Bachelor thesis
Computing Science

Radboud University

E-mail phishing prevention proposal: CEPP

Author:
Lars Jeurissen
s1022856
lars.jeurissen@student.ru.nl

First supervisor/assessor:
dr. ir. B.J.M. Mennink
B.Mennink@cs.ru.nl

Second assessor:
prof. dr. J.J.C. Daemen
J.Daemen@cs.ru.nl

June 1, 2021

Abstract

One of the biggest problems in e-mail communication is phishing. Thousands of people lose money to phishing attacks every year. Although many techniques have been created that attempt to stop phishing attacks, none are very effective, and most techniques miss a lot of phishing e-mails.

In this thesis, we present CEPP: Certificate-based E-mail Phishing Prevention. CEPP is a new, simple protocol that can significantly reduce the number of phishing attacks. CEPP stops phishing attacks by requiring each e-mail to contain a certificate signed by a 'Mail Authority' (MA). This certificate signs the domain, along with some additional parameters. Upon receiving an e-mail, a user can then verify the CEPP certificate, and therefore conclude that the domain from which the mail is sent can (in principle) be trusted. If a mail with a CEPP certificate turns out to be a phishing mail, i.e., the certificate should not have been attached to this mail, users can report the e-mail to the responsible MA, who can decide to remove the e-mail sender's trust. This way, an environment will be created where the MA keeps a timely view on which domains can be trusted. We develop CEPP by first giving an overview of the current status of phishing attacks in e-mail. Then, we will provide the specification of the CEPP protocol. We also provide a proof of concept implementation in the form of a Thunderbird add-on.

Contents

1 Introduction **4**
- 1.1 What is phishing? . . . 4
- 1.2 The impact of phishing attacks . . . 5
- 1.3 Types of phishing . . . 6
- 1.4 Attempts to prevent phishing . . . 6
- 1.5 Goal . . . 7
- 1.6 Proposal and outline . . . 7

2 Preliminaries: E-mail and phishing **8**
- 2.1 Basic e-mail terminology . . . 8
- 2.2 Why phishing attacks are so effective . . . 9
  - 2.2.1 Trustworthy sources . . . 9
  - 2.2.2 Money . . . 10
  - 2.2.3 Speed . . . 11
- 2.3 Types of phishing attacks . . . 11
  - 2.3.1 Mass e-mails . . . 11
  - 2.3.2 Spear phishing . . . 12
  - 2.3.3 Whaling . . . 12
  - 2.3.4 Business e-mail compromise . . . 12
  - 2.3.5 Clone phishing . . . 12
- 2.4 Techniques to stop phishing attacks . . . 13
  - 2.4.1 Communication level . . . 13
  - 2.4.2 Target level . . . 17
  - 2.4.3 User level . . . 19

3 Preliminaries: Certificates **20**
- 3.1 Asymmetric cryptography . . . 20
- 3.2 Public key certificates . . . 21
  - 3.2.1 Certificate chains . . . 21
- 3.3 Certificate validation levels . . . 22
  - 3.3.1 TLS . . . 22
  - 3.3.2 S/MIME . . . 23
- 3.4 Certificate problems . . . 23

3.4.1 Advantages . . . 24
3.4.2 Disadvantages . . . 24
3.4.3 Extended Validation Certificates . . . 24

4 Requirements and goals **26**
4.1 Functional requirements . . . 26
4.2 Non-functional requirements . . . 27
4.3 Security . . . 27
4.3.1 Trust model . . . 27
4.3.2 Attacker model . . . 27
4.3.3 Security goals . . . 28

5 Proposal **29**
5.1 Certificates . . . 29
5.2 The certificate authorities . . . 30
5.2.1 Problem with existing CAs . . . 30
5.2.2 Mail Authorities . . . 30
5.3 Certificate levels . . . 31
5.3.1 Level 1: No certificate . . . 31
5.3.2 Level 2: Basic certificate . . . 31
5.3.3 Level 3: Extended certificate . . . 31
5.4 Removing company trust . . . 32
5.4.1 Reporting spam . . . 32
5.4.2 Remarks . . . 32

6 Specification **33**
6.1 Digital Signatures . . . 33
6.1.1 Signature scheme . . . 33
6.2 Certificate generation . . . 34
6.2.1 Certificate data . . . 34
6.2.2 Certificate signature . . . 36
6.2.3 Example . . . 37
6.3 Certificate validation . . . 39
6.3.1 Validating the DMARC . . . 39
6.3.2 Validating the CEPP signature . . . 39
6.3.3 Validating the CEPP data . . . 39
6.3.4 Example . . . 40
6.4 Spam report generation . . . 40
6.4.1 Overview . . . 41
6.5 Spam report validation . . . 41
6.5.1 Report filtering . . . 42
6.6 CEPP usage locations . . . 42

7 Proof of concept **44**
7.1 The CEPP addon . . . 44
7.1.1 The basic functionality . . . 44
7.1.2 Development assumptions . . . 45
7.2 Usage showcase . . . 45
7.2.1 Generating a certificate . . . 45
7.2.2 Adding a certificate . . . 45
7.2.3 Validating a certificate . . . 46
7.2.4 Reporting an e-mail . . . 46

8 Discussion **48**
8.1 Evaluation of functional requirements . . . 48
8.2 Evaluation of non-functional requirements . . . 48
8.3 Overview of advantages and disadvantages . . . 49
8.3.1 Advantages . . . 49
8.3.2 Disadvantages . . . 49

9 Conclusion **51**
9.1 Future research . . . 51

A Thunderbird addon code **56**
A.1 Project structure . . . 56
A.1.1 The 'API' part . . . 57
A.1.2 The 'cepp' part . . . 57
A.1.3 The 'data' part . . . 57
A.2 Code files . . . 58
A.2.1 background.html . . . 59
A.2.2 compose_popup.html . . . 60
A.2.3 manifest.json . . . 62
A.2.4 certificate.js . . . 63
A.2.5 compose.js . . . 66
A.2.6 display.js . . . 70
A.2.7 ca_private_keys.js . . . 75
A.2.8 trusted_ca_public_keys.js . . . 76

Chapter 1

Introduction

The date is June 26th, 2014. Keith McMurtry, the corporate controller of the American company 'Scoular', received an e-mail from company CEO Chuck Elsea. The instructions were simple; McMurtry was asked to transfer $780.000 to a bank in China for a Chinese company's takeover. McMurtry was requested to 'only communicate (...) through this e-mail, in order for us not to infringe SEC regulations'. Additionally, McMurtry received an e-mail from one of the company's accounting firm employees, telling him where to send the money to. This e-mail also contained a phone number that McMurtry called to make sure the transaction was valid. The call was answered by someone with the correct name, indicating that the transfer was indeed legitimate. That day, Keith McMurtry transferred 780.000 dollars to the Shanghai Pudong Development Bank. The next day, McMurtry received a second e-mail, asking him to transfer 7 million more dollars, which McMurtry did. Three days later, another e-mail came in with the request to transfer an additional 9.4 million dollars. McMurtry wired the money.

Over four days, Scoular lost 17.2 million dollars. The e-mails were illegitimate, as was the phone number. The e-mails were not sent by the company CEO and an employee from the company's accounting firm but by two e-mail addresses that were later resolved to a Russian server. The phone number was also not legitimate; McMurtry's call was answered by one of the attackers. Scoular fell victim to a spear-phishing attack, a type of phishing attack where the attackers first spend a lot of time getting to know their victim. This attack could have easily been prevented, but it happened anyway [Jou15] [Mun15].

1.1 What is phishing?

First things first. What is actually meant by the term 'phishing'? The term 'phishing' was first recorded in 1995 in a cracking toolkit created by

Rekouche [WAB16], as a combination of the words 'fishing' and 'phreaking'. In 2006, Dhamija et al. defined phishing as 'the practice of redirecting users to fraudulent web sites' [DTH06]. One year later, in 2007, Jagatic et al. have described phishing as '(...) a form of deception in which an attacker attempts to fraudulently acquire sensitive information from a victim by impersonating a trustworthy entity.' [JJJM07]. Then in 2019, Sumner and Yuan described phishing as the act in which 'an attacker poses as a trustworthy source in an attempt to have the victim release personal or private information.' [SY19]. Additionally, non-scientific (online) dictionaries and other types of non-scientific sources (e.g., websites) have a pretty distributed opinion on the actual definition of phishing.

In general, there is no consensus on what exactly phishing means. Most definitions say phishing is fraudulent and has to do something with someone posing as a trustworthy entity, but not all definitions contain this information. In this thesis, we will be defining phishing as follows:

Phishing is the fraudulent act of attempting to get private information from a victim, via online communication, by posing as a trustworthy entity.

This definition is a combination of most other definitions. According to us, this definition represents phishing accurately and succinctly.

1.2 The impact of phishing attacks

Phishing is a huge problem in today's society. Millions of people and companies fall victim to it every year. According to Verizon's 2020 Data Breach Investigation Report, 25% of all data breaches involved phishing attacks [Ver20]. Furthermore, 94% of all malware is delivered using e-mail. Symantec's Internet Security Threat Report 2019 claims that 65% of all attack groups used spear-phishing as the primary infection vector [Sym19]. The end goal is usually to install malicious software, with a quoted 96% of e-mails being focused on gathering intelligence.

In 2020, phishing attacks reached new heights. The Anti-Phishing Working Group (APWG) Phishing Activity Trends Report reported over 165.000 phishing sites detected in the first quarter of this year and over 145.000 in the second quarter [APW20]. Although this thesis focuses on phishing e-mails, these numbers are still relevant, since many phishing attacks attempt to get victims to go to a phishing website. The report claims a quantity of phishing attacks that have not been seen since 2016. This most likely has to do something with the COVID-19 virus since many COVID-19 themed phishing attacks have started since then. On top of this, phishing attacks are becoming more and more sophisticated. According to the same report,

78% of all phishing sites now use TLS. People have been thought that 'a lock icon in the browser means that the website is secure'. Although the connection to the website might be secure, the website itself can still be malicious. Since so many phishing websites now use TLS, people are easier tricked into believing that they can trust these websites.

1.3 Types of phishing

Phishing can be conducted over multiple communication media. The most prominent are e-mail-, phone-, and text-based phishing. In this section, we will briefly discuss phishing via these three communication media. In the next section, we will give a general overview of attempts to prevent phishing.

1. E-mail phishing is the most common type of phishing attack. In e-mail phishing, an e-mail is sent to the victim, which will usually attempt to get victims to click a link to a malicious website, where the attackers will try to get the victim's login credentials for a real website.

2. Phone call phishing (also called 'voice phishing' or 'vishing') is a way of phishing where the attacker will call the victim and will attempt to get a victim's PIN code, social security number, or other private information.

3. Text message phishing (also called 'SMS-phishing' or 'smishing') is a way of phishing where the attacker attempts to get the victim to click a link to a website via a text message. Usually, SMS messages are used, but other text platforms like WhatsApp are also possible.

There are more types of phishing attacks, like attacks via social media or search engine optimization, but they are not used very often.

1.4 Attempts to prevent phishing

Phishing can be prevented on multiple levels in multiple ways. Some techniques rely on the user's ability to recognize phishing, whereas others try to prevent the phishing attack from ever reaching the user. The techniques to stop phishing attacks can be roughly divided into three categories (levels):

1. The communication level. This level represents the actual communication medium used between the attacker and the victim. Usually, the medium is e-mail, but phone calls and text messages are possible as well. A common measure to stop attacks at the communication level is a spam filter, but this only works for e-mail-based phishing attacks.

2. The target level. This level represents the place where users will give their private information to an attacker. This place is almost always a malicious website. Standard measures to stop these attacks include browser (plugin) warnings (explained in Section 2.4.2) and TLS (although TLS is not really effective anymore, as we will see in Section 2.4.2).

3. The user level. This level represents the user's ability to recognize and stop phishing attacks themselves. Many companies try to protect themselves from phishing attacks by educating their employees/customers to recognize and stop phishing attacks.

1.5 Goal

This thesis aims to find a way to stop as much e-mail based phishing attacks as possible, in the communication level. In this thesis, we will look at a way to prevent phishing e-mails, but more generally, malicious e-mails. Therefore, the research question is:

What measures can we implement to structurally protect humans from phishing e-mails without disrupting their normal e-mail behaviour?

1.6 Proposal and outline

In this thesis, we propose Certificate-based E-mail Phishing Prevention (CEPP): a protocol that allows for effective combat against phishing attacks in e-mail. We will do this by first providing the preliminaries: in Chapter 2, we will look at the current e-mail structure and anti-phishing measures. In Chapter 3, we will give an overview of how certificates work, since we will need them for the proposal. The next step is to specify all goals of CEPP. This will be done in Chapter 4. Then, we need to give an overview of how CEPP will work (Chapter 5), followed by a formal specification (Chapter 6). To prove that CEPP works, we will then give a proof of concept. We will do this by creating an add-on for Thunderbird that implements the CEPP specification. We will provide an overview of this add-on in Chapter 7. In Chapter 8, we will discuss how accurately the protocol meets the requirements specified in Chapter 4. Chapter 9 concludes the research. Appendix A contains a description of the Thunderbird add-on code.

Chapter 2

Preliminaries: E-mail and phishing

In this chapter, we will be looking at some of the essential preliminaries. We will first provide a general overview of how e-mail works in Section 2.1. In Section 2.2, we will explain why phishing attacks are so effective. In Section 2.3, we will provide the most common types of phishing attacks. In Section 2.4, we will list widely used techniques to prevent phishing attacks.

2.1 Basic e-mail terminology

In this section, we will quickly go through the most important components of the e-mail process. An abstract overview of the e-mail process is depicted in Figure 2.1.

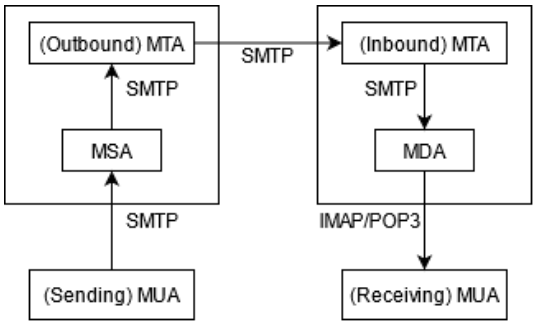

Figure 2.1: Abstract overview of the e-mail process

When someone writes an e-mail, it all starts at the sending Mail User Agent (MUA). An MUA is simply an application that can be used to send and receive e-mails. Popular MUAs include Mozilla Thunderbird and Microsoft Outlook Desktop, but also browser-based MUAs like Gmail and Yahoo Mail.

When the e-mail is sent, the Simple Mail Transfer Protocol (SMTP) is used to deliver the e-mail to the Mail Submission Agent (MSA). The MSA then forwards the mail to the Mail Transfer Agent (MTA) [Gel11], also using SMTP. The MSA and MTA are usually part of the same system and are generally called the 'mail server'. Each domain has a mail server that manages its inbound and outbound e-mails. A lot of mail server software exists. Microsoft, for example, uses their own Microsoft Exchange Server. Google also uses their own mail server software.

The outbound MTA (the MTA belonging to the domain of the e-mail sender) then uses SMTP to send the mail to the receiving domain's mail server (the recipient). This mail is delivered to the (inbound) MTA of the receiving mail server. This MTA then forwards the mail to the mail server's Mail Delivery Agent (MDA), again using SMTP. The MDA then stores the e-mail and waits for the recipient to request the e-mail.

The last step in the process is the recipient requesting the e-mail. This is usually done using either the IMAP or POP3 protocol. The main difference between these protocols is that POP3 usually deletes the e-mail from the MDA, whereas IMAP keeps the e-mail on the MDA. After the receiving MUA has received the e-mail from its MDA, it can then display the e-mail to the recipient.

2.2 Why phishing attacks are so effective

Phishing is a form of social engineering. Social engineering is (among others) the art of manipulating people into giving away their private information. Social engineering attacks are conducted in many ways, but phishing is one of the most well-known ones. Social engineering is based on the idea that it is a lot easier to find private information by manipulating people into giving it away themselves than finding it by actually breaking into a house or hacking a computer.

There are various reasons why phishing attacks (and social engineering attacks in general) are so effective. In general, phishing is effective because the perspective of gaining something (money, or other valuable things) blinds people. In this section, we will look at the different aspects of phishing attacks that make them so effective.

2.2.1 Trustworthy sources

Usually, social engineering attackers pretend to be a trustworthy entity. In phishing, e-mails will (for instance) appear to be from a bank, the govern-

ment, a good friend, or a family member. This is already the first step: people trust these sources. After all, the first step into checking the trustworthiness of an e-mail is by checking the e-mail address that the e-mail was sent from. It is, however, trivial to forge the source e-mail address when sending an e-mail. This is called e-mail spoofing. Measures to prevent e-mail spoofing include SPF, DKIM and DMARC, which we will discuss in Section 2.4.1 (Spoof Protection).

Even if e-mails are protected against e-mail spoofing, there are still problems with sources appearing to be trustworthy. For one, the e-mail accounts of other people can be hacked. Secondly, if we have a trustworthy bank `moneybank.com`, it is trivial to claim a domain like `money-bank.com` and use that one to send e-mails. People are bad at detecting these kinds of tricks, which is one of the main reasons they will trust the e-mail.

Furthermore, people can often have a hard time detecting whether an e-mail is phishing or not, since many legitimate e-mails also look.

2.2.2 Money

Most phishing attacks are about money. Although at first glance not every attack might seem to be directed towards earning money, a deeper look often reveals that this is the final goal. If the attackers want to get a victim's password, they will probably use it to log in to the victim's e-mail to find banking details. They might also use a victim's data to target other victims.

Money is (and has always been) a reason for people to act before thinking clearly. A good and recent example of this is the Twitter Bitcoin scam conducted on July 15th, 2020 [Iye20]. Over 130 high-profile Twitter accounts sent messages, claiming to send back twice the amount of money that people sent them, via Bitcoin. Funnily enough, this was possible because of another social engineering attack on Twitter employees. In the end, about $110.000 was sent to the associated Bitcoin addresses. Although this is not much money, it does show how vulnerable people are to these kinds of attacks. The line 'If it seems too good to be true, it probably is.' once again turned out to be correct.

Most phishing attacks use similar ideas to bait people into acting without thinking. Usually, the e-mails indicate that the victim can earn or lose much money. This is immediately one of the most significant phishing attacks indicators: is there much money involved? If this is the case, it is likely that the e-mail is a phishing e-mail.

2.2.3 Speed

The last main reason why social engineering attacks are so effective is all about speed. Usually, these attacks will encourage victims to act quickly. Many people might panic slightly, which means they will not think clearly and will not check the message's legitimacy.

The 2020 Twitter attack is again an excellent example of this. The tweets mentioned that the Bitcoin deal would only be live for 30 minutes. This made people forget about the message's weirdness; why would someone double *any* amount? If someone were to do that legitimately, they would be out of money instantly. Because of the 30 minute indication, some people did not think about this, and that is why over 320 transactions to the bitcoin addresses were conducted.

Most phishing attacks have some sort of time warning in them. Deals might only last for a short amount of time, or something happens to the victim if they do not act quickly enough. This is the last significant indicator of phishing e-mails: do they want the victim to act as soon as possible? Not thinking clearly about issues like these causes people to fall for social engineering attacks.

2.3 Types of phishing attacks

In this section, we will be looking at the most used types of phishing attacks. Many of these techniques are covered by Ramzan [Ram10].

As mentioned before, phishing attacks can be conducted over multiple communication media. In this thesis, we will mainly discuss e-mail-based phishing. This section will list all types of phishing attacks that can be conducted using e-mail.

2.3.1 Mass e-mails

Mass e-mails (or e-mail blasts) are mails sent to many people, hoping a few people will respond or interact with the mail. Typically, these e-mails are all the same, but some parts of the e-mail can be altered. Minor tweaks to mass e-mails can include personal information like the target's name in an attempt to come across as more trustworthy. Spam filters have gotten pretty good at detecting mass e-mails, and most people do not even respond to non-personalized e-mails anymore. Therefore, phishing attackers are switching to other, more successful phishing techniques. There are still a few techniques that work very well for mass e-mails, however. Usually, if the mail states that quick action is required, more people will fall for the scam. So, although

mass e-mails are used less and less to attack companies, they are often used to attack individual households.

2.3.2 Spear phishing

Spear phishing is the most common type of phishing attack [Sym19]. Spear phishing attacks are directed at specific targets, usually certain individuals in a company. In spear phishing, the attackers usually spend time gathering information about their target to create an e-mail that appears highly trustworthy. For example, the attack on Scoular (that we described in the introduction) is a good example of a spear phishing attack.

2.3.3 Whaling

Whaling is a phishing attack that is often directed at top executives. They are high-value targets since they have much power in the company. Whaling can be seen as a form of spear phishing. Usually, a lot of information is collected about a target, where attackers sometimes also look for vulnerabilities in the target's emotional status. Using all this information, a highly personalized e-mail (or set of e-mails) is constructed. Often, they will try to convince executive offers to install certain malware, set up a business e-mail compromise attack, or transfer a significant sum of money.

2.3.4 Business e-mail compromise

An attack with possibly devastating consequences for companies is a business e-mail compromise (BEC) attack. In this attack, attackers impersonate a senior executive to get company employees to transfer money to the attacker. Usually, this involves taking over the executive officer's e-mail. This can be done through vulnerabilities in the companies software or by using spear phishing. Victims will then receive an e-mail that usually states an urgent money transfer is required. Furthermore, victims are often instructed to not tell anyone about it because of a variety of reasons. According to the Phishing Activity Trend Report, the amount of BEC attacks is increasing [APW20].

2.3.5 Clone phishing

Clone phishing is a phishing attack where the attacker will copy a previously sent message's contents and send the same message again, but with a few alterations. For example, the e-mail attachments or links in the mail could have been changed to malicious ones. Often, the attacker then indicates that the new e-mail is an update or the old e-mail contained a mistake. Usually, clone phishing requires either the sender or the receiver to have been hacked before. Otherwise, the attacker cannot get their hands on the original mail.

2.4 Techniques to stop phishing attacks

In this section, we will take a look at the techniques that have been developed to stop phishing attacks. Since none of these attacks can prevent phishing completely, they are usually combined.

2.4.1 Communication level

The communication level represents the actual communication between the attacker and the victim. For e-mail based phishing attacks, these are the e-mails themselves. For phone-based phishing attacks, these are the phone calls; for text-based phishing attacks, the messages themselves. A variety of techniques have been developed to prevent victims from ever receiving the phishing message. We will discuss those techniques in this subsection.

Spam filters

The most well-known technique used to stop phishing attacks at the communication level is a spam filter. Spam filters are most commonly used to filter e-mails, but they can also be deployed on other communication mediums. Popular mail providers like Gmail and Outlook have deployed a spam filter in their MTA.

Each e-mail will always pass through an MTA at least once. Therefore, an MTA is a good place to filter spam. Most mail providers use both inbound and outbound filtering; they will filter mail sent to their users, but they will also filter mail that their users send to other users. This is why it is hard to send a malicious e-mail with a Gmail or Outlook account.

Techniques have been developed to detect whether an e-mail is spam or not. Dozens of methods exist to stop spam [Ram10][HvdHS08][BESH07], some more effective than others. Filtering can be divided into two parts: content-based filtering and header-based filtering (spoof protection). In this section, we will discuss content-based filtering. These methods can be divided into roughly three categories:

1. **User-based filters**
   User-based filters require the user to set up their spam protection. Examples include disabling HTML in e-mails and reporting spam e-mails. In general, user-based filters are not super effective to protect against sophisticated phishing attacks since most users do not take the time to set up good spam protection.

2. **Automated filters**
   Automated filters are spam filters that are usually set up by the e-mail

administrator or the company providing the mail server. This can be a public e-mail provider or a dedicated group of employees for a certain company. There are many types of automated spam filters, and it is an ongoing topic of research [KNL20]. Some examples are the following.

- Rule-based filtering, where e-mails with certain words will not be accepted. Often, these are checked by attempting to match a regular expression (which specifies a search pattern). For example, say one might want to block e-mails containing the word 'nigeria'. The regular expression would then look like `/b(nigeria)/b`. After checking the mail content for these regular expressions, one can then decide whether the e-mail should be blocked.
- DNS blacklists, where the IP of the sender MTA (stored in the e-mail header) of the incoming e-mail can be checked against a DNS blacklist (DNSBL) and where the mail will be rejected if the IP has been blacklisted [DNS20].
- URL filtering, where all URLs in the mail header and body are looked up in blocked-domain lists like Spamhaus DBL, SURBL, and URIBL [MMDCL09]. Mails will be rejected if any of the URLs in the mail are malicious.
- Statical (or Bayesian) spam filtering is a naive Bayes classifier that uses the bag-of-words-model (which counts the number of times each word occurs) to determine whether a mail is spam or not. It depends on the user to mark e-mails as spam (or not) and uses that information to train the classifier. A big advantage of this method is that it will conform more and more to the user's needs.
- Spam traps, or honeypots, are e-mail addresses that anti-spam organizations like Project Honey Pot create. They will, for example, hide these addresses in a websites source code. When a user visits the website, they will not see the mail address, but when an attacker uses a website scraper, they might find the mail address in the code. After they send an e-mail to the address, the anti-spam organization will know that the e-mail sender was a spammer. This can then be used to blacklist the e-mail address.

Most e-mail providers use a combination of many techniques to attempt to protect their users/employees against spam e-mails.

3. **Automated sender filters**
   Automated sender filters are spam filters that will prevent users from sending spam themselves. These filters are applied in most widespread e-mail providers like Gmail and Outlook, and there is a wide variety in the types of filters. An evident approach is to do a background

check on newly registered users and limiting the number of e-mails a user can send, but there are also more sophisticated approaches. Note that for widespread e-mail providers (like Gmail and Outlook), there is an incentive to not send spam e-mails. If they do, their domains (`gmail.com`, `outlook.com`, etc.) might get blacklisted, resulting in a considerable portion of users not being able to send e-mails to certain target domains.

Bypassing spam filters

Similarly, a lot of creative techniques to bypass spam filters have been created by researchers and attackers. Some techniques that occur often are the following.

1. **Simple e-mail formatting**
   Simple e-mail formatting is probably the most straightforward way of bypassing spam filters. If an e-mail is free of spelling/grammatical errors and doesn't contain big chunks of capitalized text or multiple punctuation marks, it is more likely that a spam filter will not categorize that e-mail as spam.

2. **Spam obfuscation**
   Spam obfuscation is a general technique representing all methods to obfuscate text so that the victim can still read it, but the spam filter cannot. Numerous techniques exist to obfuscate spam. A few techniques are the following.

   - **Basic obfuscation**
     One can purposefully misplace spaces, misspell words, or add strange characters to the text. This way, spam filters might fail to parse the input or recognize what text is actually displayed [Tho04].
   - **Unicode-obfuscated spam**
     This is a simple technique that can be used very easily. Unicode contains many characters that appear the same but have different byte codes. When substituting characters in an e-mail with other Unicode characters that appear the same, filters like Bayesian classifiers might not work anymore. Liu and Stamm [LS07] wrote an informative paper about this technique, and they discuss methods to combat it, including a way to combat this type of obfuscation.

   It is trivial to come up with other 'simple' spam filter bypassing techniques. Some techniques that we came up with are the following.

- **HTML-obfuscated Spam**
  It is trivial to obfuscate text using HTML tags. Tags like `<p></p>` will not render anything special to the user but might disrupt a spam filter.
- **Image-based obfuscation**
  It is possible to put the text inside an image and include it with HTML or attach it to the mail. Word-based spam filters will be unable to read this text since there is no actual text.
- **Attachment-based obfuscation**
  It is possible to hide text from spam filters by including the text in the e-mail attachment. In the mail, the victim will then be instructed to open the attachment and read the phishing text.

These are only a few techniques that attackers use. There are dozens of methods that are combined to create e-mails that bypass spam filters. Spam filter detection is a constant battle between attackers (and researchers) on the one hand and spam filter developers on the other.

Spoof protection

Spoof protection can be seen as part of a spam filter. Many attackers attempt to make their e-mails more believable by changing the `From` field in the mail header to the actual e-mail address of an organization they are trying to impersonate. The SMTP protocol does not protect against this by default. Many techniques have been developed and proposed to protect these (and other) SMTP envelope and mail header fields. The most common techniques have been described thoroughly by Wallis de Vries [Ste20]. We will give a summary.

1. **Sender Policy Framework**
   The Sender Policy Framework (SPF) is a protocol that is used in MUAs and MTAs to authenticate e-mails by verifying the domain of the sender's e-mail address. Providers that work with SPF have specific DNS records that specify which IP addresses are allowed to send mails with the given e-mail domain [Ste20]. The problem with SPF is that the `From` part in the e-mail header can still be forged since SPF does not check that. SPF only works for the `MAIL FROM` field in the SMTP header.

2. **Domain Keys Identified Mail**
   Domain Keys Identified Mail (DKIM) is a protocol that was established in 2007 [All07]. In this protocol, the e-mail sender uses a digital signature algorithm. They generate a public/private key pair, and publish the public key via a DNS record. They then add a signature to each e-mail, using the generated key pair. This is a signature over

the important header fields and the hash of the e-mail body. The receiver can then verify the mail by requesting the public key from the e-mail sender's DNS and checking the signature.

3. **Domain-based Message Authentication, Reporting, and Conformance**
   Domain-based Message Authentication, Reporting, and Conformance (DMARC) is a solution to a problem with both SPF and DKIM: if an e-mail does not use them, the recipient will still receive and accept the mail. When using DMARC, the sender will publish DNS records indicating whether SPF and DKIM are used. DMARC will also check the `From` field, so spoofing becomes nearly impossible.

Wallis de Vries describes quite a lot of problems with all of these protocols [Ste20]. The solution he proposes is SMTPsec, a protocol that fixes these (and other) problems in electronic mailing. Opposed to SMTPsec (which attempts to make SMTP better and more secure), our protocol attempts to prevent phishing attacks.

2.4.2 Target level

The target level represents the place where users will give their private information to the attackers. For phone-based phishing, this is usually given via the phone call. For e-mail and text-based phishing, this is often happening in web browsers. To be precise, a lot of e-mail and text-based phishing attacks instruct victims to click a link that opens a seemingly trustworthy website. In this section, we will examine the techniques developed to counter phishing attacks in the web browser.

Browser functionality/plugins

Browsers and browser plugins can implement methods to warn users about certain websites. Some of these methods still require the user to check something, whereas others explicitly deny access to a website unless the user accepts the risk. We give discuss two measures that are implemented a lot.

1. **TLS**
   TLS, previously known as SSL, is a protocol that encrypts traffic in client/server applications [Res18]. It is best known for encrypting traffic between users and web servers. All widely used browsers have an indication for when a website uses TLS. Usually, this indication is in the form of a 'lock' icon next to the address bar. Browsers will also display a warning if the TLS certificate (which is used to prove that the public key of a website is correct) is not (or no longer) valid. Checking whether a website uses TLS used to be a good indicator for

a legitimate website, but it is not anymore. The quarter 2 AWPG trends report of 2020 claimed that over 78% of phishing websites now use TLS [APW20].

2. **URL blacklisting**
   An easy solution that is used by nearly all browsers is a URL blacklist. This is a vast database containing URLs of phishing websites that were blacklisted. Upon visiting websites like these, most browsers will display a warning and ask the user if they want to continue. There also exists a lot of anti-virus software that does the same; they keep track of blacklisted URLs and warn users before they continue to those websites. Most browsers also have functionalities to report phishing websites. If a reported website is indeed a phishing website, it is then added to the blacklist. Sadly, URL blacklisting is a method that is lagging behind events. It is unable to protect victims against websites that are not (yet) blacklisted. Attackers can quickly get themselves a new website, and it is not expensive anymore to protect a website with TLS.
   Some companies have taken measures to blacklist unfinished phishing websites. For example, if someone is getting a certificate for `trustworthy-bank.com`, the owners of `trustworthybank.com` can already send a message to the website provider of `trustworthy-bank.com`, asking them if the website could be taken offline. This can (for example) be done by using Google's Certificate Transparency project, which logs all issued TLS certificates. This method is not fail-proof either. Websites can always slip through, and not all relevant companies are doing these kinds of checks.

Protocols

A few protocols have been created to prevent phishing, and more generally, leaking password attacks. Leaking password attacks are attacks where the attacker attempts to get the password of the victim. In this section, we will describe two protocols that attempt to prevent leaking password attacks: SRP and TLS-PSK.

1. **SRP**
   SRP [Tay07] [BESH07] is a protocol that prevents users from actually sending the password to the server. Instead, the user only shows to the server that they have the password via a challenge-response. This is much more secure than sending the password over TLS. When using this protocol, phishing websites cannot use the password since they do not know it. SRP under TLS has been standardized. Unfortunately, not many companies use this protocol.

2. **TLS-PSK**
   TLS-PSK [Ero05] [BESH07] is a protocol that uses a key that has been established between two parties in advance of the communication. The protocol uses symmetric key cryptography, which has two advantages. Firstly, symmetric cryptography is less performance-demanding than asymmetric cryptography. On systems where performance is essential, this helps a great deal. Secondly, attackers cannot get sensitive data since they do not have the pre-shared key.

2.4.3 User level

Finally, we have the last set of methods to prevent phishing attacks: making sure people know about the threat and how to avoid it. People are unreliable and are very prone to be scammed. In this section, we will discuss the education of people and the pitfalls in this education.

One of the most common measures against phishing attacks is education. Reminders in the news, warning mails and app warning messages all tell users to be careful when trusting people. Banks have mentioned over and over again that they never ask for credentials through e-mails or over phone calls, but people still fall for the scams. That is the main problem with education: people tend to forget what they have learned [SY19].

Young-McLear et al. [YMWBYM16] conducted an experiment on a university. An e-mail was sent to 198 students asking them to log in to a website because there was information about their grades. Precisely 146 students opened the e-mail, clicked the link, and entered their credentials. The strange part: nearly all students were active in the cybersecurity area of computing science. Most of them knew about the threat of phishing, but they fell for it nonetheless. This shows that even trained (young) professionals can have trouble distinguishing phishing e-mails from trustworthy e-mails.

Summer and Yuan [SY19] argue that education and training for particular groups (such as employees that just finished their study) should be emphasized. Furthermore, they argue that education should constantly be repeated since people tend to forget what they have learned. Many approaches exist to teach people about the risks of phishing attacks, but the best underlying principle is repetition. Many automated measures can be taken, but in the end, there is always the user. A well-educated user can spot and prevent phishing attacks easily.

Chapter 3

Preliminaries: Certificates

The problem of phishing is essentially a problem of message origin authentication. How can one be sure that the entity they think they are communicating with is the entity they are actually communicating with? A standard cryptographic solution (in public-key cryptography) to this problem involves public key certificates. In this chapter, we will be looking into public key certificates (also called digital certificates), their function in TLS and S/MIME, and their strengths and weaknesses. We hope to apply the obtained knowledge about certificates to find a solution to the phishing problem.

3.1 Asymmetric cryptography

We will first introduce some important concepts in digital security.

- **Confidentiality** means that communication data cannot be read by those that are not allowed to.
- **Integrity** means that communication data cannot be modified or deleted by those that are not allowed to.
- **Authenticity** means that it is not possible to impersonate a different entity.

When information is sent to a website, we do not want anyone other than the website to read (confidentiality) or modify (integrity) that information. For example, if someone wants to log in to a website, they would not want an attacker to read their password. This is where asymmetric cryptography (also known as public key cryptography) comes in. In asymmetric cryptography, the receiver of a message generates two *keys*, a *public* key and a *private* key. These keys are linked via a mathematical calculation.

If entity X now wants to send confidential information to entity Y, they

can *encrypt* that information using the public key of Y (or using symmetric cryptography to encrypt the information with a key derived from X's private key and Y's public key). If the underlying cryptography is reliable, it is then infeasible for an attacker to read this information. However, entity Y can easily *decrypt* the information using its private key! This way, it is possible to send confidential information to someone without anyone other than the receiver being able to read (or modify) it.

This also works the other way around. If entity Y sends a message to entity X and Y wants to prove that they are really the sender of that message, they can do that by attaching a *digital signature* to that message. We will provide a detailed explanation of digital signatures in Section 6.1.

3.2 Public key certificates

However, there is a catch. How can one be sure that the public key they are using to encrypt a message is the public key of the entity they want to communicate with and not the public key of an attacker? This is where public key certificates come in. A certificate is a document serving as evidence of something [Dic20]. A public key certificate is a certificate that proves the ownership of a public key. A public key certificate contains a section with information about the key and subject, and a section containing the certificate issuer's digital signature. A certificate authority (CA) can issue a certificate that says: 'Public key X is owned by entity Y'. If someone wants to validate that a public key is owned by entity Y, they can check that via the CA that issued the certificate. This is done by validating the certificate's correctness. This means that both the sender and the receiver need to trust the CA. We will talk about the problems that CAs cause in Section 3.4.

Certificate types

There are many certificate types. We have certificates for TLS clients and servers, e-mail (S/MIME), source code validation, and more. They all have the same function: they link a public key to an entity.

3.2.1 Certificate chains

As mentioned before, one can obtain a certificate by going to a CA. The CA will then provide a certificate, and (for example) web browsers can then verify that certificate. A problem arises though: What CAs can we trust? This is where certificate chains come in. There are two types of CAs: root CAs and intermediate CAs.

1. Root CAs. Root CAs are CAs that issue certificates for intermediate CAs. Root CAs use self-signed certificates (root certificates), and they need to be trusted by operating systems and browsers before they can be used. They have to use self-signed certificates, since no-one can issue a certificate for a root CA.

2. Intermediate CAs. Intermediate CAs are CAs that are in between the website certificates and root CAs. Most website certificates have at least one intermediate CA. Root CAs issue the certificates of these intermediate CAs. Furthermore, intermediate CAs can issue certificates for other intermediate CAs. Lastly, intermediate CAs can issue certificates for individual entities.

This structure is called a certificate chain, and it is an example of a chain of trust. Users trust one or multiple sources (root CAs), and these root CAs trust intermediate CAs. Following that logic, if one trusts a root CA, one can trust any website with a certificate that ends up in that root CA. Certificate validation is done by checking the certificate chain and checking if it ends up in a trusted root CA. If this is the case, the certificate is valid.

An interesting note to these certificate chains is the (arguable) mess they have created. As of December 8th 2020, Mozilla trusts 140 root certificates for TLS and 99 root certificates for S/MIME [Moz20]. Microsoft trusts 256 root certificates in Windows [For20]. This means that the intentions of a lot of root CAs have to be carefully monitored.

3.3 Certificate validation levels

In many circumstances, validation levels can be very useful. Validation levels indicate how much an organization is trusted. For example, an organization that can show that they are legally registered (in, for example, a company register) is more trustworthy than an organization that only owns a domain.

3.3.1 TLS

Depending on the type of certificate that is needed, there are multiple validation levels. In TLS, we have three validation levels [Coc13]:

1. Domain validation. One can obtain a domain-validated certificate by proving that one has the administrative rights to modify the domain's DNS settings. This is the lowest validation level, and companies like Let's Encrypt and Cloudflare now offer free domain-validated certificates.

2. Organization validation. To obtain an organization-validated certificate, one needs to prove that they have the administrative rights to

modify the domain's DNS settings. They also have to show that the organization legally exists. Both domain- and organization-validated certificates can be obtained automatically, without the need for human verification. For organization validation, the check that an organization legally exists is also done automatically. This is done by (for example) checking the company name with a company register.

3. Extended validation. To obtain an extended validation (EV) certificate, one needs to prove their identity to a CA, where actual human involvement is required. Extended validation certificates can not be obtained without at least one (real) person's involvement on the CA's end.

3.3.2 S/MIME

In S/MIME certificates, we have two validation levels:

1. Class 1 certificates. These can be compared to domain-validated certificates in TLS. They will protect the `From` field in an e-mail. If one receives an S/MIME protected mail with a class 1 certificate, one can be sure that the sender in the `From` field is the actual sender of the mail.

2. Class 2 certificates. These can be compared to organization-validated certificates in TLS. Although, in theory, human involvement in these certificates would be better, it is not required. Both class 1 and class 2 certificates can be obtained automatically, and neither verifies a company's validity or purpose.

Although it seems like S/MIME certificates can solve the phishing problem, they cannot. S/MIME guarantees the e-mail sender's authenticity, but we still have the same problem; if the legitimate company is `trustworthybank.com`, and the e-mail is sent by `trustworthy-bank.com`, many people think the mail is legitimate. S/MIME does not protect against this.

3.4 Certificate problems

Certificates are helpful, and they are used everywhere. However, there are also a few big problems with certificates. In this subsection, we will give an overview of the advantages and the problems that certificates have. We will also take a look at the problems with implementing extended validation level certificates into S/MIME.

3.4.1 Advantages

1. The most obvious advantage is the purpose of certificates: that it facilitates secure communication. Public key certificates enable a secure solution to communication. Asymmetric cryptography, together with certificates, enables confidentiality, integrity and authenticity over communication.

2. Certificates are simple, require no configuration from individual users, and they are cheap.

3.4.2 Disadvantages

1. There are huge risks involved with certificates. If a certificate authority is compromised, attackers can trick people into thinking they are talking to a certain party, whereas they are actually talking to (for example) the attackers. A notable example is the attack in 2011 on DigiNotar, a Dutch digital certificate authority [FI12]. In this attack, attackers compromised the CA and issued fake certificates for `*.google.com`. There are signs that these certificates have been used in Iran to create a man-in-the-middle attack for Gmail.

2. Maintaining a certificate authority is a lot of work. Security protocols constantly have to be updated, and a potential attack is devastating.

3. Certificate authorities cause a single-point-of-failure. In the DigiNotar attack, all major web browsers and operating systems had to blacklist all certificates signed by DigiNotar, which caused a lot of Dutch companies to suddenly have invalid certificates. If a CA is compromised, there is no sound system in place that offers additional protection. This is precisely why a CA compromise is terrible for the internet, and this structural issue is partially responsible for the problems caused by the DigiNotar CA compromise.

3.4.3 Extended Validation Certificates

Extended Validation (EV) certificates from TLS could solve the problem of phishing. If we add a similar certificate level to S/MIME, we could make sure that only companies like banks and governments could get these certificates. Mail service providers could then add an EV check and reject any financial e-mails if they were not signed with EV certificates.

This solution sounds good, but it has a problem. EV certificates are not as good as they might seem. There is much criticism on EV certificates. For instance, Caroll [Com17] realized that there were no name collision checks in the EV certificate registration process. He illustrated this by buying an

EV certificate for a company that already bought their EV certificate. He obtained a legitimate certificate, showing his website as being the website of a company that he did not own. Furthermore attackers have used the identities of other companies to get EV certificates. These would then trick users into believing a website was legitimate. Another example is the creation of an EV certificate for a company called 'Verified Identity' by security researcher Burton in 2017 [Com17]. Web browsers would then show 'Verified Identity' next to the lock icon in the browser search bar.

It is important to note that, theoretically, a measure such as EV certificates would be a good solution, under the assumption that CAs make no mistakes. However, CAs make mistakes, and no one holds them accountable.

Because of these (and other) reasons, companies like Mozilla, Google, Microsoft, and Apple decided to stop giving additional visual signs for EV certificates since they did not help. We are left with a system where certificate authorities earn a lot of money over certificates that do not provide additional authenticity, with most people not even noticing it anymore. It would not be wise to copy this failed system to S/MIME.

Chapter 4

Requirements and goals

As we have seen, many measures are already in place to protect people against phishing attacks. One of the most implemented techniques is S/MIME. This protocol does provide message origin authenticity and integrity, but it still fails to protect people against phishing attacks. This is partially due to it not being easy to implement and maintain, and in most e-mail clients, it requires significant efforts from the user to enable.

Moreover, even if everyone were to use S/MIME without problems, it would still not prevent phishing attacks. Most phishing attacks do not originate from a company's domain but rather from a fake look-a-like domain. For example, the trustworthy company would be named `trustworthybank.com`, and the impersonated fake company `trustworthy-bank.com`. There is not much that can be done about this with S/MIME (or any existing protocol, for that matter) since the attackers *are* the fake domain's legitimate owners.

The main goal that we need to achieve is deploying a system that prevents spam e-mails from ever being displayed to the end-user. In this chapter, we will list the requirements that our proposal needs. We will discuss the functional and non-functional requirements. We will also take a look at the system's security, particularly the trust model, attacker model, and security goals.

4.1 Functional requirements

1. Each domain should either be trusted or not, and e-mail clients should be able to filter e-mails on whether their sender is trusted.
2. End users should be able to report phishing e-mails from a trusted domain to the organization that trusted the domain. This way, the organization responsible for trusting the domain can check the domain, and if necessary, stop trusting it.

3. The system should be secure. It should not be possible to send an e-mail from a domain that is not trusted and have it displayed as being trusted.

4. The spam report system should be secure. It should not be possible to send a report of an e-mail that was never actually sent by a given domain.

4.2 Non-functional requirements

1. No single organization should be responsible for the determination of which domains are trusted.

2. The responsibility of a domain's trustworthiness should be with an organization that has a very high interest in keeping the trustworthiness of each domain high. They should attempt to prevent phishing from domains they trust at all costs.

3. The system should allow for gradual deployment. It should also interoperate with the existing infrastructure. It is infeasible to expect that everyone in the world can deploy a system at once.

4.3 Security

4.3.1 Trust model

The trust model of our proposal is greatly based on the companies that will trust domains. If any of these companies fail to do their job, people might receive 'trusted' e-mails, which might have a counterproductive effect. For example, say an e-mail user was to receive a trusted phishing e-mail. Usually, they might have spotted that the e-mail was not trustworthy, but since this mail is 'trusted', they might think it is okay. These companies must do their job correctly.

Of course, as we have seen with existing CAs, mistakes are made. Because of this, it is impossible to create a flawless system. This is precisely why the mail reporting system is required: this system allows trusting companies to fix their mistakes before the attack gets out of control.

4.3.2 Attacker model

For the attacker model, we will assume an active man-in-the-middle attacker. The attacker can locate itself between two MTAs, or between an MUA and MTA. This attacker can read and modify any traffic. The attacker is

unable to break digital signing algorithms that are considered secure at the beginning of 2021.

4.3.3 Security goals

The security goals of our proposal are:

1. An MTA or MUA receiving an e-mail should be able to verify its trust status. This means that the MTA/MUA can check that a message has a valid trust status and may therefore conclude with reasonable certainty that the mail is not a phishing e-mail.

2. Guaranteeing the integrity of the trust status of the e-mail. It should be impossible to alter the status without the receiving side noticing this. Confidentiality is not relevant for this thesis.

Chapter 5

Proposal

In this chapter, we propose CEPP: Certificate-based E-mail Phishing Prevention. CEPP is a solution to the problem described, conforming to the functional and non-functional requirements of Chapter 4.

5.1 Certificates

In CEPP, a certificate will be attached to each e-mail by the e-mail sender. The certificate is similar to the one used in TLS. In TLS, the certificate proves the ownership of a public key. In CEPP, the purpose of the certificate is for the recipient to be able to verify that the e-mail was sent by an organization that it trusts. For this purpose, a list of new organizations (similar to root CAs) must be established and added to the e-mail client (MUA) or transfer agent (MTA) software. We will call these organizations 'Mail Authorities (MAs)'. The difference between CAs and MAs is explained in Section 5.2. These MAs sign certificates for domains used to send e-mails.

Upon receiving an e-mail, the recipient will check the certificate by retrieving the MA's public key and verifying that the expected MA indeed signed the certificate. After the signature is verified, the recipient has to check all certificate fields. The most important field is the domain field, indicating what domain the certificate can be used on. More information on the validation procedure is available in Chapter 6.

Now, there is one more essential step that we must not forget. If an attacker spoofs the e-mail address and adds a valid CEPP certificate for that e-mail address, the recipient will trust the e-mail. There are two approaches we can take to counter this attack:

1. Enforce the use of DKIM (using DMARC, explained in Section 2.4.1).
2. Add functionality to CEPP that does the same as DKIM, so we do

not have to enforce DKIM.

We have opted for option 1 in our proposal. Option 2 would just be double work, and DMARC is already a popular and widely implemented technique. Therefore, before verifying the CEPP certificate, the recipient must first validate the DMARC. If and only if DMARC uses DKIM and it is successfully validated, the CEPP validation may occur. If the DMARC validation is unsuccessful, the e-mail can not be trusted.

5.2 The certificate authorities

5.2.1 Problem with existing CAs

Initially, it might seem like we can just use the existing CAs to solve this problem. Unfortunately, it is not that easy. The purpose of CAs in TLS is to prove to the client that someone is the owner of a public key. For higher-level certificates, it also proves that the domain belongs to an existing and valid organization. In CEPP, however, a different function is expected from the MA: They must verify that the company they issue a certificate for does not use that certificate to send phishing e-mails (or any other form of malicious e-mail).

This is not an easy thing to do, and therefore not something that we can expect from existing CAs. The role of MAs is changed from showing that some entity is who they claim to be (which is what existing CAs do), to trusting the entity to be legitimate, e.g., not sending malicious content to the end-user.

5.2.2 Mail Authorities

In CEPP, we propose that several MAs (for example, one per country), get to issue certificates that can be used to send e-mails. An MA should only issue a certificate once they trust that company to not use the certificate for sending malicious e-mails. For this purpose, we recommend MAs charge companies for signing their domains. It is also recommended to check the legitimacy of the company with a company register. For example, in the Netherlands, the 'Kamer van Koophandel (KVK)' maintains the company register. A company is only valid once they register with the KVK. A measure like this ensures that a company is valid and prevents attackers from quickly registering many domains and using them to send malicious e-mails.

5.3 Certificate levels

For CEPP, we specify three different certificate levels. The purpose of this is to allow e-mail clients to distinguish between the trustworthiness of e-mail senders. We need to be careful that we do not make the same mistake that was made in TLS for the extended validation (EV) certificates (Section 3.4.3). We propose the following three certificate levels:

5.3.1 Level 1: No certificate

Although it is debatable whether this should be a level at all, the first level is simply having no certificate. If no certificate is sent along with the e-mail, this is the level we default to. We propose that e-mail clients (by default) reject e-mails that do not have a certificate. This might turn out to be infeasible, especially in the beginning. Therefore, some sort of warning indicator would also be good. Although total rejection would be ideal, it might not be possible in the long run.

5.3.2 Level 2: Basic certificate

A basic certificate is the primary certificate level, intended for use by most companies. These certificates are issued by an MA, as mentioned in Section 5.2.2. The certificate proves that the mail sender can be trusted. Although CEPP will drastically decrease the number of phishing attacks, we still recommend a spam filter for e-mail clients. Legitimate companies still send many spam e-mails, and it would be weird for an MA to remove the trust of a company because of an e-mail that could be categorized as (harmless) spam. CEPP is aimed at stopping phishing attacks and not necessarily harmless spam.

5.3.3 Level 3: Extended certificate

An extended certificate is an additional certificate level intended for use by companies that relay critical information to clients, and therefore usually are impersonated by attackers. Companies include (but are not limited to) banks, authorities, hospitals, and in general, any company that works with much money or has very personal information about the client. The advantage of having a level 3 certificate might be an additional visual indicator for the client. Also, spam protection in e-mail clients could ignore e-mails with level 3 certificates.

The use of this certificate level must be restricted to only companies that need it. Otherwise, the same problems as in TLS EV certificates occur (Section 3.4.3).

5.4 Removing company trust

In CEPP, the MA that issued the certificate is responsible for the e-mails of the company. In an ideal world, mail CAs and MAs would only issue certificates for companies that are legitimate. However, we do not live in an ideal world, and humans make many mistakes. Therefore, a system should be in place to report malicious e-mails from trusted sources so that the MA can take action. In this section, we propose a basis for this spam reporting system. If an MA repeatedly fails to remove the trust of illegitimate companies, the MA can no longer be trusted, and their public key should be removed from the CEPP software.

5.4.1 Reporting spam

Every modern e-mail client has a system for reporting spam. Right now, these systems do not have a functionality besides training some spam classifier. In CEPP, we suggest that when a user reports spam, a message is sent to the responsible MA. This message will contain all information required for the MA to make a decision. A specification for both the construction and the validation of a spam report will be in Chapter 6.

An important side note with these spam reports is their security. As mentioned in the functional requirements, it should not be possible to generate and send a spam report of an e-mail that was never actually sent. Furthermore, it should also not be possible for an e-mail sender to modify the e-mail data so that generating valid spam reports becomes impossible. For example, it should not be possible to change the CEPP header in a way that it is still valid, but generating a spam report becomes impossible. These security notes need to be taken into account while constructing a concrete specification of CEPP.

5.4.2 Remarks

There is a significant disadvantage to these spam reports that has to be noted. The privacy of the person that reported the e-mail *could* be violated. For example, say someone (possibly accidentally) reports an e-mail that contains sensitive or personal information. The MA that received the report then also receives this information. In CEPP, we assume that MAs treat possibly sensitive information with care, but that might not be the best solution. Furthermore, MTAs are also able to see the content of a spam report. This could be prevented by using public key cryptography to encrypt a spam report's content, but this method is not 100% foolproof: MTAs would still be able to see the number of spam reports, as well as their destinations. Further research should be done to create a proper solution to this problem.

Chapter 6

Specification

In this chapter, we will establish a specification of CEPP. In Section 6.1, we will give more detailed information about digital signatures in the context of CEPP. In section 6.2, 6.3, we will provide a specification for certificate generation and validation, respectively. In section 6.4 and 6.5, we will provide a specification for spam report generation and validation, respectively. In Section 6.6, we will provide a detailed explanation as to where each of these steps is conducted.

6.1 Digital Signatures

We will first give a more specific explanation of digital signatures and exactly how they are used in CEPP. As explained in Section 3.1, a digital signature is used to verify the authenticity of digital messages. In CEPP, we will use these signatures to show that an MA trusts a company. By appending a certificate (which consists of data, and a signature of that data) to an e-mail, we know that the issuer of that certificate trusts the data subject.

6.1.1 Signature scheme

A signature scheme consists of a signing function and a verification function. In the context of CEPP, the first step is for the MA to generate a public/private key pair. Then, they publish the public key (for example, via their website) and keep the private key secure.

Signing a certificate

The signing function takes the following parameters.

- The message that is being signed,
- The private key of the MA.

Since the length of an input message is arbitrary, we typically first use a hash function on the message.

The signing function takes the hashed message and the private key and uses asymmetric cryptography to create the signature from this input.

Verifying a certificate

The verification of a certificate takes the following parameters.

- The message that is being validated,
- The signature of that message,
- The public key of the MA.

As with signing the certificate, we again use a hash function to hash the message that is being validated.

The verification function then uses asymmetric cryptography to verify that the signature is correct, using the hashed message, public key, and signature. The verification function returns either `true` or `false`, indicating whether the signature was correct.

6.2 Certificate generation

In CEPP, we will use two separate headers to add the certificate to the e-mail. Firstly, we will add a header containing all relevant certificate data. Secondly, we will add a header containing the signature of this certificate data.

6.2.1 Certificate data

The CEPP certificate data header uses `CEPP-Data` as header key. The header value is a string with all parameters. The parameters that are required in a CEPP certificate are listed in Table 6.1.

Type	Key	Value description
Version	v	The certificate version that is used, default is 1.
Serial number	s	The certificate serial number, used by MAs to track certificates.
Algorithm	a	The signature algorithm that is used.
Issuer	i	The full name of the MA that issued this certificate.
Minimum validity date	nb	The date before which the certificate is not yet valid.
Maximum validity date	na	The date after which the certificate is not valid anymore.
Subject domain	d	The domain of the sender of the e-mail.
Certificate level	l	The certificate level (as described in Section 5.3).

Table 6.1: CEPP certificate data parameters

Parameter format

Each parameter has a certain format that it has to follow. After using this format, the parameter is then converted to a string. The format for each of the parameters is as follows:

- **Version**: The version is a non-negative number, and in the current version, it should always be 1.
- **Serial number**: The serial number is a number that is greater than 0. The number is generated randomly and must be unique: it should not be used in other certificates.
- **Algorithm**: The algorithm is the name of the used signature algorithm. Supported algorithms are listed in Section 6.1.2.
- **Issuer**: The issuer is the full name of the MA that issued the certificate.
- **Minimum and maximum validity date**: The minimum and maximum validity dates are formatted with the UTCTime format, which is also used in X.509 certificates [Coo08, Section 4.1.2.5.1]. This means that the time format is `YYMMDDHHMMSSZ`. This time format has to be expressed in Greenwich Mean Time. If the `YY` field is greater than or equal to 50, then the actual year is `19YY`. If `YY` is less than 50, the actual year is `20YY`.

- **Subject domain**: The exact domain of the certificate subject (the e-mail sender).
- **Certificate level**: Level '2' represents a basic certificate, and '3' represents an extended certificate (as explained in Section 5.3).

Note that all fields are mandatory; a certificate is rejected if any fields are malformed or left out.

Combining parameters

All parameters are then subsequently combined into a single string, in the following format:

```
[Key]=[Value]; [Key]=[Value]; ...; [Key]=[Value].
```

The data string is then added to the e-mail headers, with `CEPP-Data` as header key.

6.2.2 Certificate signature

The CEPP certificate signature header uses `CEPP-Signature` as header key. The header value is the signature of the entire certificate data string. The signing uses the MA's private key and specified signature algorithm:

$$\text{CeppSignature} = A_{\text{sign},PrK}(\text{CeppData}),$$

where A is the specified signature algorithm, and PrK the private key of the MA. The signature is then converted to a string (if it was not already) using binary to hex conversion and added to the e-mail headers.

Supported signature algorithms

All supported signature public key algorithms are listed in Table 6.2. We have chosen algorithms that are considered secure at the beginning of 2021.

Algorithm	State	Comments
RSA	Not recommended	Key length should be at least 2048 bits. Because of this, RSA is not recommended.
DSA	Not recommended	DSA is supported, but ECDSA is preferred, since ECDSA achieves an equal level of security with shorter keys.
ECDSA	Supported	Required curve: secp256k1
EDDSA	Supported	Required curve: Curve25519 (also known as Ed25519)

Table 6.2: List of signature public key algorithms that CEPP supports.

The following hashing algorithms are supported for CEPP signature algorithms. We have chosen algorithms that are considered secure at the beginning of 2021.

- SHA_224
- SHA_256
- SHA_384
- SHA_512
- SHAKE128
- SHAKE256

Algorithms are combined into a string in the following way:

```
[PUBLIC KEY ALGORITHM]_WITH_[HASH ALGORITHM]
```

Each hash/public key algorithm combination is supported. This means that there is a total of $3 \times 10 = 30$ supported signature algorithms in CEPP.

6.2.3 Example

We will demonstrate how the construction of a certificate takes place:

1. The first step is to create an MA, together with a public/private keypair (as explained in Section 6.1). This is required, since otherwise we would not be able to create a certificate. The name of our test MA is 'Test Certificate Authority'. We use ECDSA, and obtain the following keypair:

```
 Public:
049a55a04ad8538e460dc175bb027859d32eb88208b8ecb5ac2d16afaf19079af
008db4d349cd1098bc758796c40b4fb2b75da3557d4887c77ece7af759f2a7143,
 Private:
4f77a087f11c319df6842928e92c1a3941d4d94386a0209a777d81bca6461f6d.
```

2. The second step is determining all required parameters. In this step, we actually start creating a certificate.

   v : 1
   s : 12345 (this is usually randomly generated)
   a : ECDSA_WITH_SHA_256
   i : Test Certificate Authority
   nb : 210101000000Z (January 1st, 2021, 00:00h)
   na : 210107000000Z (January 7th, 2021, 00:00h)
   d : student.ru.nl
   l : 2

   (see Table 6.1).

3. We then convert all parameters to a single string. This results in: `v=1; s=12345; a=ECDSA_WITH_SHA_256; i=Test Certificate Authority; nb=210101000000Z; na=210107000000Z; d=student.ru.nl; l=2`.

4. Then, we compute the signature:

   `CEPP-Signature` = $\text{ECDSA\_WITH\_SHA\_256}_{\text{sign,4f7...f6d}}$('v=1; ...; l=2')

   =

   ```
   304502210093d5a3c8453ab43d66b9c0eaef0d18cf6bf18
   826e044b6b7ba8de1e79933b43602207fa964e12505d74c
   29302dd88fdb2e8981473a3c5816b8def109f037762ce156.
   ```

5. The final step is adding the two CEPP headers to the e-mail. The following headers are added:

   ```
   CEPP-Data: v=1; ...; l=2,
   CEPP-Signature: 304...156.
   ```

6.3 Certificate validation

The validation of certificates is a bit easier and consists of three steps.

1. Validating the DMARC.
2. Validating the CEPP signature.
3. Validating the CEPP data.

In this section, we will explain each of these steps. We will also give an example CEPP certificate validation.

6.3.1 Validating the DMARC

As mentioned in Section 5.1, we need to enforce the use of DKIM using DMARC. If we do not do this, an attacker can spoof an e-mail address and attach a valid CEPP certificate for that domain to the mail. Therefore, before a CEPP certificate can be validated, DKIM must be used. If the DMARC policy is absent or specifies that DKIM is not used, the CEPP certificate is considered invalid.

6.3.2 Validating the CEPP signature

After the DMARC is validated, we can continue to validate the CEPP signature. This is easily done using the public key of the MA that signed the certificate:

$$\text{CertificateValid} = A_{\text{verify},PK}(\text{CeppData}, \text{CeppSignature}),$$

where A is the specified signature algorithm, PK the public key of the MA, CeppData the value of the `CEPP-Data` header and CeppSignature the value of the `CEPP-Signature` header. The certificate signature is valid if and only if the certificate validation result (CertificateValid) is `true`.

6.3.3 Validating the CEPP data

After the CEPP signature is validated, the last step is to validate the actual CEPP data. This step consists of checking all CEPP data fields. For CEPP version 1, validating the data consists of the following checks:

1. Checking that the not before (nb) date field is before the current date,
2. Checking that the not after (na) date field is after the current date,
3. Checking that the e-mail sender (d) is equal to the actual sender of the e-mail.

If and only if all of these checks are successful, the CEPP certificate is valid.

6.3.4 Example

We will demonstrate how the validation of a CEPP certificate takes place. We will use the data obtained in the previous example (Section 6.2.3).

1. The first step is to validate the DMARC. If it was valid (if it used DKIM and the DKIM was valid), we continue to the next step.

2. The next step is the validation of the CEPP signature:

$$\text{CertificateValid} = A_{\text{verify},049...143}(\text{`v=1; ...; l=2'}, 304...156) = \texttt{true}.$$

3. Now, since the certificate signature was valid, we need to check the certificate data. We assume the current date is February 1st, 2021, 00:00h, and that the e-mail was sent by the `student.ru.nl` domain.

   - The version is 1.
   - The not before date is before the current date.
   - The not after date is after the current date.
   - The certificate subject domain is equal to the e-mail sender domain.

   Since all checks passed, the certificate is valid.

6.4 Spam report generation

As mentioned in Section 5.4, CAs (and MAs) can occasionally make mistakes. We might also have a company that was initially trustworthy but is no longer. For these cases, e-mail spam reports exist. By reporting an e-mail, a client can indicate to an MA that an e-mail sender might not (or no longer) be trustworthy. The MA can then decide to stop giving out certificates for that company.

The generation of spam reports is trivial. We have looked at the following two options:

- Create a new application-level protocol for the indication of malicious e-mail.
- Use e-mail to send a spam report to a MA.

In CEPP, we have opted for the latter. Creating, managing, and describing a new protocol is a lot of work, but it is mostly unnecessary since e-mail already meets all of our requirements.

6.4.1 Overview

The construction of a spam report consists of the following steps:

1. Construct a new e-mail.
2. Set the target of this e-mail equal to the spam report e-mail of the MA that issued the certificate for the malicious e-mail.
3. Set the body of this e-mail equal to the raw content of the malicious e-mail. This includes all headers, attachments and the e-mail body.
4. Add DMARC (and DKIM) headers to the e-mail.
5. Send the e-mail.

After following each of these steps, the MA receives an e-mail containing the exact e-mail and can choose to act based on the contents.

There are a few important points we need to take into consideration. Note that for this approach to work, each MA has to publish a working e-mail address that is used for reporting malicious e-mails. This can easily be done together with the publication of their public key. It is also essential that the MA actually checks the incoming spam reports and does not just ignore them. Furthermore, note that adding the DMARC and DKIM headers is essential. If we do not do this, it is possible to spoof the source e-mail address, which means someone would be able to generate hundreds of e-mail reports without the MA being able to ignore duplicate reports.

6.5 Spam report validation

The validation of a spam report is also trivial and consists of the following steps:

1. Validate the DMARC of the received spam report e-mail. More specifically, the e-mail should have DKIM enabled, and the certificate should be valid. This step is required to make sure that a spam report e-mail sender address has not been spoofed.
2. Do a full CEPP certificate validation on the body of the received report. The body contains the raw source of the reported e-mail. This validation consists of validating the DMARC, CEPP signature and CEPP data (see Section 6.3). This step is required to ensure that the actual spam report is a legitimate e-mail, not an e-mail that was constructed to look like a malicious e-mail.

If both of these steps are successfully completed, the e-mail report is valid. It is up to the MA to take further action.

6.5.1 Report filtering

It is highly recommended to filter e-mail reports. The most obvious filter is treating reports of a level 2 (basic) certificate as more probable than a report of a level 3 (extended) certificate. It is also helpful to ignore (or count) duplicate spam reports. If, for some reason, someone decides to report the same e-mail multiple times, an MA would not want to spend more time than necessary on those reports. Therefore, filtering out, grouping, and counting reports can be very beneficial.

Techniques can also be used to check the contents of the e-mail. Think about spam filters ((Bayesian) classifiers) and URL/DNS blacklists. These techniques can be used to decrease the amount of human labour that MAs have to do.

6.6 CEPP usage locations

The generation and validation of CEPP certificates and spam reports happen at different locations. Some of the steps in the CEPP protocol can be executed at multiple locations. In this section, we will give the locations at which each CEPP step can take place.

Certificate generation

The generation of CEPP certificates happens at the responsible MA. They then deliver these certificates to their customers (domain owners).

Attaching a certificate to an e-mail is done by an (outbound) MTA. They have the CEPP certificate, and they attach it to all outgoing e-mails. The MUAs do not have this information since they often do not own the domain they are using. For example, popular e-mail providers (not clients!) like Outlook and Gmail provide e-mail addresses for users. Since these users do not own the `outlook` or `gmail` domains, they would be unable to request a certificate from an MA. For this reason, MTAs always attach the certificates to the e-mails.

Certificate validation

The certificate validation can take place at two locations. The (inbound) MTA and the receiving MUA can validate the CEPP certificate. If an MTA validates the certificate, they can communicate to the MUA that an e-mail is trusted. An MUA can also validate the CEPP certificate and then display the result to the end-user.

Spam report generation

The generation of spam reports can, once again, take place at two locations. Either the MUA or MTA can do this. Especially in browser-based e-mail clients, it could be logical that the MTA is responsible for sending spam reports. However, for standalone MUAs like Mozilla Thunderbird, spam report generation and sending occur at the MUA.

Spam report validation

The validation of spam reports only ever happens at the responsible MA. MTAs should not look at the contents of a spam report. As mentioned in Section 5.4.2, this is also part of a problem; MTAs can see which e-mails are being reported.

Chapter 7

Proof of concept

To illustrate CEPP's functionality, we have created a proof of concept. We did this by creating an add-on for the e-mail client Thunderbird. In this chapter, we will explain what the add-on does (Section 7.1) and discuss an example (Section 7.2).

7.1 The CEPP addon

Mozilla's Thunderbird e-mail client offers functionality to create add-ons. These add-ons are developed in Javascript and can directly interact with many Thunderbird parts through their API. An overview of the code and the actual code of the most important parts can be found in Appendix A.

7.1.1 The basic functionality

The CEPP add-on allows for a demonstration of how CEPP works. It contains three major parts:

1. Functionality to add a CEPP certificate to an e-mail that is being sent.
2. Functionality to display, for incoming e-mails, whether they are valid according to the CEPP specification.
3. Functionality to report incoming e-mails.

If CEPP becomes a standard, part 2 and 3 of this add-on can (after some alterations) be used. Part 1 would never be used in the software of an MUA since (as explained in Section 6.6) MTAs add the CEPP certificate to the e-mail. Concretely, this means that part 1 is only conducted to suit the proof of concept.

7.1.2 Development assumptions

While developing this add-on, we have taken some measures to make the process easier. After all, the add-on is a proof of concept and not a full implementation. We have taken the following measures to make the development process easier:

- In the add-on, we do not check the DMARC of an e-mail. We assume that the DMARC check (as described in Section 6.2.1) passed successfully.
- In the add-on, we only offer ECDSA with SHA256 signatures. Although multiple public key and hashing algorithm combinations are supported in CEPP, we have only implemented the `ECDSA_WITH_SHA_256` option.
- In the add-on, we have hard-coded the (only) MA information into a Javascript file. Usually, the public MA information would be stored securely by the responsible application.

After using these simplifications, we have successfully created a proof of concept of the CEPP protocol, as given in Appendix A.

7.2 Usage showcase

In this section, we will give an overview of all visual elements in the Thunderbird add-on.

7.2.1 Generating a certificate

In the Thunderbird e-mail compose window, one can click the 'CEPP' button. This will create a small popup that is displayed in Figure 7.1. In our proof of concept, we have hard-coded a single MA: the 'RU Certificate Authority'. In Figure 7.1, we can see that this MA is selected. Furthermore, we can see all relevant encryption details in the 'Issuer Encryption Details' field.

7.2.2 Adding a certificate

After all the certificate generation popup settings have been set, one can add the certificate to the e-mail headers by clicking the 'Attach Certificate' button. This button will check all input data, and if all data is valid, it will add the certificate to the e-mail. This is displayed in Figure 7.2.

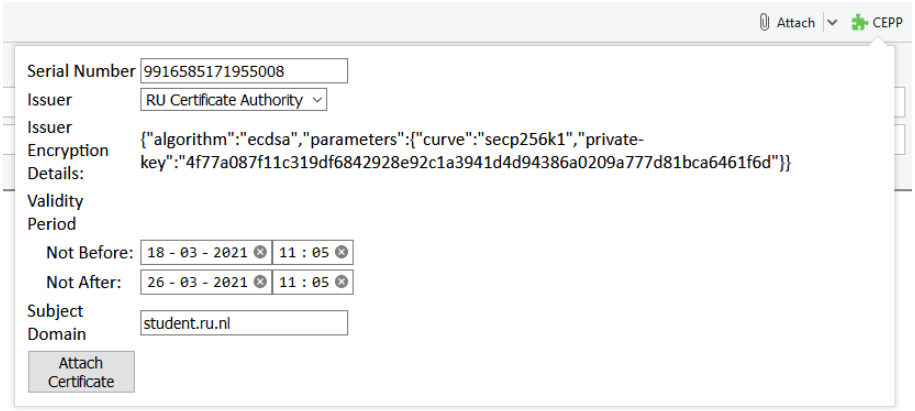

Figure 7.1: Popup window for generating a CEPP certificate

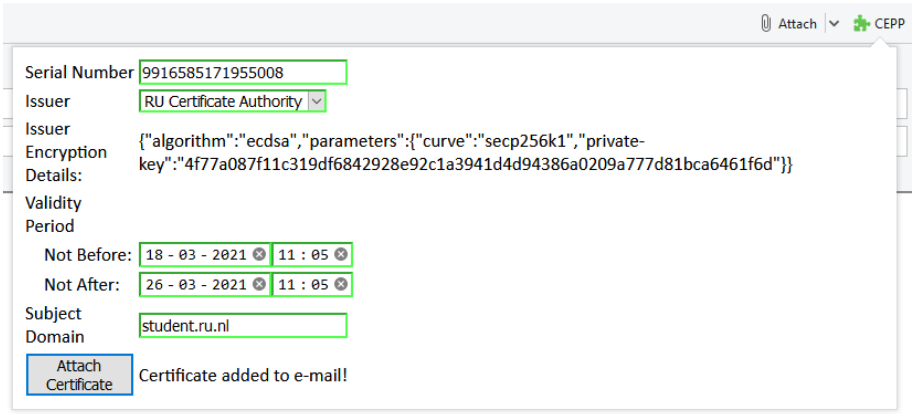

Figure 7.2: Popup window after the CEPP certificate is added to the e-mail headers

7.2.3 Validating a certificate

Now, when receiving an e-mail, the add-on will display whether the e-mail is trusted or not. We have created two example e-mails, only one of which had a valid CEPP certificate attached. The valid e-mail is displayed in Figure 7.3, and the invalid e-mail is displayed in figure 7.4.

7.2.4 Reporting an e-mail

Lastly, in the Thunderbird mail display screen, we have added the spam report button. This button is displayed in Figure 7.5. After this button is clicked, a new mail screen will be opened that already contains all information. The client only has to hit 'send' to send the e-mail to the responsible MA.

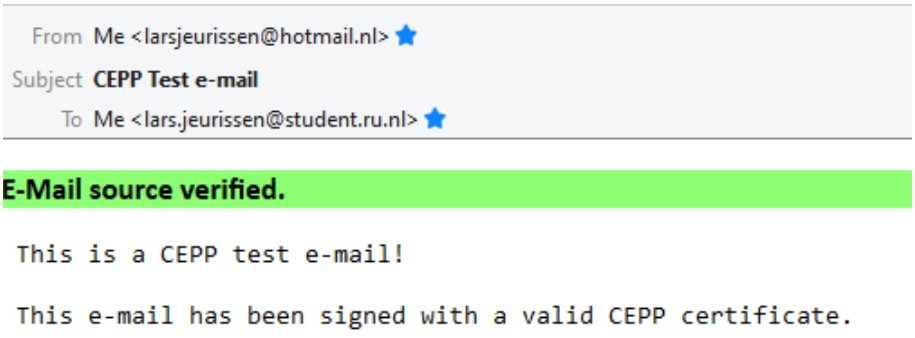

Figure 7.3: Addon visualisation for an e-mail that had a valid CEPP certificate attached to it

Figure 7.4: Addon visualisation for an e-mail that did not have a CEPP certificate attached to it

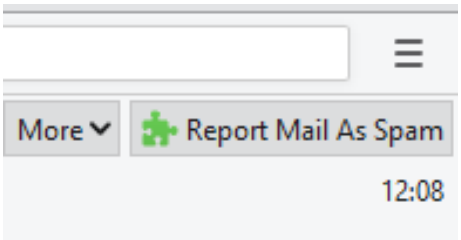

Figure 7.5: Addon button for reporting a possibly malicious e-mail

Chapter 8

Discussion

In this chapter, we will look at the functional and non-functional requirements mentioned in Chapter 4. Furthermore, we will list the advantages and disadvantages of CEPP.

8.1 Evaluation of functional requirements

1. CEPP guarantees that each domain is either trusted or not, through the use of MAs. If the client trusts an MA and that MA trusts a particular domain, then the client can also trust that domain. By validating the CEPP certificate, e-mail clients can determine whether they can trust an e-mail or not.

2. Using the e-mail reporting system, e-mail users can let the responsible MA know when one of their trusted domains sends malicious e-mail. MAs can then decide to take action.

3. CEPP is secure, assuming that the underlying cryptographic protocols are secure. If the protocol is followed correctly, it is not possible to send an e-mail from a domain that is not trusted and have it displayed as being trusted. Note that for this to work, DMARC (and in particular; DKIM) are required: the client needs to trust the public keys that are published in the domain's DNS.

4. The spam report system is also secure (assuming that the underlying cryptographic protocols are secure). It is not possible to send a report of an e-mail that was never sent.

8.2 Evaluation of non-functional requirements

1. Since CEPP uses multiple MAs, there is no single organization responsible for determining which domains are trusted.

2. Since an MA wants to stay trusted by e-mail clients, its main purpose will always be to only trust domains that are trustworthy.

3. CEPP allows for gradual deployment. E-mail clients can add visual indicators for CEPP e-mails. In the long run, it would be better to only accept e-mails that have a valid CEPP certificate, but this is infeasible in the beginning.

8.3 Overview of advantages and disadvantages

In this section, we will provide an overview of all advantages and disadvantages. It is useful to mention that the CEPP protocol is a protocol that is executed at the 'Communication level' of preventing phishing attacks (Section 2.4.1).

8.3.1 Advantages

- CEPP can greatly reduce the number of phishing e-mails (and malicious e-mails in general). We will always have companies that are trusted by MAs mistakenly, but that is why the spam report system is introduced.

- Theoretically, CEPP can also be applied to other areas than just e-mail. In general, CEPP is just a proposal where MAs are also required to check what their clients use the certificates for. This same principle can also be applied to (for example) websites.

8.3.2 Disadvantages

- The current e-mail report system is not very privacy-friendly. MAs can read the full content of reported e-mails, and this is not very ideal. It is necessary, though, since otherwise, MAs cannot determine whether an e-mail is malicious.

- CEPP is unable to stop **all** malicious e-mail. However, we do not think it is possible to create a system that stops all malicious e-mail. At one point or another, a human always has to decide whether an e-mail is malicious or not. Even if a computer decides this (through, for example, a Bayesian classifier), a human always has to configure *how* the computer decides which e-mail is malicious. Since humans make mistakes, and many tricks exist to bypass spam filters, we do not think it is possible to create a system that always works perfectly. CEPP comes pretty close to an ideal system.

- CEPP is unable to stop certain types of phishing attacks. For example, attacks where the attacker gets control over someone's e-mail account can not be stopped by CEPP.

Chapter 9

Conclusion

We have seen that phishing is a huge problem in today's society. It causes many people to lose money or leak passwords. With CEPP, we have provided a general proposal that can prevent a lot of these attacks. We have given a proof of concept implementation that proves CEPP's functionality. This implementation, in the form of a Thunderbird add-on, shows that CEPP works.

9.1 Future research

A problem with the current approach is that responsible MAs can see each spam report's content. This has a significant impact on the privacy of the person who reports the e-mail. Furthermore, MTAs can see which e-mails are being reported by clients. Research can be done to alter the current approach to prevent these problems.

We have provided a 'proof of concept' add-on but no full implementation yet. It is possible to write both MUA and MTA software (possibly in the form of a GitHub pull request) that add CEPP support to major e-mail clients and MTA's.

Lastly, it might also be worth checking possible integrations of CEPP into already existing protocols. For example, protocols like SMTPsec [Ste20] or S/MIME can significantly benefit from a CEPP integration. S/MIME, in particular, is already used, so integration might be easier there. Research can be done to check these possibilities.

Bibliography

[All07] E. Allman. DomainKeys Identified Mail (DKIM) Signatures. RFC 4871, IETF, 5 2007.

[APW20] APWG. Phishing Activity Trend Report Q2 2020. 27 August 2020. https://docs.apwg.org/reports/apwg_trends_report_q2_2020.pdf.

[BESH07] Mohamad Badra, Samer El-Sawda, and Ibrahim Hajjeh. Phishing Attacks and Solutions. In *Proceedings of the 3rd International Conference on Mobile Multimedia Communications*, MobiMedia '07, Brussels, BEL, 2007. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

[Coc13] Dean Coclin. What Are the Different Types of SSL Certificates? Certificate Authority Security Counsil, Aug 2013.

[Com17] Bleeping Computer. Extended Validation EV Certificates Abused To Create Insanely Believable Phishing Sites. December 12, 2017. https://www.bleepingcomputer.com/news/security/extended-validation-ev-certificates-abused-to-create-insanely-believable-phishing-sites/.

[Coo08] D. Cooper. Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile. RFC 5280, IETF, 5 2008.

[Dic20] Dictionary.com. Certificate. Certificate.com, Retrieved December 8, 2020. https://www.dictionary.com/browse/certificate.

[DNS20] DNSBL.info. What is a DNSBL. Retrieved 15 November 2020.

[DTH06] Rachna Dhamija, J. D. Tygar, and Marti Hearst. Why Phishing Works. In *Proceedings of the SIGCHI Conference*

on Human Factors in Computing Systems, CHI '06, page 581–590, New York, NY, USA, 2006. Association for Computing Machinery.

[Ero05] P. Eronen. Pre-Shared Key Ciphersuites for Transport Layer Security (TLS). RFC 4279, IETF, 12 2005.

[FI12] Fox-IT. Black Tulip Report of the investigation into the DigiNotar Certificate Authority breach. 13 August 2012. https://github.com/juliocesarfort/public-pentesting-reports/raw/master/Fox-IT/Fox-IT_-_DigiNotar.pdf.

[For20] Force.com. Microsoft Included CA Certificate List. Retrieved December 8, 2020. https://ccadb-public.secure.force.com/microsoft/IncludedCACertificateReportForMSFT.

[Gel11] R. Gellens. Message Submission for Mail. RFC 6409, IETF, 11 2011.

[HvdHS08] Andrew Harding, Timothy W. van der Horst, and Kent E. Seamons. Wireless Authentication Using Remote Passwords. In *Proceedings of the First ACM Conference on Wireless Network Security*, WiSec '08, page 24–29, New York, NY, USA, 2008. Association for Computing Machinery.

[Iye20] Rishi Iyengar. Twitter accounts of Joe Biden, Barack Obama, Elon Musk, Bill Gates, and others apparently hacked. CNN Business, July 15, 2020. https://edition.cnn.com/2020/07/15/tech/twitter-hack-elon-musk-bill-gates/index.html.

[JJJM07] Tom N. Jagatic, Nathaniel A. Johnson, Markus Jakobsson, and Filippo Menczer. Social Phishing. *Commun. ACM*, 50(10):94–100, October 2007.

[Jou15] Stu Jouwerman. Spear Phishing Attack Makes $17.2 Million In Three Days. KnowBe4, 7 February 2015. https://blog.knowbe4.com/spear-phishing-attack-makes-17.2-million-in-three-days.

[KNL20] Bhargav Kuchipudi, Ravi Teja Nannapaneni, and Qi Liao. Adversarial Machine Learning for Spam Filters. In *Proceedings of the 15th International Conference on Availability, Reliability and Security*, ARES '20, New York, NY, USA, 2020. Association for Computing Machinery.

[LS07] Changwei Liu and Sid Stamm. Fighting Unicode-Obfuscated Spam. In *Proceedings of the Anti-Phishing Working Groups 2nd Annual ECrime Researchers Summit*, eCrime '07, page 45–59, New York, NY, USA, 2007. Association for Computing Machinery.

[MMDCL09] Jose Marcio Martins Da Cruz and John Levine. URL filtering. Anti-Spam Research Group, May 2009. https://wiki.asrg.sp.am/wiki/URL_filtering.

[Moz20] Mozilla. Mozilla Included CA Certificate List. Retrieved December 8, 2020. https://wiki.mozilla.org/CA/Included_Certificates.

[Mun15] Phil Muncaster. Email Scam Netted $17m From Single Firm. InfoSecurity, 9 February 2015. https://www.infosecurity-magazine.com/news/email-scam-netted-17-million-from/.

[Ram10] Zulfikar Ramzan. Phishing Attacks and Countermeasures. In *Handbook of Information and Communcation Security*, pages 433–448. Springer, 2010.

[Res18] E. Rescorla. The Transport Layer Security (TLS) Protocol Versoin 1.3. RFC 8446, IETF, 8 2018.

[Ste20] Steven Wallis de Vries. Designing a simple and secure delivery protocol: SMTPsec. Radboud University, June 28, 2020.

[SY19] Alex Sumner and Xiaohong Yuan. Mitigating Phishing Attacks: An Overview. In *Proceedings of the 2019 ACM Southeast Conference*, ACM SE '19, page 72–77, New York, NY, USA, 2019. Association for Computing Machinery.

[Sym19] Symantec. Internet Security Threat Report. volume 24. February 2019. https://docs.broadcom.com/doc/istr-24-2019-en.

[Tay07] D. Taylor. Using the Secure Remote Password (SRP) Protocol for TLS Authentication. RFC 5054, IETF, 11 2007.

[Tho04] B. Thorson. How Spammers Bypass E-mail Security. EE Times, 19 July 2004. https://www.eetimes.com/how-spammers-bypass-e-mail-security/#.

[Ver20] Verizon. Data Breach Investigations Report. 2020. https://enterprise.verizon.com/resources/reports/2020-data-breach-investigations-report.pdf.

[WAB16] Adam Wright, Skye Aaron, and David W. Bates. The Big Phish: Cyberattacks Against U.S. Healthcare Systems. *Journal of General Internal Medicine*, 31(10):1115–1118, Oct 2016.

[YMWBYM16] K. Young-McLear, G. Wyman, J. Benin, and Y. Young-Mclear. A White Hat Approach to Identifying Gaps Between Cybersecurity Education and Training: A Social Engineering Case Study. In *Advances in Human Factors in Cybersecurity*, pages 229–237. Springer, 2016.

Appendix A

Thunderbird addon code

A.1 Project structure

The project structure of the Thunderbird addon is displayed here.

```
|   background.html
|   compose_popup.html
|   manifest.json
|
\---scripts
    +---api
    |   +---compose_message_headers
    |   |       implementation.js
    |   |       schema.json
    |   |
    |   \---elliptic
    |           elliptic.js
    |
    +---cepp
    |   |   certificate.js
    |   |   compose.js
    |   |   display.js
    |   |
    |   \---mail_display_injections
    |           default.js
    |           mail_invalid_incorrect.js
    |           mail_invalid_no_header.js
    |           mail_valid.js
    |
    \---data
            ca_private_keys.js
            trusted_ca_public_keys.js
```

The `manifest.json` is the home file for each Thunderbird Addon. It contains all general information, like the addon name, ID, version, and more. Furthermore, this file contains functionality for adding buttons, using permissions and adding experiment APIs.

The `compose_popup.html` file is the HTML file displayed when the CEPP button in the Thunderbird e-mail compose window is clicked. It contains all required fields for adding a certificate to an e-mail.

The `background.html` file is loaded immediately when the addon is loaded. It links to a few Javascript files that, together, create permanent background functionality.

The `scripts` folder contains all Javascript files that the addon uses. It is split into three parts:

A.1.1 The 'API' part

This part contains Javascript files that we did not write but are used as an API. The API consists of the `compose_message_headers` API, which allows us to add custom headers to an e-mail, and the `elliptic` API, which enables us to do elliptic curve cryptography.

A.1.2 The 'cepp' part

This part contains the main CEPP functionality. The `certificate.js` file contains all required functionality for generating and validating CEPP certificates. The `compose.js` file is used by the `compose_popup.html` file, and allows us to create and add certificates. The `display.js` file is ran from the `background.html` file, and contains functionality for validating the correctness of a CEPP certificate. It visualises this through e-mail injections, all of which are in the `mail_display_injections` folder.

A.1.3 The 'data' part

This part contains all (hard-coded) mail CA data. The `ca_private_keys.js` file contains all private mail CA data. It contains the private keys of the mail CA signatures.

The `trusted_ca_public_keys.js` file contains all public mail CA data. Usually, this data would be in MUA's or MTA's that want to verify CEPP certificates. It contains the public keys of the mail CA signatures.

A.2 Code files

In this section, we will go on to list the most important files. All files can also be found on GitHub:

`https://github.com/MeItsLars/CEPP/`

The files that we will be showing here (in this order) are:

- `background.html`,
- `compose_popup.html`,
- `manifest.json`,
- `certificate.js`,
- `compose.js`,
- `display.js`,
- `ca_private_keys.js`,
- `trusted_ca_public_keys.js`.

A.2.1 background.html

```
1  <html lang="en">
2  <head>
3      <meta charset="utf-8">
4      <script src="scripts/data/trusted_ca_public_keys.js"></
           script>
5      <script src="scripts/api/elliptic/elliptic.js"></script>
6      <script src="scripts/cepp/certificate.js"></script>
7      <script type="module" src="scripts/cepp/display.js"></script
           >
8  </head>
9  <body>
10 </body>
11 </html>
```

A.2.2 compose_popup.html

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="utf-8">
        <title>Add CEPP certificate</title>
    </head>
    <body>
        <table style="width:100%">
            <tr>
                <td><label for="serialNumber">Serial Number</label></td>
                <td><input id="serialNumber" type="text"></td>
            </tr>
            <tr>
                <td><label for="issuer">Issuer</label></td>
                <td>
                    <select id="issuer"></select>
                </td>
            </tr>
            <tr>
                <td><label for="issuerEncryptionDetails">Issuer Encryption Details:</label></td>
                <td><span id="issuerEncryptionDetails">-</span></td>
            </tr>
            <tr>
                <td>Validity Period</td>
            </tr>
            <tr>
                <td><label for="notBeforeDate"> Not Before:</label><label for="notBeforeTime"></label></td>
                <td><input id="notBeforeDate" type="date"><input id="notBeforeTime" type="time"></td>
            </tr>
            <tr>
                <td><label for="notAfterDate"> Not After:</label><label for="notAfterTime"></label></td>
                <td><input id="notAfterDate" type="date"><input id="notAfterTime" type="time"></td>
            </tr>
            <tr>
                <td><label for="subjectDomain">Subject Domain</label></td>
                <td><input id="subjectDomain" type="text"></td>
            </tr>
            <tr>
                <td><button id="attachCertificateButton" type="button">Attach Certificate</button></td>
                <td><span id="attachCertificateResult"></span></td>
            </tr>
```

```
42          </table>
43          <script src="scripts/data/ca_private_keys.js"></script>
44          <script src="scripts/api/elliptic/elliptic.js"></script>
45          <script src="scripts/cepp/certificate.js"></script>
46          <script type="module" src="scripts/cepp/compose.js"></
                script>
47      </body>
48  </html>
```

A.2.3 manifest.json

```
{
  "manifest_version": 2,
  "name": "CEPP",
  "description": "An example proof of concept addon for
     preventing phishing attacks via e-mail.",
  "version": "1.0",
  "author": "Lars Jeurissen",
  "applications": {
    "gecko": {
      "id": "cepp@cs.ru.nl",
      "strict_min_version": "78.0"
    }
  },
  "compose_action": {
    "default_title": "CEPP",
    "default_popup": "compose-popup.html"
  },
  "message_display_action": {
    "default_title": "Report Mail As Spam"
  },
  "background": {
    "page": "background.html"
  },
  "permissions": [
    "messagesRead",
    "messagesModify",
    "compose",
    "tabs",
    "accountsRead"
  ],
  "experiment_apis": {
    "composeMessageHeaders": {
      "schema": "scripts/api/compose_message_headers/schema.json
         ",
      "parent": {
        "scopes": [
          "addon_parent"
        ],
        "paths": [
          [
            "composeMessageHeaders"
          ]
        ],
        "script": "scripts/api/compose_message_headers/
           implementation.js"
      }
    }
  }
}
```

A.2.4 certificate.js

```
1  const EC = require('elliptic').ec;
2
3  /**
4   * Generates a certificate signature from given certificate data
        .
5   * @param {Object} certificateData The certificate data object
6   * @returns {null|String}          The signature if the
        certificate data was valid, null otherwise
7   */
8  function generateCertificateSignature(certificateData) f
9      // Choose which implementation to take based on the
           certificate algorithm
10     switch (certificateData.a) f
11         case 'ecdsa':
12             return generateECDSACertificateSignature(
                   certificateData);
13         default:
14             return null;
15     g
16 g
17
18 /**
19  * Generates an ECDSA signature given certificate data.
20  * @param {Object}   certificateData
21  * @returns {String} The signature
22  */
23 function generateECDSACertificateSignature(certificateData) f
24     const parameters = caPrivateKeys[certificateData.i].
           parameters;
25
26     // Initialize a new ECDSA instance from the parameters
27     const ec = new EC(parameters.curve);
28
29     // Parse the ECDSA key from the parameters
30     const key = ec.keyFromPrivate(parameters["private-key"], '
           hex');
31
32     // Sign the input with the key and return the result
33     return key.sign(createCertificateDataString(certificateData)
           ).toDER('hex');
34 g
35
36 /**
37  * Verifies a CEPP signature given its data and signature
        headers.
38  * @param {String} certificateDataString The content of the CEPP
        -Data header
39  * @param {String} signature             The content of the CEPP
        -Signature header
40  * @returns {boolean}                    True if the signature
        was valid, false otherwise
41  */
```

```
42 function verifyCertificateSignature(certificateDataString,
       signature) {
43     // Parse and check the certificate data
44     const certificateData = parseCertificateDataString(
           certificateDataString);
45     if (certificateData == null) return false;
46
47     switch (certificateData.a) {
48         case 'ecdsa':
49             return verifyECDSACertificate(certificateData,
                   certificateDataString, signature);
50         default:
51             return false;
52     }
53 }
54
55 /**
56  * Verifies an ECDSA CEPP signature given its data and signature
        headers
57  * @param {Object} certificateData       The parsed certificate
       data object
58  * @param {String} certificateDataString The content of the CEPP
       -Data header
59  * @param {String} signature             The content of the CEPP
       -Signature header
60  * @returns {boolean}                    True if the signature
       was valid, false otherwise
61  */
62 function verifyECDSACertificate(certificateData,
       certificateDataString, signature) {
63     // We use a try-catch to make sure that, if the certificate
          signature was forged, the output is still 'false'
64     try {
65         // Check that the given certificate authority is trusted
66         const caData = trusted_ca_public_keys[certificateData.i
               ];
67         if (caData == null) return false;
68         const parameters = caData.parameters;
69
70         // Initialize a new ECDSA instance from the parameters
71         const ec = new EC(parameters.curve);
72
73         // Parse the ECDSA key from the parameters
74         const key = ec.keyFromPublic(parameters["public-key"], '
               hex');
75
76         // Verify and return the result
77         return key.verify(certificateDataString, signature);
78     } catch (err) {
79         console.error(err);
80         return false;
81     }
82 }
83
```

```
84  /**
85   * Creates a certificate data string given a certificate data
         object
86   * @param {Object} certificateData The certificate data object
87   * @returns {string}               The formatted certificate
         data string
88   */
89  function createCertificateDataString(certificateData) f
90      return `v=${f}certificateData.v${g}; s=${f}certificateData.s${g}; a=${f
            }certificateData.a${g}; i=${f}certificateData.i${g};` +
91          ` nb=${f}certificateData.nb${g}; na=${f}certificateData.na${g}; d=
              ${f}certificateData.d${g}; l=${f}certificateData.l${g}`;
92  g
93
94  // An array holding the expected order of certificate data
        parameters
95  const expectedOrder = ['v', 's', 'a', 'i', 'nb', 'na', 'd', 'l'
        ];
96
97  /**
98   * Secure method for parsing a CEPP data header and returning an
         object containing all CEPP details.
99   * If the header was malformed, this function will return null.
100  * @param {String} input The content of the CEPP-Data header
101  * @returns {Object}     An object containing the parsed string
       if parsing succeeded, or null if parsing failed
102  */
103 function parseCertificateDataString(input) f
104     // Split the parameters
105     const args = input.split(';');
106     let i = 0;
107     // Check that the parameter count is exactly the expected
          amount of parameters
108     if (args.length !== expectedOrder.length) return null;
109
110     const result = f${g};
111
112     // Loop through all expected parameters, and add them to the
          result
113     for (const expectedKey of expectedOrder) f
114         const arg = args[i++];
115         const argParts = arg.split('=');
116         // If the formatting was not a=b or the key was not
              equal to the expected key, fail
117         if (argParts.length !== 2 ${jj} argParts[0].trim() !==
              expectedKey) return null;
118         result[expectedKey] = argParts[1];
119     g
120
121     return result;
122 g
```

A.2.5 compose.js

```
1  // Retrieving all elements by their id's:
2  const serialNumber = document.getElementById('serialNumber');
3  const issuer = document.getElementById('issuer');
4  const issuerEncryptionDetails = document.getElementById('
       issuerEncryptionDetails');
5  const notBeforeDate = document.getElementById('notBeforeDate');
6  const notBeforeTime = document.getElementById('notBeforeTime');
7  const notAfterDate = document.getElementById('notAfterDate');
8  const notAfterTime = document.getElementById('notAfterTime');
9  const subjectDomain = document.getElementById('subjectDomain');
10 const attachCertificateButton = document.getElementById('
       attachCertificateButton');
11 const attachCertificateResult = document.getElementById('
       attachCertificateResult');
12
13 // Setting a random Serial Number:
14 serialNumber.value = Math.floor(Math.random() □ 10E15);
15
16 // Setting a default Issuer & set details:
17 for (const key in caPrivateKeys) f
18     issuer.options[issuer.options.length] = new Option(key);
19 g
20 updateIssuerEncryptionDetails();
21 issuer.addEventListener("change", () =>
       updateIssuerEncryptionDetails());
22
23 function updateIssuerEncryptionDetails() f
24     issuerEncryptionDetails.textContent = JSON.stringify(
           caPrivateKeys[issuer.value]);
25 g
26
27 // Setting an example Validity Period:
28 const date = new Date();
29 date.setDate(date.getDate() - 1);
30 notBeforeDate.value = date.toISOString().slice(0, 10);
31 notBeforeTime.value = date.toISOString().slice(11, 16)
32 date.setDate(date.getDate() + 8);
33 notAfterDate.value = date.toISOString().slice(0, 10);
34 notAfterTime.value = date.toISOString().slice(11, 16)
35
36 // Setting the current Subject Domain:
37 getCurrentEmailAddress().then(address => f
38     if (address == null) return;
39     subjectDomain.value = address.split('@')[1];
40 g);
41
42 // Add a button for attaching the certificate:
43 attachCertificateButton.addEventListener("click", () => f
44     // Initialize values
45     const success = '#52ff4c';
46     const failure = '#ff4c4c';
47
```

```
48      // Util function to check element validity
49      const checkValidity = element => {
50          const valid = element.value.length > 0;
51          element.style.borderColor = valid ? success : failure;
52          return valid;
53      };
54
55      // Check that all input fields are correct
56      const serialNumberValid = !isNaN(serialNumber.value) &&
            serialNumber.value.length > 0;
57      serialNumber.style.borderColor = serialNumberValid ? success
            : failure;
58      // The issuer is always correct (Since it's a choose box)
59      issuer.style.borderColor = success;
60      // Check validity for all remaining default elements
61      const notBeforeDateValid = checkValidity(notBeforeDate);
62      const notBeforeTimeValid = checkValidity(notBeforeTime);
63      const notAfterDateValid = checkValidity(notAfterDate);
64      const notAfterTimeValid = checkValidity(notAfterTime);
65      const subjectDomainValid = checkValidity(subjectDomain);
66      // Check whether all of them were correctly entered
67      const allSuccess = serialNumberValid && notBeforeDateValid
            && notBeforeTimeValid
68          && notAfterDateValid && notAfterTimeValid &&
                subjectDomainValid;
69
70      // If all were correctly entered, add the certificate to the
            mail.
71      // Send a message containing info whether input was correct.
72      if (allSuccess) {
73          getCurrentTabId().then(tabId => {
74              if (tabId == null) return;
75
76              const certificateData = {
77                  'v': '1',
78                  's': serialNumber.value,
79                  'a': caPrivateKeys[issuer.value].algorithm,
80                  'i': issuer.value,
81                  'nb': notBeforeDate.value.replaceAll('-', '').
                        substring(2) +
82                      notBeforeTime.value.replaceAll(':', '') + '
                            00Z',
83                  'na': notAfterDate.value.replaceAll('-', '').
                        substring(2) +
84                      notAfterTime.value.replaceAll(':', '') + '00
                            Z',
85                  'd': subjectDomain.value,
86                  'l': '2'
87              }
88
89              // Add the certificate headers to the e-mail headers
90              messenger.composeMessageHeaders.addComposeHeader(
                    tabId, 'CEPP-Data',
91                  createCertificateDataString(certificateData));
```

```
92              messenger.composeMessageHeaders.addComposeHeader(
                    tabId, 'CEPP-Signature',
93                  generateCertificateSignature(certificateData));
94
95              attachCertificateResult.innerText = 'Certificate
                    added to e-mail!';
96          });
97      } else {
98          attachCertificateResult.innerText = 'Failed to add
                certificate.';
99      }
100 });
101
102 /**
103  * Returns a promise that delivers the currently opened
        ThunderBird tab ID, or null of no such tab is available
104  * @returns {Promise<Number|Null>} A promise that delivers the
        ID of the tab that is currently opened by Thunderbird
105  */
106 function getCurrentTabId() {
107     return new Promise(resolve => {
108         // Retrieve the current tab
109         messenger.tabs.query({active: true, currentWindow: true
                }).then(tabs => {
110             // If we failed to get the current tab for some
                    reason, return null
111             if (tabs.length === 0) {
112                 resolve(null);
113                 return;
114             }
115             const tab = tabs[0];
116             resolve(tab.id);
117         });
118     });
119 }
120
121 /**
122  * Returns a promise containing the e-mail address that is
        displayed in the currently opened Thunderbird compose tab
123  * If no tab is available, or the tab is not a compose tab, this
        function returns null
124  * @returns {Promise<String|Null>} The e-mail address that is
        displayed in the opened Thunderbird compose tab
125  */
126 function getCurrentEmailAddress() {
127     return new Promise(resolve => {
128         // Retrieve the current tab
129         getCurrentTabId().then(tabId => {
130             if (tabId == null) {
131                 resolve(null);
132                 return;
133             }
134
135             // Retrieve the compose details from the current tab
```

```
136                 messenger.compose.getComposeDetails(tabId).then(
                        details => f
137                     // Retrieve the ID of the currently used account
138                     const id = details.identityId;
139                     // Loop through all user acounts to find the
                            account with the id
140                     messenger.accounts.list().then(accounts => f
141                         accounts.forEach(account => f
142                             account.identities.forEach(identity => f
143                                 if (identity.id === id) f
144                                     resolve(account.name);
145                                 g
146                             g);
147                         g);
148                         resolve(null);
149                     g);
150                 g);
151             g);
152         g);
153 g
```

A.2.6 display.js

```
1  /**
2   * Given a unique e-mail ID, returns a promise that delivers the
        parsed CEPP header, or null if the header was
3   * not available or invalid.
4   * @param {Number} messageId         The ID of the e-mail that
      we want to get the CEPP header from
5   * @returns {Promise<Object|Number>} A promise containing the
      parsed CEPP data, or null
6   */
7  function getCEPPData(messageId) {
8      return new Promise((resolve, reject) => {
9          messenger.messages.getFull(messageId).then(messagePart
               => {
10             messenger.messages.get(messageId).then(messageHeader
                   => {
11                 const ceppData = messagePart.headers['cepp-data'
                       ];
12                 const ceppSignature = messagePart.headers['cepp-
                       signature'];
13
14                 if (!(ceppData && ceppSignature && ceppData.
                       length === 1 && ceppSignature.length === 1))
                       {
15                     // No CEPP signature available
16                     reject(-1);
17                     return;
18                 }
19
20                 const certificateData =
                       parseCertificateDataString(ceppData[0]);
21                 if (certificateData == null) {
22                     // Invalid certificate data
23                     reject(-2);
24                     return;
25                 }
26
27                 if (!verifyCertificateData(certificateData,
                       messageHeader.author)) {
28                     // Invalid CEPP certificate data
29                     reject(-2);
30                     return;
31                 }
32
33                 if (!verifyCertificateSignature(ceppData[0],
                       ceppSignature[0])) {
34                     // Invalid CEPP certificate
35                     reject(-2);
36                     return;
37                 }
38
39                 // If all checks passed, the mail is CEPP
                       protected
```

```
40                  resolve(certificateData);
41              g);
42          g);
43      g);
44  g
45
46  // We add an event listener that checks when a new message
        display tab is opened in Thunderbird.
47  // When this happens, this listener will attempt to retrieve the
        CEPP data and display the validity.
48  messenger.messageDisplay.onMessageDisplayed.addListener((tab,
        message) => f
49      getCEPPData(message.id).then(result => f
50          showMessageResult(tab.id, 0);
51      g, error => f
52          showMessageResult(tab.id, error);
53      g);
54  g);
55
56  /**
57   * Given a Thunderbird tab and a CEPP header validity code,
         injects a script into the given tab that alters the
58   * looks of that tab to indicate the trustworthiness
59   * @param {Object} tab       The Thunderbird tab
60   * @param {Number} validCode The CEPP header validity code
61   */
62  function showMessageResult(tab, validCode) f
63      messenger.tabs.executeScript(tab.id, f file: 'scripts/cepp/
            mail_display_injections/default.js' g);
64
65      // Set a file path based on the validity code
66      let filePath;
67      switch (validCode) f
68          case 0:
69              filePath = 'scripts/cepp/mail_display_injections/
                    mail_valid.js';
70              break;
71          case -1:
72              filePath = 'scripts/cepp/mail_display_injections/
                    mail_invalid_no_header.js';
73              break;
74          case -2:
75              filePath = 'scripts/cepp/mail_display_injections/
                    mail_invalid_incorrect.js';
76              break;
77      g
78
79      // Inject the file into the tab
80      messenger.tabs.executeScript(tab.id, ffile: filePathg);
81  g
82
83  /**
84   * Given a certificate data object and a sender domain,
         validates the certificate data (not signature)
```

```
85   * @param {Object} certificateData The CEPP certificate data
         object
86   * @param {String} domain          The sender of the e-mail
87   * @returns {boolean}              True if the certificate data
         was valid, false otherwise
88   */
89  function verifyCertificateData(certificateData, domain) {
90      // Create a string for the current date
91      const date = new Date();
92      const dateString = date.getFullYear().toString().substring
            (2)
93          + ('0' + (date.getMonth().toString() + 1)).slice(-2)
94          + ('0' + date.getDay().toString()).slice(-2)
95          + ('0' + date.getHours().toString()).slice(-2)
96          + ('0' + date.getMinutes().toString()).slice(-2)
97          + ('0' + date.getSeconds().toString()).slice(-2)
98          + 'Z';
99
100     // Check certificate version and serial number:
101     if (certificateData.v !== '1' || isNaN(certificateData.s)) {
102         return false;
103     }
104
105     // Check the date correctness of the certificate
106     if (!checkDates(certificateData.nb, dateString,
            certificateData.na)) {
107         return false;
108     }
109
110     // Check the domain and sender correctness
111     const domainParts = domain.split('@');
112     if (domainParts.length !== 2) return false;
113
114     let mailDomain = domainParts[1];
115     if (domainParts[1].endsWith('>') && domainParts[1].length >
            1) {
116         mailDomain = domainParts[1].substring(0, domainParts[1].
                length - 1);
117     }
118     return certificateData.d === mailDomain;
119 }
120
121 /**
122  * Given three CEPP-formatted date strings, checks if the first
        date is before the second date, and the second date
123  * before the third
124  * @param {String} before  The 'before' date
125  * @param {String} current The 'current' date
126  * @param {String} after   The 'after' date
127  * @returns {boolean}      True if 'before <= current <= after',
        false otherwise
128  */
129 function checkDates(before, current, after) {
130     // Check that the before and after date strings are
```

```
        formatted correctly.
131     // We don't need to check the current date string, since we
            can be sure that that string is already correct.
132     if (before.length !== 13 || after.length !== 13 ||
133         isNaN(before.substring(0, 12)) || isNaN(after.substring
                (0, 12))) {
134         return false;
135     }
136
137     // Validate and return
138     return parseInt(before.substring(0, 12)) < parseInt(current.
            substring(0, 12))
139         || parseInt(after.substring(0, 12)) > parseInt(current.
                substring(0, 12));
140 }
141
142 /**
143  * Opens an e-mail compose window with a copy of the e-mail that
        is currently being watched, with the purpose
144  * of reporting a spam e-mail
145  */
146 async function onSpamReportButtonClicked() {
147     // Get the current tab
148     messenger.tabs.query({active: true, currentWindow: true}).
            then(tabs => {
149         if (tabs.length !== 1) return;
150         const tab = tabs[0];
151
152         // Get the currently displayed message
153         messenger.messageDisplay.getDisplayedMessage(tab.id).
                then(messageHeader => {
154             // Get the CEPP data of the currently displayed
                    message
155             getCEPPData(messageHeader.id).then(certificateData
                    => {
156                 // If the CEPP data was correct, retrieve the
                        responsible CA's spam e-mail
157                 const caData = trusted_ca_public_keys[
                        certificateData.i];
158                 if (caData == null) return;
159                 const spamMail = caData['spam'];
160
161                 // Retrieve the raw e-mail content of the
                        currently watched e-mail
162                 messenger.messages.getRaw(messageHeader.id).then
                        (messageString => {
163                     // Construct a new e-mail (the spam report)
164                     messenger.compose.beginNew({
165                         to: spamMail,
166                         subject: 'Spam Report for trusted source
                                .',
167                         body: messageString
168                     });
169                 });
```

```
170                 g, error => f
171                     // Do nothing
172                 g);
173             g);
174         g);
175 g
176
177 // Add a spam report button listener to the message display
178 messenger.messageDisplayAction.onClicked.addListener(
        onSpamReportButtonClicked);
```

A.2.7 ca_private_keys.js

```
caPrivateKeys = {
    'RU Certificate Authority': {
        'algorithm': 'ecdsa',
        'parameters': {
            'curve': 'secp256k1',
            'private-key': '4f77a087f11c319df6842928e92c1a39' +
                '41d4d94386a0209a777d81bca6461f6d'
        }
    }
}
```

A.2.8 trusted_ca_public_keys.js

```
const trusted_ca_public_keys = {
    'RU Certificate Authority': {
        'spam': 'spam@ca.ru.nl',
        'algorithm': 'ecdsa',
        'parameters': {
            'curve': 'secp256k1',
            'public-key': '049a55a04ad8538e460dc175bb02785' +
                '9d32eb88208b8ecb5ac2d16afaf1907' +
                '9af008db4d349cd1098bc758796c40b' +
                '4fb2b75da3557d4887c77ece7af759f2a7143'
        }
    }
}
```