Automated Key Management for End-To-End Encrypted Email Communication

Thomas Maier
Technical University of Munich (TUM)
Email: ga85how@mytum.de

***Abstract*—Studies show that problems appear in the scope of end-to-end encrypted mailing. Key exchange procedures are hard to understand and difficult to use for end users. This paper proposes protocols enabling an automated key exchange by using the existing mail ecosystem. End users put trust in their mail provider regarding the storage of their emails already. The proposed protocols leverage this trust relationship by seeing the provider as a trusted third party. The proposed key exchange sequence reuses authentication mechanisms that are already implemented and used within the Simple Mail Transfer Protocol. Therefore the mail provider takes care of user authentication in order to make uploaded public keys publicly available. Previously designed protocols do not offer the possibility of a fully-automated exchange of authenticated public keys. The hereby defined protocols ensure the key authenticity by using the existing trust relationship. Therefor this paper introduces two protocols – the Key Publication Protocol for submitting public keys and the Key Retrieval Protocol for requesting public keys. Both protocols are designed highly scalable since each of them makes use of two emails. The protocols may be implemented in a way to be usable since they enable a fully-automated key exchange sequence. Furthermore a Prove of Concept implementation within the paper points out that the protocols are deployable in a simple way. Therefore reaching an higher adoption rate and low expenditures are favored.**

I. Introduction

Studies show that it is hard for end users to use end-to-end email encryption securely [1]–[3]. One of the major usability issues is the inability to execute manual tasks like sending, retrieving and verifying public keys. Therefore automating the key exchange may lead to better usability. That leads to the following research question:

How is it possible to automatically exchange authenticated public keys in order to make end-to-end encrypted mailing more usable?

This paper presents two novel protocols as a solution to this problem. They assist the end user by automating the key publication and key retrieval workflows. The proposed automation may contribute to the simplification of today's secure email encryption.

First related approaches of open source and commercialized projects and products are introduced in section II. Afterwards the difficulties of existing solutions are analyzed in order to identify problems (section III). Then the protocol design is getting developed in section IV based on the beforehand explained problems and design goals. Section V presents an exemplary implementation. This Prove of Concept is used to evaluate the proposed protocols. The penultimate section VI explains further issues to be discussed in the future. Finally, section VII summarizes the results and provides conclusions regarding the designed protocols.

II. Related Work

Within this section existing solutions are introduced, that concentrate on making end-to-end encryption more usable or secure. The following categories are chosen to classify the variety of different solutions. Those deviate strongly regarding implemented features and addressed problems. On the one hand the approaches concentrate on practical implementations. On the other hand theoretical concepts were designed. There are solutions provided in a public manner but there are still specific implementations offered by individual vendors only.

A. Transparency Frameworks

Projects like Certificate Transparency (CT) [4], CONIKS [5] and Key Transparency [6] emerged with the goal to improve the recognition of forged certificates or public keys. CT makes it possible to monitor issued certificates using a public log in order to detect suspicious activities of Certificate Authorities. The authors of CONIKS add additional objectives like the provision of privacy-preserving key directories and the capability to monitor logs efficiently. KT follows the objectives of CT and adjusts this concept to apply it on plain public keys instead of whole certificates.

One of the disadvantages of the previously mentioned mechanisms is to verify the validity of the user-key-binding after it has been published. The user-key-binding is only considered retroactively via monitoring. Contrary to the solution we propose, these projects try to ensure the consistency of the cryptographic keys over time.

B. Manual Key Verification

Another option to verify the key authenticity is the manual confirmation over an additional secure channel. This opportunity is given by the OpenPGP system implicitly for instance. In this case, the key fingerprint can be verified by confirming it via the mentioned additional channel.

A project called *pretty Easy Privacy* (pEp or p≡p) uses a similar approach with manual verification [7]. Also OpenPGP is used but with well pronounceable *Trustwords*. Those *Trustwords* are generated out of the public key.

The drawback of manual key verification is that it is not possible to fully automate those solutions. Thereby the key exchange workflow forces users to verify the key authenticity out of band.

C. Web of Trust

With OpenPGP some standardizations and common practices, like signatures of public keys were developed. The idea builds upon key verification by independent entities to establish trust. The standard provides two options in order to accomplish the verification. People have to either (1) check the user-key-binding off-the-record (e.g., personally) or (2) by putting confidence in key signatures. End users generally do not accept both options because they decrease the grade of usability [1]–[3].

D. Key Servers

Public key servers offer the service to publish and retrieve keys. Authenticating users regarding the uploaded key is not possible as key servers are not necessarily hosted by a specific provider. The *HTTP KeyServer Protocol* (HKP) is intended for spreading public keys but has no mechanism for authentication [8], [9].

The *GnuPG Web Key Directory* (WKD) solves this problem by implementing an HTTP server as an additional service within the mail provider's infrastructure [10]. This HTTP server holds the users' keys. The authenticated submission of those keys is under discussion during this paper as the project website and a RFC draft state [11], [12]. One idea is to use the same HTTP server. This demands the service to authenticate the user, meaning the HTTP server needs to know the user's credentials as well. This leads to a lower degree of security because credentials are accessible by an additional service. Another suggested approach is to use email in order to submit the public key. The project did not release a final specification about the key publication process.

E. Mail Provider Approaches

Some providers try to support their users in end-to-end encryption. Most of them use isolated applications in order to cover their own users only.

1) Public Key Upload and Retrieval: During research we found three common ways – the first is uploading keys via a web interface [13]–[15]. The providers ensure authenticity by checking the user's account credentials via a web application. All found providers implemented their own solution for this feature. The main drawback is that those proprietary implementations hinder a widespread adoption of the used concepts.

A second option providers offer, is making use of their own or other public key servers [13], [16], [17]. A German project called *Vertrauenswürdige Verteilung von Verschlüsselungsschlüsseln* (VVV) developed a solution to publish keys via provider's key servers [18]. Users' are allowed to upload their key after authentication via their account credentials. Thus providers offer additional services to the *Simple Mail Transfer Protocol* (SMTP). Hereby security implications arise because of the fact that users' credentials are handled in places additional to the SMTP server.

A third solution is using the *Domain Name System* (DNS) [13]. The `OPENPGPKEY` DNS Resource Record is used to publish OpenPGP public keys. *Domain Name System Security Extensions* (DNSSEC) are necessary to avoid forged responses. There are still several disadvantages regarding DNSSEC [19]. For instance *Extension Mechanisms for DNS* (EDNS) [20] are needed because a higher packet size is required for DNSSEC responses. Those amplified packets lead to a higher risk of DNS reflection attacks and therefore *Denial of Service* [21].

2) Approaches to Assist in Encryption: Some providers support their users in fetching public keys from provider's services and internal/external key servers [22]. Those providers offer their own web interface for that (e.g., mailbox.org Guard [23]) or make use of browser addons (like Mailvelope [13], [24]–[26]). All found providers assist in encryption via web interfaces only. Hence it is mandatory for users to use specific web interfaces instead of using another *Mail User Agent* (MUA).

3) Service Discovery: Only one way was found to discover key servers or other key services hosted by providers (e.g., mailbox.org). For such it is necessary to fetch the DNS Resource Record `SRV` with the Domain prefix `_hkps._tcp.` to get the domain of the provider's HKP key server. As mentioned, it is necessary to verify this record with DNSSEC. The usage of DNSSEC leads to drawbacks as mentioned in section II-E1.

4) Deployment Distribution: As mentioned all providers develop isolated applications. Therefore it is not possible to reuse most of the existing solutions. Despite of that some providers have made some parts of their solutions open source (e.g., Roundcube integration [27] or the ProtonMail Web Client [28]).

F. Client-side Approach

There is only one client-side approach with no respect to desktop mail clients. Mailpile is a web-mail client, that tries to simplify end-to-end encryption. It assists the user with retrieving new public keys. The software uses the *Trust on First Use* concept (TOFU) [29], i.e., they put confidence in the assumption that the initially retrieved key is the right one. Mailpile does not provide any native feature to authenticate the user-key-binding.

G. Guidelines

After looking into practical implementations the *autocrypt* guidelines [30] need to be introduced. The goal of the project is to replace cleartext with end-to-end encrypted e-mail. The published specification only protects against passive attackers until now. Therefore they cannot be used for practical use yet.

III. Analysis

The last section describes existing approaches with their drawbacks. Those issues are analyzed within this section.

The central problem to be solved is that end-to-end encryption is difficult for end users [1]–[3]. The lack of knowledge leads to severe usability issues and security problems. One of the major impediments is the key exchange between end users. To go more into detail this is about the inability to accomplish (1) sending and receiving public keys and (2) verifying keys and signatures.

A solution to this issue is to automate the key exchange completely with two workflows in order to accomplish the exchange of a public key K_{pub} between the MUA of Alice and the MUA of Bob.

- The *Key Publication Workflow* describes how Alice is able to upload K_{pub} after authenticating herself towards her mail provider.
- The *Key Retrieval Workflow* requires Bob to find the key-holding server (used by Alice) first of all. Afterwards Bob has to be able to request K_{pub} from that server without any authentication.

Both workflows need to ensure integrity and authenticity of the exchanged key K_{pub}.

The following sections come up with several problems (indicated by *P* and an identifier). Every single one of them has been derived from solutions described within the related work section II. Design goals are introduced (indicated by *G* and an identifier) in order to solve those problems.

A. P1 – Trust Establishment

Manual verification of keys would corrupt the automation of the process. Additionally a manual verification is not accepted by end users as mentioned in section II-C. Therefore the automated key exchange process requires a trusted third party to ensure the authenticity of the exchanged key. The drawback of central trust providers is that they are attractive targets for attackers as they maintain large quantities of keys.

G1 – Opportunistic Distributed Approach: The workflows ensure integrity and authenticity by putting trust into the own mail provider and its infrastructure. Therefore the end user's confidence about their mail provider's trustworthiness is beneficial. Putting trust in the own mail provider is more secure than not making use of encryption (resp. the key exchange) at all. The mail provider has the opportunity to verify the user's identity by executing an authentication process. Hence the same credentials could be used by the mail provider to authenticate regarding the acceptance of the user's key. This is how the provider can ensure that the received K_{pub} is bound to the user's account. Therefore the mail provider can announce the key publicly. Anybody else who asks the same provider for K_{pub} can assume that the provider ensures the authenticity of the key with appropriate mechanisms.

This approach requires end users' trust to their mail provider and therefore the mail provider's infrastructure. The MUA can use this trust relationship to make K_{pub} publicly available without fear of forgery. This trust relationship is not possible with public key servers because of the lack of authentication as mentioned in section II-D.

B. P2 – Adoption and Deployment

This work proposes two workflows, that need to be adopted by users. The benefits of the presented workflows are less useful without a certain level of adoption. In order to increase the adoption rate it is necessary to simplify the deployment of all required software components as well. The following design goals solve those problems.

G2a – Scalability: The higher the adoption rate the more scalable the software components should be. It is necessary to consider scalability since fast adoption is desired. Therefore the amount of necessary messages per key publication and retrieval is crucial.

Scalability is important for the proposed workflows as they need to maintain performance regardless of the amount of users and mail providers. Proprietary approaches offered by individual mail providers are isolated (section II-E). Therefore they do not need to specify how to deal with scalability publicly.

G2b – Cost Efficiency: In order to reach a wide-spread and consistent solution it is necessary to provide cost-efficient handling of software packages. Providers and users should have low costs regarding expenditure of time and money. This means that simple deployment should be possible – both on the client-side and on the server-side. The same should apply regarding the maintenance for the end user and for the provider.

C. P3 – Key Authenticity and Integrity

Assurance of integrity and authenticity is mandatory regarding the detection of forged keys. All existing approaches described in section II are either isolated applications or are not accepted by end users because of the lack of automation.

G3 – Message Authentication Codes: A solution is to use *Message Authentication Codes* (MAC) generated by the mail provider. In both the *Key Publication* as well as the *Key Retrieval* it is necessary to send responses. While doing so, the provider is unable to ensure that the message will not be modified on its way to the original MUA. This is why the MUA has to send randomly-generated keys, that can be used for the MAC. After receiving the response message the MUA is able to verify that the content of the message matches the generated MAC. This is how active attacks can be detected, meaning that authenticity and integrity of the key can be checked.

D. P4 – Service Discovery

The automation of the described workflows require an automatable way of service discovery as well. Many existing solutions are isolated applications as described in the related work section. It is unnecessary to search and find key-holding servers in those cases. In openly available solutions like HKP key servers DNS resource records are used to announce the ip address of key-holding servers.

G4 – Service Lookup: The bottom line is that clients need to find the provider's key-holding server. Afterwards those servers can be used for key publication and key retrieval. The

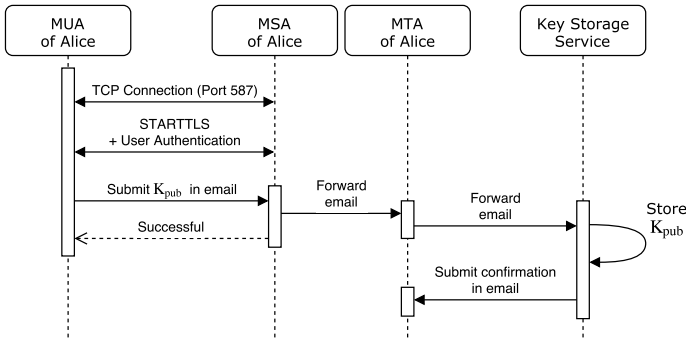

Fig. 1. Key Publication Protocol

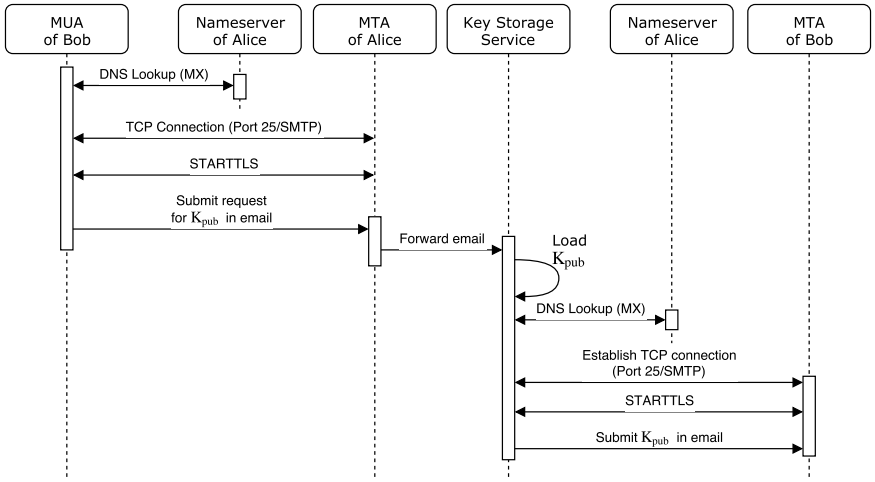

Fig. 2. Key Retrieval Protocol

proposed workflows described herein need a way to disclose the key-holding server's ip address.

IV. Protocol Design

The protocols defined in this part enable the fulfillment of the design goals for the analyzed problems. Both protocols are based on publishing and retrieving public keys utilizing emails transferred on top of SMTP.

The proposed protocols demand the transfer of a few necessary emails. This manageable amount is how the protocols can ensure a high level of scalability (*G2a*).

Additionally the protocols are addressed to keep costs under control (*G2b*) by using as much existing technology as possible. Both protocols are based on the existing email ecosystem that is distributed over the internet. The email format is used for all messages sent within the protocols as explained afterwards in detail. Additionally neither new complex nor existing applications need to be integrated or changed. This ensures a low maintenance effort (*G2b*) as well. Furthermore users look after themselves, meaning they care about publishing and retrieving keys with no provider interference (*G2b*).

Regarding goal *G4* (*Service Discovery*) it is required that clients are able to discover the right mail server (resp. its ip address). This issue is solved particularly for the specified protocols afterwards utilizing the DNS. Moreover they need to find out whether the described workflows are supported. This problem is solved by sending and receiving emails. The MUA knows whether the service is supported if receiving a properly formatted email. Hence bounce emails are handled as improperly formatted emails, i.e. the service is not supported by the contacted provider.

A. Key Publication Protocol

The first protocol handles the key publishing sequence (see figure 1). Initially clients need to connect to the SMTP submission port (587) of the *Message Submission Agent* (MSA) as defined for the usual submission of emails [31]. It should be considered that the MUA knows the submission port already in order to send usual emails. If the MUA does not know the submission server, DNS could be used to request the appropriate `SRV` resource record [32].

The MSA has to demand user authentication before submitting a new email as specified for the submission port [31]. This is how the provider verifies the identity of the MUA and thereby ensures the authenticity of the email to be transfered. This sequence can be seen as a warranty of authenticity and integrity (*G3*) if the transport is secured by

making use of *Transport Layer Security* (TLS). The MSA accepts the email (with K_{pub}) and adds the `Authenticated sender` header, that contains the mailbox account id of the authenticated user [33], [34]. Furthermore the email must contain a base64-encoded randomly generated key K_{auth}. This key is required to guarantee integrity in a later phase.

The *Key Publication Protocol* defines that both K_{pub} as well as K_{auth} have to be sent within the email body. K_{pub} has to be in the form of *ASCII Armor*, that uses the base64 encoding. This data format is defined within the *OpenPGP Message Format* specification [35].

The defined recipient address is defined with `keys` as the local part [36]. This address triggers the MSA/MTA to channel out the email to a *Key Storage Service* (KST). The KST checks whether the MSA set the `Authenticated sender` header and reuses the containing mailbox account to store the key (*G3*). After the storage process the KST has to send a confirmation email. This affirmation contains a *Keyed-Hash for Message Authentication* (HMAC). Finally, the MUA must verify $HMAC(K_{auth}, K_{pub})$ to ensure the successful storage process.

This protocol is highly scalable since two emails are necessary for the whole publication protocol (*G2a*). Even big provider infrastructures could be managed with more than one mail server. Therefore a provider has to synchronize keys between its own mail servers.

B. Key Retrieval Protocol

The second protocol enables fetching of someone's public key K_{pub} (see figure 2). The *Key Retrieval Protocol* uses a DNS lookup in order to learn the ip address of the key-holding MTA at first (`MX` resource record). Afterwards the MUA is able to establish a SMTP connection by connecting to port 25. That port is typically used for server-to-server communication without authentication mechanisms [37]. In the described use case the user has no mailbox account at the key-holding provider necessarily. This is why the protocol has to circumvent authentication. Additionally the `MX` resource record is defined as the record, that provides the ip address serving port 25 [37]. The established connection has to be secured by making use of TLS (as mentioned for the *Key Publicaton Protocol*).

Now the email has to be submitted with the key query within the email body. The email request must contain the queried email address and a base64-encoded randomly generated key K_{auth}, that is used later for the key response. The MUA keeps K_{auth} for later verification. Then the MTA receives the email and forwards it to the KST based on the recipient email address. This address contains the string `keys` as the local part as seen in the *Key Publication Protocol* already. The KST loads K_{pub}, calculates $HMAC(K_{auth}, K_{pub})$ and sends both back to the sender's mailbox. Now the sender is able to fetch the response and to extract K_{pub}. Finally, the MUA can ensure key authenticity and integrity by verifying the $HMAC$ (*G3*).

```
1   From alice@example.net Sun Mar 4 16:47:07 2018
2   Return-Path: <alice@example.net>
3   Received: from [10.0.0.1] (unknown [12.80.3.21])
4   (Authenticated sender: alice@example.net)
5   by example.net (Postfix) with ESMTPSA id 53F3A161B01
6   for <keys@example.net>; Sun, 4 Mar 2018 16:47:07 +0100 (AT)
7   To: keys@example.net
8   From: Alice <alice@example.net>
9   Message-ID: <0efca192...bb11e87d9978@example.net>
10  Date: Sun, 4 Mar 2018 16:47:09 +0100
11
12  -----BEGIN PGP PUBLIC KEY BLOCK-----
13  VGhlIHF1aWNrIGJyb3duIGZveCBqdW1wcyBvdmVyIHRoZSBsYXp5
14                          ⋮
15  -----END PGP PUBLIC KEY BLOCK-----
16  4QPeQ/MY6VCZ/8TczqZemjLKP2rvbCwxXGhC7S2v80E=
```

Fig. 3. Key publication email

```
1   From keys@example.net Sun Mar 4 16:04:33 2018
2   Return-Path: <keys@example.net>
3   Received: by example.net (Postfix, from userid 1000)
4   id 984191607DE; Sun, 4 Mar 2018 16:04:33 +0100 (AT)
5   To: <alice@example.net>
6   Message-Id: <201803...91607DE@example.net>
7   Date: Sun, 4 Mar 2018 16:04:33 +0100 (AT)
8   From: keys@example.net (Key Storage Service)
9
10  AnqfPJRZeXjOJHZ4BEkJiJ4tXn4vRM5D81dtcH3bb0Q=
```

Fig. 4. Key confirmation email

The *Key Retrieval Protocol* is highly scalable as making use of two emails for the whole workflow (*G2a*). Hence low network traffic is induced for this protocol as well.

V. Implementation and Evaluation

The last sections point out the analyzed problems and the designed protocols. A *Prove of Concept* (PoC) has been implemented within the scope of this paper. This part explains how the proposed protocols were implemented. Additionally the implementation is used to evaluate the introduced concepts and design goals.

A. Prove of Concept

The mail server *Postfix* is used for the mentioned authentication sequence. The implementation is possible with every mail server that is able to route emails to external services. Therefore it is used as the MSA and MTA in order to receive the emails for submitting keys and key queries.

If the MUA tries to submit a new key after the authentication sequence, the server appends the `Authenticated sender` header (line 4 in figure 3). The MUA sends K_{auth} within the email as well. Then it forwards the email to the KST. The KST is a Python script used to parse the received email. It then stores the key on the file system in case of a successful authentication. The containing K_{auth} (line 16) is used to generate the HMAC utilizing K_{pub} (line 12-15). For this PoC `hmac-sha256` is used as the HMAC algorithm with a 32-byte-key. Then the confirmation email (figure 4) is sent by the script in reply containing the HMAC (line 10).

The *Key Retrieval Protocol* has been implemented as follows. The key request email (figure 5) has to be parsed if no

```
1   From bob@acme.net Sun Mar 4 16:34:15 2018
2   Return-Path: <bob@acme.net>
3   Received: by example.net (Postfix, from userid 112)
4   id 61A40160923; Sun, 4 Mar 2018 16:34:15 +0100 (AT)
5   Received: from [10.0.0.1] (unknown [12.80.3.21])
6   by example.net (Postfix) with ESMTP id C151F1607DC
7   for <keys@example.net>; Sun, 4 Mar 2018 16:33:53 +0100 (AT)
8   Message-Id: <2018031...160923@example.net>
9   Date: Sun, 4 Mar 2018 16:34:15 +0100 (AT)
10  From: bob@example.net
11
12  alice@example.net
13  fjGYoKb6RUEmx1EWKSj6KkKR5U6hTzIi/FH5HFfdgSs=
```

Fig. 5. Key request email

```
1   Return-Path: <keys@example.net>
2   Delivered-To: bob@acme.net
                  ⋮
3   To: <bob@acme.net>
4   Message-Id: <201803...160923@example.net>
5   Date: Sun, 4 Mar 2018 16:50:33 +0100 (AT)
6   From: keys@example.net (Key Storage Service)
7
8   -----BEGIN PGP PUBLIC KEY BLOCK-----
9   VGhlIHF1aWNrIGJyb3duIGZveCBqdW1wcyBvdmVyIHRoZSBsYXp5
                              ⋮
10  -----END PGP PUBLIC KEY BLOCK-----
11  RLgV0ga4A11yEAqSAHOsVwO6lNsZou0V2nMzzOD9XbY=
```

Fig. 6. Key response email

`Authenticated sender` header has been found within. It must contain both the queried email address (line 12) and K_{auth} (line 13) again. The queried email is used to find K_{pub} on the file system. The HMAC has to be generated over K_{pub} (`hmac-sha256` with a 32-byte-key again). The KST sends the key response email (figure 6) containing K_{pub} (line 9-12) and the HMAC (line 13).

We developed a Python application on the client-side to test the server implementation. The Python script implements both protocols and behaves in the same way as a MUA would. Two different modes allow the script to handle the key publication and the key retrieval sequences.

It has become evident that both protocols work as specified in section IV. Concepts used in the protocol design are well-understood in the security community. Utilized cryptographic algorithms and trust relationships are very simple to integrate because of existing frameworks. It turned out that the effort of development, installation and maintenance is very low. The low cost effort may lead to a higher adoption and deployment rate (*G2b*).

B. SMTP Reverse Lookup

The PoC shows that implications can occur in case of requesting a key from the MTA (port 25). For instance the *Postfix* mail server implements an optional feature called `reject_unknown_client_hostname` [38]. Setting this option leads to checks of the client ip address. If the DNS reverse lookup does not match with the sender email address, the email can be rejected. As a consequence this security feature can not be used with the designed protocols. Reverse lookups would be possible only if the MUA uses an ip address where the reverse lookup ends in the right domain. This has to be the domain in the last part of the email address for every request. The end user could ensure that the `PTR` resource record is set right for the used internet connection [39]. This is unlikely, especially for mobile devices. The usage of different gateways at different locations leads to changing ip addresses.

C. Key Validity and Trustworthiness

The OpenPGP standard supports validity mechanisms for key expiration and revocation [35]. Those features are not part of this work and were not implemented within the PoC. As a first proposal both mechanisms can be covered by the protocols implicitly. If a key expires, the MUA can include an implementation to automatically submit a new key pair. The public key can be published with the specified *Key Publication Protocol*. The same applies for key revocation. If problems like the leakage of the published private key occur, a new public key can be published. This action has to be triggered by the end user.

Another feature OpenPGP offers is signing trusted public keys to establish higher trust [35]. This part of the specification is not considered within this paper as well. Nevertheless the protocols do not exclude the feature. It is still possible to submit keys including their signatures to the provider's mail server.

D. Complementary Protocols

As described in section II there are existing approaches related to the proposed protocols. Especially the transparency frameworks can be used as a suitable enhancement. The proposed protocols offer an authenticated way of key distribution. While being in possession of a public key of another person, transparency frameworks can be used to monitor whether the key material has been changed.

The proposed protocols still show up with some drawbacks regarding performance. Compared to lower-level protocols email transport leads to a higher amount of transmitted information, meaning the overhead is higher. For instance the usage of plain SMTP connections for the publication and retrieval protocol is much more efficient. Email transmission consists of the transport over multiple hops. This could lead to unreliable performance, especially caused by the fact that queuing and caching mechanisms are in use. Therefore it is recommended to use the proposed protocols before the end user wants to send an encrypted message. As a consequence the user is not forced to wait until the key retrieval has been finished.

The proposed protocols qualify for subsequent verification as well, meaning key authenticity check in retrospect. The original source of the public key does not matter for that purpose.

E. Big Mail Providers

The PoC consists of an implementation for infrastructures with a single mail server. Implications may emerge despite of the scalable nature of the protocols. Mail providers do not

handle the same amount of mailboxes as statistics show [40]. Therefore it can be expected that they would have to handle a different amount of public keys if they implement the proposed protocols. Hence there would be mail providers, that handle more public keys than others. Being in the possession of more keys leads to a higher attractiveness of active attackers. Therefore it can be assumed that big mail providers are more threatened by attackers than smaller ones.

VI. Future Work

The last chapter points out how the developed workflows solve the previously analyzed problems. Nevertheless a few points still need to be conducted. The following sections show topics to be discussed in the future.

A. DNS Security

DNS is used within the proposed protocols as figures 1 and 2 show. DNS lookups are neither secured against passive nor active attacks. Avoidance of passive attacks is not mandatory for the designed protocols because privacy was not a goal of DNS. Certainly active attacks are more problematic. In the current design, the protocols are vulnerable against the forgery of DNS responses. Therefore `MX` resource records can be spurious.

There are several solutions that should be considered in the future to harden the protocols. One of them would be to use DNSSEC in order to secure the `MX` records. Currently, this approach is not conceivable as DNSSEC is not commonly deployed as research shows [41]. *DNS-based Authentication of Named Entities* (DANE) could enable the verification of the used X.509 certificates additionally. DANE is not commonly implemented as well [41].

Another solution is called *DNS over TLS* and is specified to avoid passive and active attackers [42]. Some common client and server applications' feature implementation is in progress already [43].

Those solutions are necessary to stabilize the key publication and key retrieval.

B. Cryptographic Primitives

This paper uses `hmac-sha256` for the HMAC generation with a 32-byte sized key. This decision has to be considered in the future. The same applies for the mentioned keys.

The protocols can be improved in order to be more future-proof regarding security. A more dynamic way of choosing cryptographic primitives could ensure a long-lasting adoption. The protocols should be altered in order to enable the replacement of broken algorithms quickly. Therefore new protocol versions have to be specified.

C. Blocked Email Traffic

An effective resistance against spam is blocking outgoing traffic with the destination port 25 [44], [45]. This causes problems regarding the *Key Retrieval Protocol*. The protocol is based on requesting keys by submitting an email without authentication. Therefor the MUA queries the MTA of another person's provider using port 25. Future work may consider this issue since blocking this port is pretty common [44], [45].

D. Key Synchronization

The PoC has been implemented for a single mail server instance. Problems could emerge regarding multiple mail servers (resp. key repositories) on behalf of a mail provider. Synchronization of submitted public keys is necessary to avoid or resolve conflicts between servers. This issue has to be conducted in the future.

VII. Conclusions

The research of commercial and non-commercial software and projects has shown that existing solutions have very diverse approaches concerning assistance regarding key exchange. Most of the provider's implementations are proprietary and therefore developed for individual mail providers only. Non-commercial approaches do not solve problems regarding the automation of the key exchange.

The hereby presented protocols solve the analyzed problems derivated in section III. Both protocols handle the key exchange automatically with no user interaction. The provider is used in the sense that a trust relationship can be established (*G1*). A distributed approach (*G2a*) and the little installation and maintenance effort required (*G2b*) may enable a high adoption rate and a simple way of deployment. The use of mailbox credentials and the HMAC generation ensure key authenticity and integrity (*G3*). The established trust relationship with the mail provider can be utilized for that goal. DNS is used to discover whether the protocols are supported by the provider (*G4*).

The PoC implementation of the client and server software shows that the designed protocols can be integrated into existing mail infrastructures in a simple way. Hence existing infrastructures do not need to be changed. The costs of the implementation can be kept low because of the usage of common concepts like email.

The usage of DNS within the proposed protocols still leads to security issues to be resolved in the future. However, the automated workflows may lead to a high adoption rate by end users. This is how the automation of the key exchange may enable improvements of the current security level within the mailing ecosystem. The practical usage of the developed protocols can be extended by using other protocols in combination. Using protocols like *Key Transparency* may ensure the fast detection of forged keys for example.

References

[1] A. Whitten and J. D. Tygar, "Why Johnny Can'T Encrypt: A Usability Evaluation of PGP 5.0," in *Proceedings of the 8th Conference on USENIX Security Symposium - Volume 8*, ser. SSYM'99, Berkeley, CA, USA: USENIX Association, 1999, pp. 14–14. [Online]. Available: http://dl.acm.org/citation.cfm?id=1251421.1251435 (visited on 01/14/2018).

[2] S. Ruoti, J. Andersen, D. Zappala, and K. Seamons, "Why Johnny Still, Still Can't Encrypt: Evaluating the Usability of a Modern PGP Client," *arXiv:1510.08555 [cs]*, Oct. 2015, arXiv: 1510.08555. [Online]. Available: http://arxiv.org/abs/1510.08555 (visited on 11/22/2017).

[3] S. Sheng, L. Broderick, J. J Hyland, and C. Alison Koranda, *Why Johnny still can't encrypt: evaluating the usability of email encryption software*. ACM - Proceedings of the second symposium on Usable privacy and security, Nov. 2017.

[4] A. Langley, E. Kasper, and B. Laurie, "Certificate Transparency," RFC Editor, RFC 6962. [Online]. Available: http://www.rfc-editor.org/rfc/rfc6962.txt (visited on 11/26/2017).

[5] M. S. Melara, A. Blankstein, J. Bonneau, E. W. Felten, and M. J. Freedman, "CONIKS: Bringing Key Transparency to End Users," in *Proceedings of the 24th USENIX Conference on Security Symposium*, ser. SEC'15, Berkeley, CA, USA: USENIX Association, 2015, pp. 383–398, ISBN: 978-1-931971-23-2. [Online]. Available: http://dl.acm.org/citation.cfm?id=2831143.2831168 (visited on 11/28/2017).

[6] *Key Transparency: A transparent and secure way to look up public keys*, Nov. 2017. [Online]. Available: https://github.com/google/keytransparency (visited on 11/26/2017).

[7] *Pretty easy privacy documentation*. [Online]. Available: https://prettyeasyprivacy.com/docs/ (visited on 03/06/2018).

[8] M. Horowitz, "A PGP Public Key Server," 1997. [Online]. Available: http://www.mit.edu/afs/net.mit.edu/project/pks/thesis/paper/thesis.html.

[9] D. Shaw, "The OpenPGP HTTP Keyserver Protocol (HKP)," en, IETF Secretariat, Internet-Draft draft-shaw-openpgp-hkp-00.txt, 2003.

[10] *Web Key Directory - Pubkey Distribution Concept*. [Online]. Available: https://wiki.gnupg.org/EasyGpg2016/PubkeyDistributionConcept (visited on 03/06/2018).

[11] *WKD - GnuPG wiki*. [Online]. Available: https://wiki.gnupg.org/WKD (visited on 03/08/2018).

[12] W. Koch, "OpenPGP Web Key Directory," en, IETF Secretariat, Internet-Draft draft-koch-openpgp-webkey-service-05.txt.

[13] *Posteo - Encryption*. [Online]. Available: https://posteo.de/en/site/encryption (visited on 11/30/2017).

[14] *Posteo - How do I publish the public PGP key for my Posteo email address in the Posteo key directory?* [Online]. Available: https://posteo.de/en/help/publishing-public-pgp-key-for-posteo-email-address (visited on 11/30/2017).

[15] *Mailbox.org - Eigenen PGP-Schlüssel verwenden*. [Online]. Available: https://support.mailbox.org/knowledge-base/article/kann-ich-eigene-pgp-schluessel-im-mailbox-org-guard-importieren (visited on 11/30/2017).

[16] *Einfache und sichere Verteilung Ihres öffentlichen PGP-Schlüssels*. [Online]. Available: https://mail.de/hilfe/nachrichten-pgp-schluessel-veroeffentlichen (visited on 11/30/2017).

[17] *Der mailbox.org HKPS-Keyserver*. [Online]. Available: https://support.mailbox.org/knowledge-base/article/der-mailbox-org-hkps-keyserver (visited on 11/30/2017).

[18] *Verschlüsselt E-Mails versenden künftig leichter*, Oct. 2017. [Online]. Available: https://mailbox.org/verschluesselung-von-e-mails-kuenftig-leichter-bedienbar/ (visited on 11/21/2017).

[19] P. Fischer, T. Kunz, K. Lorenz, and U. Waldmann, "Verfahren zur vertrauenswürdigen Verteilung von Verschlüsselungsschlüsseln," *INFORMATIK 2017*, 2017.

[20] J. Damas, M. Graff, and P. Vixie, "Extension Mechanisms for DNS (EDNS(0))," en, RFC Editor, RFC 6891. [Online]. Available: http://www.rfc-editor.org/rfc/rfc6891.txt (visited on 03/03/2018).

[21] *DNSSEC and DNS Amplification Attacks*. [Online]. Available: https://technet.microsoft.com/en-us/security/hh972393.aspx (visited on 03/30/2018).

[22] *Mailbox.org - verschlüsselte E-Mails versenden*. [Online]. Available: https://support.mailbox.org/knowledge-base/article/verschluesselte-e-mails-mit-guard-versenden (visited on 11/30/2017).

[23] *Einführung in mailbox.org Guard*. [Online]. Available: https://support.mailbox.org/knowledge-base/article/einfuehrung-in-mailbox-org-guard (visited on 11/30/2017).

[24] *Mailvelope Add-on verwenden*. [Online]. Available: https://support.mailbox.org/knowledge-base/article/mailvelope-add-on-mit-guard-verwenden (visited on 11/30/2017).

[25] *GMX - Verschlüsselte Kommunikation mit PGP im Detail*. [Online]. Available: https://www.gmx.net/mail/sicherheit/pgp/details/ (visited on 11/21/2017).

[26] *Mail.de - PGP mit Mailvelope*. [Online]. Available: https://mail.de/hilfe/nachrichten-pgp-mit-mailvelope (visited on 11/30/2017).

[27] *Mailvelope_client: Roundcube plugin to use Mailvelope's OpenPGP-support*, https://github.com/posteo/mailvelope_client, Jan. 2017. [Online]. Available: https://github.com/posteo/mailvelope_client (visited on 11/30/2017).

[28] *WebClient: Official AngularJS web client for the ProtonMail secure email service*, Nov. 2017. [Online]. Available: https://github.com/ProtonMail/WebClient (visited on 11/30/2017).

[29] *Mailpile: A free & open modern, fast email client with user-friendly encryption and privacy features*, Nov. 2017. [Online]. Available: https://github.com/mailpile/Mailpile (visited on 11/29/2017).

[30] *Autocrypt Level 1: Enabling encryption, avoiding annoyances — Autocrypt 1.0.0 documentation*. [Online]. Available: https://autocrypt.org/level1.html (visited on 03/04/2018).

[31] J. C. Klensin and R. Gellens, "Message Submission for Mail," en, RFC Editor, RFC 6409, Nov. 2011. [Online]. Available: http://www.rfc-editor.org/rfc/rfc6409.txt (visited on 03/02/2018).

[32] C. Daboo, "Use of SRV Records for Locating Email Submission/Access Services," en, RFC Editor, RFC 6186. [Online]. Available: http://www.rfc-editor.org/rfc/rfc6186.txt (visited on 01/14/2018).

[33] *Postfix Configuration Parameters*. [Online]. Available: http://postfix.cs.utah.edu/postconf.5.html#smtpd_sasl_authenticated_header (visited on 03/02/2018).

[34] *Exim documentation - chapter 33 - SMTP authentication*. [Online]. Available: https://www.exim.org/exim-html-current/doc/html/spec_html/ch-smtp_authentication.html#SECTauthparamail (visited on 03/02/2018).

[35] D. Shaw, L. Donnerhacke, R. Thayer, H. Finney, and J. Callas, "OpenPGP Message Format," en, RFC Editor, RFC 4880. [Online]. Available: http://www.rfc-editor.org/rfc/rfc4880.txt (visited on 03/03/2018).

[36] P. W. Resnick, "Internet Message Format," en, RFC Editor, RFC 5322. [Online]. Available: http://www.rfc-editor.org/rfc/rfc5322.txt (visited on 03/03/2018).

[37] J. C. Klensin, "Simple Mail Transfer Protocol," en, RFC Editor, RFC 5321. [Online]. Available: http://www.rfc-editor.org/rfc/rfc5321.txt (visited on 03/04/2018).

[38] *Postfix Configuration Parameters*. [Online]. Available: http://www.postfix.org/postconf.5.html (visited on 03/09/2018).

[39] P. V. Mockapetris, "Domain names - implementation and specification," en, RFC Editor, RFC 1035. [Online]. Available: http://www.rfc-editor.org/rfc/rfc1035.txt (visited on 03/10/2018).

[40] *Statista - Leading U.S. consumer e-mail providers 2016, by age*, en, 2018. [Online]. Available: https://www.statista.com/statistics/547531/e-mail-provider-ranking-consumer-usa-age/ (visited on 03/10/2018).

[41] T. Maier, T. Schreck, and H.-J. Hof, "Kurzvortrag: Aktuelle Umsetzung von SMTP over TLS – Ein Realitätscheck," 23. DFN-Konferenz - Sicherheit in vernetzten Systemen, 2016, [Online]. Available: https://www.dfn-cert.de/veranstaltungen/vortrage-vergangener-workshops/23Siko2016.html.

[42] D. Wessels, J. Heidemann, L. Zhu, A. Mankin, and P. Hoffman, "Specification for DNS over Transport Layer Security (TLS)," en, RFC Editor, RFC 7858. [Online]. Available: http://www.rfc-editor.org/rfc/rfc7858.txt (visited on 03/11/2018).

[43] *DNS Privacy Implementation Status*. [Online]. Available: https://dnsprivacy.org/wiki/display/DP/DNS+Privacy+Implementation+Status (visited on 03/11/2018).

[44] M. Xie, H. Yin, and H. Wang, "An effective defense against email spam laundering," in *Proceedings of the 13th ACM Conference on Computer and Communications Security*, ser. CCS '06, New York, NY, USA: ACM, 2006, pp. 179–190, ISBN: 978-1-59593-518-2. DOI: 10.1145/1180405.1180428. [Online]. Available: http://doi.acm.org/10.1145/1180405.1180428 (visited on 03/31/2018).

[45] A. Ramachandran and N. Feamster, "Understanding the network-level behavior of spammers," in *Proceedings of the 2006 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, ser. SIGCOMM '06, New York, NY, USA: ACM, 2006, pp. 291–302, ISBN: 978-1-59593-308-9. DOI: 10.1145/1159913.1159947. [Online]. Available: http://doi.acm.org/10.1145/1159913.1159947 (visited on 03/31/2018).