Subscription Email

Interim report

Student	David Lowe
Module	CM0343
Credits	40
Year	2012/2013
Project supervisor	Dr. Frank Langbein
Moderator	Prof. Ralph Martin

Abstract

Email is one of the most widely used applications of the Internet, but it suffers from two flaws in particular: insecurity and unsolicited bulk email. Various other online mail systems have been developed and adopted, but none of them provide authentication and confidentiality, while eliminating spam and keeping email's decentralised nature. In this report, I examine the functionality offered by email, Facebook private messaging, Diaspora private messaging and Stubmail. I then propose an alternative: PigeonMail. Inspired by Stubmail and its related protocols, it is a pull-based protocol, that is, the burden of storing messages lies on the sender, not the recipient. To eliminate spam, recipients only receive messages from senders they have selected and approved. In addition, the protocol mandates that all messages be signed and encrypted to provide a base for confidentiality and authentication. Following this, I explain my approach in designing the protocol, and I end by summarising the progress made in implementing an initial prototype.

Table of Contents

Abstract....2
Introduction....4
Background....5
1 Email....5
2 Facebook private messaging....7
3 Diaspora private messaging....9
4 StubMail....10
Proposal....15
Requirements for PigeonMail....19
Protocol design....21
1 Name....21
2 Sending a message....21
3 Location distribution....21
4 Public key and verification key distribution....22
5 User identifiers....22
6 User identifier distribution....23
7 User identifier authentication and certificate distribution....23
8 Contact recommendation....24
9 Conversations....24
10 Network architecture....25
11 Extensibility....25
Initial prototype....27
Glossary....28
References....29

> *Note: Throughout this report, when referring to a generic human being, the third feminine singular person is used. It should be understood by the reader that the gender of the person referred to is intended to be neutral.*

Introduction

Electronic messaging has a vast history on the Internet. Indeed, one could consider all communication on the Internet to be electronic messaging. This project focuses on one particular category of electronic messaging: the online and asynchronous exchange of textual messages between two or more humans, intended for private view. For brevity, this type of electronic messaging shall be referred to as *online mail systems*. Email fits in this category, but instant messaging, would not, nor would Internet message boards.

Existing online mail systems serve the needs of users well enough to be widely deployed and utilised. However, all popular online mail systems suffer from at least one of these two flaws: insecurity and spam. Email, for example, suffers from both of these problems. The ratio of legitimate messages over spam messages is extremely low, and messages are sent in plain-text unauthenticated over the network. These flaws are discussed in detail in the background section of this report.

Bernstein (2000) proposed a protocol named Internet Mail 2000. Its distinguishing and core tenet is that burden of storing a message is laid on the sender, not the recipient. Recipients request and fetch messages from the sender as frequently as they wish. The advantages of this approach, as listed by Bernstein, are the elimination of bounce messages, the elimination of complicated mailing list software, and the reduction of the total amount of disk space used across mail servers. Bernstein never completed the specification of Internet Mail 2000, but his work proved inspiring to others, such Brett Keith Watson.

Watson (2002, section 2) described some advantages of a pull-based approach not listed by Bernstein. To begin with, a pull-based approach has the advantage of requiring at least a weak form of identity from the sender, unlike SMTP, which allows anonymous sending. It also implies consensual communication, as both recipient and sender would have to co-operate to communicate. Consensual communication of course rules out spam by definition. He proposed (2002, section 4) a new protocol named Message Transfer Protocol that enable recipients to notify recipients of new messages. A recipient would then use POP to download the new messages from the sender's server. Watson's approach was evolutionary, not revolutionary, that is, he did not seek to replace SMTP with a completely incompatible protocol, but to add a new protocol to it, hoping that eventually SMTP would be phased out of use.

Influenced by Bernstein's and Watson's work, Haight and Wong (2006) proposed another pull-based protocol named Stubmail. The focus of that project was to transfer the cost of storing large volumes of spam from the recipient to the sender. In addition, it was hoped that the increased cost would prove to be a deterrent to spammers. They developed a prototype in Perl, but no further work has been done since. Like Watson's proposal, Stubmail took an evolutionary approach, keeping compatibility with existing email systems a priority. In the background section, I will examine Stubmail, choosing it as a representative of all the family of proposed pull-based email protocols.

Following the background section, I will propose a new protocol named PigeonMail. Like Internet Mail 2000, MTP and Stubmail, it takes the pull-based approach, but it is deliberately designed to be incompatible with email. Because all communication would be consensual, this would allow PigeonMail to defeat spam. In addition, PigeonMail would mandate the use of end-to-end encryption to provide a base for authenticated and confidential communication.

In the final section of this interim report, I discuss the progress I have made with an initial protype, and the lessons learnt from the early implementation.

Background

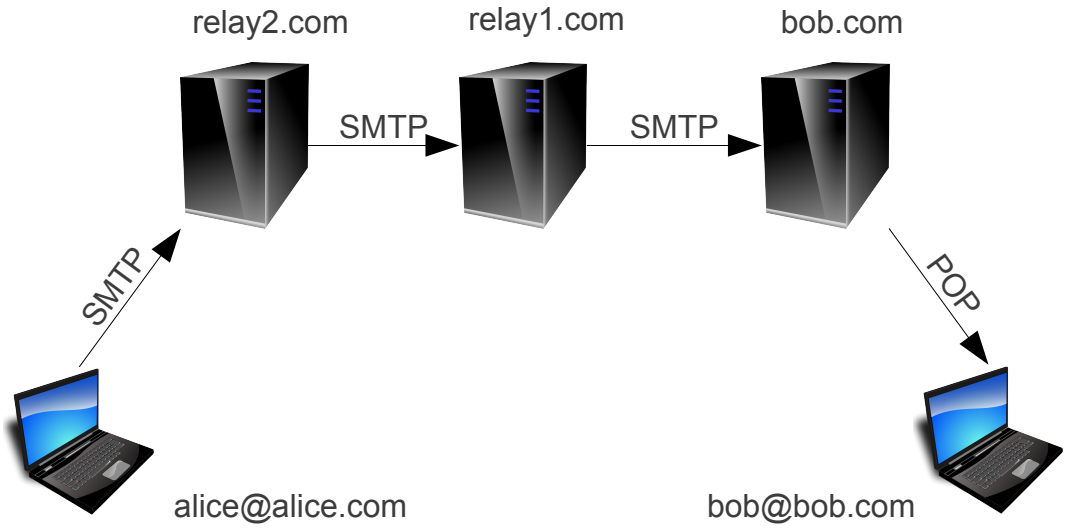
In this section, I shall evaluate various existing and proposed online electronic messaging systems. I shall use these criterion:

- Reliability
- Security
- Message relevance
- Popularity and barrier to entry
- Trust requirements and vendor lock-in

1 Email

Email has a long history, longer than that of the world wide web. Because of its age and its popularity, several different implementations and protocols have been built around it. Its evolution is a complex topic, and is not one I will address here.

As things stand today, all the different email servers and clients have one thing in common: their support of the SMTP protocol. Every mail transfer agent uses SMTP to send and receive electronic messages. Each message must contain at least three pieces of information: the email address of the intended recipient, the email address of the sender, and the message content. Email addresses are used to identify and locate a recipient. Each address must be composed of two parts: a user identifier, and a domain name, separated by the @ symbol (Resnick 2008).



Drawing 1: Sending an email with SMTP

Typically, the recipient uses an email client to connect to her email server over a protocol such as POP or IMAP. Her email server acts as a buffer. It is expected to be online permanently, while the user's email client is not.

If the recipient wishes to reply to the message, she can send a message by SMTP to the address specified in the electronic message. She has no guarantees that the message is indeed from the user that owns that email address, though.

So how does email evaluate?

Reliability

SMTP is a best-effort protocol. Each jump from relay to relay is not guaranteed to succeed. The DSN extension to SMTP allows relays to communicate back errors to the original sender (Kucherawy 2012). However, since one cannot be sure whether the relay servers implement this extension, one cannot be certain that the message has been delivered to the recipient, or even to the recipient's mail server.

Security

Upon receiving the message, the recipient cannot be certain that the message arrived from the claimed sender. Nor can she be certain that the message was not viewed by a third party: any one of the relays could have viewed and tampered with the message, or anybody with access to the underlying network. Similarly, the sender cannot be certain that the message will be confidential and read only by the intended recipient.

To combat these glaring security flaws, various extensions to SMTP have been proposed, and implemented to various degrees:

- SSL may protect the confidentiality of communication between one relay and another, providing authentication for the machines.
- The use of open relays as illustrated in the diagram is more and more rare. In practise, the initial relay will require authentication credentials from the user, such as a user name and password, and only allow certain users to use the relay. For example, Gmail requires that all senders register for a Google account and provide a user name and password upon sending a message.
- DomainKeys Identified Mail (DKIM) uses cryptographic proof that a received email came from the domain name claimed in the From header.
- A user may choose to digitally sign her message with a technology such as OpenPGP or S/MIME. The recipient must opt in to using the technology to benefit from the authentication verification this provides.
- A user may choose to encrypt her message with a technology such as OpenPGP or S/MIME. This requires that the recipient previously opted in to the technology, distributing a public key to the user in one way or another.

Message relevance

Email is well known for the large volumes of unwanted and indiscriminate messages it delivers. These messages are referred to as spam.

Spam is not in itself a security flaw, since it does not necessarily negate confidentiality and authenticity of the communication.

Email was designed to allow for anonymous delivery of messages in bulk. Spam exists in such volume, because a small proportion of its users desire to use email to advertise indiscriminatingly. The designers of SMTP did not anticipate that the anonymous features of the protocol would be abused to such a degree.

It is possible to disable this characteristic of SMTP by white-listing allowed senders. However, this is rarely done for the following reasons:

- Senders expect to be able to send messages without requesting approval beforehand. If their message is not delivered, they may not attempt to contact the recipient by other means, to

ensure that they have been approved.

- The identity of the sender of the message often cannot be authenticated.
- Other spam reducing technologies filter out enough spam messages without requiring creating a white-list.

Popular spam reducing techniques include statistical content filtering and black-lists. They are all only partially successful.

Popularity and barrier to entry

Email is, without doubt, the most widespread online mail system. Thanks to the network effect, this makes email highly attractive to existing and new users.

Email's barrier to entry is low. Creating an email account is easy: ISPs often provide one to the user, and there are many respected free email providers as well. Typically, all that is required to register is a name, an address, and a minimal proof that the user is human, rather than a computer.

In addition, beginning a conversation with someone is easy is well. All that is required is the email address of the intended recipient. Email addresses are memorable and simple to distribute over the phone, in person, or online.

Trust requirements and vendor lock-in

A user of email is required to trust her mail provider with the contents of her outgoing and incoming email, to trust the recipient's mail provider with the contents of her outgoing email, and to trust any other parties on the network.

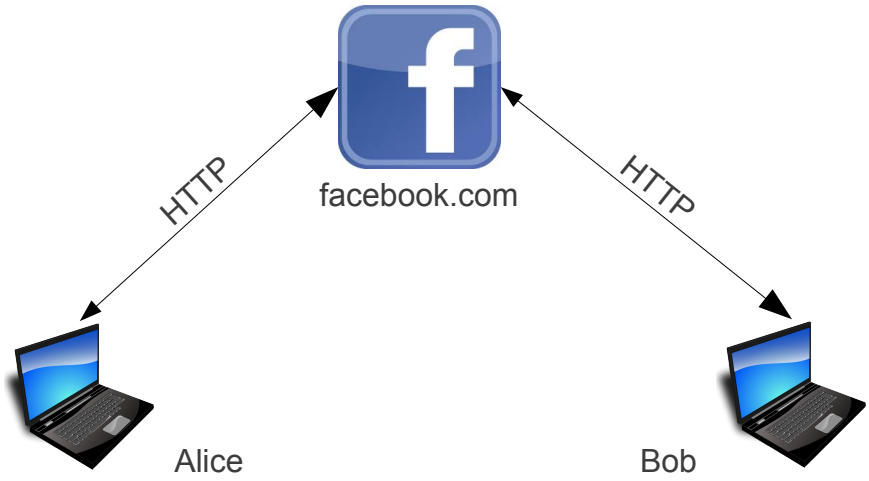
A user may change vendors, but will be obligated to inform all her contacts of her change of email address.

2 Facebook private messaging

Facebook is a social network on the web, with an estimated one billion users. Its popularity make it an unavoidable candidate for consideration. It provides many communication tools, but only its private messaging functionality concerns us.

Its network architecture is simple: users log in to `facebook.com` in their web browser, and leave messages to each other. The details of how the server handles messages internally are not relevant.

Facebook also provides a compatibility layer with SMTP. However, it is partial and relatively new,



Drawing 2: Sending and receiving messages on Facebook

so it will not be considered part of the online mail system examined.

Reliability

All users of the system must rely on Facebook at all times even to view their messages.

Security

Users must opt in to using HTTPS. Because of this, there is no assurance that messages are kept confidential from attackers on the network. As for authenticity, Facebook acts as the verifier that messages originate from the specified Facebook account. It is up to the user to verify that the Facebook account is indeed associated with the person claimed to own that account.

Message relevance

Because Facebook can view, tamper with and delete all of the private messages sent by users, there are in a powerful position to stop spam. They can gather very accurate statistics to filter out likely spam messages. In addition, users can opt to block messages from unapproved Facebook accounts if they wish.

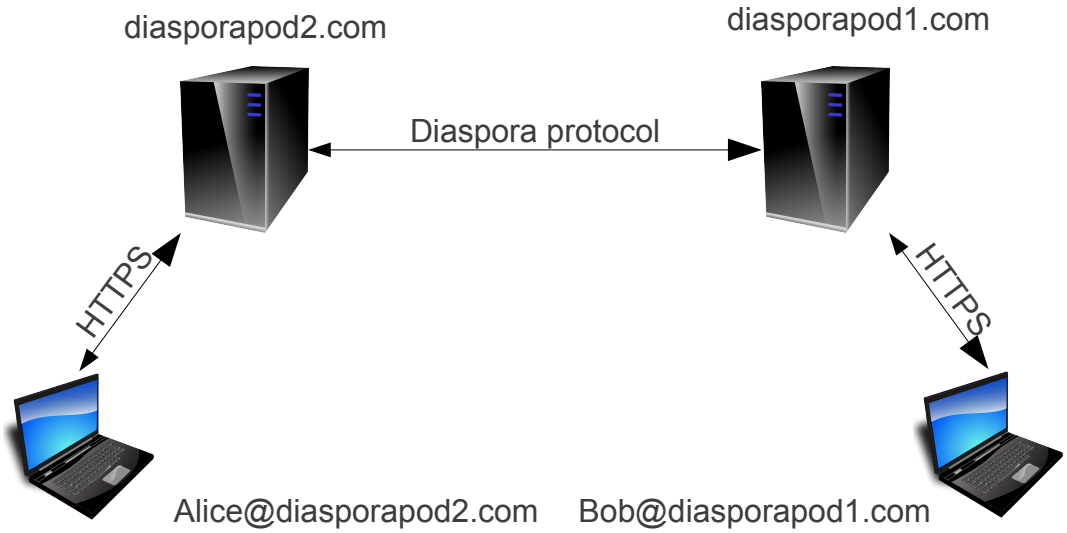
Popularity and barrier to entry

It is estimated that over one billion people use Facebook actively. With the exception of some countries, it is almost guaranteed that a sizeable number of anyone's acquaintances possess a Facebook account. Facebook have worked very hard to ensure that a new user's barrier to entry is very low. After sign-up, Facebook can use information that the user enters such as location and home town to find user accounts that are likely to be real-word acquaintances. Facebook requires users to share their list of friends to all friends, enabling easier discovery of friends.

Trust requirements and vendor lock-in

All users are required to trust Facebook completely with the contents of their messages. They cannot change vendor.

3 Diaspora private messaging



Drawing 3: Sending and receiving messages with Diaspora

Driven by the increasing awareness of the privacy risks Facebook presents, Diaspora was born to develop a web-based decentralised social network. Similarly to Facebook, it provides users with the possibility of sharing status updates, photos and private messages with other users. Unlike Facebook, there is not a central website, rather, users must choose a server (or a "pod") to host their data. Pods regularly exchange relevant data so as to maintain one global social network, rather than many independent networks.

Security

Diaspora requires SSL in the browser, and between pods. This prevents attackers on the network from viewing or forging Diaspora private messages. Pods are authenticated using well-known certificate authorities, but users are not authenticated.

Message relevance

Because Diaspora is still relatively young and unpopular, it has not attracted much attention from spammers. Every pod can view and modify all data it receives from its users and from connected users in other pods. It theoretically could use this information to perform statistical analysis of messages, to prevent spam. In the currently released version of Diaspora, users cannot block messages from all unapproved contacts.

Popularity and barrier to entry

Compared to Facebook, Diaspora is unknown. Its barrier to entry is higher than Facebook, because users do not publish their list of contacts, not even to their "friends".

Trust requirements and vendor lock-in

A Diaspora user must trust the pod with all of her data, and must trust her contacts' pods with all of the data that she has shared with those pods. She may change pods if she wishes, but she must her identifier changes, meaning that she must re-establish her social network from scratch.

4 StubMail

StubMail is an online mail system inspired by Internet Mail 2000. It was designed by Meng Weng Wong and Julian Haight (2006), two engineers with experience in the field of filtering spam. An open source prototype was released in 2006.

Stubmail is the newest addition to the family of theoretical pull-based online mail systems such as Internet Mail 2000, Weemail and HTMP. All of these systems attempt to improve email by focussing on one idea: the burden of storing the message should be on the sender, not the recipient. The hope is that this would increase the cost of unsolicited bulk email, and thus decrease spam. It would also force senders to identify themselves, another hindrance to spam. Since none of these systems have been widely deployed, the efficacy of these proposals remain untested. Because Stubmail is the most recent of these pull-based online mail systems, it was chosen to be representative of all of them.

The protocol for sending and receiving messages between two Stubmail supported users is discussed on separate pages 11 and 12, due to the large diagrams. The reader is directed to read those pages before continuing.

Stubmail assumes each email mail server points to an OpenPGP public key server. If at any point, a user's OpenPGP public key is missing from the client's machine, the client software will fetch it automatically from the public key server. Validation of the public keys is left to the user.

If either the sender or the recipient's email server does not announce Stubmail support by omitting a SRV `stub` record, Stubmail will simply fall back to SMTP, and it will not encrypt any outgoing messages using GnuPG. This means that Stubmail support can be gently introduced one email server at a time, without breaking compatibility with other email servers. However, the benefits of Stubmail are only realised once both recipient and sender enable Stubmail. Because of this, there is little reward for early adopters: they will still suffer from the flaws of email.

Stubmail provides a compatibility layer for email clients, such as Thunderbird. The executable `smtp2stub` distributed with the open source prototype converts the email client's SMTP messages into HTTP post requests conforming to Stubmail's protocol. The executable `http2mbox2` fetches messages from senders' Stubmail servers over HTTP and stores them in the user's UNIX spool directory, a scheme compatible with many email clients such as Thunderbind and Evolution. To adopt Stubmail, a user only has to install this compatibility layer and publish her OpenPGP public key, assuming her email provider is offering Stubmail support. She can continue to use the email address and the email client she is accustomed to.

Sending a message with Stubmail:

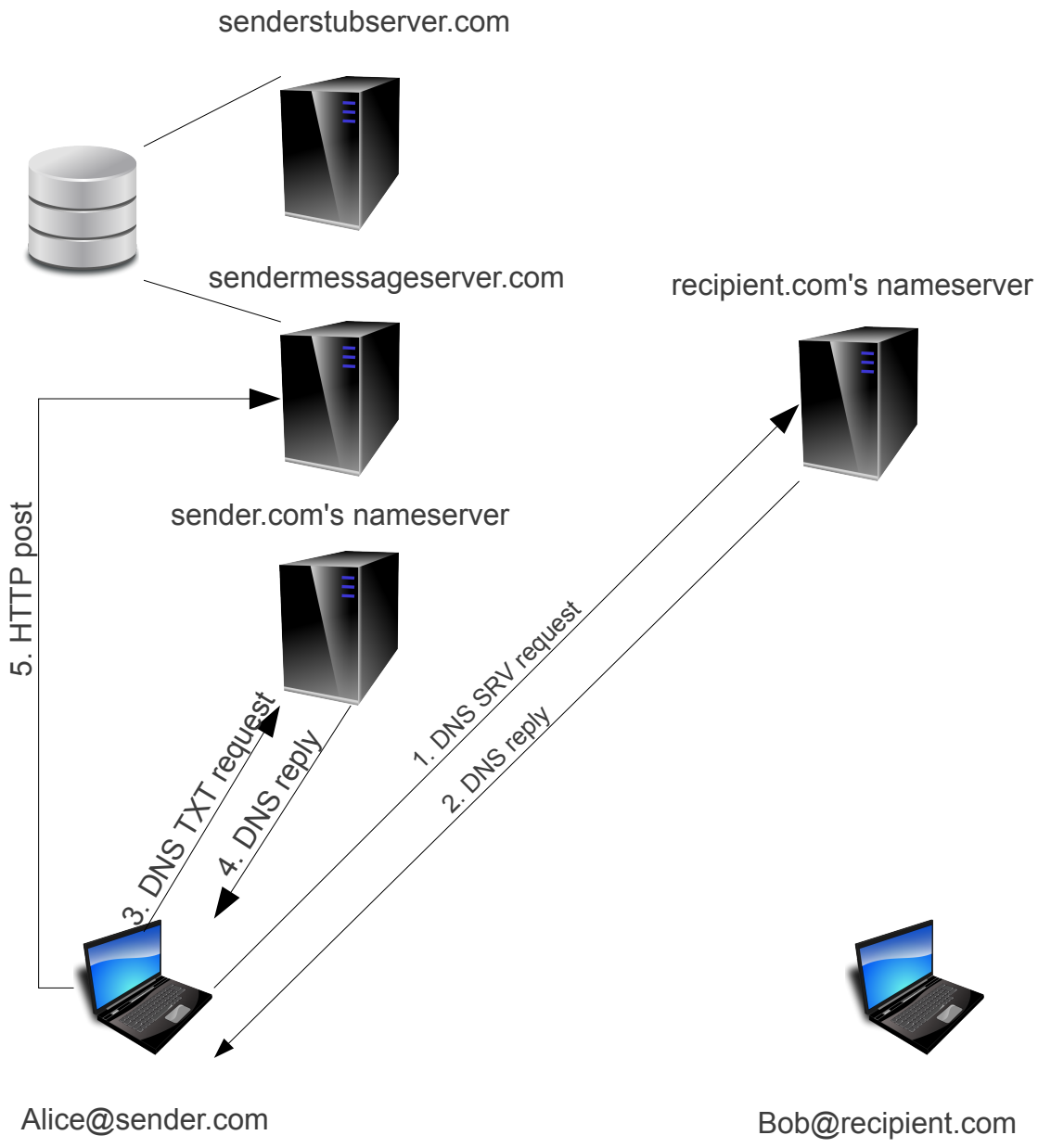
1. The sender (Alice) identifies the domain name of the email server of the recipient (Bob), from his email address. She sends a DNS query to the name server of the recipient's email server, requesting the SRV record for the `stub` service over UDP (`_stub._udp.recipient.com`). The purpose of this query is to verify that the recipient's server supports StubMail, otherwise, SMTP is used as a fallback.

2. The name server replies with the appropriate domain name and port, in this case, `recipientstubserver.com` over port 2225, thus confirming that the recipient supports Stubmail.

3. Alice sends a DNS query to her email server's mail server, requesting the TXT record for the `post` service over TCP (`_post._tcp.sender.com`).

4. The name server replies with the appropriate domain name and port, in this case, `sendermessageserver.com` over port 80, as well as the post URL, in this case, `stubmail/post_manager.cgi`.

5. The sender signs the message using her private signature key, and encrypts the message using the recipient's public key, using GnuPG. She then posts the message over HTTP to `http://sendermessageserver.com/stubmail/post_manager.cgi`.



Receiving a message with Stubmail:

1. The recipient (Bob) sends two DNS queries to the name server of the sender's email server, identified from her email address. He requests both the SRV record for the `stub` service, and the TXT record for the `post` service.

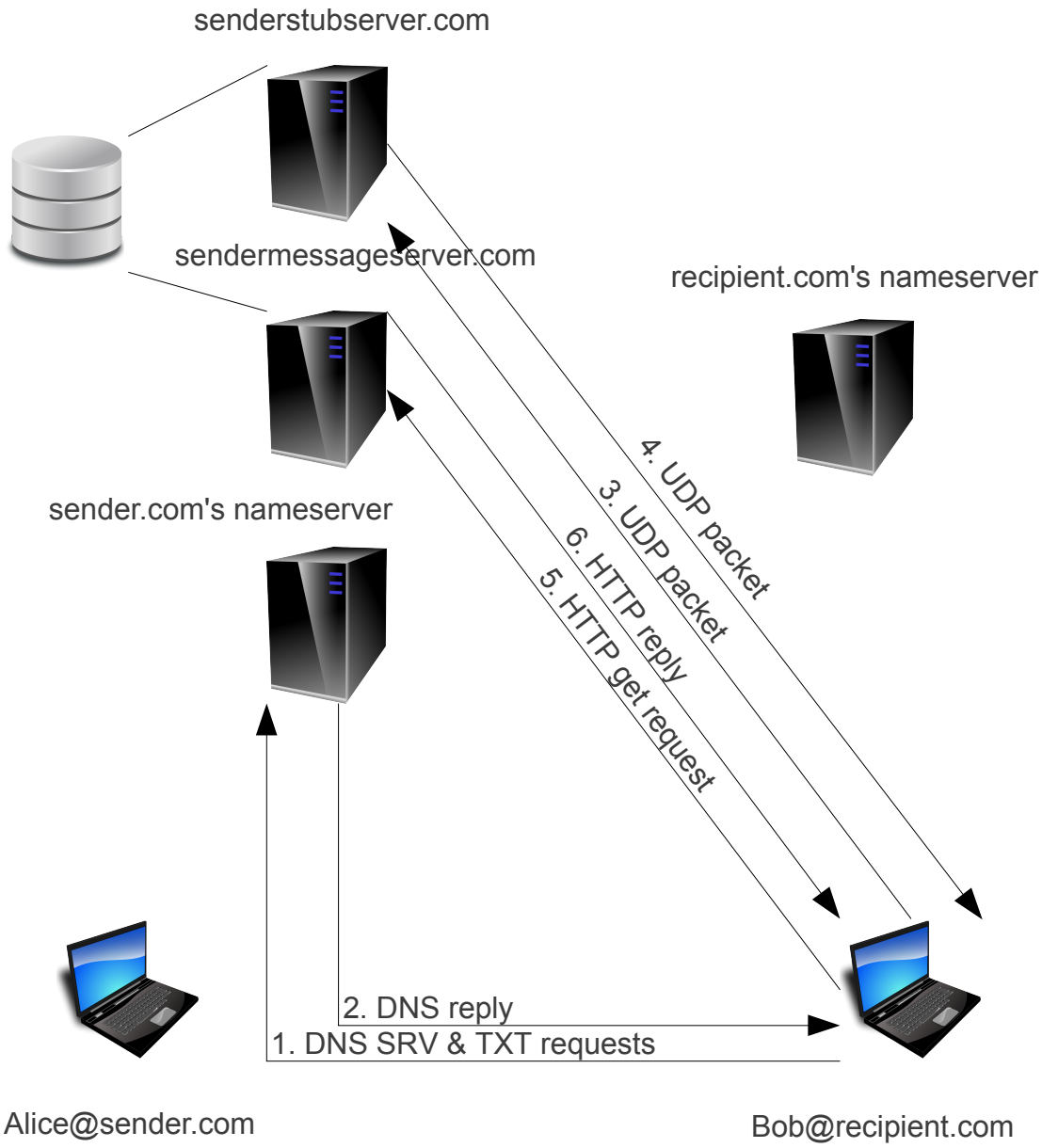
2. The sender's name server replies with the domain name and port of the `stub` service, in this case `senderstubserver.com` over port 2225, and with the domain name, port and URL of the `post` service, in this case, `senderstubserver.com` over port 80 with the URL `stubmail/post_manager.cgi`.

3. The recipient then sends a UDP packet to `senderstubserver.com` with a cookie value. The stub server can use this cookie value to quickly determine whether it has any new messages for the recipient.

4. The stub server sends a UDP packet back to the recipient confirming the presence of new messages and setting a new cookie value.

5. The recipient sends an HTTP request to the sender's message server, the parameters of which are signed with the recipient's private key using GPG.

6. The sender's message server verifies the authenticity of the request using GPG, fetches the message and optionally saves the used cookie, sharing it with the `stub` server.



Reliability

For a Stubmail enabled communication, the burden of storing and delivering the message rests on the sender's server. It must keep each message until the recipient first retrieves it, at minimum. If the sender's server loses the message, or loses connectivity, the message is not delivered. The recipient does not need to send back a bounce message, because the sender's server should be able to track which messages have been retrieved and which messages have not.

Naturally, if the sender's server does not support Stubmail, the communication falls back to normal SMTP, and SMTP's previously discussed reliability assurances apply.

Security

All Stubmail messages are encrypted and signed using OpenPGP keys. If a user can be confident that the keys used belong to their claimed owners, OpenPGP provides assurances of the confidentiality and the authenticity of the messages. In practise, many users do not understand the importance of verifying OpenPGP keys, and email clients often fail at making that task as easy and comprehensible as possible.

If either the recipient or the sender does not support Stubmail, the implementation will fall back to SMTP, without encrypting outgoing messages. In this case, all the previously discussed problems with email's security apply.

While Stubmail does not use SSL to protect its HTTP communications, all of the email messages passed through HTTP are signed and encrypted using OpenPGP, making this less of a concern.

Message relevance

Once a recipient has enabled Stubmail and disabled all POP and IMAP interfaces, the recipient can be guaranteed to only receive email from senders in her GnuPG keyring. This would eliminate unsolicited bulk email.

In practise, users are unlikely to disable POP and IMAP, as they have the expectation that unapproved receiving should be supported. All the message relevance issues of normal email would apply.

Popularity and barrier to entry

Stubmail is only a prototype and is unlikely to be widely deployed in its current form. In theory, email providers could begin supporting Stubmail, allowing users to opt in. Once enough email providers and users have opted in, the benefits of Stubmail concerning security and message relevance are actualised. Because the benefits of Stubmail do not apply to early adopters, it may be prove extremely difficult to convince enough early email providers and users to switch early on, making the protocol irrelevant and unpopular.

For users, the barrier to entry is low. They can continue to use their existing email client, to correspond with their existing contacts and to keep their existing email address. However, while Stubmail remains unpopular, there may be little benefit to adopting it, making the barrier to entry low but unrewarding.

Trust requirements and vendor lock-in

The trust requirements characteristics of Stubmail are similar to that of email. The only reduction in

required trust is that the messages are signed and encrypted, denying email providers and any attackers the opportunity to read or forge any emails. This benefit applies only once all compatibility layers with SMTP have been disabled, if not, an email provider or an attacker could simply disable Stubmail and force the user to return to plain email.

As with email, a user may change vendors, but will be obligated to inform all her contacts of her change of email address.

Proposal

The goal of this project is to develop an alternative online mail system that does not suffer from the two glaring flaws of the email, which are insecurity and spam, but that retains some of its benefits, such as decentralisation and no vendor lock-in. To achieve this goal, a trade-off is made: unapproved sending is disallowed, and compatibility with email is ruled out.

The proposed protocol, PigeonMail, would in essence be very similar in architecture to Stubmail and other pull-based theoretical online mail systems. The crucial difference between PigeonMail and Stubmail is that no attempt is made to provide compatibility with email. Compatibility with the SMTP protocol engenders the drawbacks of SMTP that are meant to be avoided in the first place. In addition, no compatibility with existing email clients is attempted. This is to make it painfully obvious to the user that PigeonMail is not compatible with email, and that unapproved sending is not supported. The goal of any technological system should be to provide the user with a pleasant experience, and false expectations must be eliminated to avoid disappointment.

Reliability

The reliability of PigeonMail should be equivalent to the reliability of Stubmail without its compatibility layer. The burden of storing sent messages rests on the sender's server, and no bounce messages are needed because the sending server can record when the recipient has fetched the message.

Security

The security of PigeonMail should be equivalent to the security of Stubmail without its compatibility layer. A public-key cryptographic protocol like OpenPGP should be used to sign and encrypt all messages between users, providing end-to-end encryption. A user who verifies all the keys used should be confident that all ingoing and outgoing messages are confidential and authenticated.

Message relevance

The relevance of the messages received by PigeonMail users should be equivalent to that of Stubmail, disregarding its compatibility interface. Users can only receive messages from senders they have selected. This white-list should eliminate all unsolicited bulk email. The cost of this approach is that of creating a “closed community” (Waston, 2004), often an undesirable consequence. To minimise this cost, efforts should be made to make expanding the closed community as easy and inviting as possible.

Popularity and barrier to entry

Because PigeonMail provides no compatibility with email, it cannot benefit directly from email's popularity. PigeonMail would begin with low popularity, and hence limited use. Some small communities with significant security concerns may prove to be the early adopters. However, the goal of this project is not to achieve popularity, but to design an online mail system that achieves security and blocks spam without compromise.

The barrier to entry is the highest of all the online mail systems previously discussed, because of the lack of support for unapproved sending. A user must approve specific senders before PigeonMail is of any use, which may prove difficult without receiving recommendations from at least one other

PigeonMail user.

Trust requirements and vendor lock-in

The trust requirements should be greatly reduced compared to other online mail systems: the user only has to trust that their online mail provider will store and serve their outgoing messages reliably. Because of the security requirements, the user should not have to trust any third party with the content of her messages.

Like email, there should not be any vendor lock-in with PigeonMail. A user should be free to switch online mail providers with as little inconvenience as possible. Indeed, a user should be able to switch vendors without even being required to inform her contacts of her migration.

Comparison of online mail systems

The following table summarises the differences between the existing online mail systems, and the proposed PigeonMail protocol. The requirements and design choices for PigeonMail will be discussed in the upcoming sections.

Criteria	Email	Facebook	Diaspora	Stubmail	Proposed PigeonMail
Reliability	Best-effort	Eventual consistency	Eventual consistency	Good	Good
Security	Plain-text, non-authenticated with optional SSL or OpenPGP	Plain-text, non-authenticated with optional HTTPS	HTTPS	OpenPGP, but the compatibility layer does not require OpenPGP	OpenPGP
Message relevance	Spam prevalent	Little spam	Spam prevalent potentially	Spam prevalent until compatibility layers are disabled	Non-existent
Popularity and barrier to entry	Popular and very low barrier to entry	Popular and low barrier to entry	Unpopular and medium barrier to entry	Unpopular and low barrier to entry	Unpopular and high barrier to entry
Trust requirements	Senders must trust their host and the recipient's host and the network	Senders must trust Facebook and possibly the underlying network	A sender must trust their pod and the recipient's pod	Senders can rely on OpenPGP's guarantees of confidentiality and authenticity when used	A sender can rely on OpenPGP's guarantees of confidentiality and authenticity
Vendor lock-in	Can choose between competing email providers. Email address is tied to a particular provider	Locked in to Facebook	Can choose between competing pods. Identifier is tied to a particular pod.	Can choose between competing providers. Identifier is tied to a particular provider.	Can choose between competing providers. Identifier is not tied to a provider, enabling easy migration.
Unit of communication	Message	Conversation	Conversation	Message	Conversation

Criteria	Email	Facebook	Diaspora	Stubmail	Proposed PigeonMail
Anonymous sending	Allowed	Messages are always tied to a Facebook account	Messages are always tied to a Diaspora account	Allowed in order to be compatible with email	Messages are always tied to an OpenPGP key
Unapproved sending	Allowed. White-listing may be implemented but it is rare.	Allowed, although users commonly block messages from "non-friends".	Allowed.	Allowed in order to be compatible with email	Impossible
Identifier	username@hostname	User number or user URL	username@poddomain name	username@hostname	OpenPGP key fingerprint
OpenPGP key distribution	Manual or using key servers	N/A	N/A[1]	Key server	Each PigeonMail provider would also act as a key server
Data loss and recovery	Many providers automatically backup emails	Behind the scenes back-ups that are not necessarily available to the user	Some pods may provide backups	Theoretically possible, as long as the OpenPGP private key is not permanently lost	Theoretically possible, as long as the OpenPGP private key is not permanently lost
Identity loss and recovery	Providers may provide a new password after verification over the phone or by post	Facebook may provide a new password after verification through another means	Unknown	Losing an OpenPGP private key means permanent identity loss	Losing an OpenPGP private key means permanent identity loss

1 OpenPGP is used between pods, but not for end-to-end encryption

Requirements for PigeonMail

The set of requirements for PigeonMail is not a superset of the functionality of email, rather it intersects with it.

Similarly to email, the design of PigeonMail must allow for these requirements:

- It must meet the criteria of an online mail system. That is, it must allow the online and asynchronous exchange of messages between two or more humans, intended for private view.
- It must allow users to choose between competing vendors or to host their own mail server.
- Competing vendors and multiple hosts must be able to communicate with each other. This is what is meant by decentralisation.
- It must allow for binary data to be exchanged, as well as textual information.
- It must be open to improvements, evolution and extensions.
- It must be documented well enough and simply enough to encourage the development of multiple implementations.

In addition, PigeonMail should meet the following requirements, that email does not guarantee to meet:

- It must require end-to-end encryption. That is, every message must be encrypted on a machine the sender trusts and owns, and decrypted on a machine the recipient trusts and owns. No other machine or person should be able to view the contents of messages.
- It must support message author authentication: every message must be cryptographically proven to originate from someone holding the sender's signature key.
- All messages that are received should be relevant to the user. If a user has approved a sender, it is acceptable to consider all messages from that particular sender to be relevant, since the recipient may remove the sender from her approved list at any time.
- For usability reasons, PigeonMail should be built around sequences of messages, or conversations, rather than individual messages.

It is important to iterate that PigeonMail does not aim to provide every piece of functionality that email provides. In particular:

- PigeonMail must not allow for anonymity, every message must be tied to its author, even if the author never divulges her real name.
- PigeonMail must not allow unapproved sending, that is, senders cannot send messages to recipients without being explicitly approved beforehand. This may mean that some relevant messages cannot be delivered, however, it is the only way to guarantee the previous requirement that all received messages are relevant. In other words, it is the only way to completely eliminate spam.
- It must not be compatible with email. This means PigeonMail cannot directly benefit from email's popularity, however, it is extremely difficult to meet the previous requirements while staying compatible with email. Furthermore, compatibility with email engenders false user expectations, especially the false expectation that it should allow unapproved sending.

In order to further clarify the goals of PigeonMail, these non-requirements are mentioned:

- PigeonMail is not required to defeat traffic analysis. Technologies such as Tor may be used instead.
- PigeonMail is not required to provide instant messaging, only asynchronous messaging.
- PigeonMail is not required to provide data and identity recovery, but it should allow another system to fulfil this role.
- PigeonMail is not required to provide synchronisation between multiple devices and clients, and it should allow another system to fulfil this role.
- PigeonMail is not required to be more efficient or scalable than email, as this project is focussing exclusively on the problems of insecurity and spam, not performance.

Protocol design

1 Name

A name is a small but important detail of any design. The name of the protocol should naturally indicate that the protocol is an online mail system, while emphasising the differences between it and SMTP email, particularly the fact that unapproved sending is impossible.

Email takes its metaphor from the postal mail service. Similarly to the post, a sender simply creates a message, addresses it to the recipient, and submits it. This metaphor is unsuitable for the proposed protocol, because unapproved sending is not allowed. Rather, the recipient fetches messages from the sender.

Carrier pigeon messaging is a better metaphor for the proposed protocol. In order to begin receiving messages from a sender, a recipient must first provider her with homing pigeons, ready to return to the recipient. The sender can then use the homing pigeons to send messages to the recipient. It is of course impossible to send messages to uncooperative recipients.

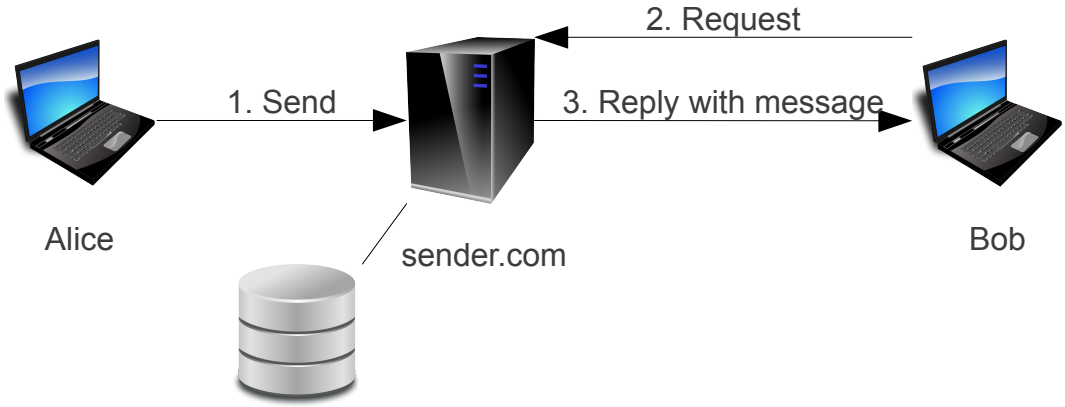
This imperfect analogy is a much better fit for the new protocol, inspiring the name: PigeonMail.

2 Sending a message

To send a message, Alice creates the message, signs it using her signature key and encrypts it using Bob's public key. She then uploads it to her PigeonMail server.

When Bob needs to receive new messages from Alice, he sends a request to her server with credentials proving his identity. The server will then return all messages for Bob from Alice. Bob can then use his private key to decrypt the message, and Alice's public verification key to verify the authenticity of the message.

For this communication to happen, Alice must know Bob's public key, and Bob must know Alice's verification key and the location of her server. The distribution of this knowledge is discussed below.



Drawing 4: Sending and receiving a message with PigeonMail

3 Location distribution

To make sure that Bob has the ability to receive her messages as soon as he may need them, Alice must somehow let Bob know the location of her PigeonMail server. Furthermore, Bob must be informed should the location of that server change.

Alice's first option is to communicate this information to Bob out-of-band. For example, she may

tell Bob this information in person. She will need to update Bob out-of-band as well. This option is very inconvenient.

A better option is to use an online public directory. Alice creates a location record with the IP address or the domain name of her PigeonMail server, a creation date and an expiry date. She then signs it with her signature key, and uploads it to the centralised public directory. Bob will then request the location record from the centralised directory, and will verify its authenticity using Alice's verification key. Bob now knows the location of Alice's PigeonMail server, and can request a new location record from the centralised server should the location record's expiry date pass, or should the server fail to respond to Bob's requests.

When Alice chooses to migrate from one vendor to another, she can simply publish a new signed location record, with a more recent creation date. She can also leave a pointer at her old PigeonMail server, locating the new location of her PigeonMail server.

To avoid centralisation, the role of public directory could be fulfilled by a peer-to-peer distributed hash table.

4 Public key and verification key distribution

Once Bob has identified the location Alice's PigeonMail server, he can request from that server Alice's public key and verification key. An OpenPGP public key contains both of these keys. (The verification key is known as the master key and the encryption public key is known as a subkey.) This does not provide Bob with assurance that the returned OpenPGP public key is in fact Alice's key. However, due to the fact that OpenPGP fingerprints are immune to preimage attacks, Bob can be confident that the returned verification key is unique to the fingerprint ID. Every subkey, every user ID and every photo ID included in the OpenPGP is self-signed, so Bob can be confident of the association between the master verification key and these pieces of data.

5 User identifiers

A user identifier for Email consists of a user name, followed by the @ symbol, followed by the user's mail provider's domain name. The advantages of this schema of identifiers are clear: an email address is easily communicated and memorised by humans, and it establishes the location of the user's mail server.

In contrast, a PigeonMail user identifier consists of a fingerprint of the user's verification key. It is frequently transcribed as a sequence of 40 hexadecimal digits. Unlike email addresses, it does not easily memorised by humans, nor does it establish the location of the user's mail server. This is an example of a OpenPGP key fingerprint:

05E7 A7D1 A5DC 08D6 4835 B111 AF27 6528 5ADB DC4A

Note that the fingerprint of an OpenPGP key is exactly the same as the fingerprint of the main verification key it contains. Encryption subkeys can be added and removed at any time without changing the fingerprint of the OpenPGP key.

A PigeonMail user's identifier does not include the location of the user's PigeonMail server. This is to meet the requirement of allowing the user to easily migrate between PigeonMail servers without losing her identity.

According to Zooko's triangle (Wilcox-O'Hearn 2006), names cannot be securely unique, global and memorable at the same time. Any naming scheme can only provide three of those properties at once. Email addresses are global and memorable, but they are not securely unique: they are

susceptible to mimicry attacks. For example, although a computer can easily distinguish between `alice@yahoo.com` and `a1ice@yahoo.com`, a human may not. This situation is made worse by the introduction of visually similar Unicode characters, as in this example: `alice@yahoo.com` and `alice@yahoo.com` (the former begins with U+0430, while the latter begins with U+0061).

PigeonMail user identifiers are securely unique and global, but deliberately not memorable. In order to ensure that PigeonMail user identifiers are handled correctly, they must be difficult to remember, so that a human will always opt to allow computers to handle the identifiers in her place.

To solve the usability problems fingerprints introduce, a petnames system shall be adopted (Stiegler 2005). A petnames system overcomes Zooko's triangle by using three identifiers called keys, nicknames and petnames, in Stiegler's terminology. In PigeonMail, a “key” would correspond to a user's fingerprint. It is both global and securely unique, but it is not memorable. A “nickname” would correspond to a user's published name, possible her real name. It is global and memorable, but it is not securely unique, as multiple people may share similar or identical published names. These published names are visible to the user when viewing search engine results, verifying OpenPGP keys or looking up OpenPGP keys in a public directory. A “petname” would correspond to an alias the user creates in a PigeonMail client, it is memorable and securely unique, but not memorable, as it would only be displayed to a specific user with a specific PigeonMail client. When a user adds a sender to her approved list for the first time, she must be prompted to create an alias for that user. For many users, this will be a familiar process, as instant messaging and VoIP applications such as Skype offer this functionality. However, to ensure that the petnames remain securely unique, the application must not allow the user to create duplicate aliases. If Alice has two contacts with the publish name “Bob”, she may alias one “Bob”, and the other “Bob from France”. In this way, she will never confuse the two contacts.

6 User identifier distribution

Because PigeonMail user identifiers are not memorable, they must be distributed electronically. Like Email addresses, they can be shared online on the web, with instant messaging or with any online mail system, since a PigeonMail user ID can be represented as a short sequence of text. They can be published in an online public directory, mapping names to IDs. They can also be shared offline with technologies such as QR codes, or Bump.

Specialised search engines or online public directories could be set up to allow users to find the user identifier for a particular person. A user could search for “Larry Wall”, narrowing down the results to only display the creator of the Perl language. Of course, it is perfectly possible for fraudulent results to appear, so the resulting user ID must be authenticated using another mechanism. During the early stages of PigeonMail adoption, search engines and public directories must deliberately return fraudulent results to train the user to understand that the results are not necessarily accurate, even before the network attracts the attention of malicious users.

7 User identifier authentication and certificate distribution

Unless Bob obtains Alice's fingerprint from Alice in person or through another secure and authenticated channel, Bob cannot be sure that this fingerprint does indeed belong to Alice. Bob must obtain assurances about the fingerprint's authenticity from elsewhere.

If Charlie can verify that the Alice's fingerprint does indeed belong to Alice, he can issue a certificate matching Alice's name to her fingerprint, signing it with his signature key. If Bob trusts Charlie to perform the verification thoroughly, and if Bob has previously obtained Charlie's verification key in a secure and authenticated manner, Bob can be confident that Alice does own her

fingerprint upon verifying the certificate.

To solve the problem of certificate distribution, two categories of solutions have been proposed by the cryptographic community: CA authorities and webs of trust. Because PigeonMail is required to be a decentralised protocol, the web of trust model is preferred.

Receiving messages without verifying the authenticity of the sender's fingerprint must be allowed. The reasoning behind this is best explained by an example. Bob has obtained Alice's fingerprint from a web page. He has not verified himself that Alice does indeed own that fingerprint, nor has any of his trusted users in the web of trust. He now receives a message from Alice's fingerprint. He is naturally curious to read the contents of the message. If the software prevents him from viewing the message because the sender's fingerprint is unverified, he will mark the sender's fingerprint as verified personally, even if he has not in fact verified the fingerprint in person with Alice or over the phone. The software must not encourage Bob to lie, so it must either allow Bob to view the unauthenticated message (with a warning), or not inform Bob that a message has been sent to him at all. The latter option is only feasible once the web of trust has grown very strong and very popular, which is not the case for early adopters.

Users that care deeply about the authenticity of their received messages will heed the warnings of the software and will perform the steps necessary to verify the sender's fingerprint. If authenticity is important to Bob, he can phone Alice and verify the key. In an email communication, Bob would have to convince Alice to create an OpenPGP key, to remember to encrypt and sign her messages, and to exchange public keys in a secure manner. To a security conscious user, this makes PigeonMail a more attractive protocol than email with OpenPGP, all other things being equal.

Every user can issue certificates validating the association between the publicised name of a contact and their OpenPGP fingerprint. In OpenPGP terminology, this is known as signing a user ID of an OpenPGP key. These certificates can be published on key servers. It is important that every user frequently refresh the OpenPGP keys of their contacts, to download new certificates thus potentially increasing the validity of the imported OpenPGP keys, or to download new revocation certificates, potentially decreasing the validity of the keys.

8 Contact recommendation

Because PigeonMail is based on a white-list approach, it is important that users add each other as soon as possible. Once a recipient adds a sender to her approved list, the sender could be notified of this. The sender could then be prompted to send a list of recommended contacts to the recipient.

It is important that these prompts be reduced or disabled as PigeonMail grows in popularity, otherwise, spammers could overwhelm users with illegitimate notifications. To begin with, a good rule of thumb would be to only display these prompts when the user has fewer than ten contacts. This would allow the user to bootstrap an initial set of contacts, without causing her to expect continual notifications. If the user is interested in knowing who is requesting her messages, she can consult this list by intentionally requesting it from her PigeonMail client software.

Two users who have approved each other should be able to easily recommend contacts to each other. This could be done by delivering a message in a specific format. To the end user, the experience should be as pleasant and easy as contact recommendation in Facebook or Skype.

Users should also be able to ask a recipient to recommend her to a third user. For example, if Alice and Bob have approved each other, and if Bob and Charlie have approved each other, Alice send a message to Bob, requesting that he send a message to Charlie which recommends that Charlie approves Alice. Bob would act as an intermediary, introducing new contacts to Charlie.

The user should also be able to explore the web of trust. If one of Alice's contacts has issued a

certificate for another user, there is the possibility that the certified user is of interest to Alice. Alice might choose to approve that user.

9 Conversations

Inspired by Gmail's web interface and by web forums, the fundamental unit in PigeonMail is not a message, but a conversation. PigeonMail clients should not display a list of new messages in the “inbox”, but a list of conversations. Conversations rather than messages can be flagged and marked as read or unread, conversations rather than messages have subjects, conversations rather than messages have a list of recipients. When replying to a message, there is no need to quote its content in the body of the reply, since replies will always be displayed with the original message. The popularity of Gmail's web interface proves that this user interface is at least minimally usable, and while it would be interesting to perform an HCI analysis and usability tests on it, this project is not focussed on that aspect of the problem.

The implications of this design are not only presentational, they extend to the protocol as well. One user may initiate a conversation by creating a new message untied to any pre-existing conversations. By doing so, she becomes the “owner” of that conversation. She assigns a list of recipients to the conversation, and may choose to limit the permissions granted to each recipient. For example, she may choose to only grant read permissions to one recipient, while granting read and reply permissions to another recipient. She may also choose to add recipients at a later date if she wishes. She may grant this capability to other users if she wishes as well. The owner may close the conversation to any further replies, and users may unsubscribe from individual conversations if they so choose.

When recipients reply to a message, they do not store their message on their own server, but they post it to the owner's server. This is the only case where PigeonMail resembles a push-based protocol such as SMTP. The potential for spam is negligible, as the recipients have been approved by virtue of being grating reply permissions to the conversation. In addition, the owner of the conversation may remove them from the list of recipients if needed.

An interesting consequence of this is that participants of a conversation can in effect communicate with each other, even if they have not approved each other beforehand. This will help new users discover potential contacts, and may lower the barrier to entry.

10 Network architecture

A fully feature PigeonMail network would include the following components:

- A public directory mapping OpenPGP fingerprints to IP addresses or domain names, implemented in one centralised server or a DHT.
- A search engine or online public directory allowing users to search for OpenPGP fingerprints of a particular user.
- At least one PigeonMail mail server. It hosts outgoing mail from a sender, and provides it to recipients who fetch new mail. It can also act as an OpenPGP key server.
- Every user should have access to a machine running a PigeonMail client. It would be able to connect to a PigeonMail server.

All communication between clients and servers should be protected with SSL. Although messages should be encrypted with an OpenPGP implementation, an attacker on an unsecure network channel could still deduce private information from those encrypted messages. For example, every

OpenPGP encrypted message contains the recipient's key ID unencrypted. SSL will make this information confidential between the client and the server. The performance cost of SSL is negligible, making SSL an obvious and natural choice.

11 Extensibility

As a consequence of using end-to-end encryption, it may prove difficult to create a web interface for PigeonMail. A web interface would need access to a user's private key, but it must not be shared with the hosting server in order to keep end-to-end encryption. If the user is willing to trust the hosting server with the contents of her messages and to trust the server not to masquerade as her, she may choose to share her private key with the server in order to gain a web interface.

Another useful extension would be to have a permanently online server act as a proxy for a PigeonMail client. It could fetch messages in advance of the user's device connecting to the Internet, possibly improving the speed of delivery as perceived by the user, and providing better privacy to the user, as the sending servers will not be able to record a history of recipients' connectivity. To achieve this, the protocol must allow for a mechanism for users to authorise proxies for certain actions. For example, a user could issue a certificate indicating that a particular proxy is authorised to fetch messages on behalf of the user until a certain expiry date.

The proxy could also perform the role of a synchronisation authority. Users typically have multiple devices, and it would be desirable to have their PigeonMail information synchronised. For example, if a user fetches a message on one device, that message may be deleted from the sending server before the user is able to fetch it again on another device. To take another example, if a user adds an approved recipient on one device, it would be convenient to have it automatically added on all other devices. The protocol should be built with this extension in mind, although the extension itself is not listed in the requirements.

Initial prototype

In the first few stages of the project, an initial prototype was implemented. The implementation of a PigeonMail server was named Loft, and the implementation of a PigeonMail client was named Dove.

The software was written in Python. Initially, the Bottle framework was used, as it was thought that all communication with the PigeonMail server would occur over HTTP. HTTP provides many convenient functions applicable to a PigeonMail message server, such as caching, content-type declaration, URLs and extensibility, and it is widely supported by many libraries and frameworks. However, mixing HTTP Auth with OpenPGP proved to be inelegant. HTTP Auth expects a username and password retransmitted with every request, while OpenPGP simply signs any data with a signature key, expecting the recipient to posses the verification key. In addition, to be hold true to the principle of least surprise, the OpenPGP signature should be transmitted in base-64 encoded ASCII, rather than binary, increasing the size of each HTTP request. SSL would protect the signed request from being replayed by an attacker, if SSL was not used, a nonce or a counter would have to be used.

I began development of a new protocol on the application layer, on top of TCP/IP. As expected, this proved a lot more difficult to write, as I could not benefit from the HTTP libraries that exist for Python. Instead, I interacted directly with Unix sockets. This should hopefully mean a more efficient and more elegant protocol.

While experimenting with OpenPGP, I discovered late in the game that OpenPGP encrypted messages contain the recipient's OpenPGP key ID unencrypted. This was quite disappointing to me, as I had hoped that the identity of an OpenPGP encrypted message would be entirely concealed, diminishing the amount of trust a user needs to give to a PigeonMail server. It is because of this discovery that I dropped the requirement that PigeonMail servers be unaware of who the recipients are of outgoing messages. My project is limited in scope, and I do not have the time and expertise necessary to attempt to improve OpenPGP for this project.

Another discovery is that OpenPGP does not offer authentication without non-repudiation. Non-repudiation is the cryptographic property that an authenticated message can be passed to a third party, and that third party can be confident that it originated from the sender. This is an unwanted property in this case, it would be preferable if the recipient could be confident of the authenticity of the sender's message, without being able to provide a third party with this confidence. Because of this, I also dropped the requirement that PigeonMail messages be authenticated without non-repudiation. A sender could still plausibly deny authorship of the message in any case, by claiming that her signature key had been compromised.

At of writing, the prototype itself does very little. The server implementation, Loft, only checks to see if the requester is authorised by checking against a database table. The client implementation, implemented with Python and Gobject, only displays a list of conversations retrieved from the server. All communication occurs through SSL, although certificate verification is disabled.

Glossary

Blacklist	A list of objects to exclude, implying all other objects are allowed
MTP	Message Transfer Protocol, a theoretical protocol proposed by Watson (2002)
Online mail system	A system enabling the online and asynchronous exchange of textual messages between two or more humans, intended for private view.
OpenPGP	An end-to-end message encryption and authentication standard, using asymmetric cryptography, originally developed to be deployed on top of email
Private key	In asymmetric cryptography, private keys are used to decrypt messages encrypted with the corresponding public key. In this document, a private key is distinguished from a signature key, even though they both should remain private.
Public key	In asymmetric cryptography, public keys are used to encrypt messages that only the corresponding private key can decrypt. In this document, a public key is distinguished from a verification key, even though they both should be publicly distributed.
S/MIME	Secure/Multipurpose Internet Mail Extensions, an end-to-end message encryption and authentication standard for email, similar in purpose to OpenPGP
Signature key	In asymmetric cryptography, signature keys are used to digitally sign messages, allowing anyone with the corresponding verification key to verify the signature.
SMTP	Simple Mail Transfer Protocol
SSL	Secure Sockets Layer
Tor	An anonymity network
Verification key	In asymmetric cryptography, verification keys are used to verify signatures on digitally signed messages.
Whitelist	A list of objects to exclusively include.

References

Bernstein, D. J. 2000. *Internet mail 2000* [Online]. Available at: http://cr.yp.to/im2000.html [Accessed 14 December 2012]

Crocker D., Hansen T. and Kucherawy M. 2011 *DomainKeys Identified Mail (DKIM) Signatures* [Online]. Available at: http://tools.ietf.org/html/rfc6376 [Accessed 14 December 2012]

Haight, J. and Wong, M. W 2006. *Stubmail Readme* [Online]. Available at: http://www.stubmail.com/stubmail/docs/README.html [Accessed 14 December 2012]

Kucherawy M. 2012. *The Multipart/Report Media Type for the Reporting of Mail System Administrative Messages* [Online]. Available at: http://tools.ietf.org/html/rfc6522 [Accessed 14 December 2012]

Martin, K. M. 2012. *Everyday Cryptography*. Oxford University Press, New Delhi.

Resnick, P. 2008. *Internet Message Format* [Online]. Available at: http://tools.ietf.org/html/rfc5322 [Accessed 14 December 2012]

Shaw, D. et al. 2007 *OpenPGP Message Format* [Online]. Available at http://tools.ietf.org/html/rfc4880 [Accessed 14 December 2012]

Stiegler M. 2005. *An Introduction to Petname Systems* [Online]. Available at http://www.skyhunter.com/marcs/petnames/IntroPetNames.html [Accessed 14 December 2012]

Watson, B.K. 2002. *Proposed Method to Combat Internet Mail Abuse*. BSc dissertation, Macquarie University. [Online] Available at http://web.archive.org/web/20041205032024/http://www.comp.mq.edu.au/~brett/bschons/index.html [Accessed: 14 December 2012]

Watson, B.K. 2004. Beyond identity: Addressing problems that persist in an electronic mail system with reliable sender identification. In: *CEAS 2004: First Conference on Email and Anti-Spam*

Wilcox-O'Hearn B. and Zooko. 2003. *Names: Decentralized, Secure, Human-Meaningful: Choose two* [Online] Available at: https://zooko.com/uri/URI:DIR2-RO:d23ekhh2b4xashf53ycrfoynkq:y4vpazbrt2beddyhgwcch4sduhnmmefdotlyelojxg4tyzllhb4a/distnames.html [Accessed 14 December 2012]