation into hardware is the perfect intersection between mathematics and computer science. Any computer scientist can quickly assess from the images above, that the XORing, bit shifting, table lookups, and bit permutations are exactly the kind of operations that are executed super fast on computers. [Dooley, 2008, p. 178]

Chapter 3

Implementation

3.1 Javascript Cryptography

The developer decided that the best solution for the cryptography implementation was the utilization of a pre-existing cryptography library.

For this task, there were three necessary conditions that had to be met:

- A clear, easily identifiable, and reputable source/owners
- Existing, easily obtainable, and comprehensive documentation
- Open source, easily available code analysis

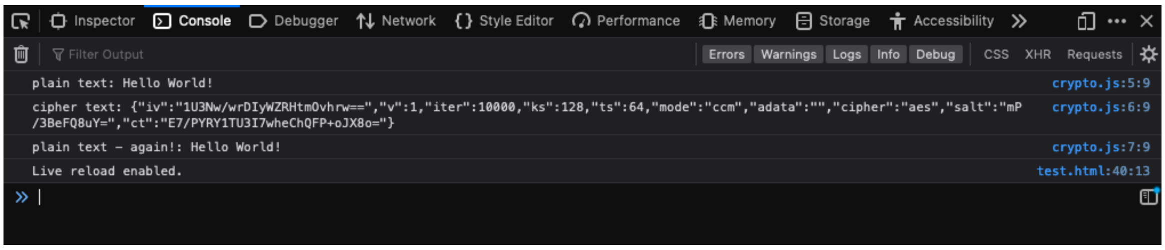
and, slightly less important, actively maintained.

Meeting the above criteria was not as simple as it would seem. There were many options that met some of the conditions, but meeting all of them was more challenging. Ultimately, however, the researcher was satisfied with the Stanford Javascript Crypto Library (SJCL), as it met all the above conditions. Additionally, it appeared to be well developed, and still actively maintained (See: 3.1. [Stanford Security Lab, 2009, Website]

```
var ciphertext = sjcl.encrypt("reallyHardPasswordNoOneCouldEveryGuess",
 ↪  "Hello World!");
var plaintext = sjcl.decrypt("reallyHardPasswordNoOneCouldEveryGuess",
 ↪  ciphertext);
console.log("plain text: " + plaintext);
console.log("cipher text: " + ciphertext);
console.log("plain text - again!: " + plaintext);
```

Figure 3.1: Example of javascript SJCL

The usage is uncomplicated, and will work for this implementation. The library is available in a minified version, and will be packages with the add-on:



```
plain text: Hello World!                                                                                    crypto.js:5:9
cipher text: {"iv":"1U3Nw/wrDIyWZRHtmOvhrw==","v":1,"iter":10000,"ks":128,"ts":64,"mode":"ccm","adata":"","cipher":"aes","salt":"mP   crypto.js:6:9
/3BeFQ8uY=","ct":"E7/PYRY1TU3I7wheChQFP+oJX8o="}
plain text - again!: Hello World!                                                                           crypto.js:7:9
Live reload enabled.                                                                                        test.html:40:13
```

Figure 3.2: Example output to Firefox console

giving the following result (See: 3.2):

This usage will meet our needs.

3.2 WebExtensions

WebExtensions are web technologies built with the tools that are natural to any web developer: HTML, CSS, and Javascript. Each extension must have a *manifest.json* file, which essentially holds all the vital information like the author of the software, permissions required to use the add-on, the software version, and so on. [Mozilla, 2022, Webpage]

Starting with the release of Thunderbird version 68 (August 2019), Thunderbird moved to only support WebExtensions for add-ons and themes development, with all previous versions no longer working. Even the long standing Enigmail cryptography add-on that the author used for years, no longer functioned.

Here is an image from Mozilla's Thunderbird Add-on Webpage that gives a quick glance of how the extensions might look like (See: 3.3. [Mozilla, 2022, Webpage]

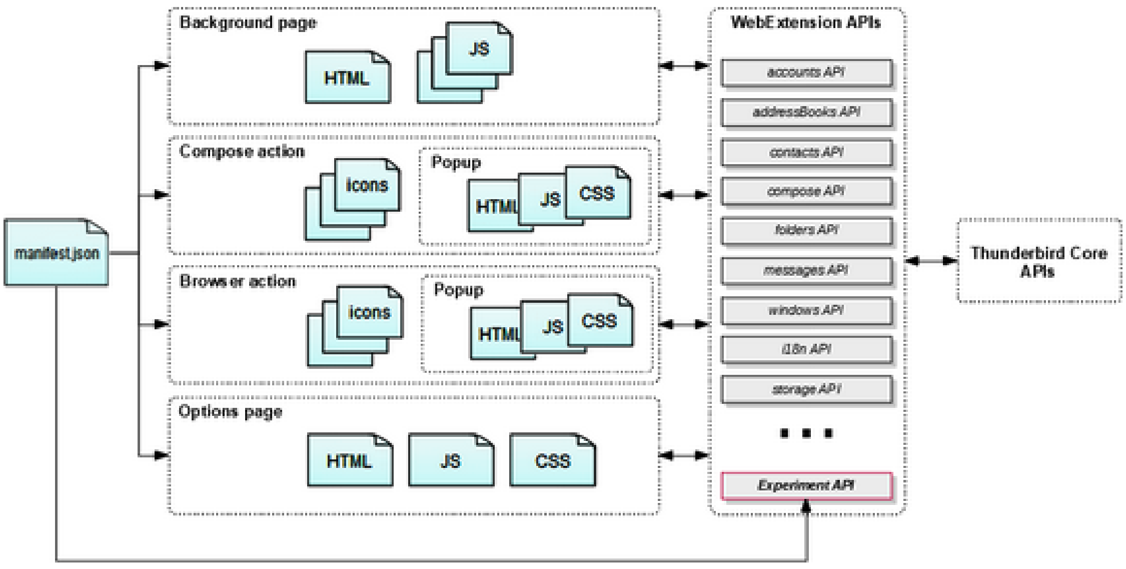


Figure 3.3: WebExtension overview

3.3 Implementation Details

3.3.1 Developer Tools

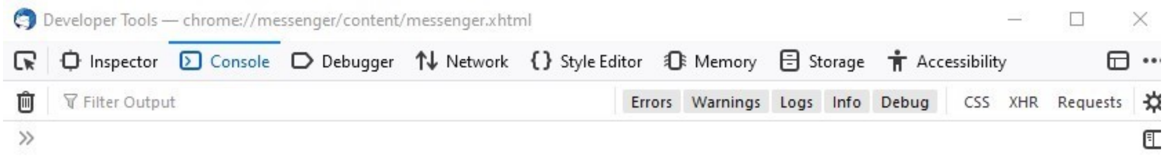
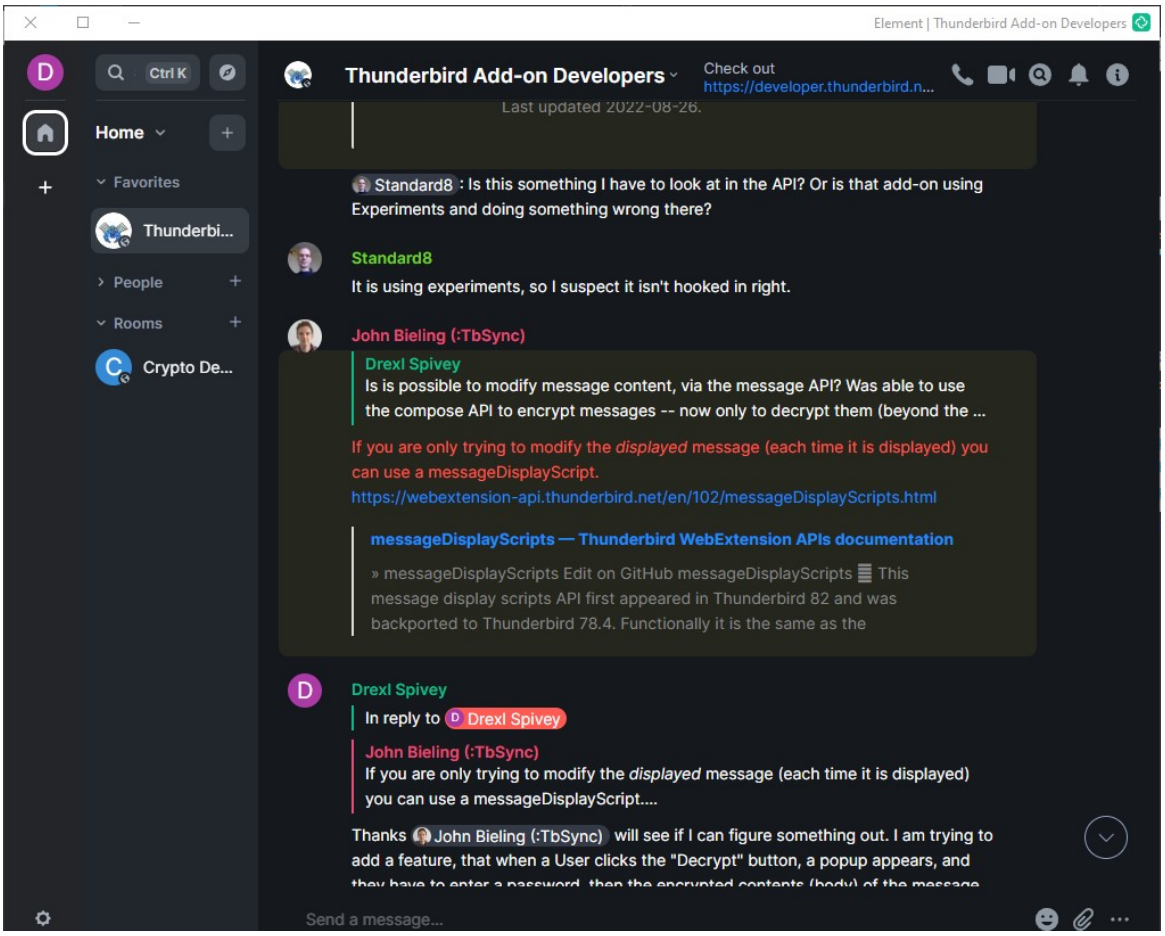


Figure 3.4: Developer Toolkit

For this project, the developer used a Test Driven Development model to go step-by-step through each implemented item. This allows for immediate checking if something works or not, and enhancement of the code in an incremental way. Most every step will be sent to the console.log, to verify success or failure. The first invaluable tool was Tunderbird's mail client Developer Toolkit, which shows a great deal of information, and includes the console. It was an excellent tool which helped with debugging, and seeing what was working and not working through each step (See: 3.4). The developer had it open at all times. Note: Not incidentally, it looks like a typical web browsers developer toolkit, since Thunderbird's core is based on Web Extension technology.

The second valuable tool was the communication channel Element. It an excellent channel to ask questions to other Thunderbird developers. It was where I found answers to

most issues that I had (See: 3.5). [1]


Element | Thunderbird Add-on Developers
Ctrl K
Home
Favorites
Thunderbi...
People
Rooms
Crypto De...
Thunderbird Add-on Developers
Check out https://developer.thunderbird.n...
Send a message...

Last updated 2022-08-26.

Standard8 : Is this something I have to look at in the API? Or is that add-on using Experiments and doing something wrong there?

Standard8
It is using experiments, so I suspect it isn't hooked in right.

John Bieling (:TbSync)
> Drexl Spivey
> Is is possible to modify message content, via the message API? Was able to use the compose API to encrypt messages -- now only to decrypt them (beyond the ...

If you are only trying to modify the *displayed* message (each time it is displayed) you can use a messageDisplayScript.
https://webextension-api.thunderbird.net/en/102/messageDisplayScripts.html

> **messageDisplayScripts — Thunderbird WebExtension APIs documentation**
> » messageDisplayScripts Edit on GitHub messageDisplayScripts This message display scripts API first appeared in Thunderbird 82 and was backported to Thunderbird 78.4. Functionally it is the same as the

Drexl Spivey
> In reply to Drexl Spivey
> John Bieling (:TbSync)
> If you are only trying to modify the *displayed* message (each time it is displayed) you can use a messageDisplayScript....

Thanks John Bieling (:TbSync) will see if I can figure something out. I am trying to add a feature, that when a User clicks the "Decrypt" button, a popup appears, and

Figure 3.5: Developer uses Element (Drexl Spivey)

As mentioned in the acknowledgements, John Bieling was always able to point me in the right direction, within a reasonably short period of time. So, this type of quick responses were paramount to getting the author back on track.

The third valuable resource gained during the development process was a Github repository that had many sample extensions. The examples that provided the most help are listed below: [Bieling, 2020, Website]

1. awaitPopup
2. myComposeBody
3. messageDisplayScript

[1]https://element.io/

3.3.2 Creating a compose window encryption button

First, we'll create a button that will appear in the "Compose Window," the window that appears when you start to write an email.

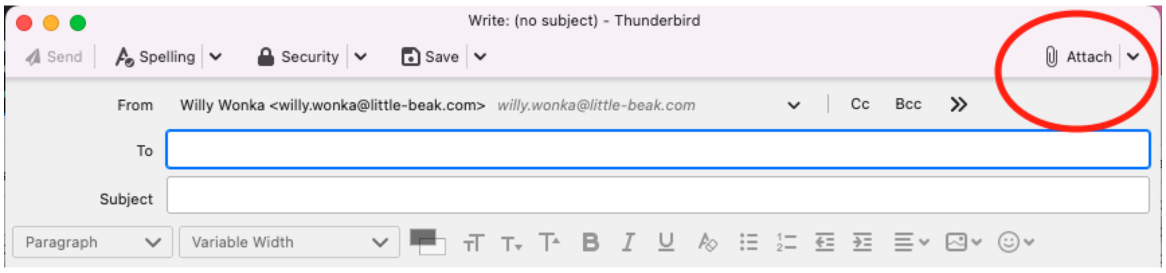

Write: (no subject) - Thunderbird
Send
Spelling
Security
Save
Attach
From
Willy Wonka <willy.wonka@little-beak.com> willy.wonka@little-beak.com
Cc
Bcc
To
Subject
Paragraph
Variable Width

Figure 3.6: Normal compose window

Before add-on implementation: (See: 3.6)

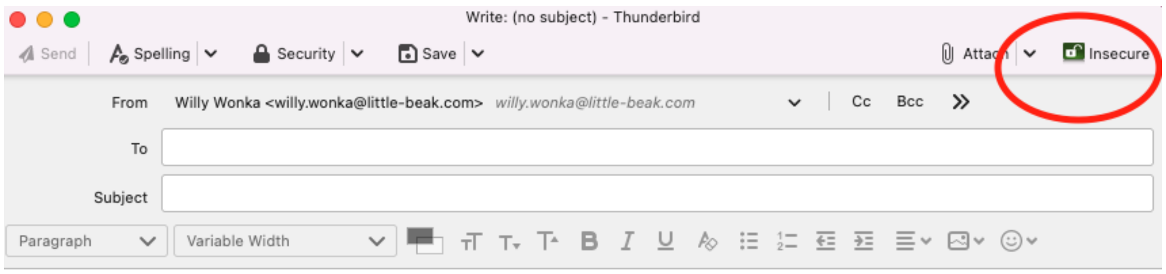

Write: (no subject) - Thunderbird
Send
Spelling
Security
Save
Attach
Insecure
From
Willy Wonka <willy.wonka@little-beak.com> willy.wonka@little-beak.com
Cc
Bcc
To
Subject
Paragraph
Variable Width

Figure 3.7: Button added to the compose window.

with add-on button implemented (See: 3.10).

```json
{
    "manifest_version": 2,
    "name": "Crypto add-on",
    "description": "A password AES cryptographic addon",
    "version": "1.5",
    "author": "Esteban Licea",
    "applications": {
        "gecko": {
            "id": "esteban@little-beak.com",
            "strict_min_version": "78.0"
        }
    },
    "compose_action": {
        "default_title": "Insecure",
        "default_icon": "images/unlocked_64px.png"
    },
    "permissions": [
        "menus"
    ]
}
```

Figure 3.8: Basic manifest.json file

And, here is the manifest.json file that was created for this button. Most of the references in the manifest.json file are self-explanatory. One thing to note about the manifest file is the version number. It has to be set to 2. The rest is composed of information about the add-on, location of images used, and the permissions required for the add-on to function, which is not insignificant. Different permissions are required to access different areas of the user interface, e.g. the compose window, browser window, messages, modifyMessages, etc. (See reference figure: 3.8).

3.3.3 Prompt for password

Managing the password was an early obstacle encountered. Namely, the developer was put on notice that he simply could not just use straight Javascript to get the desired results, but instead had to work within the context of Mozilla's API. So, this was an initial challenge.

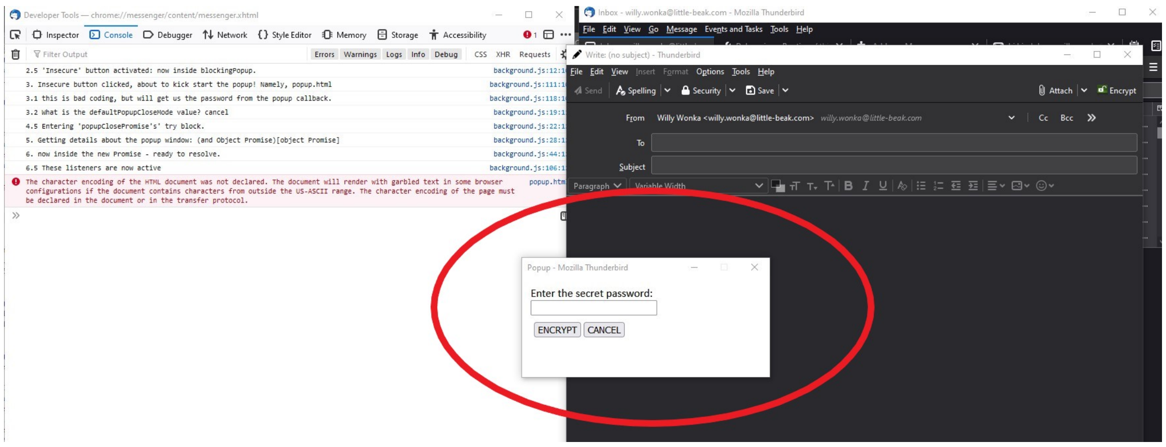
I wanted to highlight the process one time, just so it's clear. This was the process throughout. First, if necessary, the manifest.json file has to be updated. Then, the web extension files (HTML, CSS, and Javascript) should be kept orderly. But, additionally, Mozilla web extension API rules needed to be followed.

```
{
    "compose_action": {
        "default_popup": "passwordPrompt/passwordPrompt.html",
        "default_title": "Insecure",
        "default_icon": "images/unlocked_64px.png"
    },
}
```

Figure 3.9: Add a Password Prompt to manifest file

Step one: updated the manifest.json file. The developer needed to add the *compose_action* so that the button would be added on the compose window.

But, in the code, several files needed to be created, in addition to the pop up folder. A *background.js* file needed to be created. The background.js file acts, as the name suggests in the background. It is loaded when the add-on is loaded, and it is running, or shall I say, ”listening” for events. The code will be available on github, but I will explain the basic process.



```
2.5 'Insecure' button activated: now inside blockingPopup.                              background.js:12:1
3. Insecure button clicked, about to kick start the popup! Namely, popup.html          background.js:111:1
3.1 this is bad coding, but will get us the password from the popup callback.          background.js:118:1
3.2 What is the defaultPopupCloseMode value? cancel                                     background.js:19:1
4.5 Entering 'popupClosePromise's' try block.                                           background.js:22:1
5. Getting details about the popup window: (and Object Promise)[object Promise]         background.js:28:1
6. now inside the new Promise - ready to resolve.                                       background.js:44:1
6.5 These listeners are now active                                                     background.js:106:1
The character encoding of the HTML document was not declared. The document will render with garbled text in some browser    popup.htm
configurations if the document contains characters from outside the US-ASCII range. The character encoding of the page must
be declared in the document or in the transfer protocol.
```

Popup - Mozilla Thunderbird

Enter the secret password:

Developer Tools — chrome://messenger/content/messenger.xhtml
Inspector
Console
Debugger
Network
Style Editor
Memory
Storage
Accessibility
Filter Output
Errors
Warnings
Logs
Info
Debug
CSS
XHR
Requests
Inbox - willy.wonka@little-beak.com - Mozilla Thunderbird
Write: (no subject) - Thunderbird
File Edit View Insert Format Options Tools Help
Send
Spelling
Security
Save
Attach
Encrypt
From
Willy Wonka <willy.wonka@little-beak.com>
To
Subject
Paragraph
Variable Width
ENCRYPT
CANCEL

Figure 3.10: Pop up Window with Developer Toolkit

The background.js listeners are just waiting for the encrypt button to be clicked. This activates the pop up window, which will launch the popup.html file. (See: 3.10

The reader can notice how I used the Developer Toolkit, with console.log used literally throughout the development process.

3.3.4 Encrypt the message

The challenge with the password was not simply collecting it. But, instead, it was passing the variable from the popup.html and its accompanying popup.js to the background.js.

The way the API works, it wasn't as straight forward as the developer originally thought. The password had to be sent as a message, to listeners in the background.js.

Fortunately, there were some examples that were reasonably close on Mr. Bieling's github sample-extensions repo that gave me some ideas of how it could work.

The encryption was a three part process:

- Get the password variable
- Load the Stanford Javascript Cryptography Library
- encrypt away! Hopefully.

After we were able to move the password variable to the background.js, we could go with step two.

```
var imported = document.createElement('script');
imported.src = 'sjcl.js';
document.head.appendChild(imported);
```

Figure 3.11: Loading external library into background.js

For this step, the developer loaded the minimized version of the library, which is quite small, and still powerful. This is how it was loaded into the background.js file. (See: 3.11)

During development, every step was sent to the console.log (later cleaned up). There are a few different elements here to take note of. Firstly, some of the elements are part of the Mozilla API, like the details.body, messenger.compose.setComposeDetails. Apparently, the messenger is the namespace used for Thunderbird, replacing what was previously used (the browser namespace).

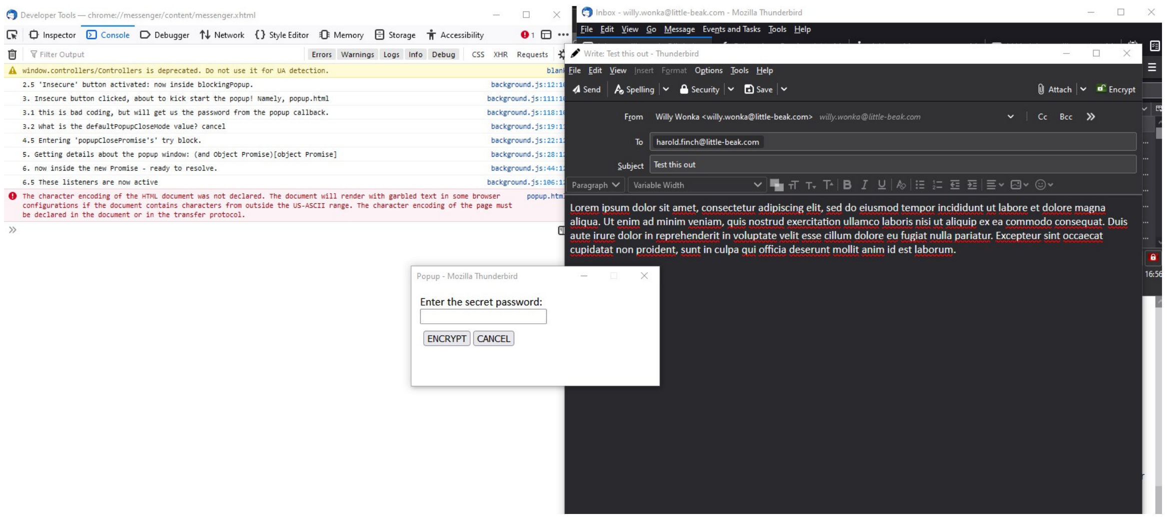
Another interesting component is the actual execution of the encryption, by the SJCL. [2]

[2]Stanford Javascript Cryptography Library

```
let password = popupCloseMode;
console.log("16. Encrypting now! " + sjcl.encrypt(password,
 ↪  details.body));
console.log("12. Random text to encrypt is: " + details.body);
let newBody = sjcl.encrypt(password, details.body);
console.log("17. newBody is " + newBody);
messenger.compose.setComposeDetails(currentComposeTabId, { body: newBody
 ↪  })
```

Figure 3.12: Encryption code

It's usage is simple, it takes two arguments, one with a password, and another with the text to be encrypted.


Developer Tools — chrome://messenger/content/messenger.xhtml
Inspector
Console
Debugger
Network
Style Editor
Memory
Storage
Accessibility
Filter Output
Errors
Warnings
Logs
Info
Debug
CSS
XHR
Requests
Inbox - willy.wonka@little-beak.com - Mozilla Thunderbird
File Edit View Go Message Events and Tasks Tools Help
Write: Test this out - Thunderbird
File Edit View Insert Format Options Tools Help
Send
Spelling
Security
Save
Attach
Encrypt
Paragraph
Variable Width
Popup - Mozilla Thunderbird
ENCRYPT
CANCEL

```
window.controllers/Controllers is deprecated. Do not use it for UA detection.          blank
2.5 'Insecure' button activated: now inside blockingPopup.                       background.js:12:1
3. Insecure button clicked, about to kick start the popup! Namely, popup.html    background.js:111:1
3.1 this is bad coding, but will get us the password from the popup callback.    background.js:118:1
3.2 What is the defaultPopupCloseMode value? cancel                              background.js:19:1
4.5 Entering 'popupClosePromise's' try block.                                    background.js:22:1
5. Getting details about the popup window: (and Object Promise)[object Promise]  background.js:28:1
6. now inside the new Promise - ready to resolve.                                background.js:44:1
6.5 These listeners are now active                                               background.js:106:1
The character encoding of the HTML document was not declared. The document will render with garbled text in some browser   popup.html
configurations if the document contains characters from outside the US-ASCII range. The character encoding of the page must
be declared in the document or in the transfer protocol.
```

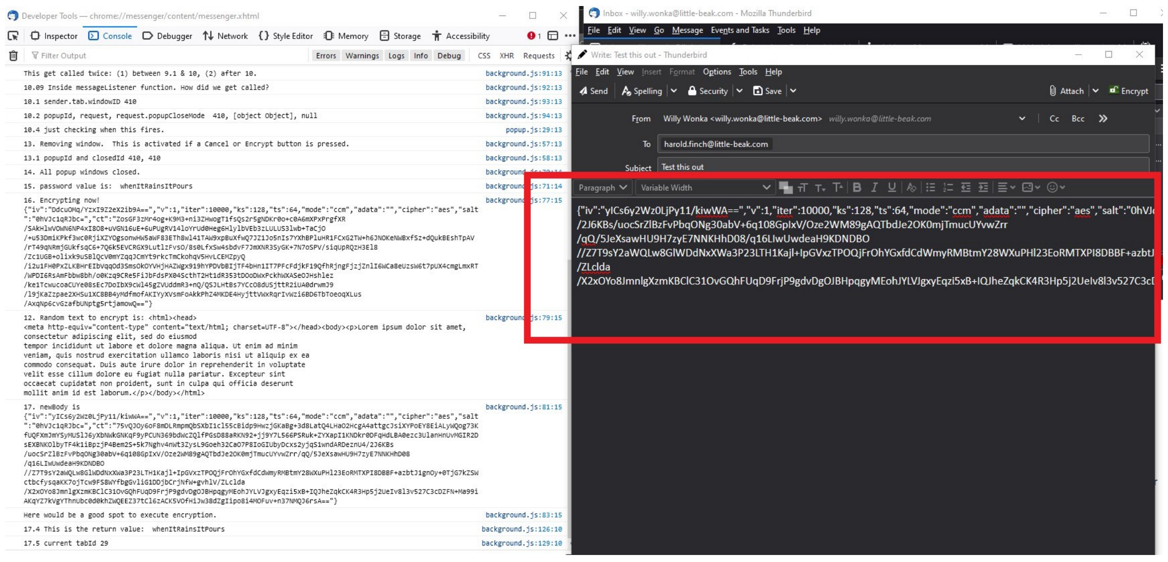
From: Willy Wonka <willy.wonka@little-beak.com> *willy.wonka@little-beak.com*
To: harold.finch@little-beak.com
Subject: Test this out

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Enter the secret password:

Figure 3.13: The popup window, in front of a completed email

The expected and natural behavior for a user is for them to write an email, then click on the "encrypt" button, like so. (See: 3.13), and enter a password.



```
This get called twice: (1) between 9.1 & 10, (2) after 10.                          background.js:91:13
10.09 Inside messageListener function. How did we get called?                        background.js:92:13
10.1 sender.tab.windowID 410                                                         background.js:93:13
10.2 popupId, request, request.popupCloseMode  410, [object Object], null            background.js:94:13
10.4 just checking when this fires.                                                     popup.js:29:13
13. Removing window.  This is activated if a Cancel or Encrypt button is pressed.    background.js:57:13
13.1 popupId and closedId 410, 410                                                   background.js:58:13
14. All popup windows closed.                                                        background
15. password value is:  whenItRainsItPours                                           background js:71:14
16. Encrypting now!                                                                  background js:77:15
{"iv":"DdcuOMq/YzxI9Z2eX2ib9A==","v":1,"iter":10000,"ks":128,"ts":64,"mode":"ccm","adata":"","cipher":"aes","salt
":"0hVJc1qRJbc=","ct":"ZosGF3zMr4og+K9M3+ni3ZHwogT1fsQs2r5gNDKr0o+c0A6mXPxPrgfXR
/SAkHlwVOWN6NP4xI8O8+uVGN16uE+6uPUgRV14loYrUd0Heg6HlylbVEb3zLULUS3lwb+TaCjO
/+u53DmiKPkf3wc0RjiXZYOgsonwHW5aWF83ETh8wl41TAW9xpBuXfwQ7JZ1Jo5nIs7YXhBPluHR1FCxG2TW+h6JNOKeNWBxfSz+dQukBEshTpAV
/rT49qNRmjGUkfsqC6+7Q6k5EVCRGX9LutlzFvsO/8s0LfxSw4sbdvF7JmXNR3SyGK+7N7oSPV/siqUpRQzH3El8
/Zc1UGB+olixk9uSBlQcV0mYZqqJCmYt9rkcTmCkohqV5HvLCEMZpyQ
/i2u1FH0PxZLKBHrEIbVqqOd3SmsOkOYVHjHAZWgx919hYPDVbBIjTF4bHn1IT7PFcFdjkF19QfhRjngFjzjZnlI6WCa8eUzsW6t7pUX4cmgLmxRT
/WPDI6RsAmFbbw8bh/o0Kzq9CRe5FiJbFdsPX04ScthT2Ht1dR353tDOoDWxPckhWXASeOJHshlez
/ke1TcwucoaCUYe08sEc7DoIbX9cWl45gZVUddmR3+nQ/QSJLHtBs7YCcO8dUSjttR2iUA0drwmJ9
/l9jKaZzpae2XHSu1XC8BB4yMdfmofAKIYyXVsmFoAkkPhZ4MKDE4HyjttVWxRqrIvWzi6BD6TbToeoqXLus
/AxqNp6cvGzafbUNptg5rtjamowQ=="}
12. Random text to encrypt is: <html><head>                                          background js:79:15
<meta http-equiv="content-type" content="text/html; charset=UTF-8"></head><body><p>Lorem ipsum dolor sit amet,
consectetur adipiscing elit, sed do eiusmod
tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim
veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea
commodo consequat. Duis aute irure dolor in reprehenderit in voluptate
velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint
occaecat cupidatat non proident, sunt in culpa qui officia deserunt
mollit anim id est laborum.</p></body></html>
17. newBody is                                                                       background.js:81:15
{"iv":"yICs6y2Wz0LjPy11/kiwWA==","v":1,"iter":10000,"ks":128,"ts":64,"mode":"ccm","adata":"","cipher":"aes","salt
":"0hVJc1qRJbc=","ct":"75vQJOy6oF8mDLRmpmQbSXbI1cl55cBidp9HwzjGKaBg+3d8LatQ4LHaO2HcgA4attgcJsiXYPoEY8EiALyWQog73K
fUQFXmJmYSyMUSlJ6yXbNWkGNKqF9yPCUN369bdWcZQlfPGsD88aRKN92+jj9Y7L566PSRuk+ZYXapI1KNDkr0DFqHdLBA0ezc3UlanHnUvMGIR2D
sEXBNKOlbyTF4k1iBpzjP4Bem2S+5k7Nghv4nWt3ZysL9Goeh32CaO7P8IoGIUbyDcxs2yjqS1wndARDeznU4/2J6KBs
/uocSrZlBzFvPbqONg30abV+6q108GpIxV/Oze2WM89gAQTbdJe2OK0mjTmucUYvwZrr/qQ/5JeXsawHU9H7zyE7NNKHhD08
/q16LIwUwdeaH9KDNDBO
//Z7T9sY2aWQLw8GlWDdNxXWa3P23LTH1Kajl+IpGVxzTPOQjFrOhYGxfdCdWmyRMBtmY28WXuPHl23EoRMTXPI8DBBF+azbtJ1gnOy+0TjG7kZSW
ctbcfysqaKK7ojTcw9FS8WYfbgGvliG1DDjbCrjNfW+gvhlV/ZLclda
/X2xOYo8JmnlgXzmKBClC31OvGQhFUqD9FrjP9gdvDgOJBHpqgyMEohJYLVJgxyEqzi5xB+IQJheZqkCK4R3Hp5j2UeIv8l3v527C3cDZFN+Ma99i
AKqYZ7kVgYThnUbc0d0khZWQEEZ37tCl6zACK5VOfHiJw38dZgIipo8i4MOFuv+n37NMQJ6rsA=="}
Here would be a good spot to execute encryption.                                     background.js:83:15
17.4 This is the return value:  whenItRainsItPours                                  background.js:126:10
17.5 current tabId 29                                                               background.js:129:10
```

From Willy Wonka <willy.wonka@little-beak.com> willy.wonka@little-beak.com

To harold.finch@little-beak.com

Subject Test this out

```
{"iv":"yICs6y2Wz0LjPy11/kiwWA==","v":1,"iter":10000,"ks":128,"ts":64,"mode":"ccm","adata":"","cipher":"aes","salt":"0hVJ
/2J6KBs/uocSrZlBzFvPbqONg30abV+6q108GpIxV/Oze2WM89gAQTbdJe2OK0mjTmucUYvwZrr
/qQ/5JeXsawHU9H7zyE7NNKHhD08/q16LIwUwdeaH9KDNDBO
//Z7T9sY2aWQLw8GlWDdNxXWa3P23LTH1Kajl+IpGVxzTPOQjFrOhYGxfdCdWmyRMBtmY28WXuPHl23EoRMTXPI8DBBF+azbtJ
/ZLclda
/X2xOYo8JmnlgXzmKBClC31OvGQhFUqD9FrjP9gdvDgOJBHpqgyMEohJYLVJgxyEqzi5xB+IQJheZqkCK4R3Hp5j2UeIv8l3v527C3cD
```

Developer Tools — chrome://messenger/content/messenger.xhtml
Inspector Console Debugger Network Style Editor Memory Storage Accessibility
Filter Output
Errors Warnings Logs Info Debug CSS XHR Requests
Inbox - willy.wonka@little-beak.com - Mozilla Thunderbird
File Edit View Go Message Events and Tasks Tools Help
Write: Test this out - Thunderbird
File Edit View Insert Format Options Tools Help
Send Spelling Security Save Attach Encrypt
Cc Bcc
Paragraph Variable Width

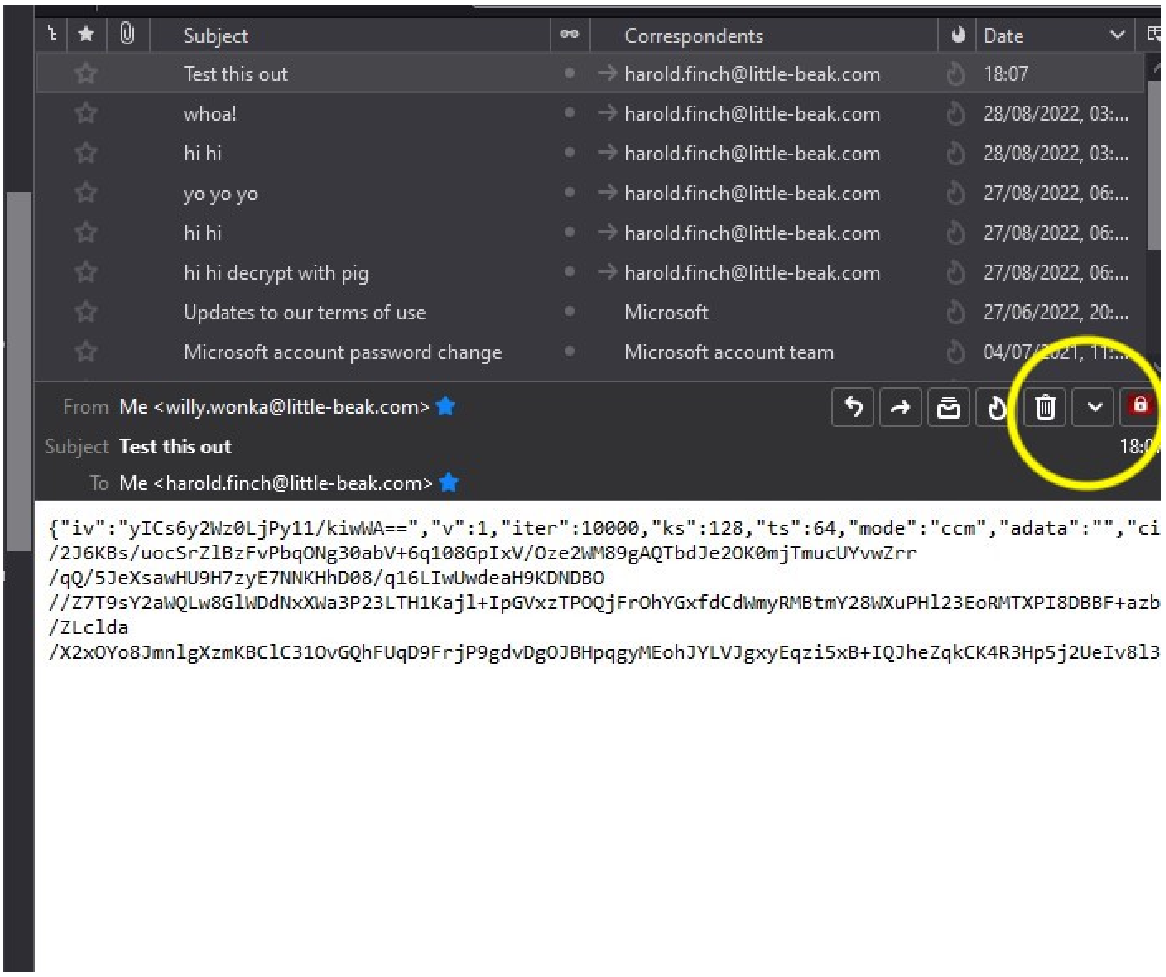
Figure 3.14: Voila! Encrypted text

After that, a password is entered, and the magic happens. (See: 3.14)

Note: the console.log keeping track of where we are, and also verifying that the decryption also works with the same password. From here, the message can be sent as normal, or saved as a draft. There was some developmental pondering, if the encryption button should also send the message in one go, since it's not like the email is really able to be edited or read at this point, but for now left it as is, giving the user the option to add attachments or email recipients.

3.3.5 Decrypt the Message

Now, sending the email to another account, we can see two very interesting elements.



Subject	Correspondents	Date
Test this out	harold.finch@little-beak.com	18:07
whoa!	harold.finch@little-beak.com	28/08/2022, 03:...
hi hi	harold.finch@little-beak.com	28/08/2022, 03:...
yo yo yo	harold.finch@little-beak.com	27/08/2022, 06:...
hi hi	harold.finch@little-beak.com	27/08/2022, 06:...
hi hi decrypt with pig	harold.finch@little-beak.com	27/08/2022, 06:...
Updates to our terms of use	Microsoft	27/06/2022, 20:...
Microsoft account password change	Microsoft account team	04/07/2021, 11:...

From Me <willy.wonka@little-beak.com>
Subject **Test this out** 18:0
To Me <harold.finch@little-beak.com>

```
{"iv":"yICs6y2Wz0LjPy11/kiwWA==","v":1,"iter":10000,"ks":128,"ts":64,"mode":"ccm","adata":"","ci
/2J6KBs/uocSrZlBzFvPbqONg30abV+6q108GpIxV/Oze2WM89gAQTbdJe2OK0mjTmucUYvwZrr
/qQ/5JeXsawHU9H7zyE7NNKHhD08/q16LIwUwdeaH9KDNDBO
//Z7T9sY2aWQLw8GlWDdNxXWa3P23LTH1Kajl+IpGVxzTPOQjFrOhYGxfdCdWmyRMBtmY28WXuPHl23EoRMTXPI8DBBF+azb
/ZLclda
/X2xOYo8JmnlgXzmKBClC31OvGQhFUqD9FrjP9gdvDgOJBHpqgyMEohJYLVJgxyEqzi5xB+IQJheZqkCK4R3Hp5j2UeIv8l3
```

Figure 3.15: Decrypt icon

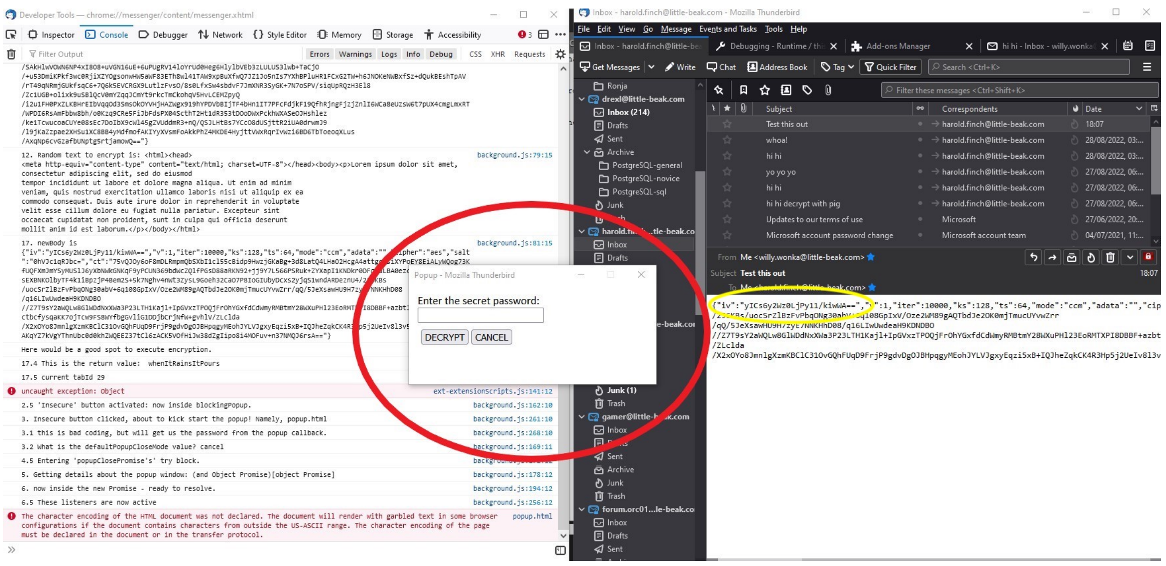
First, there is a "Decrypt" icon on the far right of what is known as the *message_display_area* another UI component of the Thunderbird interface. (See: 3.15)

```
"message_display_action": {
  "default_title": "Decrypt",
  "default_icon": "images/locked_64px.png"
},
"permissions": [
    "compose",
    "messageRead",
        "messagesModify"
]
```

Figure 3.16: Updated manifest.json components

This is another required manifest.json addition, as well as updating the permissions section (See: 3.16 & 3.17).

An interesting encryption component that has not been explained, is the way the encrypted message is displayed. Most of the encrypt text describes the protocols involved, like mode of operation (in this case, CCM, or *Counter with Cipher Block Chaining-Message Authentication Code*), type of cipher used (AES), the version, etc. What leads the block, however, is important. The "iv" or the *Initialization vector*. This is essentially used as a seeding value. And what is so interesting about it is: that even if a would be hacker was given the "iv" value, without the password they would be no closer to decrypting the message. (See 3.17) [Schneier, 2015, p.170]



```
/SAkHlwVOWN6NP4xI8O8+uVGN16uE+6uPUgRV14loYrUd0Heg6HlylbVEb3zLULUS3lwb+TaCjO
/+u53DmiKPkf3wc0RjiXZYOgsonwHW5aWF83ETh8wl41TAW9xpBuXfwQ7JZ1Jo5nIs7YXhBPluHR1FCxG2TW+h6JNOKeNWBxfSz+dQukBEshTpAV
/rT49qNRmjGUkfsqC6+7Q6k5EVCRGX9LutlzFvsO/8s0LfxSw4sbdvF7JmXNR3SyGK+7N7oSPV/siqUpRQzH3El8
/Zc1UGB+olixk9uSBlQcV0mYZqqJCmYt9rkcTmCkohqV5HvLCEMZpyQ
/i2u1FH0PxZLKBHrEIbVqqOd3SmsOkOYVHjHAZWgx919hYPDVbBIjTF4bHn1IT7PFcFdjkF19QfhRjngFjzjZnlI6WCa8eUzsW6t7pUX4cmgLmxRT
/WPDI6RsAmFbbw8bh/o0Kzq9CRe5FiJbFdsPX04ScthT2Ht1dR353tDOoDWxPckhWXASeOJHshlez
/ke1TcwucoaCUYe08sEc7DoIbX9cWl45gZVUddmR3+nQ/QSJLHtBs7YCcO8dUSjttR2iUA0drwmJ9
/l9jKaZzpae2XHSu1XC8BB4yMdfmofAKIYyXVsmFoAkkPhZ4MKDE4HyjttVWxRqrIvWzi6BD6TbToeoqXLus
/AxqNp6cvGzafbUNptg5rtjamowQ=="}

12. Random text to encrypt is: <html><head>                                    background.js:79:15
<meta http-equiv="content-type" content="text/html; charset=UTF-8"></head><body><p>Lorem ipsum dolor sit amet,
consectetur adipiscing elit, sed do eiusmod
tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim
veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea
commodo consequat. Duis aute irure dolor in reprehenderit in voluptate
velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint
occaecat cupidatat non proident, sunt in culpa qui officia deserunt
mollit anim id est laborum.</p></body></html>

17. newBody is                                                                 background.js:81:15
{"iv":"yICs6y2Wz0LjPy11/kiwWA==","v":1,"iter":10000,"ks":128,"ts":64,"mode":"ccm","adata":"" [illegible]ipher":"aes","salt
":"0hVJc1qRJbc=","ct":"75vQJOy6oF8mDLRmpmQbSXbI1cl55cBidp9HwzjGKaBg+3d8LatQ4LHaO2HcgA4attg[illegible]iXYPoEY8EiALyWQog73K
fUQFXmJmYSyMUSlJ6yXbNWkGNKqF9yPCUN369bdWcZQlfPGsD88aRKN92+jj9Y7L566PSRuk+ZYXapI1KNDkr0DF[illegible]LBA0ez
sEXBNKOlbyTF4k1iBpzjP4Bem2S+5k7Nghv4nWt3ZysL9Goeh32CaO7P8IoGIUbyDcxs2yjqS1wndARDeznU4/2[illegible]KBs
/uocSrZlBzFvPbqONg30abV+6q108GpIxV/Oze2WM89gAQTbdJe2OK0mjTmucUYvwZrr/qQ/5JeXsawHU9H7zy[illegible]NNKHhD08
/q16LIwUwdeaH9KDNDBO
//Z7T9sY2aWQLw8GlWDdNxXWa3P23LTH1Kajl+IpGVxzTPOQjFrOhYGxfdCdWmyRMBtmY28WXuPHl23EoRMT[illegible]I8DBBF+azbtJ
ctbcfysqaKK7ojTcw9FS8WYfbgGvliG1DDjbCrjNfW+gvhlV/ZLclda
/X2xOYo8JmnlgXzmKBClC31OvGQhFUqD9FrjP9gdvDgOJBHpqgyMEohJYLVJgxyEqzi5xB+IQJheZqkCK4R[illegible]p5j2UeIv8l3v5
AKqYZ7kVgYThnUbc0d0khZWQEEZ37tCl6zACK5VOfHiJw38dZgIipo8i4MOFuv+n37NMQJ6rsA=="}

Here would be a good spot to execute encryption.

17.4 This is the return value:  whenItRainsItPours

17.5 current tabId 29

uncaught exception: Object                                          ext-extensionScripts.js:141:12
2.5 'Insecure' button activated: now inside blockingPopup.                   background.js:162:10
3. Insecure button clicked, about to kick start the popup! Namely, popup.html background.js:261:10
3.1 this is bad coding, but will get us the password from the popup callback. background.js:268:10
3.2 What is the defaultPopupCloseMode value? cancel                          background.js:169:11
4.5 Entering 'popupClosePromise's' try block.                                [illegible]
5. Getting details about the popup window: (and Object Promise)[object Promise] background.js:178:12
6. now inside the new Promise - ready to resolve.                            background.js:194:12
6.5 These listeners are now active                                           background.js:256:12
The character encoding of the HTML document was not declared. The document will render with garbled text in some browser   popup.html
configurations if the document contains characters from outside the US-ASCII range. The character encoding of the page
must be declared in the document or in the transfer protocol.
```

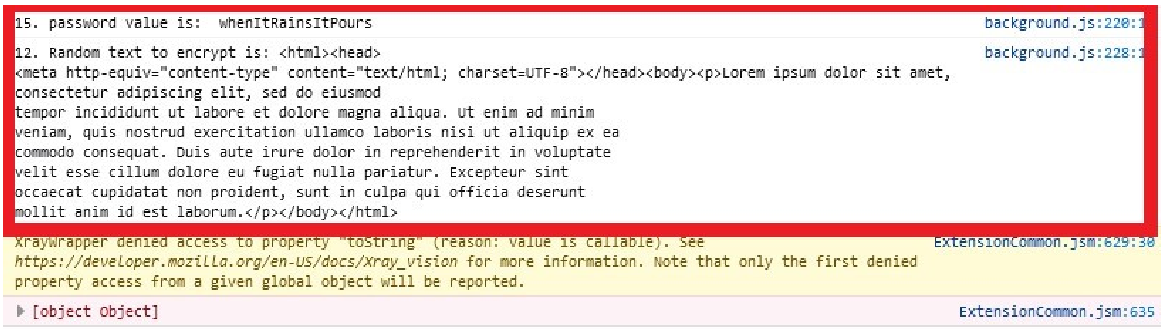
Subject	Correspondents	Date
Test this out	harold.finch@little-beak.com	18:07
whoa!	harold.finch@little-beak.com	28/08/2022, 03:...
hi hi	harold.finch@little-beak.com	28/08/2022, 03:...
yo yo yo	harold.finch@little-beak.com	27/08/2022, 06:...
hi hi	harold.finch@little-beak.com	27/08/2022, 06:...
hi hi decrypt with pig	harold.finch@little-beak.com	27/08/2022, 06:...
Updates to our terms of use	Microsoft	27/06/2022, 20:...
Microsoft account password change	Microsoft account team	04/07/2021, 11:...

From Me <willy.wonka@little-beak.com>
Subject **Test this out** 18:07
To Me <harold.finch@little-beak.com>

```
{"iv":"yICs6y2Wz0LjPy11/kiwWA==","v":1,"iter":10000,"ks":128,"ts":64,"mode":"ccm","adata":"","cip
/2[illegible]KBs/uocSrZlBzFvPbqONg30ab[illegible]6q108GpIxV/Oze2WM89gAQTbdJe2OK0mjTmucUYvwZrr
/qQ/5JeXsawHU9H/zyE/NNKHhD08/q16LIwUwdeaH9KDNDBO
//Z7T9sY2aWQLw8GlWDdNxXWa3P23LTH1Kajl+IpGVxzTPOQjFrOhYGxfdCdWmyRMBtmY28WXuPHl23EoRMTXPI8DBBF+azbt
/ZLclda
/X2xOYo8JmnlgXzmKBClC31OvGQhFUqD9FrjP9gdvDgOJBHpqgyMEohJYLVJgxyEqzi5xB+IQJheZqkCK4R3Hp5j2UeIv8l3v
```

Figure 3.17: Decryption button activating pop-up

The following image displays the activation of the "decrypt button" which displays a similar pop-up window as was shown during the encryption pop-up, with the only difference between "ENCRYPT" or "DECRYPT," naturally. (See 3.17)



```
15. password value is:  whenItRainsItPours                              background.js:220:1

12. Random text to encrypt is: <html><head>                             background.js:228:1
<meta http-equiv="content-type" content="text/html; charset=UTF-8"></head><body><p>Lorem ipsum dolor sit amet,
consectetur adipiscing elit, sed do eiusmod
tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim
veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea
commodo consequat. Duis aute irure dolor in reprehenderit in voluptate
velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint
occaecat cupidatat non proident, sunt in culpa qui officia deserunt
mollit anim id est laborum.</p></body></html>
XrayWrapper denied access to property "toString" (reason: value is callable). See   ExtensionCommon.jsm:629:30
https://developer.mozilla.org/en-US/docs/Xray_vision for more information. Note that only the first denied
property access from a given global object will be reported.
▶ [object Object]                                                       ExtensionCommon.jsm:635
```

Figure 3.18: Console showing the decrypted text

Finally, in the console.log, we are able to see that from the receiving end, we are able to decrypt the emailed text when the correct password is given. (See: 3.18)

Ultimaetly, however, the implementation was not completed. The developer wasn't not quite able to get the decrypted code to replaced the cipher code in the body of the message in a reliable way. Further development will continue to resolve this. Unfortunately, the Mozilla API is a living, breathing, in progress project, so the various elements do not behave in a uniform manner. Namely, the API that worked on the compose window did not have a corresponding message window API.

Chapter 4

Challenges Encountered

4.1 Live development environment

There were many challenges encountered, especially at the start. First, the developer wanted to work through the four available Thunderbird add-on development beginner tutorials to build an understanding of how the basics worked. There was only one problem, after two simple "hello world" level tutorials, the last two did not work. A week or so of trying to figure it out, far too long, before I finally asked for help. There were bugs. Thus, I became acquainted with the Mozilla bug tracker, *Bugzilla*. Even though Thunderbird is considered to be an agnostic piece of software, that should work in Windows, Linux and Apple operating systems, it apparently is not always the case. Development is chiefly done in Windows, and bugs for other systems are worked out over time. One of the most helpful developers, John Bieling, mentioned to me that, he himself, used Windows, and that he did not even have access to a Apple computer to debug some of the issues.

The developer's preferred platform is Apple OS, followed closely by Linux Mint. However, both proved to be unreliable and I had to fallback to a Windows system, which became the system I used for this project. Even though, sometimes, during this process, I would revert to Apple or Linux platforms, ultimately, I came to the grim realization that it was not advisable, because I could code something – 100% as it should be – and it wouldn't work.

4.2 Backlog

There were many areas where the developer's efforts were incomplete, in one regard or another. In some cases, it was simply a juvenile and painful oversight at the start. There were many components that should have been included in the original specifications that the developer simply did not consider. This included common features of email clients that were completely overlooked:

1. How to handle decrypted messages:
   - How long do they stay decrypted, after they have been decrypted?

- Do they re-encrypt after a certain amount of time?

There are a slew of other issues that only became evident to the developer *after* the development process began. Small things could be immediately resolved–or turned out not to be issues at all (like handling messages that fall into the draft window). But, many other things were not included in the original specifications–so the developer stayed the course, and remained focused on meeting the minimum specified requirements, rather than stray too far in different directions.

These are additional items that could be added, improved or enhanced in the future:

- Password handling
  - minimum requirement
  - character restrictions
  - empty character considerations – like an empty password

Lastly, the code needs to be re-factored to be cleaner. The developer concedes some redundancy.

Chapter 5

Summary

5.1 Retrospective

First, the developer has to acknowledge that *now* he feels like he knows Mozilla's Thunderbird API. Additionally, the experience and knowledge in Javascript has expanded by leaps and bounds. So, professionally, the experience has been of significant importance for someone interesting in calling themselves a professional in internet technologies.

This was the developer's very first full fledged project where he was not supported, by other more talented programmers. Additionally, it marked the first time that the developer worked in a real-time environment with other developers. I was not intimidated by this, at the time, because I did not know any better. Only later, when I saw developers arguing about something (stylistic), with one throwing down his "...in 40 years of software engineering experience" card, to which the respondent conceding defeat with his own "...that he only had 30+ years of software engineering experience" card. I knew then, I was swimming with the sharks. Fortunately, it came towards the end of my project.

Regrettably, the implementation is not complete. It falls just short of meeting the specifications defined by the developer at the beginning, and further short of what should be expected in a complete turn-key solution. There were a number of things that the developer did not foresee before development began which, in itself, is a failing that inhibited a robust solution.

5.2 Next steps

Nevertheless, the developer intends to smooth out the project, enhance shortcomings, and use it as an example of what the developer is fully capable of. The developer can take a break from academic writing, and just focus on hacking, debugging, testing, refining, and re-factoring.

Chapter 6

Software Requirements 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:

AES Advanced Encryption Standard.

API application programming interface.

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

CBC Cipher Block Chaining, a AES encryption mode.

CFB Cipher Feedback Mode, a AES encryption mode.

cipher text The encrypted text.

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

CTR Counter Mode, a AES encryption mode.

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

ECB Electronic Codebook, a AES encryption mode.

extensions An extension adds features and functions to a browser.

IEEE Institute of Electrical and Electronics Engineers.

JS JavaScript.

OFB Output Feedback Mode, a AES encryption mode.

plain text The text that we wish to encrypt.

SRS Software Requirements Specification.

symmetric encryption Encryption that only uses one key for encryption.

6.1.4 Software Requirements References

Author used the IEEE document:

1. IEEE Std 803-1998

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

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. 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 to this proposal do have the advantage that they use asymmetric 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 some security.

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

System interfaces

The 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 like Debian 10+ or Mint 21+
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 and tested on Windows 10, Linux Cinnamon Mind 21, and macOS Monterey.

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.

6.2.2 User characteristics

See Personas in the appendix.

6.2.3 Constraints

There will be various constraints within this project listed below:

- Security: It will not be possible to account for all attack vectors. Thus, they will not be explored at this time.

6.3 Specific Requirements

6.3.1 Use Cases

The Use Cases used in this project will be defined, and 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 will not describe unanticipated errors 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:	1
Actor(s):	Alice
Description:	Alice will encrypt an email
Preconditions: 1. Thunderbird Email client installed. 2. Thunderbird Email client configured to send and receive emails. 3. Add-on installed. 4. Email written	
Basic Flow: 1. Alice writes an email in Thunderbird. 2. Alice locates and clicks on the "encrypt" button. 3. A pop-up window opens. 4. Alice is prompted to enter a password. 5. Alice enters a password. 6. Alice clicks on the Encrypt button. 7. The pop-up window closes. 8. The email is encrypted. 9. Alice is returned to the compose window. 10. Alice sends the email.	
Exceptions: 1. N/A	
Postconditions: 1. The email is encrypted. 2. The add-on window closes. 3. Alice is returned to the compose window. 4. Alice shares a password with Bob: *offline*.	

Use Case ID:	2
Actor(s):	Bob
Description:	Bob decrypts an email
Preconditions: 1. Thunderbird email client installed. 2. Thunderbird email client configured to send and receive emails. 3. Add-on installed. 4. Email received that has been encrypted by our add-on 5. Bob gets the password from Alice: *offline*.	
Basic Flow: 1. Bob opens his email client. 2. Bob notices that he has received a new email. 3. Bob observes that the email is encrypted. 4. Bob clicks on the "decrypt" button located in the message window. 5. Bob observes a pop-up window open. 6. Bob enters the password. 7. Bob clicks on decrypt button.	

8. Bob observes the pop-up window close.
9. Bob observes the body of the message in plaintext.
Exceptions:
1. N/A
Postconditions:
1. The email is decrypted and can be read.
2. The add-on pop-up closes.
3. Bob is returned to the Thunderbird client.

Use Case ID:	3
Actor(s):	Alice
Description:	Alice cancels email encryption
Preconditions:	
1. Thunderbird Email client installed.	
2. Thunderbird Email client configured to send and receive emails.	
3. Add-on installed.	
4. Email written.	
Basic Flow:	
1. Alice writes an email in Thunderbird.	
2. Alice locates and clicks on the ”encrypt” button.	
3. A pop-up window opens.	
4. Alice is prompted to enter a password.	
5. Alice clicks on the Cancel button.	
6. The pop-up window closes.	
7. The email is *not* encrypted.	
8. Alice is returned to the compose window.	
Exceptions:	
1. N/A	
Postconditions:	
1. The email is *not* encrypted.	
2. The add-on window closes.	
3. Alice is returned to the compose window.	

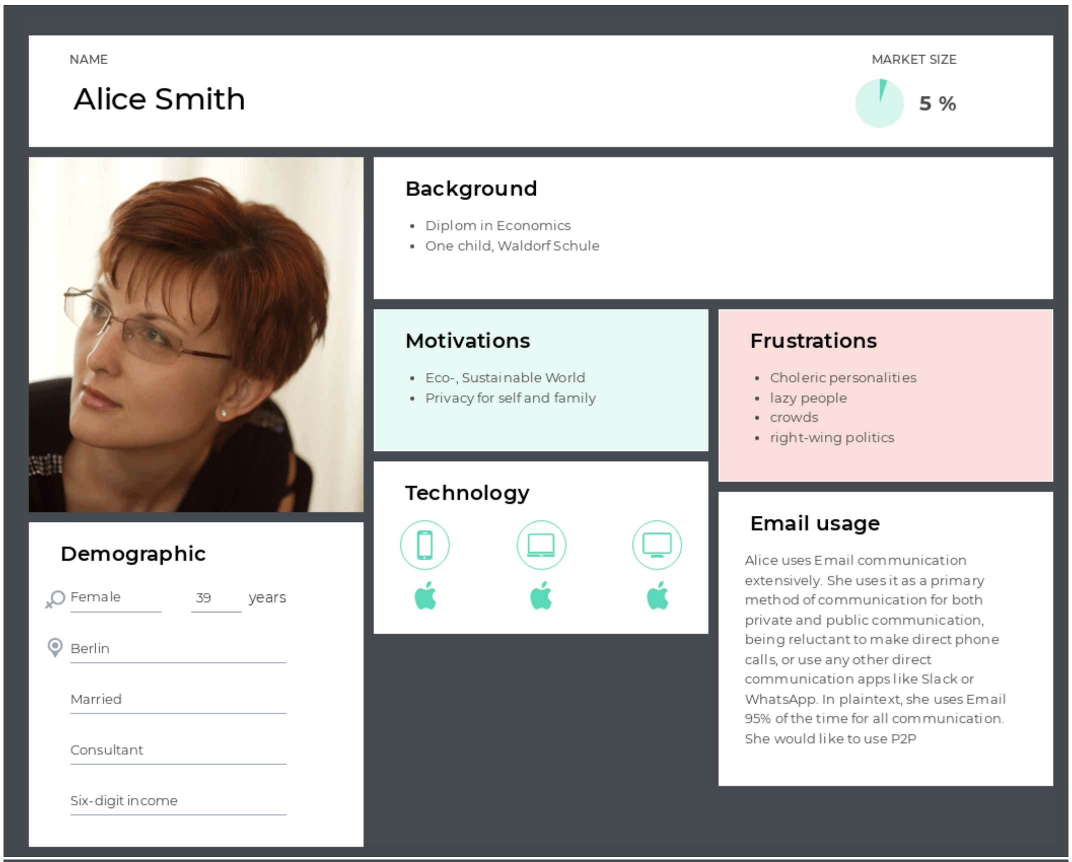
Use Case ID:	4
Actor(s):	Alice
Description:	Alice saves encrypted draft
Preconditions:	
1. Thunderbird Email client installed.	
2. Thunderbird Email client configured to send and receive emails.	
3. Add-on installed.	
4. Email written	
Basic Flow:	
1. Alice writes an email in Thunderbird.	
2. Alice locates and clicks on the ”encrypt” button.	

3. A pop-up window opens.
4. Alice is prompted to enter a password.
4. Alice enters a password.
5. Alice clicks on the Encrypt button.
6. The pop-up window closes.
7. The email is encrypted.
8. Alice is returned to the compose window.
9. Alice saves the email.
10. Alice observes the encrypted email saved as a draft.

Exceptions:
1. N/A

Postconditions:
1. The email is encrypted.
2. The add-on window closes.
3. Alice observes the email saved as a draft.
4. Alice is returned to the compose window.

6.3.2 Personas



NAME

Alice Smith

MARKET SIZE

5 %

Background

- Diplom in Economics
- One child, Waldorf Schule

Motivations

- Eco-, Sustainable World
- Privacy for self and family

Frustrations

- Choleric personalities
- lazy people
- crowds
- right-wing politics

Technology

Demographic

Female 39 years

Berlin

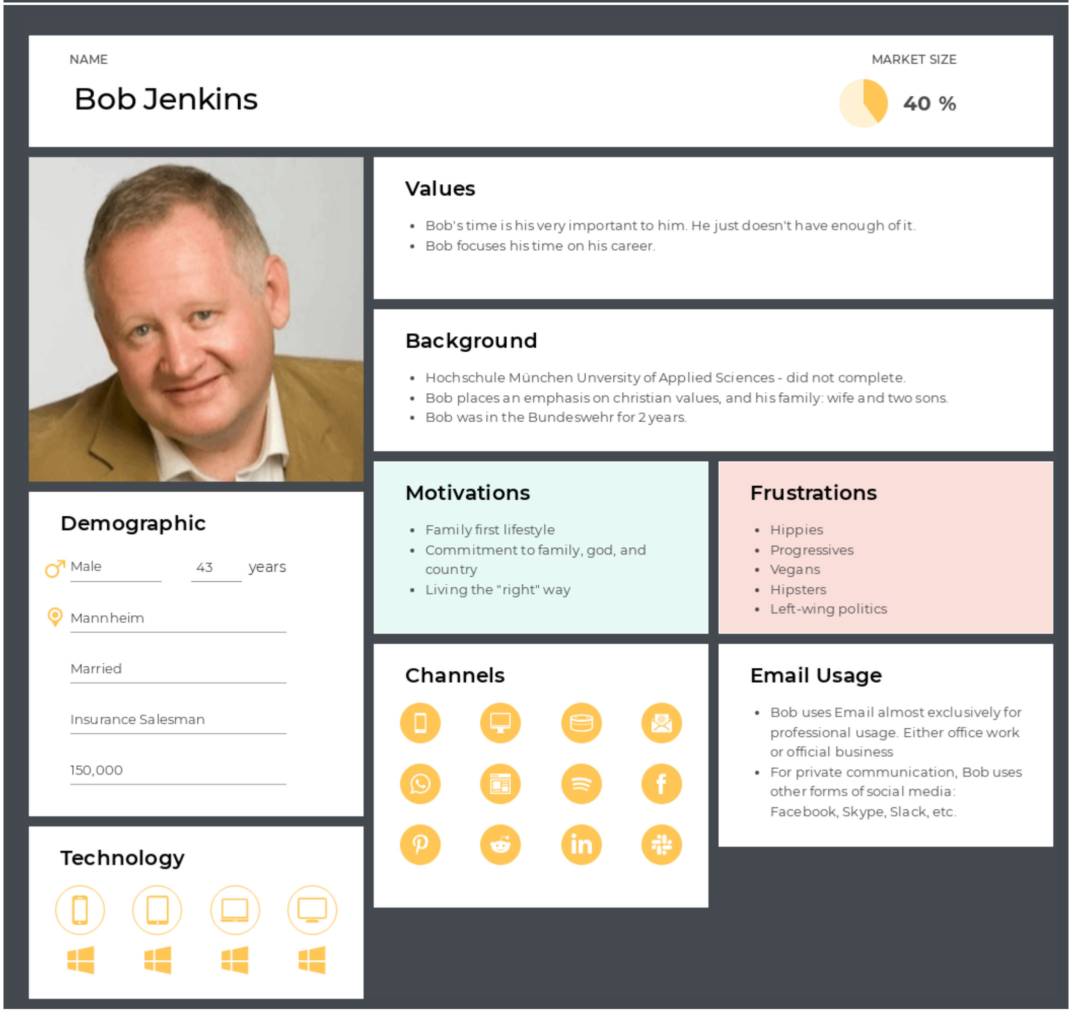
Married

Consultant

Six-digit income

Email usage

Alice uses Email communication extensively. She uses it as a primary method of communication for both private and public communication, being reluctant to make direct phone calls, or use any other direct communication apps like Slack or WhatsApp. In plaintext, she uses Email 95% of the time for all communication. She would like to use P2P



NAME

Bob Jenkins

MARKET SIZE

40 %

Values

- Bob's time is his very important to him. He just doesn't have enough of it.
- Bob focuses his time on his career.

Background

- Hochschule München Unversity of Applied Sciences - did not complete.
- Bob places an emphasis on christian values, and his family: wife and two sons.
- Bob was in the Bundeswehr for 2 years.

Motivations

- Family first lifestyle
- Commitment to family, god, and country
- Living the "right" way

Frustrations

- Hippies
- Progressives
- Vegans
- Hipsters
- Left-wing politics

Demographic

Male 43 years

Mannheim

Married

Insurance Salesman

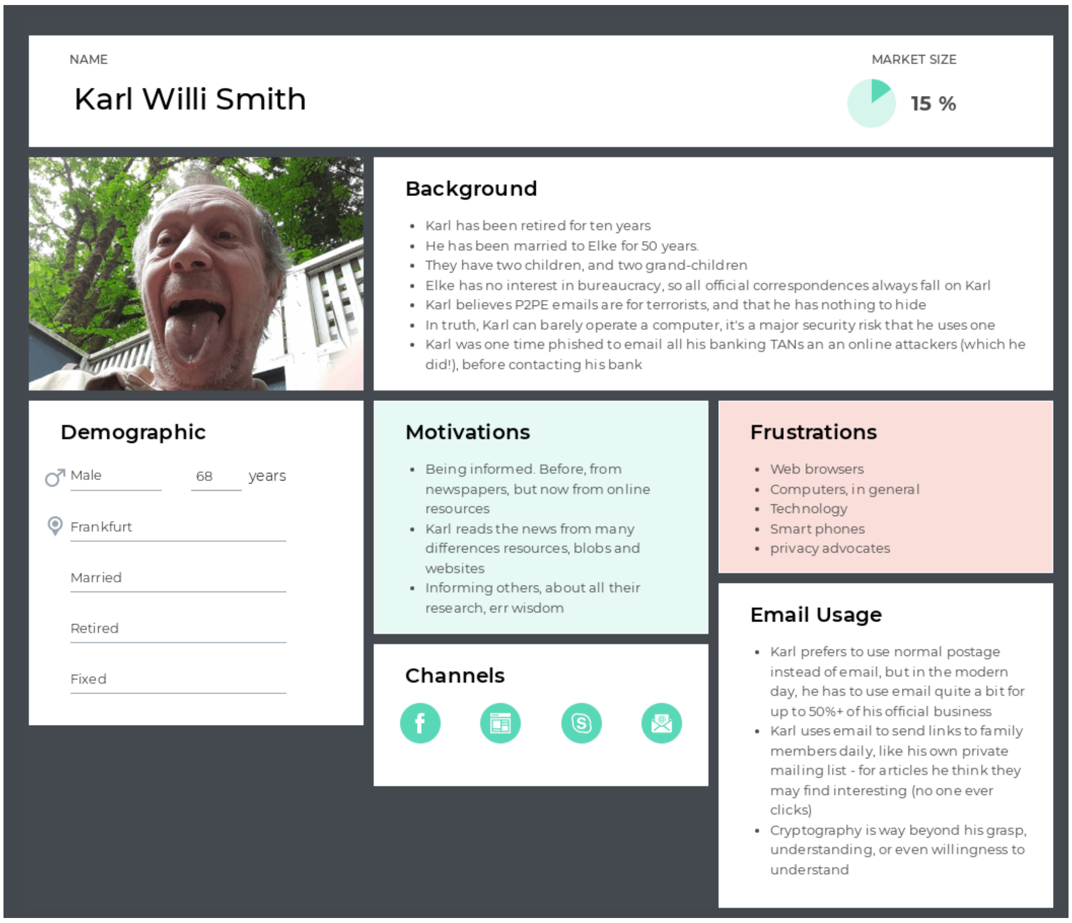
150,000

Channels

Email Usage

- Bob uses Email almost exclusively for professional usage. Either office work or official business
- For private communication, Bob uses other forms of social media: Facebook, Skype, Slack, etc.

Technology



NAME

Karl Willi Smith

MARKET SIZE

15 %

Background

- Karl has been retired for ten years
- He has been married to Elke for 50 years.
- They have two children, and two grand-children
- Elke has no interest in bureaucracy, so all official correspondences always fall on Karl
- Karl believes P2PE emails are for terrorists, and that he has nothing to hide
- In truth, Karl can barely operate a computer, it's a major security risk that he uses one
- Karl was one time phished to email all his banking TANs an an online attackers (which he did!), before contacting his bank

Demographic

Male	68 years
Frankfurt	
Married	
Retired	
Fixed	

Motivations

- Being informed. Before, from newspapers, but now from online resources
- Karl reads the news from many differences resources, blobs and websites
- Informing others, about all their research, err wisdom

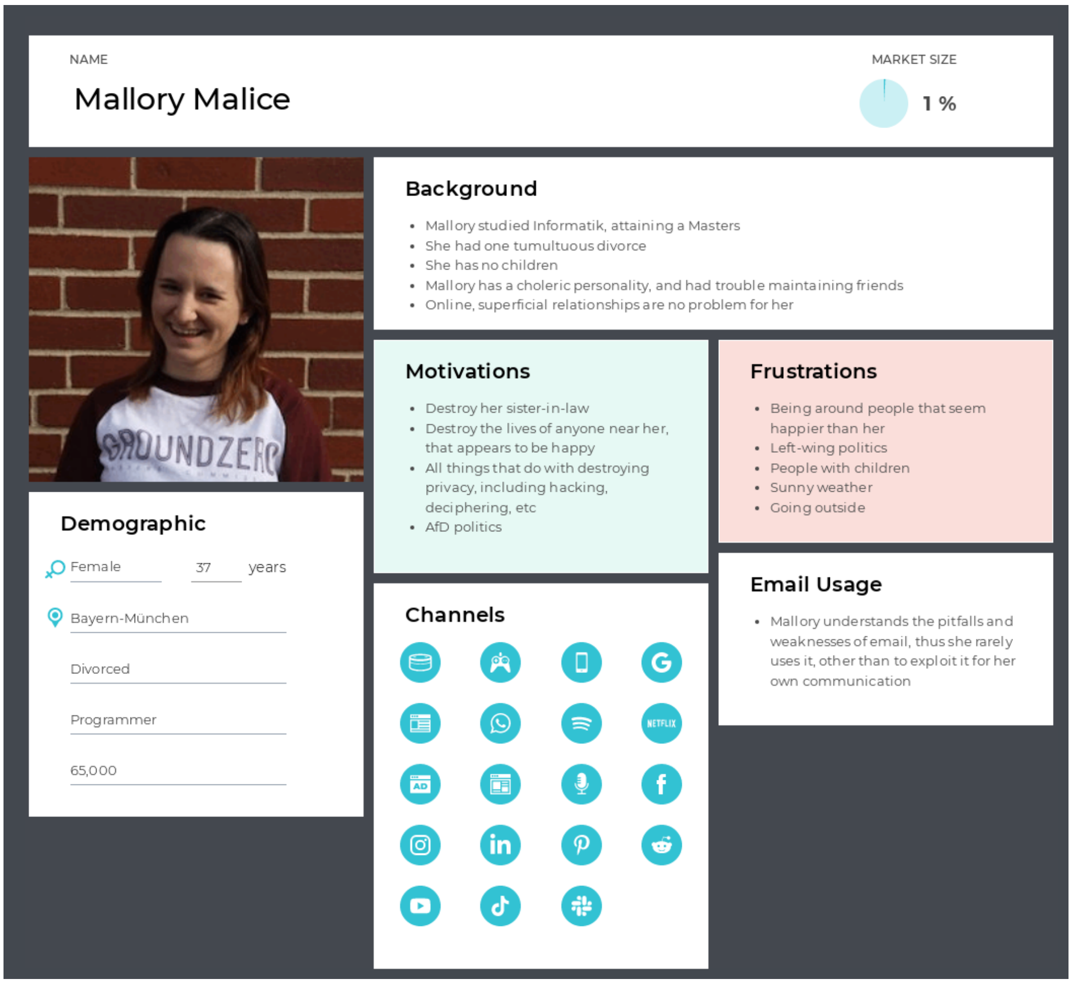
Frustrations

- Web browsers
- Computers, in general
- Technology
- Smart phones
- privacy advocates

Channels

Email Usage

- Karl prefers to use normal postage instead of email, but in the modern day, he has to use email quite a bit for up to 50%+ of his official business
- Karl uses email to send links to family members daily, like his own private mailing list - for articles he think they may find interesting (no one ever clicks)
- Cryptography is way beyond his grasp, understanding, or even willingness to understand



NAME

Mallory Malice

MARKET SIZE

1 %

Background

- Mallory studied Informatik, attaining a Masters
- She had one tumultuous divorce
- She has no children
- Mallory has a choleric personality, and had trouble maintaining friends
- Online, superficial relationships are no problem for her

Motivations

- Destroy her sister-in-law
- Destroy the lives of anyone near her, that appears to be happy
- All things that do with destroying privacy, including hacking, deciphering, etc
- AfD politics

Frustrations

- Being around people that seem happier than her
- Left-wing politics
- People with children
- Sunny weather
- Going outside

Demographic

Female 37 years

Bayern-München

Divorced

Programmer

65,000

Channels

Email Usage

- Mallory understands the pitfalls and weaknesses of email, thus she rarely uses it, other than to exploit it for her own communication

References

[Aumasson, 2017] Aumasson J., *Serious cryptography: A practical introduction to modern encryption*, No Starch Press Inc, San Francisco, California, U.S.A, 2017

[Bieling, 2020] Bieling, J., *"Thundernest/sample-extensions: Example extensions for Thunderbird webextensions apis."*, GitHub., Document Search and Retrieval Page, Retrieved August 29, 2022, from https://github.com/thundernest/sample-extensions

[Crawford, 2019] Crawford D., *"AES Encryption: Everything You Need to Know about AES."*, ProPrivacy.com, Published 4 Feb. 2019, Document Search and Retrieval Page, Retrieved August 29, 2022, from https://proprivacy.com/guides/aes-encryption

[Delfs and Knebl, 2007] Delfs H. and H. Knebl, *Introduction to Cryptography: Principles and Applications*, Springer Verlag, Berlin, Germany, 2007

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

[Katz & Lindell, 2008] Katz J., & Lindell Y., *Introduction to modern cryptography: Principles and protocols*, Chapman & Hall/CRC Press, Boca Raton, London, New York, 2008

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

[Mozilla, 2022] Thunderbird Add-on Team., *"Introduction to Add-on Development."*, Thunderbird, Mozilla, Document Search and Retrieval Page, Retrieved August 29, 2022, https://developer.thunderbird.net/add-ons/mailextensions.

[Nirula, 2022] Nirula U., *"Block Cipher — Purpose, Applications & Examples"*, Document Search and Retrieval Page, Retrieved August 29, 2022, https://study.com/learn/lesson/block-cipher-purpose-applications.html

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

[Paar, 2014] Paar, C., *"Lecture 7: Introduction to Galois Fields for the AES by Christof Paar."*, YouTube, Published 30 Jan. 2014, Document Search and Retrieval Page, Retrieved August 29, 2022, https://www.youtube.com/watch?v=x1v2tX4_dkQ.

[Schneier, 2015] Schneier B., *Applied Cryptography: Protocols, Algorithms, and Source Code in C*, Wiley, Indianapolis, Indiana, U.S.A, 2015.

[Shirey, 2007] Shirey, R., *"RFC 4949 - Internet Security Glossary, Version 2."*, Document Search and Retrieval Page, Aug. 2007, Retrieved August 29, 2022, https://datatracker.ietf.org/doc/html/rfc4949.

[Stanford Security Lab, 2009] Stanford Security Lab, *"Stanford Javascript Crypto Library (SJCL)."*, SJCL: a Javascript Crypto Library, Stanford University, California, U.S.A., Document Search and Retrieval Page, Retrieved August 29, 2022, https://crypto.stanford.edu/sjcl/.