Bachelor thesis
Computing Science


Radboud University

# Designing a simple and secure email delivery protocol: SMTPsec

*Author:*
Steven Wallis de Vries
S1011387
SWallisDeVries@science.ru.nl

*First supervisor/assessor:*
prof. dr. J.J.C. Daemen
J.Daemen@cs.ru.nl

*Second supervisor:*
dr. ir. B.J.M. Mennink
B.Mennink@cs.ru.nl

*Second assessor:*
dr. B.E. van Gastel
B.vanGastel@cs.ru.nl

June 28, 2020

**Abstract**

Email is used all over the world for all sorts of purposes. Sometimes it is used for sensitive communications, but it is also used for phishing and spam. To protect these sensitive communications and counter phishing, email should be secure. A user wants to be certain that email they receive is authentic and that the integrity of the contents is guaranteed. Additionally, email should remain confidential to attackers. Currently, the state of email security is not optimal to say the least. It is very complicated and not as robust as one would want it to be.

We present SMTPsec: a new, simpler and more robust email delivery protocol to gradually replace SMTP. SMTPsec guarantees authenticity and integrity of the email from the email domain of the author to the email domain of the recipient. It also ensures confidentiality between email servers. SMTPsec does not provide end-to-end security. Hence, its security goals are similar to that of the current infrastructure, but it achieves these in a simpler and more robust fashion. In this document we first study the current email architecture and then present the specification of SMTPsec and discuss its benefits over the current architecture.

# Contents

# Chapter 1

# Introduction

Email started out in the 1970s as a small concept where security was not an issue [1], but now, with 290 billion emails being sent worldwide per day [2], security has become very important. As a sender, you want to authenticate the recipient of the message, and as a recipient, you want to authenticate the origin of the message. Both may also want to ensure confidentiality of the contents.

The current email architecture tries to provide security on the level of the email provider, not on the level of the user. We use the term 'email provider' for a company that manages an email domain, such as `gmail.com` or `ru.nl`. This means that the email servers of the email domains of the author and recipient are authenticated, not the users themselves (this is the responsibility of their corresponding providers). Confidentiality is provided on each link, so not from user to user. However, we will see that even these goals are often not achieved.

## 1.1 Insecurity of email

One attack on email security is called 'spoofing'. Email spoofing is a process in which an attacker sends an email which seemingly originates from someone else by forging the sender address. This can be detected by properly authenticating the origin of the message. More specifically, the receiving provider has to verify that the message actually originated from the provider of the author. Often criminals spoof the address so that it appears to be a legitimate email from, say, a bank. With such an email they try to obtain the victim's login credentials, a practice known as phishing. The number of phishing cases does not seem to be decreasing any time soon [3], and for the third quarter of 2019 Kaspersky Lab reported more than 100 million cases of phishing emails with links to a malicious website [4].

As a user one wants to be sure that whatever email address is displayed as the sender's address of the email is actually the sender's address, but unfortunately spoofing is still an issue. In late 2017 even the Dutch government turned out to not have the appropriate countermeasures in place to prevent this spoofing of the sender's address. Because of this, attackers could send emails seemingly originating from the Dutch parliament (`tweedekamer.nl`) and even from the domain of the Dutch secret service, `aivd.nl` [5]. At the start of this research, also the `ru.nl` and `student.ru.nl` domains did not have any protection against spoofing, but now they have the same policies as the `cs.ru.nl` and `science.ru.nl` domains already had: some checks

were put into place but email providers are explicitly instructed to ignore the results.[1]

Besides the message origin, the recipient should also be authenticated. With the current architecture, this means verifying that the server claiming to be authoritative over the recipient's email domain is actually authoritative over that domain.

Furthermore, legitimate emails can contain sensitive information which the sender wants to protect from eavesdroppers by guarding the confidentiality. In the current architecture, this means encrypting the contents on each link. In many countries, secrecy of correspondence is regarded as a fundamental right, also if the correspondence takes place via the Internet.

However, currently not all of these goals are always accomplished, as we will see in Chapter 2.

## 1.2 Email security protocols

Since the introduction of email, many concepts have been developed to try to ensure the security of emails, such as SPF, DKIM, DMARC, TLS, DANE, and MTA-STS. All of these and more will be explained in Chapter 2, but for now you only have to know that these all operate between email providers and they each try to achieve some of the security goals we mentioned before. Multiple studies have been performed measuring how many email providers implement these protocols [6, 7, 8, 9], and the results are still not good: in 2018 more than half of the providers tested offer little to no protection against spoofing [10]. In a small test we conducted ourselves we found that 22 out of 99 email domains offer good protection against spoofing attacks using their domain name, except that none of these are protected against a spoofing attacker who can alter network traffic. Furthermore, only 6 of the 99 hosts have implemented the necessary protocols to enable other domains to properly authenticate their domain when sending email to it. See Appendix A.

In 2018, Hu et al. found that one of the reasons for the low adoption is the perceived complexity by system administrators and the fear of emails from their company getting rejected if they configure strict anti-spoofing policies [11]. With such a large set of protocols, all trying to patch the holes in the previous protocols, it has become a mess and making mistakes in the configuration is easy, causing legitimate emails to also be blocked by the recipient. Additionally, sometimes automatically forwarding emails may cause them to be blocked.

Also, because the usage of all security protocols is optional, it is difficult to decide when to mark an email as spam: maybe a legitimate email provider just has not implemented some protocols. Another problem is that many things, such as what domain name should be on the security certificate of the recipient, are unspecified. Some possible effects of this problem are that the address shown to the recipient may not have been checked by their email server and that the validity of the certificate of the recipient for their domain can often not be determined by the sender, both of which are in accordance with the relevant protocol standards.

## 1.3 Current alternatives

There already exist solutions that can provide end-to-end security (from user to user), most notably Pretty Good Privacy (PGP) [12] and Secure/Multipurpose Internet Mail Extensions (S/MIME) [13]. We will discuss these in more detail in Chapter 2, but the largest problem

[1] One can verify this on: `https://en.internet.nl/test-mail/`.

with these is that they require considerable effort from the user and their correspondents to set up. In contrast, the protocols mentioned before (SPF, DKIM, ...) operate between email providers so the user does not have to configure anything. For this reason, our research focuses on security between email providers and not on end-to-end security.

## 1.4 Our solution

Our research strives to find a solution for the security problems mentioned above. Hence, the research question is:

> How can we develop a robust and backwards-compatible delivery protocol to improve and simplify the security of email?

The protocol is meant to gradually replace SMTP. 'Backwards-compatible', in this case, means that systems should be able to exchange email with older systems that do not support the new protocol.

In this document we will present the concrete specification of our secure and simple email delivery protocol SMTPsec and the design process that led to it. We did not have the time to implement the protocol in the form of a prototype. SMTPsec guarantees authenticity and integrity of the email from the provider of the author to the provider of the recipient. It also ensures confidentiality between providers. Like the current architecture, SMTPsec does not provide end-to-end security. While its security guarantees are similar to those of SMTP with all security extensions, SMTPsec was designed with security in mind, so it can provide these in a simpler and more robust way. Unfortunately, interoperability with SMTP does require extra effort from providers.

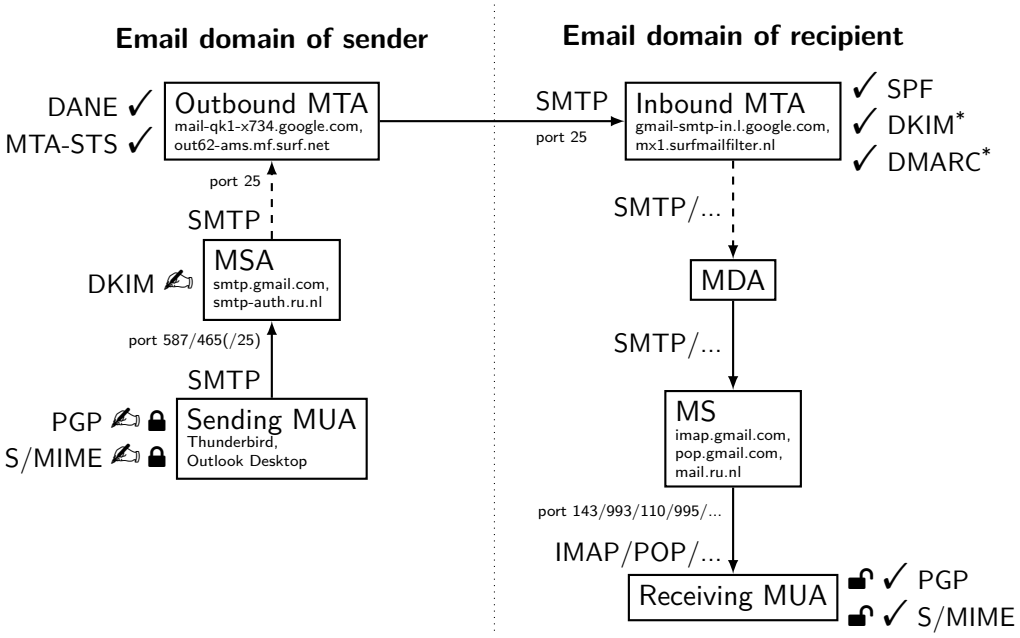
First we looked at the current email architecture and its problems so that we knew what SMTPsec should be compatible with and what it should fix, then we tried to identify common pitfalls in earlier non-successful attempts to develop new email delivery protocols so that we could try to avoid these, and with this knowledge we first made an abstract design and then a concrete specification of SMTPsec.

## 1.5 Outline

In Chapter 2 we will dive into the current email infrastructure and its problems. In Chapter 3 we will discuss previous efforts to develop a new email delivery protocol. In Chapter 4 we list the requirements and security goals for the protocol. Then, in Chapter 5 we will present several ways to tackle the various problems we face designing the protocol, and choose the combination we deem the most suitable. In Chapter 6 we will give a concrete specification for the abstract protocol designed in the previous chapter. In Chapter 7 we evaluate to what extent the protocol meets the requirements set out in Chapter 4 and discuss the benefits but also the flaws of this solution. In Appendix A we present the results of two small protocol adoption tests we performed ourselves.

# Chapter 2

# Preliminaries: current state of email




*: Check could be done later on

Figure 2.1: Overview of the email architecture

A user can compose an email message using their email client, also called Mail User Agent (MUA), for example Mozilla Thunderbird or Microsoft Outlook Desktop.[1] The MUA then submits the email to a Message Submission Agent (MSA), as shown in Figure 2.1, which is managed by an email provider like Gmail or a company using a Microsoft Exchange server. Often the MUA has to provide some form of credentials, such as a username and password. The Simple Mail Transfer Protocol (SMTP) is the protocol used to send this email. It was first standardized in 1982 in RFC (Request For Comments) 821 [14] and its newest specification is from 2008 [15].

The MSA then hands the email off to an outbound Mail Transfer Agent (MTA) of the same provider who sends it over to an inbound MTA of the recipient's provider (possibly via more internal MTAs, as indicated by the dashed arrows). Mail transfers between these MTAs also use SMTP. The MTA of the recipient's provider then sends the email to the Mail Delivery Agent

[1] The Gmail website can also be seen as a MUA, but here all transactions take place via their server.

(MDA) (again, possibly via more internal MTAs), which delivers the email to the Message Store (MS), from which the recipient can access it with their MUA, often using the Internet Message Access Protocol (IMAP) [16] or Post Office Protocol (POP) [17] to download the email [18].

In this text, we use the notion of 'provider' or 'email provider' for a company who manages an email domain. Many companies outsource the management of their email service. Often the MTA is managed by an external company such as Microsoft, or SURF for universities. This has the advantage that this other company can do all security checks and spam filtering. Also the MSA and MDA may be managed by an other company, but the problem with that is that this company will then have access to the credentials of the user, although in some cases it may be possible to use an (OAuth) authentication token instead [19]. It would technically also be possible to use TLS client authentication with a key pair belonging to the user [20, appendix A], but this is not feasible for the average user.

The syntax of an email message is described by the Internet Message Format (IMF) [21] and extended to allow for multiple parts and attachments by Multipurpose Internet Mail Extensions (MIME) [22]. An email message consists of a body and a number of headers which contain information about the message. For example, the `From` header contains the email address(es) of the author(s)[2] and the `To` header contains the address(es) of the recipient(s). The body contains the actual message, including any attachments.

The sending agent wraps the email message in an SMTP 'envelope' that contains some more information needed for delivery on top of the headers inside the message itself, see Figure 2.2. This envelope is actually nothing more than a bunch of SMTP commands, which are interpreted and then stripped off by the receiving server. The sender starts with a `HELO` (or `EHLO`, Extended `HELO`) command, which identifies the sending machine by a domain name or IP address.[3] Next, a `MAIL FROM` command is sent, which contains the address of the originator, also called the return path. This might be different from the `From` header when using automatic forwarding, see Figure 2.3. The same often holds for other mediators like mailing lists [18]. Then, recipients are listed using the `RCPT TO` command, also called the forward path [15]. Again, this might be different from the `To` header because of automatic forwarding, or the use of CC/BCC.

## 2.1 History

### 2.1.1 Authenticating the sender: preventing spoofing

#### SPF

The problem with plain SMTP is that it does not protect against spoofing. Nothing prevents an attacker from entering the email address of someone else in the `From` header or after the `MAIL FROM` command. Something had to be done, and so the Sender Policy Framework (SPF) protocol was born. SPF is used to check the authenticity of an email by verifying the domain portion (after the '@') of the sender's email address. It was standardized in 2014 [23], but the idea had been around for much longer, with RFC 4408 describing the first version in 2006 [24]. Providers that use SPF, publish special DNS records on their domain, which list IP addresses of MTAs that are allowed to send email from that domain.

[2] Yes, a message can have multiple authors according to the standard.

[3] MUAs may use local IP addresses like `[IPv6:::ffff:192.168.2.10]`.

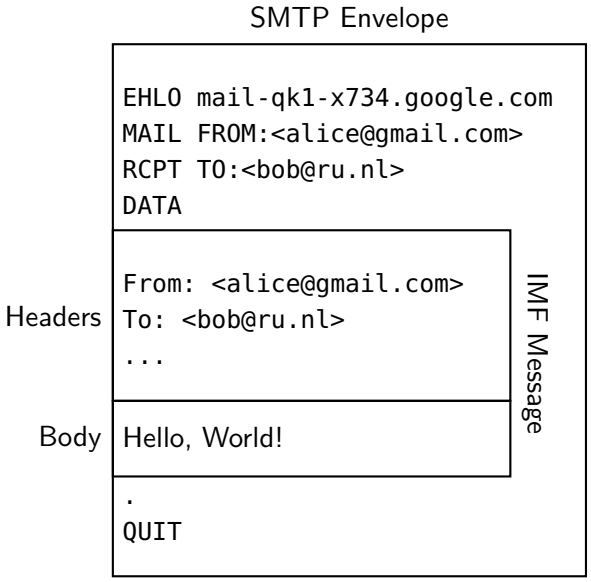



Figure 2.2: A simple SMTP transaction

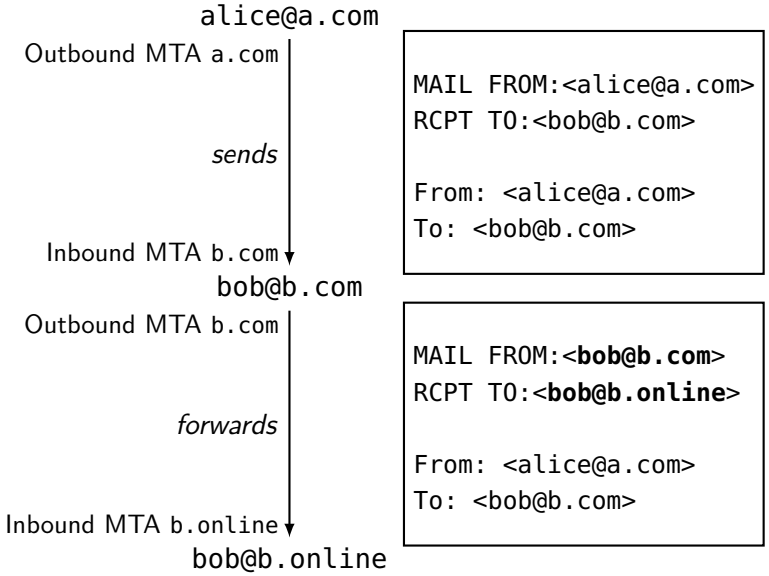



Figure 2.3: SMTP and automatic forwarding

For example, the SPF record for `ru.nl` is as follows:

```
ru.nl. 3600 IN TXT "v=spf1 redirect=spf-mail.ru.nl"
```

Which means we have to look at the SPF record of `spf-mail.ru.nl`, which is:

```
spf-mail.ru.nl. 86400 IN TXT "v=spf1 ip4:131.174.59.192/28
  include:spf-considered-harmful.science.ru.nl include:_spfout.mf.surf.net
  include:spf.protection.outlook.com include:a._spf.brightspace.com ~all"
```

As you can see, it lists a range of IP addresses that are allowed to send email originating from ru.nl and some domains such as `spf-considered-harmful.science.ru.nl` which contain more address ranges to include or exclude. Finally, `~all` means 'softfail' all other addresses, or in other words: they are "probably not authorized".

For received emails, the inbound MTA then checks the SPF DNS record of the domain sent in the `HELO`/`EHLO` and/or `MAIL FROM` commands of the SMTP envelope to see if the outbound MTA with that IP address is allowed to send email using this domain. Note that the MDA and MUA have to trust the inbound MTA to do the verification, because these do not have access to the IP address of the outbound MTA.

This does not fix the issue, however, because the end user rarely gets to see the `MAIL FROM` address. They only get to see the address in the `From` header, which is still trivially forged under SPF.

A proposed other protocol is Sender ID, which is currently still experimental and defined in 2006 [25]. Like SPF, Sender ID also does not verify the `From` header often shown to the user.

### DKIM

Later, in 2007, Domain Keys Identified Mail (DKIM) made its entrance [26]. It was designed to ensure integrity of the message. An MSA can sign parts of a message with a private key belonging to the provider. The signature is then added as a header to the message and can be verified by the receiver by using the corresponding public key, which is published via DNS. The domain name to query for this is also saved in the header. Note that there are no certificate authorities in the DKIM architecture. An advantage of DKIM over SPF is that the signature can be verified by anyone, not just by the receiving MTA, because the IP address of the sending MTA is not required.

A sample DKIM signature header is as follows:

```
DKIM-Signature: v=1; a=rsa-sha256; c=relaxed; d=student.ru.nl; h=from:to
  :subject:date:message-id:content-type:content-transfer-encoding
  :mime-version; s=canit; bh=cNqun+DE07C3PmQ62YCxtlmVojk5nhToMY4aV
  pXeLDs=; b=Xzt1kzzbrg9X6wjWZBG2nRy21CA4kcix5+38ZSvlZKkV2j5m4+LA2...
```

Here h= lists the headers to sign, bh= contains the body hash, and b= contains the signature. The d= tag specifies the signing domain and the s= tag contains the selector, which is a subdomain of the signing domain.

The corresponding DNS key record is:

```
canit._domainkey.student.ru.nl. 86400 IN TXT "v=DKIM1; t=y; k=rsa; "
  "p=MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAsAiolbF2rMTsh8EvypNq" ...
```

It contains the RSA public key used to generate the signature above. 't=y' specifies that "This domain is testing DKIM. Verifiers MUST NOT treat messages from Signers in testing mode differently from unsigned email, even should the signature fail to verify." [27, p. 28]. Hence, it might not be the best idea to keep this tag enabled.

### DMARC

The domain used for the DKIM signature does not have to equal any other domain in the message, so like with SPF, the `From` header can still be forged. Also, the actions to be taken when the SPF or DKIM checks fail, are not defined. To fix these issues, a new protocol was devised: Domain-based Message Authentication, Reporting, and Conformance (DMARC), which was specified in an informational RFC in 2015 [28]. An email provider can publish a DMARC policy record via DNS to indicate that they use SPF and/or DKIM. This way, an attacker cannot, for example, just leave out the DKIM header and pretend the provider does not sign messages. Via DMARC, providers can also specify what should happen if an email fails the SPF and DKIM checks, such as rejection of the message or classifying it as spam, with the option to send a failure report to the sending domain. Additionally, the DMARC check only succeeds if the domains used for the SPF or DKIM checks are in alignment with the `From` domain. This means that they are equal to, or a subdomain of, the `From` domain, depending on the policy.

This seems to cause a new problem, however. For example, say that `alice@a.com` sends an email to `bob@b.com`, and that Bob has configured his mailbox to automatically forward mail to `bob@b.online`. Then this means that `b.com` sends an email with `alice@a.com` in the `From` header, which will probably not be allowed by the SPF policy record of `a.com`, so the SPF check does not pass. See again Figure 2.3. To solve this, DMARC specifies that the check succeeds if SPF *or* DKIM succeeds, as DKIM has no problems with automatic forwarding.[4] But then why do we still have SPF? This is mostly for systems that do not support DKIM.

A sample DMARC policy record is as follows:

```
_dmarc.student.ru.nl. 86400 IN TXT "v=DMARC1; p=none;
  rua=mailto:dmarc-report@ru.nl; ruf=mailto:dmarc-reports@ru.nl; fo=1;"
```

This specifies that no action should be taken on failure except for sending a failure report to `dmarc-report(s)@ru.nl`.

### Reverse DNS

Then there is the odd one out that we list for completeness: the reverse DNS lookup. Some MTAs perform a reverse DNS (`PTR` record) lookup for the IP address of the sending MTA [29, section 3]. This should result in a domain name associated with the IP address. Further action differs per implementation: some just check if such a domain name exists, some also check if this domain name again resolves to the IP address we started with. The latter is called Forward-Confirmed reverse DNS (FCrDNS) or iprev. Some MTAs also check if the domain name found corresponds to the HELO/EHLO domain name.[5] Some resources claim that this domain name

[4] Email would not be email if there were no exceptions to this: some emails may contain multiple signatures of which just one matches with the `From` domain. Some email servers like Microsoft Exchange may actually strip off the valid signature when automatically forwarding, such that the email gets rejected anyway.

[5] See for example `https://github.com/smtpd/qpsmtpd/blob/master/plugins/helo`.

should also be equal to the domain portion of the `From` header, but this does not make sense, because many MTAs serve multiple companies: take `out62-ams.mf.surf.net`, which not just serves `ru.nl` and `student.ru.nl`, but also domains from many other universities.

The idea behind the reverse DNS check is that attackers, especially with computers in botnets, cannot easily obtain (valid) `PTR` records for all these IP addresses. Note that, contrary to forward DNS, a `PTR` record has to be created by the Internet Service Provider (ISP). Some MTAs also check if the obtained domain name does not look like a 'generic' domain name that an ISP would assign to residential IP addresses, such as `ip54570101.direct-adsl.nl`. However, the effectiveness of all these checks is questionable. Note that there is no official standard advising the use of this check and that not all MTAs comply with the conditions.

An example `PTR` record for `145.0.1.62` is the following:

```
62.1.0.145.in-addr.arpa. 84231 IN PTR out62-ams.mf.surf.net.
```

### 2.1.2 Authenticating the recipient and providing confidentiality

On top of these protocols, we also need Transport Layer Security (TLS, previously SSL) [30] to encrypt the SMTP connection to keep the content confidential and authenticate the recipient's MTA so that an active attacker cannot pretend to be the recipient.

## 2.2 Problems

We already touched upon some problems with the protocols: SPF and DKIM do not work well without DMARC and SPF under DMARC does not support automatic forwarding. But as you might expect, there are more problems with the current architecture.

### 2.2.1 SPF

To start off, we again consider SPF. We have seen that it checks the IP address of the sender against a list of allowed MTA IP addresses for the sender domain. The problem with this, is that it can enable IP spoofing, where an attacker uses the IP address of a legitimate MTA in the communication with the recipient MTA. The risk of this is limited though, because to receive the network packets required to set up the connection back from the target MTA, the attacker would need to be in control of the target network. Still, this is a problem that should be avoided.

There are two other things that might be considered as disadvantages of SPF. First, with SPF the inbound MTA has to be trusted to correctly perform the check, as we cannot do this without the IP address of the sender. Second, companies might find it difficult to compose a full list of IP addresses to use for the SPF policy record.

### 2.2.2 DKIM & DMARC

Now we move on to DKIM, using which providers can sign messages. Signing a message may seem simple, but there are some important problems that arise due to the way that MTAs

handle messages. The first problem is that MTAs are allowed to add and reorder some headers [21, section 3.6]. For example, most mail agents add a `Received` header to the message, indicating that the agent received the message from some other agent. The DKIM header specifies which headers in what order should be signed, so this would not be a problem, were it not the case that a header with the same name can occur multiple times [27, section 3.7, 8.12]. The second problem is that message contents may be transformed by an MTA into an equivalent other form. Most notably, messages containing something other than 7-bit ASCII may be transformed into 7-bit ASCII using Base64 encoding if the receiving MTA does not support messages using 8 bits per byte. Thus, DKIM advises implementations to only use 7-bit ASCII to try to prevent such transformations [27, section 5.3], which can slow down transmission of messages containing large binary files. Besides that, there may be changes in whitespace in the body, or whitespace and case in the headers of the message.

To try to solve part of these problems, DKIM has two canonicalization algorithms. The "simple" algorithm signs the specified headers and body as-is, while the "relaxed" algorithm allows some modifications in case and whitespace [27, section 3.4]. However, this does not solve the header reorder problem mentioned above, and it may not cover all other modifications. Moreover, with alteration of whitespace allowed by the "relaxed" algorithm, an attacker might change the conveyed message of the email [27, section 8.1] or may slightly change the content of an attachment.

Another problem with DKIM, which SPF does not have, is that it enables certain kinds of replay of signed messages: an attacker can just resend a signed message to the original recipient or multiple other recipients [31]. This can be partially mitigated by adding an expiration time to the signature, although the standard tells us that "The *[expiration time]* tag is not intended as an anti-replay defense" [27, p. 24]. In most cases, a replayed email will be recognizable by a user through inspecting the values of the `Date` and `To` headers, but this is not a very robust strategy.

Some also consider it a problem that DKIM (with DMARC) does not support mailing lists that alter the content of an email but leave the `From` header intact [32, section 5.2].[6] DKIM attempts to still allow an addition of a footer by a mailing list by enabling signers to only sign the start of a body upto a certain number of bytes. However, this feature can be badly abused by attackers using the possibilities of HTML to overwrite the entire displayed message [27, section 8.2]. Recently, a protocol called Authenticated Received Chain (ARC) was developed [34]. With this protocol, a party can sign the authentication results (such as SPF, DKIM, and DMARC validation) of the original message before modification to prevent rejection by the recipient, but there is no single good way to decide when to trust such a signature.

Something else which could be considered a problem is that there is no single answer to the question "which headers should I sign?" Ultimately, this is something that the MSA has to decide, but the receiving MDA may ignore a signature if it does not include certain headers [27, section 5.4].

Lastly, domains that do not send email also have to implement DMARC to make sure that attackers cannot send email originating from these domains.

A quick test we performed under 99 email domains tells us that 68 domains have implemented DKIM, also 68 have implemented DMARC, but only 50 have implemented both. See Appendix A for more information.

[6] DMARC makes an earlier protocol referenced in this RFC, called DKIM Author Domain Signing Practices (ADSP) [33], obsolete.

### 2.2.3 DNS

Someone who knows a bit about the DNS may already have noticed that it might be unsafe to rely on the DNS for security-critical protocols such as SPF, DKIM,[7] and DMARC. After all, the DNS is not inherently secure. Hence, to protect against modification of these records by an active attacker on the wire (a 'Man In The Middle', MITM), we also need DNS Security Extensions (DNSSEC) [35, 36, 37]. With DNSSEC, a domain's records are signed with its private key and the corresponding public key is published via a DNS record. The public key is then signed by the parent domain, which publishes the signature as a record. This chain continues up to the root domain '.', which is the trust anchor and has a known key that changes every couple of years. This key is managed by the Internet Assigned Numbers Authority (IANA).[8] DNSSEC also has a way to indicate that certain subdomains are not signed. Hence, a MITM cannot just strip off the DNSSEC records to make the resolver think that the domain does not sign their records.

DNSSEC also has some problems, however. First of all, it is perceived to be complex to implement [38]. Second, there is no specified method for key revocation in case the private key is compromised. Just uploading a new key has the problem that caching DNS servers will take a while to detect the change, and, moreover, this is vulnerable to active MITM attacks. Some solutions have been published [39, 40], but these are not implemented yet. It is also worth noting that DNS stub resolvers that just query a local recursive caching server cannot verify the DNSSEC signature chain and hence have to trust the (connection to the) recursive server, so it is often best to have the software perform the full iterative DNS lookup itself.

### 2.2.4 TLS

As mentioned before, TLS is used in SMTP for confidentiality and authentication of the recipient. TLS in SMTP is different from TLS used on the Web in HTTPS for two important reasons, which are also the two most prominent problems with TLS in SMTP.

First, with HTTPS the URL specifies whether to use TLS or not using the `https:` prefix. In SMTP (at least between MTAs, see Section 2.2.4 for the MUA→MSA case) we do not have such a prefix in the email address; we have opportunistic TLS where the client connects via an unsecured channel and the server then sends a message to the client indicating support for TLS. If the client also supports TLS, it initiates the TLS handshake. This does not protect against an active MITM attacker, because such an attacker could easily strip out the message from the server indicating support for TLS. This is called a downgrade attack.

Second, with HTTPS the URL specifies the domain name of the server to be contacted, which is also the name that must be on the certificate presented by the server. With email, however, using the domain name in the email address is impractical because companies often let other companies provide their email services, and it is not a good practice to provide an external company with a certificate with your domain name on it, including a private key for that certificate. Hence, in email the sender looks at the `MX` (Mail eXchange) DNS record(s) of this domain, which contain the domain name(s) for the MTAs of that domain [15, section 5.1].[9] For example, one of the MTAs for `ru.nl` is `mx1.surfmailfilter.nl`. Classically, there also

[7] The DKIM standard actually states that "DKIM is only intended as a "sufficient" method of proving authenticity. It is not intended to provide strong cryptographic proof about authorship or contents. Other technologies such as OpenPGP [RFC4880] and S/MIME [RFC5751] address those requirements" [27, section 8.5].

[8] The hash of this key can be found at `https://www.iana.org/dnssec/files`.

[9] If no `MX` records exist on the domain, this domain itself is implicitly the domain for the MTA.

was a problem with using these domain names for the certificate, because a MITM can just modify the DNS response with the MX records and insert their own server, so there is only protection against a passive attacker, who cannot modify traffic. In fact, the standard for the TLS extension of SMTP has no rules on how to verify the server's certificate: "The decision of whether or not to believe the authenticity of the other party in a TLS negotiation is a local matter" [30]4.1. Hence, an MTA might as well use a certificate with another name or even a self-signed certificate instead of a certificate signed by a trusted Certificate Authority (CA). See also Foster et al.'s research on provider-based email security [8]. In 2014, Friedl et al. composed a draft RFC to define how identity verification between MTAs should take place [41]. It specifies restrictions for the name on the certificate. However, it was only intended as a recommendation and it has since expired. There are two standards that are meant to solve this: DANE and MTA-STS. We discuss these in the next two sections.

### DANE

The first solution uses DNS-Based Authentication of Named Entities (DANE) TLSA records [42], of which the use in SMTP was defined in 2015 [43]. The authors Dukhovni & Hardaker argue that the use of a list of trusted CAs is not practical, because the list would need to be exhaustive as to not reject legitimate emails, while simultaneously being short to keep the attack surface for CA compromises as small as possible. With DANE, a domain owner can publish TLSA DNS records to impose restrictions for the TLS certificates used by their server for connections on certain transport ports. Additionally, the presence of these records indicate that the server supports TLS, thus preventing a downgrade to cleartext. For example, an email provider could specify that only a certificate with a certain fingerprint can be used for SMTP (on port 25). Note that for DANE to be effective for email, we need DNSSEC on both the domain containing the MX records and the domains of the MTAs that these records refer to, which contain the TLSA records. DANE even makes it possible to securely use self-signed certificates, because these are still authenticated via DNSSEC.

A quick test we performed found that just 6 out of the 99 email domains we tested have TLSA records, see Appendix A.

The DANE records of one of ru.nl's MTAs look like this:

```
_25._tcp.mx1.surfmailfilter.nl. 600 IN TLSA 3 1 1 A70AC85B96943A6A6...
_25._tcp.mx1.surfmailfilter.nl. 600 IN TLSA 2 1 1 F3AE75C0490C907E5...
_25._tcp.mx1.surfmailfilter.nl. 600 IN TLSA 3 1 1 F27635F7DA5CDC9FA...
```

Here, '1 1' means that the data following is a SHA-256 hash of the public key of the specified certificate. '3' means that the certificate for this public key must be the certificate presented by the MTA, while '2' means that the certificate for this public key must be the trust anchor of the certificate presented by the MTA. For both of the certificate usage options above, no validation by a public CA is required. The alternatives to '2' and '3' that also require validation by a CA are '0' and '1' respectively.

### MTA-STS

The second is MTA Strict Transport Security (MTA-STS) (2018) [44]. This mechanism uses a DNS record to indicate that an MTA-STS policy file can be retrieved via HTTPS. Hence, contrary to DANE, this protocol does expect MTAs to maintain a list of trusted CAs. For

example, the MTA-STS policy for `gmail.com` can be found at `https://mta-sts.gmail.com/.well-known/mta-sts.txt`. This policy file contains the domain names of the MTAs that support TLS. These MTAs must present a certificate issued to their domain by a trusted CA. The file also contains a validity period for this policy, just like with HTTP Strict Transport Security (HSTS), which is used on the Web for HTTPS. MTA-STS, unlike DANE, was designed to not mandate DNSSEC, but without it, an active MITM attacker could change the response to the DNS policy record query to signal that MTA-STS is not supported, after which the sender may fall back to using opportunistic TLS. Note that after the first response, the policy is cached until it expires, and a client can regularly make secure requests for the policy file over HTTPS to fetch the new expiration date.

Our quick test found that just 4 out of the 99 email domains we tested have an enforced MTA-STS policy, see Appendix A.

For example, the MTA-STS record of `gmail.com` is just:

```
_mta-sts.gmail.com. 300 IN TXT "v=STSv1; id=20190429T010101;"
```

Its presence indicates that a policy file is available. This policy file has the following contents:

```
version: STSv1
mode: enforce
mx: gmail-smtp-in.l.google.com
mx: *.gmail-smtp-in.l.google.com
max_age: 86400
```

This policy expires after 86400 seconds, or about 24 hours, which will in many cases be too short to effectively protect against MITM attacks, because the policy will already have expired the next time an email to this domain is sent, unless the MTA regularly actively fetches the new expiration date

### Certificate Transparency

The Certificate Transparency (CT) protocol (2013) [45], which is not necessarily related to email, tries to tackle the problem that with such a large list of CAs, there is a very large attack surface and one compromised CA generally affects the whole infrastructure. We already discussed DANE, which also has this as one of its goals, but CT is different. CT tries to prevent misissued certificates, possibly by an attacker who compromised the CA, by providing a standard for publicly logging issued certificates. A signed proof of addition to a log can be added to a TLS certificate. This proof can be verified using the public keys of trusted logs. Hence, the verifier has to maintain a list of logs that it trusts. The Google Chrome browser already requires all certificates issued since May 2018 to be publicly logged.[10] There are multiple logs, run by companies such as Google and Cloudflare, that can be audited by anyone.

### Mail submission

Mail submission, from MUA to MSA, is different from message routing between other mail agents in that message submission can use implicit TLS: on the classic message submission

[10] See `https://github.com/chromium/ct-policy/blob/master/ct_policy.md`.

port (587) TLS is optional [46], which makes it vulnerable to downgrade attacks, but when a MUA connects to port 465, a TLS handshake is initiated immediately, which counters these attacks [47]. However, not all providers support this since this was only specified in 2018, although a user can often enforce TLS usage even over port 587 by checking a box in the MUA's settings.

We have seen that to discover the MTAs of a certain domain, we use its `MX` DNS records. There is a similar method for discovering MSAs for mail submission, but it is not implemented by all email providers and it was specified only in 2011. Namely, we can request the `SRV` (service) DNS records of the domain [48, 49]. For example, this is the `SRV` record for message submission of `gmail.com`:

```
_submission._tcp.gmail.com. 85067 IN SRV 5 0 587 smtp.gmail.com.
```

The numbers '`5 0`' indicate the priority of this MSA relative to other MSAs, the MUA should try to contact the MSAs in the order specified by their priorities (in this case there is just one MSA), '`587`' specifies the port, and `smtp.gmail.com` is the domain name of the MSA.

We also saw that in the case of `MX` records, using plain DNS we cannot properly verify the identity of the server using the domain name in the `MX` record, because a MITM could change this record. We have the same problem when using the `SRV` MSA discovery system. However, for this protocol the domain names that can be on the certificate are clearly defined [50]. Moreover, there is a special `SRVName` subject name type for TLS certificates of servers that can be discovered through `SRV` records [51]. In this way, a certificate can be issued to, for example, `SRVName _submission.gmail.com`, so that DNSSEC does not need to be used, because this domain name can be derived directly from the user's email address. We checked multiple domains that have email submission `SRV` records, but unfortunately none of them presented a certificate with an `SRVName`. It should also be noted that some CAs, such as the free Let's Encrypt CA, do not support this extension. Still, a MUA can check if the domain name on the certificate corresponds with the domain name portion of the user's email address.

### Require TLS

TLS is negotiated per link and not end-to-end, which causes another problem: what do we do if we want to make sure that TLS is used on each link? For this purpose the `REQUIRETLS` SMTP extension was developed [52] (November 2019). For example, if an MSA supports this extension, a MUA can mark an email such that the MSA will only send the email to an MTA that supports TLS, MTA-STS, and the `REQUIRETLS` extension, such that this MTA will do the same, etc. Unfortunately we have yet to come across a server that supports this extension.

The extension also introduces the '`TLS-Required: No`' message header which does exactly the opposite. Clients can add it so that servers that support it will *ignore* all security checks, such as DANE and MTA-STS. This can be useful to deliver an email to a domain with a misconfiguration in these protocols, informing them about this misconfiguration.

## 2.3 PGP & S/MIME

We already briefly mentioned PGP and S/MIME in the previous chapter. Both provide end-to-end security instead of provider-to-provider security, but in slightly different ways. PGP [12] (first specified in 1991) uses public key cryptography, where a user can sign their emails using their private key and possibly encrypt them using the recipient's public key (or more specifically using a symmetric key attached to the message, encrypted with the recipients public key). A problem with PGP is that users have to manage the keys of others that they know to be correct. To ease key management, PGP introduces the concept of the web of trust, where users can sign keys they trust and publish these signed keys. The software can then look at the trusted keys and recursively trust the keys that these users trust, etc. In other words: keys trusted by people you trust are also trusted by you.

S/MIME [13] (first specified in 1998) allows users sign and/or encrypt MIME messages such as emails. It is based on the Cryptographic Message Syntax (CMS) [53]. It can be used with certificates issued by public CAs as well as other certificates. S/MIME agents should have some way to manage trusted certificates and retrieve certificates, but how exactly this should happen is not specified [54]. For the encryption of a message, a symmetric cipher is used, similar to PGP, and the symmetric key is encrypted with the public key of each recipient. The certificate chain of the sender can be attached to the message to help the recipient with verification.

# Chapter 3

# Related Work

In this chapter we list some previous proposals for new email delivery protocols. We divide these into protocols before and after the introduction of SPF, because SPF was the first anti-spoofing protocol and this division provides us with a picture of what the existing architecture at that time looked like.

## 3.1 Before SPF

Already before the standardization of the security protocols mentioned in Chapter 2, there had been a lot of proposals to improve email delivery. Below we give a few examples.

A first example is Bernstein's Internet Mail 2000 (IM2000) [55], which uses pull-based email delivery, in contrast to the push-based email-delivery that SMTP uses. With pull-based email delivery, the sender is responsible for storing the email. Advantages of this type of systems include the difficulty for 'spammers' to send many large emails to someone and the reduction of overall disk space usage when sending a large email to a lot of people. Disadvantages include possibly reduced privacy due to the sender knowing when you opened an email, the fact that emails may become unavailable if the sending server is experiencing outage, and lost emails if the sender's server were to be permanently shut down. Pull-based email-delivery, together with its advantages and disadvantages, is also discussed by Chrobok et al., together with an extension of SMTP providing it [56].

Another attempt at replacing SMTP is made by Mislove et al. in the form of ePOST (2003) [57], which is a decentralized (peer-to-peer) protocol. On the one hand ePOST may have more storage overhead than the traditional email system, because emails are replicated over multiple hosts, but on the other hand it does use a system like IM2000 where references to emails with attachments are reused and only a notification is sent to the recipient, until they request the full email. Because of the replication of emails, outage is less of a problem in this system. Messages are encrypted using their hash as key, which means identical messages yield identical ciphertexts. However, this is intentional and serves to prevent duplicate messages in message stores. When a user sends an email, the recipient gets a notification containing the hash of the encrypted message and the hash of the plaintext, encrypted with their public key and signed by the sender. The hash of the encrypted message serves as a handle to retrieve the message from a message store, while the hash of the plaintext can be used to decrypt the message. The full notification is itself signed by the sender's private key and encrypted with a symmetric session key which is sent along, encrypted using the recipients public key.

ePOST authenticates the public key of the other party via a Public Key Infrastructure (PKI) with certificates bound to email addresses signed by a trusted third party. Hence, it offers end-to-end security. For interoperability with SMTP it uses gateway servers that act to the outside as SMTP servers but to the local POST network as POST servers. This has the advantage that it is easy to communicate with SMTP MTAs, but in this case we do not have the security of POST anymore. However, because it is a peer-to-peer protocol, the paper tells us that "each participant is required to contribute a portion of her desktop's local disk", which is not something that everyone would want to do.

A different protocol is the Secure Email Transport Protocol (SETP) by LeMay and Tan (2004) [58]. SETP provides not only confidentiality, integrity, and end-to-end security via a semi-trusted key server, but also proof of delivery, message lifetime control and shared email addresses. Note that SETP does not aim to replace SMTP, but rather seeks to amend it, as to ease its adoption. With message lifetime control the authors mean that an unread message can be revoked after a certain time or only be made available after some time. This, however, would also be possible using pull-based email architectures. Email addresses can be shared to provide message flow confidentiality, where only the sender and recipient know which parties are really communicating and only the actual recipient can decrypt the message by using their private key. Furthermore, SETP tries to combat spam using a central server keeping track of user ratings of emails.

Yet another attempt is Lux et al.'s WSEmail (2005) [59], which is heavily based on Web technologies such as SOAP and XMLDSIG and is designed as a replacement for SMTP. The goal of WSEmail is to be flexible while still being secure and performant. WSEmail can also be extended to allow for a pull-based model for attachments. Due to the use of XML and due to their extensibility, WSEmail messages are arguably pretty complex, as can be seen on the (archived) website.[1]

## 3.2 After SPF

There are also some more recent proposals, defined after some of the email security protocols from Chapter 2.

An example is Ghafoor et al.'s CryptoNET (2009) [60]. CryptoNET is more than a secure protocol to replace SMTP: it also provides secure storage for emails and address books, end-to-end security, support for robust authentication of users through smart cards, a system for confirmation of delivery, and an authorization policy system to minimize spam mails. Furthermore, it uses a pull-based model for attachments, which are stored with the sender and downloaded separately from the rest of the email by the recipient. This, together with the authorization policy system, minimizes spam emails. As one can see, it provides many new features, but this also makes it difficult to implement.

Two other, more recent, proposals that provide end-to-end security are the Cipher Mail Transport Protocol (CMTP) (2016) [61] and the Dark Mail Transfer Protocol (DMTP) (2018) [62]. Both try to automate the tedious process of managing keys that users of similar protocols like PGP have to go through. CMTP does not provide authentication of the public keys of others, however: a mail server sends its key unprotected to the user, so the infrastructure uses a trust on first use (TOFU) model where the key is assumed to be correct the first time you make contact. It uses the same port as SMTP and uses similar commands, such that a server can

[1] `https://web.archive.org/web/0/http://wsemail.ws/messages.html`.

run one service that supports both protocols. In DMTP, keys are distributed via key servers and verified via DNS with DNSSEC or using the key of a trusted CA. DMTP also has the interesting feature that the outbound MTA cannot see the full address of the recipient and the inbound MTA cannot see the full address of the author, similar to SETP. It can run either in single protocol mode, where it uses a different port from SMTP, or in dual protocol mode, where it uses the same port. Just like CMTP, DMTP's commands are similar to SMTP's.

# Chapter 4

# Requirements & goals

In practice, SMTP is still used almost everywhere. We argue that previous endeavors for a more secure email infrastructure have not been widely adopted mostly due to them being too complex. For example, the prototype for WSEmail (which seems to not be publicly available), is constructed from no less than "68 interfaces and 343 classes organized into 30 projects" [59, section 5.1]. We have to bring back the *Simple* in SMTP.

But there are more problems that need to be dealt with. In this chapter, we list the requirements that we think a new email delivery protocol like SMTPsec should satisfy to succeed. These requirements are partly based on the criteria applied by Moecke et al. [63]. We distinguish between non-functional and functional requirements. In Section 4.1 we list the non-functional requirements, which concern the operation of the protocol. In Section 4.2 we list the functional requirements, which are about the functionality that the protocol should have. We also give the trust model and security goals of SMTPsec in Section 4.3.

## 4.1 Non-functional requirements

N.1 It should be *secure*: emails should remain confidential to eavesdroppers and should be authentic. However, the protocol will explicitly not provide end-to-end security like PGP or S/MIME to benefit simplicity and opacity to the user. More on this in Section 4.3.

N.2 It should be *simple* to implement and deploy. This means that it should not be too technically complex. Basing the protocol on well-established existing protocols such as TLS can benefit this requirement.

N.3 It should, however, be *extensible*, such that if more advanced features are needed, these can be added in later on.

N.4 It should be *low-cost*, so not require a significant amount of additional resources compared to the current infrastructure. This includes servers and other things that require money.

N.5 It should not be noticeably slower than SMTP, such that speed does not impede adoption.

N.6 It should be *opaque* to the user whenever possible. That is, they should barely notice the change in protocol and they should not have to change their workflow. In other words, the actions required to send and read email should remain the same as they were

without SMTPsec. In practice, this means that it should work with existing MUAs and the user should not have to configure anything new.

N.7 It should be *interoperable* with the existing infrastructure. Hence, users should be able to exchange email with others who do not use the new protocol yet, but instead just use SMTP. In practice, this means falling back to SMTP when the recipient does not support SMTP. This is unfortunate, but without this capability, gradual deployment becomes impossible, which means everyone would need to start using the new system at the same time, which is infeasible.

N.8 Users should already receive *immediate benefits* with low adoption rates. This is another requirement for gradual deployment. A system will not be implemented if it will only have benefits if nearly everyone has implemented it.

N.9 Lastly, the protocol description should be *concrete*, so not just an abstract mathematical description: it should specify what bytes actually are transmitted over the network, such that it can be implemented directly.

## 4.2 Functional requirements

F.1 The opacity requirement implies that the new protocol should have the *same functionality* as SMTP. However, we exclude functionality that would degrade security.

F.2 Another important feature is also an exception to the opacity requirement: the recipient of an email should receive an *indication* about the security of this certain email. Such a security indicator could take the form of an icon such as the lock symbol for HTTPS connections in most Web browsers. If it was sent using the new secure protocol, it should be clearly indicated that, for example, the message origin was reliably authenticated. This way, the user can take it into account when judging the authenticity of an email. For this research, however, only the facilities necessary for such an indicator system will be implemented.

F.3 Similarly, on the other end the sender should be able to *constrain* the use of SMTP in favor of the new protocol, if, for whatever reason, they find SMTP to be not secure enough.

F.4 A recurring feature present in modern protocols such as DMARC and MTA-STS is a testing mode including reporting, where the policy is not (fully) in force yet but reports of failures are sent to a system administrator, as to verify the correctness of the configuration. We think such a *failure reporting* system would also benefit deployment of a new delivery protocol.

## 4.3 Security

### 4.3.1 Trust model

SMTPsec focuses on provider-to-provider security, rather than end-to-end security. This means we have to trust the email provider of the sender and of the recipient (MTAs, MSAs, MDAs, etc.), similar to what is done in the current mail architecture, if we do not count PGP and S/MIME. Hence, the providers are trusted to keep their private keys secure and the emails of

their users confidential to third parties. Unfortunately, in reality it may be that some providers disclose emails to intelligence agencies. If one wants to make sure that emails cannot be read by a provider, one should use a system offering end-to-end security like PGP or S/MIME, with a key pair that they generated themselves, of which they keep the private key secret, and with properly authenticated public keys of their correspondents.

Besides the two providers, a list of CAs will be semi-trusted. That is, each certificate that is issued also has to be publicly logged using the Certificate Transparency (CT) protocol. A number of logs are also semi-trusted, as anyone can audit them and the certificates still have to be issued by a trusted CA.

Additionally, the DNSSEC infrastructure will be trusted. More specifically, this means that parent domains of domains on which we require DNSSEC are trusted to sign only the right public keys.

### 4.3.2 Attacker model

We assume an active MITM attacker, so a network attacker than can read and modify any traffic on any link. The attacker can only break cryptography that is considered to be broken as of today. As we design our protocol with strong cryptographic primitives, we focus on attacks on the construction of the protocol.

### 4.3.3 Security goals

The security goals of SMTPsec are:

- The recipient's MTA should be able to verify the authenticity of the message origin, or in other words, that the message was indeed originally sent by an MTA that is authoritative over the domain in the domain portion of the author's address. Checking the local-part portion of the address is the responsibility of the MSA and is out of scope for this protocol.
- The sender's MTA should be able to verify the authority of the recipient's MTA over the domain in the domain portion of the recipient's address. Routing the email to the correct user's message store based on the local-part portion of the address is the responsibility of the MDA and is out of scope for this protocol.
- Providing integrity of the message from the sending MTA up to the MTA of the final recipient, also if the message is automatically forwarded.
- Ensuring confidentiality of the message to attackers, so only the recipient and providers can read the message.

Ideally we would also want strong replay protection. However, this may be hard to achieve because we want support for automatic forwarding, but we will investigate this possibility further.

# Chapter 5

# Design

In this chapter, we present various ways to solve each problem that we face. Then, in Section 5.8, we choose the best combination of solutions, taking into account the requirements from Chapter 4.

## 5.1 Locations of changes

We have seen in Chapter 2 that SMTP is used on multiple links: MUA→MSA, MSA→MTA, MTA→MTA, MTA→MDA, and MDA→MS (see Figure 2.1). However, not all these links benefit equally from a new delivery protocol.

We already discussed message submission (MUA→MSA) in Section 2.2.4, including why the problems of the MTA→MTA case do not really apply to it. So for this case, a new protocol is not really necessary. Moreover, we want to avoid the MUA having to support the new protocol because of the *opacity* requirement.

For the MSA→MTA, MTA→MDA, and MDA→MS cases there is also no real security benefit, as an email provider can configure their services to all use TLS (or when email management is outsourced, instruct the external company to do this).

For the remaining **MTA→MTA** case, however, our protocol would of course be very useful, because providers cannot always know if the other party supports TLS, and, as we have seen in Chapter 2, currently authentication and integrity verification is a mess, which can be solved by a new delivery protocol designed with security in mind.

We also mentioned that the MSA→MTA and MTA→MDA connections might not be direct, but rather go via more MTAs. However, these other MTA→MTA cases are thus similar to the MSA→MTA and MTA→MDA connections. In the design of SMTPsec, we assume that these connections are direct.

One might also wonder if merging the MSA/MDA/MS with the MTA has any benefits. If we see this as the other agents being merged into the MTA then a problem with this is that if management of the MTA is outsourced, then this external company may also have to be trusted with the user's credentials, as already discussed at the start of Chapter 2. However, if we see it as the MTA being merged into the other agents, we could also say that a company can run the 'MTA software' on their server just like the MSA software.

## 5.2 Identifying & authenticating the recipient's server

With SMTP, MTAs are discovered by using MX records, which, as we discussed in Section 2.2.4, has various authentication problems. In this section we look at the possible alternatives for SMTPsec.

### 5.2.1 Using MX records with DNSSEC

One possibility for our protocol would be to keep using MX records, but enforce the use of DNSSEC and check that the name on the server's certificate corresponds to the name from the MX record. The MTA could then indicate in the reply to the EHLO command that it supports SMTPsec, after which the sender can signal it wants to switch to SMTPsec. However, not all domain name registrars provide support for DNSSEC, so this would mean that domain owners using such a registrar would not be able to use the protocol, but of course we could hope that an increasing number of registrars will support DNSSEC.

An advantage of this is that if it turns out that a server does not support SMTPsec, we do not have to make an extra DNS query to find the SMTP MTAs. A disadvantage is that for some providers maybe only part of the listed MTAs support the new protocol, so always all MTAs would need to be tried to decide if one supports it. Moreover, an active MITM attacker could strip support for SMTPsec from the MTA's reply so that plain SMTP is used.

### 5.2.2 Using TXT records or a new type of record with DNSSEC

Another possibility is to do the same except use a new type of DNS records, or use plain TXT records. An advantage of this is that (with DNSSEC) a fallback to SMTP due to a MITM attacker is prevented. The downside of using a new record type is that it would first need to be registered and then registrars and DNS clients need to implement support for it, which takes time and might not happen if the significance of SMTPsec is not recognized. TXT records do not have these problems but are not really intended to be used this way, plus if many new protocols start using them, a DNS reply to a query for TXT records might get unnecessarily large.

### 5.2.3 Using SRV records with DNSSEC or SRVName

We could also use SRV records to list the MTAs. In this case we can either use DNSSEC to make sure the listed name is correct, or use a certificate with an SRVName. A disadvantage of using SRVName is that many CAs, especially free ones, do not support it. Additionally, without DNSSEC a MITM attacker could remove the SRV records from the reply to make it seem like no MTA supports SMTPsec, causing a fallback to SMTP.

### 5.2.4 Authentication using certificate issued to domain of email address

Instead of using DNSSEC or SRVName for authentication of the recipient's MTA with one of the methods above, we could also require the name on the certificate to correspond with the domain portion of the email address of the recipient. However, if we do not use DNSSEC,

the protocol is again vulnerable to downgrade attacks to SMTP by a MITM. Additionally, as we discussed in Section 2.2.4, this makes it insecure to outsource the handling of email for a company, as this external company would need a certificate that is valid for the email domain, which it could also use to, for example, set up a website on that domain. Hence, we would need some sort of domain separation, limiting what the certificate can be used for. Now this might be accomplished by setting the extended key usage extension in the certificate to `id-kp-emailProtection`, but one could argue that network connections are not what this purpose value was meant for and instead it should be used for signing emails like in S/MIME. Besides that, not all CAs will issue certificates with this key purpose.

### 5.2.5 Using HTTPS

Another way to discover MTAs could be by making an HTTPS request to a predetermined URL on a (sub-domain of) the domain in the domain portion of the email address, much like MTA-STS. The advantages of this solution are that no special certificates are needed and neither is DNSSEC. However, if we do not also use some DNS record protected by DNSSEC to indicate that SMTPsec is supported, a MITM could just block the HTTPS request to make the sending MTA think the protocol is not supported, causing a downgrade to SMTP. On top of that, this solution requires extra work for a company to set up, also when outsourcing the rest. Besides that, it can make the MTA more complicated as it also needs to support HTTPS, which contradicts the *simplicity* requirement.

## 5.3 Authenticating message origin & verifying message integrity

SMTP uses SPF, DKIM, and DMARC to authenticate the message origin and verify the integrity. For SMTPsec we want to have a simple but secure alternative.

### 5.3.1 Using TLS client authentication

The first thing that comes to mind is to just use TLS client authentication for the outbound MTA. This procedure is not too complicated in general, but here we again have the problem that it is not clear what domain name should be on the certificate. We could solve that in the same way that we use for discovering the recipient's MTA.

However, we also have the problem that automatically forwarding email becomes tricky, because the forwarding domain is not allowed to send mail originating from another domain. It might be possible to instruct the original outbound MTA to redirect the message to another MTA, but it might be the case that the connection has already been closed before it is decided to forward the email. Additionally, it would mean that the original MTA gets to know the addresses that the email is forwarded to. Note that this also applies to all subscribers of a mailing list. On top of that, this method may make room for denial of service attacks if the MTA is instructed to forward the email to a lot of addresses, although a limit could be set on this number. Of course, it would be possible to drop the support for this kind of automatic forwarding altogether, but this is not in accordance with the *same functionality* requirement.

### 5.3.2 Using a signature

An alternative would be to add a signature to the email like with DKIM. This signature could either be included as a message header like with DKIM, or just in the envelope. The advantage of including the signature as a message header is that the MUA could also verify the signature, although currently this is not often done anyway. Furthermore, if we put the signature in the envelope, we could still add it as a message header before the message is delivered to the MUA, although it may be tricky to keep a signature of the whole message intact when adding this header and possibly other trace headers to the message. The advantage of putting the signature in the envelope is that the size of an email would be less when downloaded by a MUA. However, a provider must make sure to not discard the signature if it wants to forward the email.

Note that, like DKIM, this method does not protect against replay attacks. However, the `To` and `Date` headers are signed, so most attacks should be manually recognizable by inspecting the date and recipient.

When using signatures we also need a way to distribute and authenticate public keys. For this we could either use public CAs or a DNS record like DKIM does. We investigate the two possibilities below.

#### Authenticating public key using public CAs

If we choose to use public CAs, we also need to make sure that the inbound MTA has access to a certificate chain, so that it can verify the signature. This certificate chain could be attached to the signature, or could be accessible via DNS. The advantage of the former is that it saves us an extra DNS query, but if we want the MUA to also be able to verify the signature, then delivering it via DNS saves a lot of space when downloading emails. Note that we do not need DNSSEC for this because a certificate chain is inherently signed. For the leaf certificate this time the `id-kp-emailProtection` extended key usage would be suitable, but as mentioned above, not all CAs support this, so if we do not use this then we again have the problem of domain separation and choosing what domain name should be on the certificate, like we would have when using TLS client authentication.

#### Authenticating public key using DNS records with DNSSEC

If we choose to distribute public keys via DNS, then we do need DNSSEC and we would need to make extra queries, but we do not need to store a large certificate chain. However, we can actually prevent the extra queries by attaching the DNSSEC chain including key to the signature, a kind of practice which is often called 'stapling'. This is very similar to attaching the certificate chain when using CAs.

#### Canonicalization

With DKIM over SMTP, we have to apply a canonicalization mechanism to the email before the signature is computed because mail agents may transform parts of the message into equivalent other forms or add certain headers. However, this is fragile and can be insecure, as discussed in Section 2.2.2. It would be much nicer if we could just make sure that the message is not

changed at all. However, this may not be feasible if an MDA that does not support SMTPsec decides to forward an email. If we do choose to enforce that also the MDA supports the new protocol (or merge the agents) then we can prohibit changes to the message. We could still keep track of message headers like `Received` in the envelope for troubleshooting, and we can re-add these before delivering the message to the MUA, so that a tech-savvy the user could inspect them if they need to.

## 5.4 Providing confidentiality

In SMTP we use TLS on a per-link basis to provide confidentiality of the message against MITM attackers. In keeping our protocol as simple as possible, we think it is reasonable to use the same method.

## 5.5 Certificates

There are some additional restrictions that we want to impose on the certificates used with SMTPsec. If we use public CAs then we can also enforce the use of the Certificate Transparency protocol as mentioned in Section 2.2.4 to reject misissued certificates. On top of that, we can also use DANE `TLSA` records as mentioned in Section 2.2.4 to put additional restrictions on the certificates used and even allow self-signed certificates because these are authenticated via DNSSEC. However, fetching `TLSA` records does require additional queries. Note that there was a proposal to enable stapling of the DNSSEC chain including `TLSA` record onto the server's reply in the TLS handshake, but this draft has since expired [64].

## 5.6 Transaction syntax

SMTP uses an interactive text-based method for transactions, where the client can send various commands and the server replies to each command with a status code and a message. Because the server replies to each command, SMTP is sometimes described as 'chatty'. (The Command Pipelining extension [65] addresses this by allowing multiple commands inside a single network packet.)

A text-based approach also makes the parser more complicated. For example, messages not using the Chunking SMTP extension [66] are terminated by a dot ('`.`') on a single line, which means that for each line in the message, the recipient has to check if it is a single dot and if so, it knows that the message has ended. This slows down processing of large messages. Additionally, if the message itself contains an actual dot on a single line, or even a line starting with a dot, this dot has to be escaped by the sender (by another dot) and the recipient then has to remove all leading dots [15, section 4.5.2]. Not properly escaping data is a typical source of error when using many text-based protocols, causing vulnerabilities in the code that uses them (for example: SQL injections, cross-site scripting attacks, etc.).

For SMTPsec we can choose to use binary messages instead of human-readable text. The advantage of this is that it is easier to parse, does not require escaping data, and uses less space, all of which make the protocol faster, which is in accordance with the *speed* requirement. To further speed up delivery, we want to use as few roundtrips as possible, by combining

information in as few packages as possible. Furthermore, there should be a way to add new commands while retaining compatibility with agents that do not support these commands, following the *extensibility* requirement.

Now for the email message syntax we could either use IMF (and MIME) or something else. If we do not use this, we might be able to make the message more compact, possibly speeding up delivery. However, the message would need to be transformed back to IMF before access by a MUA. We could also still use IMF but transform the message to an equivalent but more compact variant when possible. For example, we could transform Base64 attachments to a binary variant and concatenate long lines that were split over multiple lines (and remove any line length limit).

We also want to provide the recipient with information about the security of the email delivery, in accordance with the *security indication* requirement. We could use a message header for this that indicates that SMTPsec was used, which can just be ignored by MUAs that do not support it. A company that wants to just pretend to use the new secure protocol, might add this header while it only supports SMTP, so an agent of the recipient should remove it if the new protocol was not actually used. This can still fool MUAs connecting to a provider that does not support the new protocol.

Lastly, according to the *constraint* requirement, there should be a way for the user to indicate that only SMTPsec may be used for the delivery of an email. To accomplish this, we could add a bit to the email command, set by the outbound MTA, indicating that the email may not be automatically forwarded over SMTP. But we also need a way in which the MUA can indicate it wants this bit to be set by the MTA. The MUA still uses SMTP, so we also need to define an SMTP extension for this. Related to this, the `REQUIRETLS` SMTP extension should also be integrated with SMTPsec, such that email marked with this flag is not forwarded over unsecured SMTP.

## 5.7 Interoperability

We need SMTPsec to be interoperable with SMTP so that emails can still be delivered to and from users whose providers do not support the new protocol, following the *interoperability* requirement. Unfortunately this means that for now protocols like DKIM and MTA-STS also still have to be implemented.

Often the MSA will sign an email using DKIM, but if the other party supports SMTPsec then this would not be necessary, so one might want to leave out the signature in this case to save space. The problem with this is, however, that if an email is automatically forwarded to a third person, and this person's provider does not support SMTPsec, then we still need the DKIM signature, which we now do not have. Hence, it seems like the best we can do is keep including the DKIM signature until more servers support the new protocol. However, this might mean that when forwarded to an SMTP server, the signature might be rendered invalid if a server decides to transform a binary attachment to Base64, for example. This problem has no clear solution, except maybe more canonicalization, and is an unfortunate example of why the infrastructure is so difficult to change.

## 5.8 Decisions

We have seen that there are many options for each aspect. Choosing the best alternative for each aspect was not easy. In this section we will present our decisions for SMTPsec.

We start off with MTA discovery, for which we will use SRV DNS records because they were made for purposes like this one. These will need to be protected with DNSSEC to prevent downgrade attacks and forged names. For authentication of the recipient MTA we will use TLS where the subject name on the certificate must be equal to the name in the SRV record or an SRVName corresponding to the service. Additionally, the Certificate Transparency protocol and optionally DANE will be used to keep the effects of a CA compromise to a minimum.

To provide message origin authentication, the sending server will add a signature in the envelope. This signature is created using the private key corresponding to a certificate of the server with the same requirements as for the certificate of the recipient's server. This means that the recipient's server will need to query the sender's SRV records to verify the domain name on the certificate, even if an SRVName is used, because we also need the port to check for DANE TLSA records. Contrary to DKIM, the signature will be calculated over the full email, so no modifications are tolerated. This also means that, for the signature mechanism to work reliably, the MSA and outbound MTA, and the MDA and inbound MTA need to be logically merged. Furthermore, of course no headers can be added, so any added trace headers will be kept track of in the envelope.

SMTPsec will use binary commands to simplify parsing and prevent escaping of data, thus increasing the speed. Emails will still be in IMF but the outbound server may compact the message (before signing) to further speed up the transaction.

For interoperability with SMTP (and hence DKIM), we could unfortunately not think of a better way then to also include a DKIM signature header.

# Chapter 6

# Specification

In this chapter we give a concrete and detailed specification of SMTPsec.

## 6.1 Server discovery

Suppose that `alice@a.com` wants to send a message to `bob@b.com`. Alice then sends the email with her MUA using SMTP to the `a.com` mail server, which may transform the message to a more compact form (e.g. transforming Base64 attachments to binary). The `a.com` mail server shall then make a DNS query for SRV records on `_smtpsec._tcp.b.com` and verify the DNSSEC chain. If the zone can be verified to be unsigned or if the zone is signed but no SRV record exists, it reverts back to SMTP. If any other DNS error or DNSSEC verification error occurs, the transaction shall be aborted. If one or more signed SRV records exist, then these are ordered as specified in [48] and we start by contacting the first host on the specified port. If this host is offline or an error occurs, we try the next one, etc. If no host can deliver the email, the transaction is aborted.

## 6.2 Handshake

Let the endpoint from the current SRV record be denoted as $B$ with domain $D$ and port $P$. When the connection to $B$ is opened, immediately a TLS handshake is initiated. The `b.com` server will present its certificate, which must be valid for either $D$ or for SRVName `_smtpsec.b.com` and be signed by one or more trusted Certificate Transparency logs. If TLSA records are present on domain $\_P.\_tcp.D$, then the certificate should comply with the restrictions imposed by these records as outlined in [42]. If the certificate presented by the server is the same as an end-entity certificate specified by a TLSA record, the Certificate Transparency check does not apply. If an error occurs then we move over to the next SRV record.

## 6.3 Transaction syntax

In this section we describe the transaction syntax using the same C-like syntax as is used in [67, section 3]. We use this syntax because it provides us with an easy way to define binary

structures that are transferred over the network. A couple of features stand out. The `uint`*xx* type contains an unsigned integer and consumes *xx* bits. Variable-length vector fields are denoted by angled brackets (`<...>`) after the field name. Written within the angled brackets are the minimum and maximum size of the vector, counted in object instances. A variable-length vector implicitly includes a size field, consuming the minimum number of bytes necessary to hold the maximum size of the vector. The value of an enumeration (`enum`) field is an integer consuming the minimum number of bytes to hold its largest value. All data structures defined below can also be found in Appendix B.

## 6.4 Transaction process

```
enum {
    error(0),
    server_hello(1),
    client_hello(2),
    email(3),
    email_delivered(4)
} MessageType;

struct {
    uint64 size;
    MessageType type;
    Extension extensions<0..2^16-1>;
    select (Message.type) {
        case error: Error;
        case server_hello: ServerHello;
        case client_hello: ClientHello;
        case email: Email;
        case email_delivered: EmailDelivered;
    }
} Message;
```

All commands in SMTPsec are wrapped inside `Message` objects. The `size` field of the message contains the total length of the message in bytes and must be equal to the sum of the size of the individual parts. Large messages may of course be split over multiple TCP packets, while on the other hand multiple messages may also be combined into a single TCP packet.

### 6.4.1 Initialization

```
struct {
    uint8 version = 0;
} ServerHello;

opaque Certificate<0..2^24-1>;
Certificate CertificateChain<0..2^16-1>;

struct {
    uint8 version = 0;
    CertificateChain certificate_chains<0..2^24-1>;
} ClientHello;
```

After the TLS handshake, `b.com` sends a `ServerHello` `Message` in which it communicates the maximum protocol version it supports. Currently, the only possible value is 0. `a.com` replies with a `ClientHello` `Message` with the highest protocol version that both parties support. The `ClientHello` also contains a list of certificate chains that are used for signatures on the emails `a.com` is about to send, such that these chains may be reused for multiple emails. Either party may reply with an `insufficient_security` `Error` `Message` if the negotiated protocol version is too low. Errors are discussed in more detail in Section 6.4.7.

### 6.4.2 Email submission

```
opaque Address<0..2^16-1>;  /* email address of the form local@domain */

enum {
    /* ECDSA with curves as defined in [68] */
    ecdsa_secp256r1_sha256(0x0403), /* with SHA-256 hash */
    ecdsa_secp384r1_sha384(0x0503), /* with SHA-384 hash */
    ecdsa_secp521r1_sha512(0x0603), /* with SHA-512 hash */

    /* PureEdDSA with curves as defined in [69] */
    ed25519(0x0807),
    ed448(0x0808),
    (0xFFFF)
} SignatureAlgorithm;

struct {
    SignatureAlgorithm algorithm;
    opaque parameters<0..2^16-1>;
} SignatureMethod;

struct {
    opaque name<0..2^16-1>;
    opaque value<0..2^24-1>;
} Header;

struct {
    uint24 certificate_chain;
    SignatureMethod signature_method;
    opaque signature[signature_length];

    Header trace_headers<0..2^24-1>;
    Address recipients<0..2^24-1>;
    opaque message<0..2^64-1>;
} Email;
```

If the initialization is successful, `a.com` can send a number of `Email` `Messages`. Each `Email` contains an IMF message, a list of recipients, a number of trace headers, and a signature. The trace headers field can contain headers like `Received` that can be used for troubleshooting and keeping track of when an email is forwarded, etc. For the signature, there is also a field with the zero-based index of the certificate chain from the `ClientHello` that was used and the signature algorithm. The size of the signature depends on the algorithm. The signature is computed over the `message` field. The trace headers field is not signed because it should be

editable and should not contain important information. The signature algorithms listed above are chosen based on [67, section 4.2.3] and have the same identifiers. We chose PureEdDSA instead of HashEdDSA because it is more resistant to collisions in the hash function used internally [69, section 4] and an MTA uses a store-and-forward approach (enabling it to verify the signature before sending it on) so we do not mind that the algorithm requires two passes over the input. For the currently listed algorithms no additional parameters are required.

### 6.4.3 Email receival

When receiving an `Email Message`, `b.com` verifies the signature with the indicated certificate chain. To validate the certificate, `b.com` queries the `SRV` records of the domain $F$ of the `From` header in the `message`. This check fails if there are multiple addresses listed in this header. It then checks if there is a DNSSEC-protected `SRV` record with a port $P$ and a domain name $D$ to which the certificate is issued and if this certificate complies with any `TLSA` records present on domain $\_P.\_tcp.D$. The certificate may also be issued to `SRVName` $\_smtpsec.F$, in which case we might need to check all domains from the `SRV` records to see if one has no `TLSA` records or `TLSA` records that allow the certificate to be used. In any case, if the certificate presented by the server is not identical to one specified by a `TLSA` record, the certificate has to be signed by a trusted Certificate Transparency log. In both cases, if there is no such `SRV` record, the check fails, and `b.com` replies with an `authentication_failed Error Message` with a reference to the email as explained in Section 6.4.7. Authentication also fails if the certificate was not signed by a trusted Certificate Transparency log, and of course if the signature is incorrect.

### 6.4.4 Email delivery

```
struct {
    uint24 email;
} EmailDelivered;
```

If this was successful, `b.com` tries to deliver the message to the listed recipients over a secure channel and sends back an `EmailDelivered Message` on success. If one or more recipients are not recognized, the email is delivered to the remaining recipients, after which an `unknown_address Error Message` is sent, with a list of the unknown recipients and a reference to the email. In both cases, `a.com` may continue sending more `Email Messages`. After `a.com` is done sending emails and received replies from `b.com`, it closes the connection.

### 6.4.5 Forwarding

If `b.com` decides to forward the message, the signature and trace headers should be retained. If the email is forwarded to a server that only supports SMTP, the signature will be stripped. Furthermore, in this case the headers from the `trace_headers` field are appended on top of the existing IMF headers of the message. Providers should enforce a whitelist for allowed trace headers, which will include headers like `Received` and nonstandard headers starting with `X-`.

### 6.4.6 Extensions

```
enum {
    require_security(0)
    /* more extensions may be defined in the future */
} ExtensionType;

struct {
    ExtensionType type;
    opaque data<0..2^32-1>;
} Extension;
```

The `extensions` field can be used in the future for non-breaking protocol extensions. It can also be seen as a place for optional fields for messages. Unrecognized extensions must be ignored by the other party. One extension, `require_security`, is already defined. Like the SMTP `REQUIRETLS` extension, it indicates that an email may only be sent over a secure channel. The extension can be added to an email message and has one byte of data, which has a value of either 0 or 1. If present, that email may only be sent via SMTPsec, or, if the byte is 0, also via SMTP with the `REQUIRETLS` extension given that the conditions from [52] are satisfied. These restrictions also hold in particular if the email is automatically forwarded.

### 6.4.7 Errors

```
enum {
    internal_error(0),
    service_unavailable(1),
    illegal_message(2),
    illegal_parameter(3),
    too_large(4),
    authentication_failed(5),
    insufficient_security(6),
    unknown_address(7)
} ErrorCode;

struct {
    none(0),
    message(1)
} ErrorReference;

struct {
    ErrorCode code;
    ErrorReference reference;
    select (Error.reference) {
        case none: struct {};
        case message: uint24 messageNumber;
    }
    select (Error.code) {
        case unknown_address: Address<0..2^16-1>;
        default: struct {}; /* for other error codes */
    };
} Error;
```

We already mentioned some errors that can occur, but we define more possible errors.

- The `internal_error` error can be sent at any time by either party to indicate that an internal error occurred, optionally accompanied by a message reference. The connection will be closed.
- The `service_unavailable` error can be sent by `b.com` at any time to indicate that this server is temporarily not operational. The connection will be closed.
- The `illegal_message` and `illegal_parameter` errors can be sent by either party to indicate that respectively an unexpected message or an invalid field value was received and includes a reference to that violating message. The offending message is ignored.
- The `too_large` error can be sent by either party if some message is too large for the other party to handle and includes a reference to that message. The connection will be closed if the offending message is not an `Email`. If it is an `Email`, the connection may also be closed if `b.com` so desires.
- The `authentication_failed` error can be sent by `b.com` if the authentication on an email message failed and includes a reference to that message.
- The `insufficient_security` error can be sent by any party in response to a hello message if the protocol version is not deemed to be sufficiently secure, and by `b.com` if the email signature algorithm or parameters are not secure enough. In the former case the connection will be closed.
- The `unknown_address` error can be sent by `b.com` if one or more recipients of an `Email` do not exist and contains a list of unrecognized addresses in addition to a reference to the email message.

The error reference message number is a zero-based index of the referenced message, where only `Messages` from the other party are counted. This includes non-`Email` messages.

## 6.5 Informing the MUA

Before the email is requested by the MUA, the `trace_headers` are added to the IMF headers like when forwarding an email as discussed above.

To inform the MUA that SMTPsec was used, we use the `SMTPsec-Used` header. Before sending an email via SMTPsec that is received from a MUA, the header will be added, with an empty value. Upon sending or receiving an email to or from an MTA via SMTP, the header must be removed.

## 6.6 Mail submission

Because SMTPsec is not meant for submission by a MUA, this agent cannot directly use the `require_security` extension. Hence, we define the 'Require SMTPsec' SMTP extension. If the MSA advertises support for this extension, the MUA may add the `REQUIRESMTPSEC` parameter to the `MAIL FROM` command with no value to indicate that SMTPsec must be used

for further transmission and that the `require_security` extension must be included with a value of 1.

Additionally, if a MUA sets the `REQUIRETLS` flag on a message, the MTA must use SMTPsec to deliver this message with the `require_security` extension with value 0, or SMTP with the `REQUIRETLS` flag set, subject to the requirements set out in [52].

# Chapter 7

# Discussion

In this chapter we discuss the benefits and flaws of SMTPsec, following the requirements from Chapter 4.

## 7.1 Evaluation of non-functional requirements

N.1 *security*: The security is guarded well. The provider of the author is authenticated and the integrity of the message ensured between the two providers using a signature on the message. The provider of the recipient is authenticated using TLS. Certificates of both parties are either signed by trusted public CAs and by a trusted Certificate Transparency log, or are authenticated using TLSA records specifying end-entity certificates. The CA can also be restricted by using TLSA records. Confidentiality is ensured on each link. All of this follows the security goals from Section 4.3. Additionally, there is no downgrade possible to SMTP by an active MITM attacker, due to the use of DNSSEC. However, just like with DKIM in SMTP, there is no strong protection against replay attacks.

N.2 *simplicity*: We tried to keep SMTPsec itself as simple as possible while preserving security, but unfortunately compatibility with SMTP requires extra effort. This will probably be an obstacle in the adoption of any new email delivery protocol.

N.3 *extensibility*: We added an extensions field to protocol messages that can be used for new features later on without breaking the protocol when communicating with hosts that do not support these features.

N.4 *cost*: SMTPsec should require about as much computing power as SMTP with the security protocols mentioned in Chapter 2, maybe even less because messages can be more easily parsed because the protocol is not text-based. Additionally, no separate HTTPS connection is necessary like with MTA-STS.

N.5 *speed*: A reasonable implementation of SMTPsec is anticipated to be at least as fast as SMTP with security protocols. This is not only due to the simplified parser, but also because signatures can be added efficiently to large binary messages without the need for a conversion to Base64 and because signature verification does not require canonicalization as the email itself cannot be modified.

N.6 *opacity*: SMTPsec does not require any extra actions from the user, as the connection to the MSA will still use SMTP and they do not need to store cryptographic keys or anything, because the protocol authentication acts only between providers.

N.7 *interoperability*: We specified how the sending host can handle recipients that only support SMTP, but unfortunately security-wise this costs extra effort, because the existing security protocols have to be implemented as well.

N.8 *immediate benefits*: The benefits of SMTPsec over SMTP are listed in Section 7.3.

N.9 *concreteness*: We gave a concrete description of SMTPsec in the previous chapter, which should be detailed enough such that it can be implemented.

## 7.2 Evaluation of functional requirements

F.1 *same functionality*: SMTPsec still fulfills the main purpose of SMTP, which is sending emails from person (or company) to person. However, mailing lists that change the contents of an email cannot resend this email using the original author's email address, as this is difficult to arrange without compromising security.

F.2 *security indication*: A security indicator could be implemented in a MUA by making use of the `SMTPsec-Used` header to show if SMTPsec was used.

F.3 *constraint*: By using the 'Require SMTPsec' SMTP extension, the MUA can enforce the use of SMTPsec.

F.4 *reporting*: In SMTPsec, the server immediately responds with an error message if authentication failed or the signature is incorrect, so no further reporting system should be necessary.

## 7.3 Incentives for switching

To make sure that email providers will switch to SMTPsec, there should be incentives for doing so. These may include the following:

- First of all, SMTPsec gives an unambiguous result for message origin authentication, because it is not optional like DKIM/SPF and DMARC in SMTP. Also, integrity checking is more robust and secure because of the elimination of canonicalization.
- Besides being more secure, the protocol will also be faster because it can transmit binary files in a signed email, contrary to DKIM. This also makes it less resource intensive for servers.
- In the end SMTPsec will be much simpler than the current infrastructure, but a disadvantage that we will have to deal with when it is first implemented is that SMTP and DKIM etc. also still have to be supported.
- If providers that implement SMTPsec also add security indicators to emails, then this might speed up the usage of the protocol, because other companies will want the indicator to be displayed on their emails as well.
- Another measure that could be taken by providers to incentivize other providers to implement SMTPsec is prioritizing transactions using the new protocol, allowing larger emails when using the new protocol, or even artificially slowing down SMTP transactions.

# Chapter 8

# Conclusions

We saw how the current email architecture came to be and that it is a complex clutter of protocols. With SMTPsec, we succeeded in creating a more secure and simpler email delivery protocol. The trust model is still the same as with the current architecture: authenticity and integrity of the email are guaranteed from the author's email provider to the recipient's email provider, and confidentiality is ensured only between email providers.

Unfortunately, we found that, because of the need for interoperability with SMTP, the option to outsource email services, and the automatic forwarding feature, some issues were harder to solve in a clean way than initially presumed. One thing we were not able to provide with SMTPsec is a strong protection against replay attacks, but the current architecture also does not offer this. A downside to switching to any new email delivery protocol, including SMTPsec, is that for the time being, providers also still have to support SMTP and its security extension protocols.

## 8.1 Future work

Before the protocol can be implemented, it should receive scrutiny from experts on the email architecture and in the field of protocol design. It would also be a good idea to compose a concrete list of allowed trace headers as referred to in Section 6.4.5. After this process, it should be standardized in the form of an RFC document. When we are convinced that the protocol is ready for implementation, the protocol should be implemented in a cross-platform manner, such that it is usable everywhere.

When the implementation is complete and thoroughly tested, adoption by email providers can begin. Optionally, they can implement artificial incentives like mentioned in Section 7.3. Additionally, MUAs should start implementing a good security indicator system as mentioned in the *security indication* requirement. In order to design a good security indicator system, a survey or interview should be conducted under potential users to evaluate various designs and pick or combine the most effective design(s), meaning that for a user it is clear what the indication means.

Note that email servers supporting SMTPsec should have a secure way to keep an up-to-date list of trusted CAs (a root certificate store) and to handle DNS root key rollover. We think this is doable, because all modern desktop computers already have such a system in place. Besides that, its own certificate has to be renewed from time to time before it expires.

It might also be worth looking into a protocol extension for stapling DNSSEC chains for SRV and TLSA records onto signatures, such that no extra requests have to be made. However, this would be more logical if DNSSEC chains for TLSA records could also be stapled on the server's reply in the TLS handshake, but this idea was dropped, as mentioned in Section 5.5.

Lastly, it might be beneficial for the adoption of SMTPsec to also look into a good way to phase out SMTP (and DKIM etc.). We already discussed some incentives in Section 7.3, but the problem that during the transition period we need to implement both protocols remains.

# Peer-reviewed references


[6] Tatsuya Mori et al. "How is E-Mail Sender Authentication Used and Misused?" In: *Proceedings of the 8th Annual Collaboration, Electronic Messaging, Anti-Abuse and Spam Conference*. CEAS 11. Perth, Australia: Association for Computing Machinery, 2011, pp. 31–37. DOI: 10.1145/2030376.2030380. https://doi.org/10.1145/2030376.2030380.

[7] Zakir Durumeric et al. "Neither Snow Nor Rain Nor MITM...: An Empirical Analysis of Email Delivery Security." In: *Proceedings of the 2015 Internet Measurement Conference*. IMC 15. Tokyo, Japan: Association for Computing Machinery, 2015, pp. 27–39. DOI: 10.1145/2815675.2815695. https://doi.org/10.1145/2815675.2815695.

[8] Ian D. Foster et al. "Security by Any Other Name: On the Effectiveness of Provider Based Email Security." In: *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*. CCS 15. Denver, Colorado, USA: Association for Computing Machinery, 2015, pp. 450–464. DOI: 10.1145/2810103.2813607. https://doi.org/10.1145/2810103.2813607.

[11] H. Hu, P. Peng, and G. Wang. "Towards Understanding the Adoption of Anti-Spoofing Protocols in Email Systems." In: *2018 IEEE Cybersecurity Development (SecDev)*. Cambridge, MA, USA: IEEE, Sept. 2018, pp. 94–101. DOI: 10.1109/SecDev.2018.00020. https://doi.org/10.1109/SecDev.2018.00020.

[39] Eric Osterweil et al. "Zone State Revocation for DNSSEC." In: *Proceedings of the 2007 Workshop on Large Scale Attack Defense*. LSAD 07. Kyoto, Japan: Association for Computing Machinery, 2007, pp. 153–160. DOI: 10.1145/1352664.1352677. https://doi.org/10.1145/1352664.1352677.

[40] Gilles Guette. "Key Revocation System for DNSSEC." In: *Journal of Networks* 3.6 (2008). Journal has been discontinued as of 2016. View the archived homepage at https://web.archive.org/web/20161010000615/http://www.academypublisher.com/jnw/, pp. 54–61. DOI: 10.4304/jnw.3.6.54-61. https://semanticscholar.org/paper/146c2eadaf72e91d066a8a13bcf8096591b00f66.

[56] Natascha Chrobok, Andrew Trotman, and Richard OKeefe. "Advantages and Vulnerabilities of Pull-Based Email-Delivery." In: *Proceedings of the Eighth Australasian Conference on Information Security - Volume 105*. AISC 10. Brisbane, Australia: Australian Computer Society, Inc., 2010, pp. 22–31. https://dl.acm.org/doi/10.5555/1862266.1862272.

[57] Alan Mislove et al. "POST: A Secure, Resilient, Cooperative Messaging System." In: *Proceedings of HotOS'03: 9th Workshop on Hot Topics in Operating Systems*. Ed. by Michael B. Jones. Lihue (Kauai), Hawaii, USA: USENIX, May 2003, pp. 61–66. https://www.usenix.org/conference/hotos-ix/post-secure-resilient-cooperative-messaging-system.

[58] M. D. LeMay and J. S. E. Tan. "Comprehensive message control and assurance with the secure email transport protocol." In: *2004 IEEE Electro/Information Technology Conference*. Milwaukee, WI, USA: IEEE, Aug. 2004, pp. 272–280. DOI: 10.1109/EIT.2004.4569392. https://doi.org/10.1109/EIT.2004.4569392.

[59] K. D. Lux et al. "WSEmail: secure Internet messaging based on Web services." In: *IEEE International Conference on Web Services (ICWS'05)*. Orlando, FL, USA: IEEE, July 2005, 75–82 vol.1. DOI: 10.1109/ICWS.2005.138. https://doi.org/10.1109/ICWS.2005.138.

[60] A. Ghafoor, S. Muftic, and G. Schmölzer. "CryptoNET: Design and implementation of the Secure Email System." In: *2009 Proceedings of the 1st International Workshop on Security and Communication Networks*. Trondheim, Norway: IEEE, May 2009, pp. 1–6. https://ieeexplore.ieee.org/abstract/document/5683054.

[63] Cristian Thiago Moecke and Melanie Volkamer. "Usable secure email communications: criteria and evaluation of existing approaches." In: *Inf. Manag. Comput. Security* 21.1 (May 2013), pp. 41–52. DOI: 10.1108/09685221311314419. https://doi.org/10.1108/09685221311314419.

# Non peer-reviewed references including standards

[1] Ray Tomlison. *The First Email*. 2002. http://openmap.bbn.com/~tomlinso/ray/firstemailframe.html (visited on 03/13/2020).

[2] *Email Statistics Report, 2019-2023 - Executive summary*. Tech. rep. The Radicati Group, 2019. https://www.radicati.com/?p=15792.

[3] NOS. *Phishing weer groeiend probleem, oplichters steeds creatiever*. Nov. 2018. https://nos.nl/artikel/2260753.

[4] *Spam and phishing in Q3 2019*. Tech. rep. Kaspersky Lab, Nov. 2019. https://securelist.com/spam-report-q3-2019/.

[5] NOS. *Iedereen kan mailen namens de AIVD dankzij 'spoofing'*. Oct. 2017. https://nos.nl/artikel/2199557.

[9] Eunice Zsu-Chnn Tan. "A Quantitative Study of the Deployment of the Sender Policy Framework." MA thesis. Brigham Young University, Oct. 2018. https://hdl.lib.byu.edu/1877/etd10375.

[10] Ignacio Sanchez and Gerard Draper-Gil. *My Email Communications Security Assessment (MECSA): 2018 Results*. Tech. rep. Joint Research Centre (European Commission), Feb. 2019. DOI: 10.2760/166203. https://doi.org/10.2760/166203.

[12] Hal Finney et al. *OpenPGP Message Format*. RFC 4880. Nov. 2007. DOI: 10.17487/RFC4880. https://tools.ietf.org/html/rfc4880.

[13] Jim Schaad, Blake C. Ramsdell, and Sean Turner. *Secure/Multipurpose Internet Mail Extensions (S/MIME) Version 4.0 - Message Specification*. RFC 8551. Apr. 2019. DOI: 10.17487/RFC8551. https://tools.ietf.org/html/rfc8551.

[14] Jonathan Bruce Postel. *Simple Mail Transfer Protocol*. RFC 821. Aug. 1982. DOI: 10.17487/RFC0821. https://tools.ietf.org/html/rfc821.

[15] Dr. John C. Klensin. *Simple Mail Transfer Protocol*. RFC 5321. Oct. 2008. DOI: 10.17487/RFC5321. https://tools.ietf.org/html/rfc5321.

[16] Mark Crispin. *Internet Message Access Protocol - Version 4rev1*. RFC 3501. Mar. 2003. DOI: 10.17487/RFC3501. https://tools.ietf.org/html/rfc3501.

[17] Dr. Marshall T. Rose and John G. Myers. *Post Office Protocol - Version 3*. RFC 1939. May 1996. DOI: 10.17487/RFC1939. https://tools.ietf.org/html/rfc1939.

[18] Dave Crocker. *Internet Mail Architecture*. RFC 5598. July 2009. DOI: 10.17487/RFC5598. https://tools.ietf.org/html/rfc5598.

[19] William Mills, Tim Showalter, and Hannes Tschofenig. *A Set of Simple Authentication and Security Layer (SASL) Mechanisms for OAuth*. RFC 7628. Aug. 2015. DOI: 10.17487/RFC7628. https://tools.ietf.org/html/rfc7628.

[20] Kurt Zeilenga and Alexey Melnikov. *Simple Authentication and Security Layer (SASL)*. RFC 4422. June 2006. DOI: 10.17487/RFC4422. https://tools.ietf.org/html/rfc4422.

[21] Pete Resnick. *Internet Message Format*. RFC 5322. Oct. 2008. DOI: 10.17487/RFC5322. https://tools.ietf.org/html/rfc5322.

[22] Ned Freed and Dr. Nathaniel S. Borenstein. *Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies*. RFC 2045. Nov. 1996. DOI: 10.17487/RFC2045. https://tools.ietf.org/html/rfc2045.

[23] Scott Kitterman. *Sender Policy Framework (SPF) for Authorizing Use of Domains in Email, Version 1*. RFC 7208. Apr. 2014. DOI: 10.17487/RFC7208. https://tools.ietf.org/html/rfc7208.

[24] Wayne Schlitt and Meng Weng Wong. *Sender Policy Framework (SPF) for Authorizing Use of Domains in E-Mail, Version 1*. RFC 4408. Apr. 2006. DOI: 10.17487/RFC4408. https://tools.ietf.org/html/rfc4408.

[25] Meng Weng Wong and Jim Lyon. *Sender ID: Authenticating E-Mail*. RFC 4406. Apr. 2006. DOI: 10.17487/RFC4406. https://tools.ietf.org/html/rfc4406.

[26] Eric P. Allman et al. *DomainKeys Identified Mail (DKIM) Signatures*. RFC 4871. May 2007. DOI: 10.17487/RFC4871. https://tools.ietf.org/html/rfc4871.

[27] Murray Kucherawy, Dave Crocker, and Tony Hansen. *DomainKeys Identified Mail (DKIM) Signatures*. RFC 6376. Sept. 2011. DOI: 10.17487/RFC6376. https://tools.ietf.org/html/rfc6376.

[28] Murray Kucherawy and Elizabeth Zwicky. *Domain-based Message Authentication, Reporting, and Conformance (DMARC)*. RFC 7489. Mar. 2015. DOI: 10.17487/RFC7489. https://tools.ietf.org/html/rfc7489.

[29] Murray Kucherawy. *Message Header Field for Indicating Message Authentication Status*. RFC 8601. May 2019. DOI: 10.17487/RFC8601. https://tools.ietf.org/html/rfc8601.

[30] Paul E. Hoffman. *SMTP Service Extension for Secure SMTP over Transport Layer Security*. RFC 3207. Feb. 2002. DOI: 10.17487/RFC3207. https://tools.ietf.org/html/rfc3207.

[31] Jim Fenton. *Analysis of Threats Motivating DomainKeys Identified Mail (DKIM)*. RFC 4686. Sept. 2006. DOI: 10.17487/RFC4686. https://tools.ietf.org/html/rfc4686.

[32] Murray Kucherawy. *DomainKeys Identified Mail (DKIM) and Mailing Lists*. RFC 6377. Sept. 2011. DOI: 10.17487/RFC6377. https://tools.ietf.org/html/rfc6377.

[33] John R. Levine et al. *DomainKeys Identified Mail (DKIM) Author Domain Signing Practices (ADSP)*. RFC 5617. Aug. 2009. DOI: 10.17487/RFC5617. https://tools.ietf.org/html/rfc5617.

[34] Kurt Andersen et al. *The Authenticated Received Chain (ARC) Protocol*. RFC 8617. July 2019. DOI: 10.17487/RFC8617. https://tools.ietf.org/html/rfc8617.

[35] Scott Rose et al. *DNS Security Introduction and Requirements*. RFC 4033. Mar. 2005. DOI: 10.17487/RFC4033. https://tools.ietf.org/html/rfc4033.

[36] Scott Rose et al. *Resource Records for the DNS Security Extensions*. RFC 4034. Mar. 2005. DOI: 10.17487/RFC4034. https://tools.ietf.org/html/rfc4034.

[37] Scott Rose et al. *Protocol Modifications for the DNS Security Extensions*. RFC 4035. Mar. 2005. DOI: 10.17487/RFC4035. https://tools.ietf.org/html/rfc4035.

[38] Derek Atkins and Rob Austein. *Threat Analysis of the Domain Name System (DNS)*. RFC 3833. Aug. 2004. DOI: 10.17487/RFC3833. https://tools.ietf.org/html/rfc3833.

[41] Stephan Friedl, Tom Kaupe, and Sriram Gorti. *TLS Certificate Identity Verification Procedure for SMTP MTA to MTA Connections*. Internet-Draft. Expired draft. Mar. 2014. https://tools.ietf.org/html/draft-friedl-uta-smtp-mta-certs-00.

[42] Paul E. Hoffman and Jakob Schlyter. *The DNS-Based Authentication of Named Entities (DANE) Transport Layer Security (TLS) Protocol: TLSA*. RFC 6698. Aug. 2012. DOI: 10.17487/RFC6698. https://tools.ietf.org/html/rfc6698.

[43] Viktor Dukhovni and Wes Hardaker. *SMTP Security via Opportunistic DNS-Based Authentication of Named Entities (DANE) Transport Layer Security (TLS)*. RFC 7672. Oct. 2015. DOI: 10.17487/RFC7672. https://tools.ietf.org/html/rfc7672.

[44] Daniel Margolis et al. *SMTP MTA Strict Transport Security (MTA-STS)*. RFC 8461. Sept. 2018. DOI: 10.17487/RFC8461. https://tools.ietf.org/html/rfc8461.

[45] Ben Laurie, Adam Langley, and Emilia Kasper. *Certificate Transparency*. RFC 6962. June 2013. DOI: 10.17487/RFC6962. https://tools.ietf.org/html/rfc6962.

[46] Dr. John C. Klensin and Randall Gellens. *Message Submission for Mail*. RFC 6409. Nov. 2011. DOI: 10.17487/RFC6409. https://tools.ietf.org/html/rfc6409.

[47] Keith Moore and Chris Newman. *Cleartext Considered Obsolete: Use of Transport Layer Security (TLS) for Email Submission and Access*. RFC 8314. Jan. 2018. DOI: 10.17487/RFC8314. https://tools.ietf.org/html/rfc8314.

[48] Arnt Gulbrandsen and Dr. Levon Esibov. *A DNS RR for specifying the location of services (DNS SRV)*. RFC 2782. Feb. 2000. DOI: 10.17487/RFC2782. https://tools.ietf.org/html/rfc2782.

[49] Cyrus Daboo. *Use of SRV Records for Locating Email Submission/Access Services*. RFC 6186. Mar. 2011. DOI: 10.17487/RFC6186. https://tools.ietf.org/html/rfc6186.

[50] Alexey Melnikov. *Updated Transport Layer Security (TLS) Server Identity Check Procedure for Email-Related Protocols*. RFC 7817. Mar. 2016. DOI: 10.17487/RFC7817. https://tools.ietf.org/html/rfc7817.

[51] Stefan Santesson. *Internet X.509 Public Key Infrastructure Subject Alternative Name for Expression of Service Name*. RFC 4985. Aug. 2007. DOI: 10.17487/RFC4985. https://tools.ietf.org/html/rfc4985.

[52] Jim Fenton. *SMTP Require TLS Option*. RFC 8689. Nov. 2019. DOI: 10.17487/RFC8689. https://tools.ietf.org/html/rfc8689.

[53] Russ Housley. *Cryptographic Message Syntax (CMS)*. RFC 5652. Sept. 2009. DOI: 10.17487/RFC5652. https://tools.ietf.org/html/rfc5652.

[54] Jim Schaad, Blake C. Ramsdell, and Sean Turner. *Secure/Multipurpose Internet Mail Extensions (S/MIME) Version 4.0 - Certificate Handling*. RFC 8550. Apr. 2019. DOI: 10.17487/RFC8550. https://tools.ietf.org/html/rfc8550.

[55] Daniel J. Bernstein. *Internet Mail 2000*. 2000. https://cr.yp.to/im2000.html (visited on 02/15/2020).

[61] Jonathan Moroney. "The Cipher Mail Transport Protocol (CMTP)." MA thesis. University of Hawaii at Manoa, Aug. 2016. https://hdl.handle.net/10125/51458.

[62] Ladar Levison. *Dark Internet Mail Environment*. June 2018. https://darkmail.info/downloads/dark-internet-mail-environment-june-2018.pdf.

[64] Melinda Shore et al. *A DANE Record and DNSSEC Authentication Chain Extension for TLS*. Internet-Draft. Expired draft. Mar. 2018. https://tools.ietf.org/html/draft-ietf-tls-dnssec-chain-extension-07.

[65] Ned Freed. *SMTP Service Extension for Command Pipelining*. RFC 2920. Sept. 2000. DOI: 10.17487/RFC2920. https://tools.ietf.org/html/rfc2920.

[66] Gregory Vaudreuil. *SMTP Service Extensions for Transmission of Large and Binary MIME Messages*. RFC 3030. Dec. 2000. DOI: 10.17487/RFC3030. https://tools.ietf.org/html/rfc3030.

[67] Eric Rescorla. *The Transport Layer Security (TLS) Protocol Version 1.3*. RFC 8446. Aug. 2018. DOI: 10.17487/RFC8446. https://tools.ietf.org/html/rfc8446.

[68] National Institute of Standards and Technology. *Digital Signature Standard (DSS)*. FIPS 186-4. July 2013. DOI: 10.6028/NIST.FIPS.186-4. https://doi.org/10.6028/NIST.FIPS.186-4.

[69] Simon Josefsson and Ilari Liusvaara. *Edwards-Curve Digital Signature Algorithm (EdDSA)*. RFC 8032. Jan. 2017. DOI: 10.17487/RFC8032. https://tools.ietf.org/html/rfc8032.

[70] Email-Verify.My-Addr.com. *List of most popular email domains*. https://email-verify.my-addr.com/list-of-most-popular-email-domains.php (visited on 06/16/2020).

[71] Moz. *The Top 500 Most Popular Websites*. https://moz.com/top500 (visited on 03/11/2020).

# Appendices

# Appendix A

# Adoption of existing protocols

As mentioned in Chapter 1, there have already been many studies examining the current adoption of email security protocols [6, 7, 8, 9, 10]. However, none of them test for the support for MTA-STS or the REQUIRETLS SMTP extension.

## A.1 Adoption among 99 email domains

We used a list of the 100 most popular email domains in 2016 by number of emails [70] and performed various tests. The results are in Table A.1. One of the domains (voila.fr) has no email-related DNS records at all because its email service has been discontinued. Hence, 99 email domains remain.

First, we tested if SPF and DMARC DNS records were present. For DMARC records we also looked at the specified policy. DKIM key records are on different subdomains of the _domainkey subdomain depending on the selector in the DKIM signature header, so we cannot directly request these without having a selector. Hence, we test if a query for the _domainkey subdomain itself does not result in an error. However, it might be that some DNS servers erroneously reply with an error anyway, even though subdomains exist. This is also the method that is used by internet.nl. For the DANE test, we checked if a TLSA record was present for port 25 on the domain in the MX record with the highest priority and that this record and the MX record are protected by DNSSEC.[1] Besides that, we tested if an MTA-STS DNS record was present and a policy file exists. We also checked inside the policy file what maximum age was used and if the mode was set to 'enforce' (and not 'testing' or 'none'). For the DNSSEC tests, we checked if an RRSIG signature record existed for the record type concerned. If CNAME aliases were used, we also checked for signatures on these records. We did not verify the validity of the signatures. For the SMTP extension tests, we connected to the host and checked if establishing a secure connection was possible and if the server advertised support for the REQUIRETLS extension. If we could not connect to any of the MTAs listed in the MX records, for example due to a filter rejecting our generic PTR record (see Section 2.1.1), this is indicated in the 'unknown support' column.

Note that it may be that some of the domains tested share the same MTAs.

For the most part, these results are comparable or sometimes slightly more optimistic than in the report from 2019 that was mentioned earlier [10]. It is remarkable that none of the

[1] We found no hosts with a TLSA record but an MX record that was not signed.

checked hosts seems to support the REQUIRETLS extensions, although it is still relative new. Furthermore, only 2 hosts have their DMARC record signed using DNSSEC.

There is no clear 'winner' here. Although comcast.net supports all features tested excluding REQUIRETLS, and has an MTA-STS maximum age of 30 days, it does not have a quarantine or reject DMARC policy. Also note that this result does not necessarily mean that Comcast itself enforces security policies for incoming or outgoing email. For a 2015 study into this question, see [8].

| **Feature** | **#domains with support** | **#domains with unknown support** |
|---|---|---|
| SPF | 91 | - |
| DKIM | 68 | - |
| DMARC | 68 | - |
| DMARC + quarantine policy[a] | 12 | - |
| DMARC + reject policy | 23 | - |
| DKIM + DMARC | 50 | - |
| DKIM + DMARC + quarantine policy[a] | 12 | - |
| DKIM + DMARC + reject policy | 22 | - |
| DMARC + DNSSEC | 2[b] | - |
| DANE (TLSA) | 6 | - |
| MTA-STS | 12[c] | - |
| MTA-STS + enforce mode | 4[d] | - |
| MTA-STS + DNSSEC | 4[e] | - |
| MTA-STS + DNSSEC + enforce mode | 1[e] | - |
| MTA-STS + max age $\geq$ 1 day | 12 | - |
| MTA-STS + max age $\geq$ 2 days | 7 | - |
| MTA-STS + max age $\geq$ 8 days | 1 | - |
| MTA-STS + max age $\geq$ 1 day + enforce mode | 4 | - |
| MTA-STS + max age $\geq$ 2 days + enforce mode | 1 | - |
| STARTTLS | 82 | 9 |
| STARTTLS + TLS 1.2 or higher | 80 | 9 |
| STARTTLS + valid certificate with MX name | 77 | 9 |
| REQUIRETLS | 0 | 9 |

Table A.1: Adoption of email security protocols among 99 email domains

[a] We only include hosts that have the policy set to affect all emails. For one host not included here, DMARC is configured such that only 50% of its emails are affected, for testing purposes.

[b] All of these hosts also support DKIM. None of them have a quarantine or reject policy.

[c] This includes 4 domains that also implement DANE.

[d] This includes 1 domain that also implements DANE.

[e] All of these domains also implement DANE.

## A.2 Adoption among 409 Web domains

Besides this, we also performed a test using the same method under the 500 top Web domains in March 2020 [71]. From domains with a `www.` prefix, we removed this prefix beforehand. The percentages in Table A.2 only include the 409 domains that have a reachable email server, to make sure that we do not include non-email domains. (In hindsight it would have given a better view to also include domains with a non-reachable email server, as long as they have explicit `MX` records.) Furthermore, we did not query the top domain for records if none were found on a subdomain, which might cause some false negatives. Even in this very large data set, we found no hosts supporting the `REQUIRETLS` extension.

| **Feature** | **Domains with support** |
|---|---|
| SPF | 93% |
| DKIM | 73% |
| DMARC | 67% |
| DMARC + quarantine policy[a] | 9% |
| DMARC + reject policy[a] | 25% |
| DKIM + DMARC | 52% |
| DKIM + DMARC + quarantine policy[a] | 7% |
| DKIM + DMARC + reject policy[a] | 17% |
| DMARC + DNSSEC | 5% |
| DKIM + DMARC + DNSSEC + quarantine policy[a] | 0 |
| DKIM + DMARC + DNSSEC + reject policy[a] | 3% (11 domains) |
| DANE (`TLSA`) | 1% (5 domains) |
| MTA-STS | 2% (7 domains)[b] |
| MTA-STS + enforce mode | 1% (4 domains)[b] |
| MTA-STS + DNSSEC | 0% (1 domain)[b] |
| MTA-STS + DNSSEC + enforce mode | 0% (1 domain)[b] |
| MTA-STS + max age $\geq$ 1 day | 2% (7 domains) |
| MTA-STS + max age $\geq$ 2 days | 0% (2 domains) |
| MTA-STS + max age $\geq$ 8 days | 0% (1 domain) |
| MTA-STS + max age $\geq$ 1 day + enforce mode | 1% (4 domains) |
| MTA-STS + max age $\geq$ 2 days + enforce mode | 0% (1 domain) |
| `STARTTLS` | 95% |
| `STARTTLS` + TLS 1.2 or higher | 95% |
| `STARTTLS` + valid certificate with `MX` name | 88% |
| `REQUIRETLS` | 0 |

Table A.2: Adoption of email security protocols among 409 top Web domains

[a] We only include hosts that have the policy set to affect all emails.
[b] This includes 1 domain that also implements DANE.

# Appendix B

# Protocol data structures

```
enum {
    require_security(0)
    /* more extensions may be defined in the future */
} ExtensionType;

struct {
    ExtensionType type;
    opaque data<0..2^32-1>;
} Extension;

struct {
    uint8 version = 0;
} ServerHello;

opaque Certificate<0..2^24-1>;
Certificate CertificateChain<0..2^16-1>;

struct {
    uint8 version = 0;
    CertificateChain certificate_chains<0..2^24-1>;
} ClientHello;

opaque Address<0..2^16-1>;  /* email address of the form local@domain */

enum {
    /* ECDSA with curves as defined in [68] */
    ecdsa_secp256r1_sha256(0x0403), /* with SHA-256 hash */
    ecdsa_secp384r1_sha384(0x0503), /* with SHA-384 hash */
    ecdsa_secp521r1_sha512(0x0603), /* with SHA-512 hash */

    /* PureEdDSA with curves as defined in [69] */
    ed25519(0x0807),
    ed448(0x0808),
    (0xFFFF)
} SignatureAlgorithm;
```

```
struct {
    SignatureAlgorithm algorithm;
    opaque parameters<0..2^16-1>;
} SignatureMethod;

struct {
    opaque name<0..2^16-1>;
    opaque value<0..2^24-1>;
} Header;

struct {
    uint24 certificate_chain;
    SignatureMethod signature_method;
    opaque signature[signature_length];

    Header trace_headers<0..2^24-1>;
    Address recipients<0..2^24-1>;
    opaque message<0..2^64-1>;
} Email;

struct {
    uint24 email;
} EmailDelivered;

enum {
    internal_error(0),
    service_unavailable(1),
    illegal_message(2),
    illegal_parameter(3),
    too_large(4),
    authentication_failed(5),
    insufficient_security(6),
    unknown_address(7)
} ErrorCode;

struct {
    none(0),
    message(1)
} ErrorReference;

struct {
    ErrorCode code;
    ErrorReference reference;
    select (Error.reference) {
        case none: struct {};
        case message: uint24 messageNumber;
    }
    select (Error.code) {
        case unknown_address: Address<0..2^16-1>;
        default: struct {}; /* for other error codes */
    };
} Error;
```

```
enum {
    error(0),
    server_hello(1),
    client_hello(2),
    email(3),
    email_delivered(4)
} MessageType;

struct {
    uint64 size;
    MessageType type;
    Extension extensions<0..2^16-1>;
    select (Message.type) {
        case error: Error;
        case server_hello: ServerHello;
        case client_hello: ClientHello;
        case email: Email;
        case email_delivered: EmailDelivered;
    }
} Message;
```