Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Master's Thesis Nr. 44

Systems Group, Department of Computer Science, ETH Zurich

Encrypting Gmail

by

Patrick Nick

Supervised by

Prof. Donald Kossmann
Tahmineh Sanamrad

February 2012 - July 2012

Informatik
Computer Science

Abstract

While cloud services have many well-known advantages such as high availability, portability, modern interfaces and redundant storage, we have identified the lack of data privacy as the main disadvantage of such services.

This thesis addresses the issue of privacy and solves it for the example of the popular email service Google Mail by encrypting all email contents on a middleware before they reach the cloud servers so that the content is no longer intelligible for the provider. A proxy and ICAP server enable the use of the Gmail web interface as before, by performing the decryption/encryption on the HTTP data stream in real time. In addition, a mailserver intercepts email messages to do the same on the SMTP level.

This solution has the benefit of being highly portable, not requiring any changes to the Google servers and only minimal changes for the users. Also, the proposed hybrid encryption scheme which combines strong encryption and hashing allows for the content to remain searchable, and Gmail's useful grouping into conversations also keeps working.

Zusammenfassung

Während Cloud-Services mit vielen bekannten Vorteilen wie zum Beispiel hoher Verfügbarkeit und Portabilität, moderner Benutzeroberflächen und redundanter Speicherung punkten, haben sie doch einen grossen Nachteil gemein: die unter Umständen privaten Daten müssen dem Service Provider anvertraut werden.

Diese Masterarbeit befasst sich mit dem Problem des Verlustes der Privatsphäre und schlägt eine mögliche Lösung am Beispiel des beliebten Email Services Google Mail vor. Dies ist möglich dank einer Zwischenschicht, die alle Benutzereingaben verschlüsselt bevor sie an Google's Server weitergeschickt werden und den Inhalt von Emails somit für Google unverständlich macht. Eine Kombination von einem Proxyserver und einem ICAP-Server ermöglichen dabei die normale Benutzung der Gmail Web-Oberfläche, indem alle verschlüsselt abgespeicherten Daten in Echtzeit auf der Zwischenschicht entschlüsselt werden bevor der Benutzer sie zu sehen bekommt, was den ganzen Vorgang der Verschlüsselung für den Benutzer transparent macht. Des weiteren übernimmt ein spezieller Mailserver die Aufgabe der Ver- und Entschlüsselung auf der SMTP Ebene.

Der grosse Vorteil dieser Lösung ist, dass keine Änderungen an den Google Servern und nur minimale Änderungen am System des Benutzers notwendig sind um die Verschlüsselung (fast) transparent einzusetzen. Zusätzlich ermöglicht die vorgeschlagene Hybrid-Verschlüsselung, dass die nützliche Suchfunktion von Gmail weiterhin verwendet werden kann, und auch das gruppieren von Konversationen funktioniert weiterhin.

Contents

Chapter 1

Introduction

1.1 Motivation

The current trend clearly shows that cloud services are rapidly taking over everything. We see that all the things which used to take up a lot of space on our desktops are moving to the cloud: photos go to Flickr/Picasa/Facebook, videos to Youtube, music to Spotify, emails/calendar/documents to Google. There are good reasons for that:

- Availability,
- Portability and Mobile access,
- Sleek interface,
- Redundant storage

are all good arguments for using cloud services. Those companies offer to store our data for us, and while for many things like music and (most) photos and videos there is little concern, we may have different opinions as far as more personal data is concerned.

If we have been making important business decisions by email and the information gets into the wrong hands, we may lose a lot of money. Equally, if our calendar contents are known to the wrong people, they will know when we are on holidays and can use this to break into our house.

Even though cloud service providers such as Google assure the public that everything is perfectly safe, by handing over this kind of private information to them, the user takes a risk of his data getting into the wrong hands. By following the news we have seen time and again how big companies for example lost thousands of credit card numbers due to security leaks. Furthermore, as long as the theoretical possibility exists, a part of the users will never believe that those cloud service provider companies do not actively inspect user data for commercial profit.

We generally accept this situation because in most cases, the advantages far outweigh the risks. However, in this paper we argue that ignoring those risks is not good enough. We show at the example of Google Mail that privacy is possible without losing any of the aforementioned advantages.

1.2 Problem statement

The problem that we are trying to solve is the following: We want to be able to use the Google Mail service like before, but make sure that all our information content is not intelligible to Google or anyone who might obtain our data from Google. "Our data" includes subjects, bodies and sender/recipient addresses of email in our mailbox.

At the same time we want to keep the aforementioned advantages of cloud services: availability, portability, redundant storage and a modern and easy-to-use web interface.

1.3 Contribution

We make several contributions towards solving the stated problem.

First, we suggest an architecture which is based on a proxy server rather than on a local browser plugin and allows to build a solution which is portable for the end user. No installation other than setting up the data redirection to the proxy server is necessary.

Second, we propose a hybrid encryption scheme which combines strong, probabilistic encryption with hashing. The main problem here was to preserve Gmail's searching functionality since we feel that it presents a big usability advantage. This way, the encryption enables the content to be securely preserved while the hashes allow it to remain searchable at the same time.

Third, we introduce a simple randomized address anonymization scheme which replaces every involved email address to make sure that Google also does not know who we are communicating with.

Fourth, we present a composition and interaction of the involved components to achieve the described functionality. Proxy server, ICAP server, mailserver and MySQL server all need to work together seamlessly to reach our goal. While using the web interface, the ICAP server makes the necessary adjustments so that the user never sees the emails' encrypted form and Google never sees their clear form. At the same time, the mailserver is responsible to encrypt/decrypt incoming/outgoing emails via SMTP.

Fifth, we evaluate how much usability has to be sacrificed for various levels of security by presenting several optional features that influence this trade-off.

A note about availability: it seems clear that the theoretical availability of the resulting system will be smaller. Whenever the decryption software for some reason is not working or not accessible, it is still possible to access our emails in encrypted form but we cannot make sense of them, which is equal to not being able to access them.

1.4 Outline of the Thesis

In chapter 2 we will talk about similar and different approaches to end user privacy, and then we will provide some general background about Email and the protocols that it uses in chapter 3. In chapter 4 we then introduce the architecture of our approach and detail the suggested encryption scheme in chapter 5, followed by some implementation aspects in chapter 6. The next chapter 7 is dedicated to evaluating the achieved results in terms of security, usability and performance, chapter 8 explains the currents status and future work and we conclude in chapter 9.

Chapter 2

Related Work

2.1 CloudProtect

Diallo et al.[1] have proposed a system called CloudProtect which takes a more general approach and tries to provide a general interface for various cloud applications rather than being specific to one service. However, the most important difference to our system is that their goal is to make usability guarantees while our goal is to make privacy guarantees.

CloudProtect allows different levels of privacy ranging from "no encryption" to "strong encryption", and it applies these on a fine-grained scale such that different "objects" of an application space can have different privacy levels over time. They have come up with an algorithm which takes the user-defined constraints about execution time and additional user interactions and from that computes the maximal level of privacy that can be offered, where more privacy means less objects having "no encryption" or "weak encryption". It is also possible for the user to specify objects which must be strongly encrypted at all times.

While we agree that such an approach may be helpful in case of a cloud service provider "misplacing" data, we feel that it is not good enough to guarantee privacy with respect to a potentially curious cloud service provider, which is a concern of many users.

Another big disadvantage is that they implemented their system as an installable desktop application, which is much less portable than our approach.

In addition, a system which attempts to change the privacy levels of different objects over time is not applicable to an email system where the principal objects are emails, since an email can no longer be modified once it has been received.

2.2 Private Google Calendar

Rather than sacrificing a variable amount of privacy for attaining the usability goals like Diallo's work, this system makes strong privacy guarantees without significant usability sacrifices. Sanamrad et al.[2] describe a system which uses a proxy based approach to provide privacy for the popular service Google Calendar. Their method is similar to ours, using a combination of proxy and ICAP server to build a middleware that intercepts HTTP traffic for encryption/decryption. Their encryption scheme also works similarly to ours in order to provide CPA-secure encryption while still allowing searching.

The calendar-specific challenges are different from an email system, and basic calendar functionality does not need to pass messages to other parties. However, they came up with an elaborate key distribution scheme which allows to support the Google Calendar feature of sharing calendars with other people, so that cooperation is possible without sacrificing any privacy.

Furthermore they also hypothesized that invitation emails to shared events could be sent in a secure way by operating a mailserver on the middleware, but this feature was not implemented.

Their system was later extended to also work with browsers on mobile devices running iOS and Android.

2.3 Encrypting Google Calendar with Firefox extensions

Harrington[3] from IBM developerWorks has built a system which offers similar levels of privacy for Google Calendar as Sanamrad's[2] solution. However, this approach is based on Firefox extensions to perform most of the computation (encryption/decryption), which will only work on one machine unless the user (in addition to installing the extensions) manually synchronizes state (such as encryption keys).

Chapter 3

Background: Email

3.1 Introduction

Even though the invention of email actually predates the internet, today email is still the most popular and most widely used application on the internet. Since most people use email almost on a daily basis and are relatively familiar with the advantages and risks of use, we will not discuss its very basics. Instead we will focus on a few particular aspects which are relevant for this project.

An email message will traverse different elements between the endpoints. End users usually only know the *Mail User Agents* (MUA) which are well-known applications such as Outlook or Thunderbird. In the normal case MUAs will send a freshly composed email to the *Mail Transfer Agent* (MTA) specified in their settings. MTAs are the elements which are responsible for the message routing and transmission, and every MTA has a domain or subdomain for which it is authoritative. The first MTA will typically perform the following steps:

1: decide if the message is valid

2: decide if the sender is authorized to send/relay from this address

3: locate the authoritative MTA of every receiving domain via DNS

4: send out one message per listed recipient

Note that the following MTAs in the relay route will not perform step 4, but only relay based on the information on the envelope (see next section). The final MTA in the relay chain will then transfer

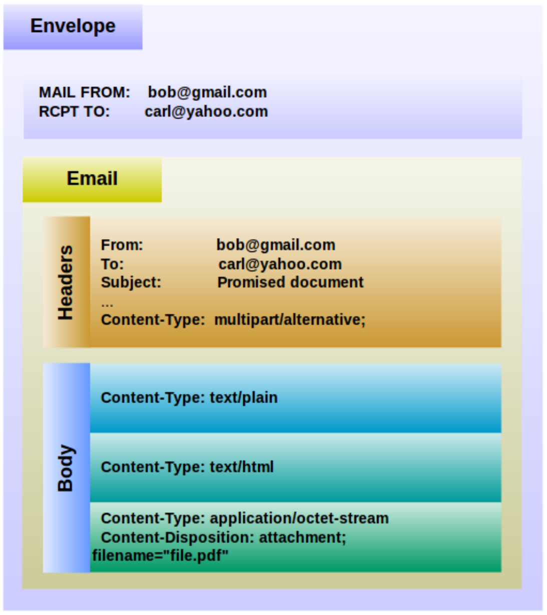


Figure 3.1: Detailed example of an email message structure including envelope and multipart body

the message to a *Mail Delivery Agent* (MDA) which is going to append it to the mailbox of the recipient.

The MTA is the only relevant unit for this thesis because we need our own MTA for our system, and we will sometimes also refer to it as the mailserver.

3.2 Message structure

The detailed structure of email messages is specified in RFC 5322[6]. The most important aspect that we would like to mention is the distinction between email envelope, headers and body. This is illustrated in Figure 3.1.

Similar to snailmail, emails have an envelope which contains information about who originally sent the message and where it's supposed to go. This envelope is created by every involved MTA and can be different for every hop of its relay chain. However, it is only used for routing on the way and the end users on both ends will never see it. The actual email message then consists of two parts: the email headers and the body. There can be many possible headers that deal with different aspects of the message and most of them are optional. Well-known headers include "From:", "To:", "Cc:", "Bcc:" and "Subject:".

Thanks to the *Multipurpose Internet Mail Extensions* (MIME) standard, the body of a message nowadays usually consists of several parts as well, each of which can be of a different data type and encoding. This enables to include different representations of the body content (for example plain text and HTML) as well as having attachments in a binary format, all in the same message.

The important thing to note in this section is that the "From:" and "To:/ Cc:/ Bcc:" headers in the email are independent of the "MAIL FROM:" and "RCPT TO:" fields which are used in the email's envelope! The headers in the message have no influence on the routing since that is based only on the envelope. The consequence of this is that spoofing email becomes quite easy, which boosts the spam problem, but also benefits this project. We will talk about it in more detail in section 4.4.

3.3 SMTP, IMAP and POP

The *Simple Mail Transfer Protocol* (SMTP)(RFC 2821)[6] and its extensions are definitely the most important protocols for email; they are what MTAs use to communicate with each other. After establishing a connection, SMTP transmits the envelope first ("MAIL FROM:" and "RCPT TO:"), followed by the message with all its headers and body parts. Once the MTA confirms having received the complete body, it assumes responsibility for the delivery of the message, meaning that it will either successfully deliver the message (to the next MTA or the final recipient's mailbox) or send a failure ("bounce") message to the address noted in the "MAIL FROM:" envelope field.

The *Post Office Orotocol* (POP)(RFC 1939)[6] and the *Internet Message Access Protocol* (IMAP, RFC 3501)[6] are used for managing mailbox contents and support operations such as downloading or deleting emails. POP does everything on the client machine and the emails are deleted on the server after downloading, while IMAP leaves the messages on the mailbox server and manages them there, allowing more flexibility by access through different client machines. However, the important point to note is that, regardless of which of the two is used, they only come into play once a message has arrived in the correct mailbox of a final recipient, and are therefore irrelevant for this project since this happens behind the Gmail web interface.

3.4 Security

With respect to (technical) email "security" there are different properties that need to be understood to enable an informed discussion.

- **Authenticity:** This refers to guaranteeing that the address listed in the "From:" header really is the technical origin address of the message.
- **Confidentiality:** Confidentiality (also called privacy sometimes) means that the intended meaning of the message content can only be understood by the sender and the intended recipients. If a third party was somehow able to acquire a copy of the message it must not be able to make sense of it.
- **Integrity:** The meaning of integrity in this context is that it is not possible to modify a message while it is on its way without the change being immediately apparent to the recipient.

Due to its empirical development, the default architecture of email over the internet unfortunately provides none of the described properties. The absence of authenticity makes it easy for spammers to conceal their identity, and since messages are generally transmitted in clear text it is easy to intercept and/or modify them.

Solutions to enforce the described properties exist, and *Pretty Good Privacy* (PGP) is probably the most important one. PGP can be used to add a hash of the message content to enforce integrity, and with the hash it can also create a digital signature to ensure authenticity (based on a public key cryptosystem and a web of trust, the recipient can check the identity of the sender). Furthermore it is also possible to encrypt the message content by using PGP to guarantee confidentiality, but this requires all recipients to have the necessary decryption software installed.

It is very important to be clear about what the approach of this thesis does and does not provide in terms of email security. We are talking about adding privacy to Google Mail, but this does not mean that we add confidentiality in the aforementioned sense. Our goal is to add privacy with respect to the cloud service operator (Google), not with respect to the entire message path. The version of privacy discussed here only makes sure that Google can not understand the content of our emails even though it is storing them, but we do **not** provide end-to-end privacy. Once a message leaves our protected "zone" it is sent on in clear and once again has none of the three mentioned properties.

Chapter 4

Architecture

In this chapter we are going to describe the different components used in the system's architecture as well as their interaction in detail. First we show an overview of the complete system and then present some particular points about the involved parts.

4.1 System overview

In the normal usage of Gmail without any modifications there are basically two relevant layers: The user's browser runs the web interface (client) and the Google cloud represents the server. We designed our system as a middleware layer which can be added between the traditional client and server. This does not require any modification of the Google servers and only minimal modification of the browser, which makes our solution highly portable. The only two changes that a user must perform is to set his browser to use a proxy server and exceptionally accept our self-signed SSL certificate.

Figure 4.1 shows the architecture overview of our system. On the top we see the browser layer, in the center we see our middleware layer and on the bottom is the Google cloud server layer. The left side represents the sender and the right side is the receiver, and the communication between left and right sides is always in clear.

It is important to understand that we designed this in such a way that a user's public email address is different from the underlying Gmail address both in user and in domain part. As far as the outside world is concerned, our email address doesn't have gmail in the name (in fact this wouldn't be possible because it is not feasible to set up an intercepting mailserver that pretends to be authoritative for the *gmail.com* domain). The goal is that the Gmail address is hidden as much as possible, we are treating Gmail like a storage system where

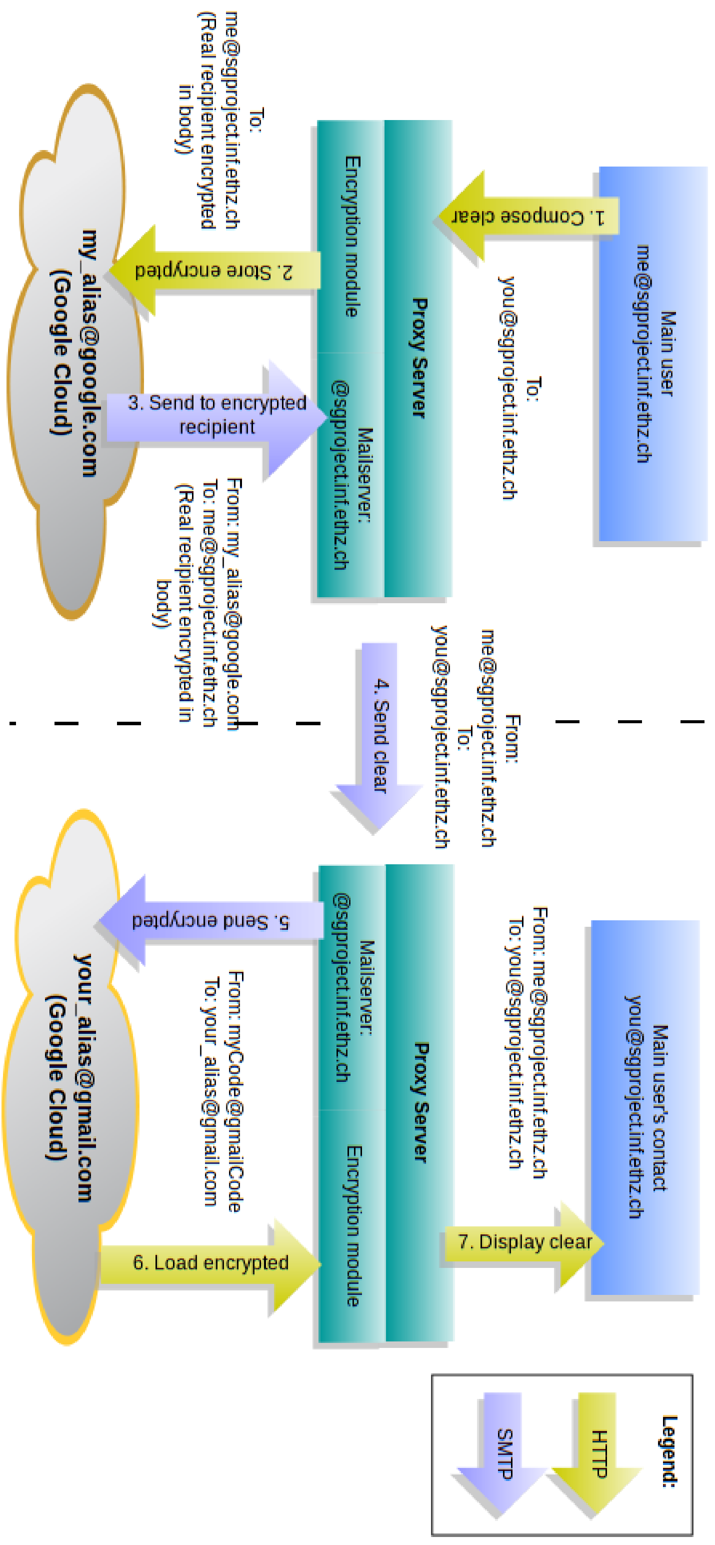


Figure 4.1: A layout of the system architecture, using sgproject.inf.ethz.ch as the publicly visible mail domain

we can send our emails to, and the ICAP server is the user interface that accesses it.

We will explain how the two main use cases of sending (steps 1-4) and receiving email (steps 5-7) work with our system based on Figure 4.1. In this example, the clear public email addresses of the two users "me" and "you" are of the domain "sgproject.inf.ethz.ch". The underlying Google accounts are "my_alias" and "your_alias".

1 **Compose clear:** In the first step, the user (after logging in to Gmail) composes a new email to *you@sgproject.inf.ethz.ch* in the web interface and clicks "Send". The entered data is transmitted to the middleware by HTTP in clear and the necessary parts get encrypted there with this user's encryption key.

2 **Store encrypted:** The now encrypted message gets sent to the Google servers by HTTP for delivery. Together with the encryption of subject and body, all effective recipients are encrypted in the body and the only remaining visible recipient is *me@sgproject.inf.ethz.ch*.

3 **Send to encrypted recipient:** The Google mailservers now send the encrypted email back to the middleware's mailserver by SMTP. This is necessary because the message needs to be decrypted, otherwise the recipients cannot read it. The middleware also finds the original recipients in the body and writes them back into the message.

4 **Send clear:** The middleware now sends the decrypted email to the recipient's mailserver by SMTP, which concludes the sending use case. Note that if there was more than one original recipient it has to send out one email per recipient (we explain this in more detail in section 4.4).

---

5 **Send encrypted:** To also illustrate the receiving use case, in our example at least one recipient happens to be someone who also uses this encryption middleware. (For non-users, the email would go to their inbox directly like in any conventional email transmission.) The middleware receives the clear email and encrypts the subject and body with this user's encryption key. In addition, it anonymizes the sender address and all recipient addresses by replacing them with codes, so that Google does not know who the user is communicating with. The encrypted email is then sent on to the Gmail alias address by SMTP. The

address codes and their respective clear addresses are stored in a database. Note that this was not necessary on the sender side since the only recipient at that point will always be the same (*me@sgproject.inf.ethz.ch*) and the sender is the Gmail address itself. If we used any codes at that stage, Gmail would try to send email to non-existing code addresses and receive bounces.

6 **Load encrypted:** When the recipient logs in to the Gmail web interface, it will load the inbox from the Google servers for display. Again, this HTTP data transfer is first intercepted by the middleware, so that it can scan for any present ciphertext and decrypt it before showing the result to the user. Also, address codes that were inserted by the middleware's mailserver will now be replaced by the real addresses which it finds in the database.

7 **Display clear:** Finally, the user will get the clear contents from the middleware.

4.2 Web Proxy

The web proxy is responsible for intercepting the HTTP traffic between the client browser and the Google servers, feeding this data to the ICAP server and forwarding the modified response.

The proxy also needs to present an authentication window before a user can access the system. This authenticates the user so that the ICAP server knows which encryption key to use, but it also has the additional benefit of preventing unauthorized usage/abuse as an "open proxy" by third parties.

The main challenge here is the termination of TLS/SSL encrypted traffic, as the proxy is using a self-signed certificate. Towards the client browser the proxy pretends to be a Google server, effectively becoming the man in the middle, but it is unable to provide a trusted certificate identifying it as such. Therefore, the user's browser will display a warning, and to use the system he will need to create a permanent certificate exception on every device that he wishes to use. The security consequences of this are explained in section 7.1.

4.3 ICAP Server

The *Internet Content Adaptation Protocol* (ICAP) is used to perform the actual real-time data modification on the traffic. The ICAP unit receives the data from the web proxy, performs encryption/decryption of the relevant parts as necessary, and then sends the data back to the proxy. It can make the decision of whether to encrypt or decrypt based on the direction of data flow; encryption is performed on specific user inputs sent to Google and decryption is applied to all data loaded from the server to the browser.

The inherent problem and main challenge here is that we are intercepting data streams about which we do not know anything. There is no public API or definition of any kind saying how the data is structured, the best we can do is apply some simple reverse engineering and make reasonable guesses. The subsequent problem is that Google may change their data formats at any time and it may break our system until we can adapt to it. More on that in section 6.2.

The nature of the Gmail web interface has also brought numerous challenges to our attention which are very specific to Gmail:

- Some of their data structures include a length field which needs to be found and adapted if content is changed, otherwise their javascript encounters an exception and the entire interface simply freezes.

- Since the fields of their data structures are delimited by double quotes we need to escape any doubles quotes and backslashes that might occur in the cleartext of an email that is being decrypted. HTML emails also need to be handled with care since different escaping methods are used at different times, as sometimes long emails are clipped and then displayed in a separate tab upon request.

- Gmail may insert HTML tags such as <b>...</b> (boldface) or <wbr> (word break) into the email body before transmission. Such modifications need to be filtered out since they break the cipher.

- The inbox overview page displays inline previews of the email subject and content up to a certain point, and then breaks it off for performance reasons (see section 7.3). Any content that used to come after the cutoff is not transmitted at this point, making decryption of the previewed part impossible because parts of the ciphertext are missing. We solved this by including a separate preview-ciphertext which is short enough (deferring content hashes to the following block).

- The available length for email subjects is limited to 250 characters in Gmail, which might be a problem for long subjects because the encryption increases their length.

4.4 Mail Transfer Agent

The MTA mainly acts as a relay for email and operates independently from the proxy and ICAP server; it only needs access to the same encryption keys. It will receive messages by SMTP, decide whether decryption, encryption or both are required, do it and then send the message on by SMTP to one or more new recipients. This process is illustrated in Figure 4.2.

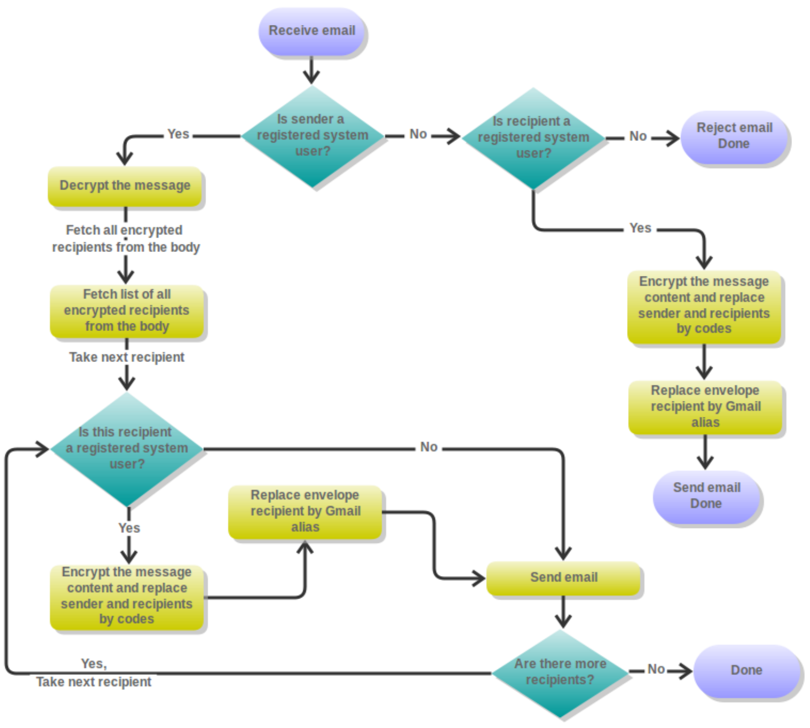

Receive email
Is sender a registered system user?
Yes
No
Is recipient a registered system user?
No
Reject email Done
Yes
Decrypt the message
Fetch all encrypted recipients from the body
Fetch list of all encrypted recipients from the body
Take next recipient
Encrypt the message content and replace sender and recipients by codes
Replace envelope recipient by Gmail alias
Send email Done
Is this recipient a registered system user?
No
Yes
Replace envelope recipient by Gmail alias
Encrypt the message content and replace sender and recipients by codes
Send email
Are there more recipients?
No
Done
Yes, Take next recipient

Figure 4.2: Decision making flowchart for the MTA

First, we examine the sender of the email. If the sender address is the *user_alias@gmail.com* address of a registered user of our system then we know that it is an outgoing email which was just composed. In that case the message must have been encrypted by the ICAP server and first needs decryption. During decryption we also fetch the encrypted list of original recipients which were included in the email body. These recipients are now regularly written to the "To:/Cc:/Bcc:" headers. For every recipient from that list, we now need to decide if it is another user of our system. If not, we will send on the email in clear, but if the recipient is a user, then we need to encrypt it again, this time with that user's encryption key, and replace all clear sender and recipient addresses by codes, before sending it to the corresponding alias gmail address.

Going back to the start of the chart: If the sender address of the incoming email is not a registered user of our system, it checks the recipient. If the recipient is not a system user either then this email has nothing to do with us and we reject it to prevent acting as an open relay. If the recipient is a system user then we need to encrypt it with this user's key, replace the sender and recipient addresses by codes and then send it on to the corresponding Gmail alias address.

Let us make an example by using the same domain as in Figure 4.1. Assume that user *bob@sgproject.inf.ethz.ch* has just composed a new email which should go to *carl@sgproject.inf.ethz.ch*, and *dave@yahoo.com* should receive a carbon copy (Cc).

The MTA will receive an encrypted email where the envelope's sender is *(bobs_ alias)@google.com* and the envelope's recipient is *bob@sgproject.inf.ethz.ch.* The MTA immediately recognizes that *(bobs_ alias)@google.com* is an alias of Bob who is a system user and decrypts the message with Bob's key. At this point the message's headers only list one recipient "To: *bob@sgproject.inf.ethz.ch*". During decryption the MTA finds the encrypted list of original recipients: "To: carl@sgproject.inf.ethz.ch" and "Cc: *dave@yahoo.com*". It writes those into the headers of the email and deletes the already present "To: *bob@sgproject.inf.ethz.ch*". First Carl is processed. The MTA finds in the database that Carl is also a system user and therefore email to him needs to be encrypted with his key. Body and subject get encrypted and now both recipient addresses get replaced by a code; let's say "To: *carl@sgproject.inf.ethz.ch*" gets replaced by "To: *-gcqd6d-@-gcq4r4-*" and "Cc: *dave@yahoo.com*" gets replaced by "Cc: *-gcq765-@-gcqwtf-*". This replacement will later be inversed by the ICAP server while Carl loads his inbox in his browser. Now the email will be sent with the envelope's recipient set to *(carls_ alias)@gmail.com.* Dave is processed next, but as he is not a known system user the email will be sent out in clear, with "To: *carl@sgproject.inf.ethz.ch*" and "Cc: *dave@yahoo.com*" in the headers and with *dave@yahoo.com* in the envelope's recipient.

Chapter 5

Encryption

5.1 Challenges

The very particular scenario of using Gmail poses numerous challenges to an encryption scheme which is to be applied to all emails. We list the main challenges in this section.

5.1.1 Search Function

According to the literature[4] , a cryptographically strong encryption scheme must not be deterministic but rather probabilistic in order to be secure against chosen plaintext attacks (CPA). This means that a given plaintext must not produce the same ciphertext when encrypted again. However, supporting a search functionality on encrypted content requires a deterministic encryption, so there is a conflict of interests.

For solving this dilemma we decided to use a hybrid scheme which on one hand uses a strong CPA-secure encryption to preserve the plaintext with its formatting, and on the other hand hashing every word and saving all hashes together with the ciphertext. This way a search query can also be hashed before sending it to the servers, which will then search for occurrences of the hash value without knowing its meaning.

5.1.2 TLS Termination

Since the Gmail web interface is secured by TLS/SSL, performing encryption and decryption on a middleware layer makes it necessary to become a man in the middle. The user will notice this due to the absence of a trusted certificate for the domain *mail.google.com*.

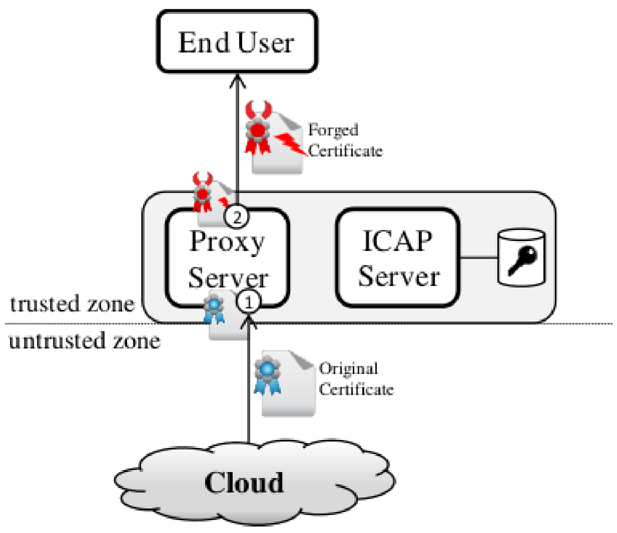


Figure 5.1: Visualization of the interrupted SSL tunnel and the false certificate

The simplest solution for this is to present a self-signed certificate and require the user to permanently store it as an exception. Sanamrad et al.[2] have visualized this nicely and we show their illustration in Figure 5.1.

We feel that the resulting potential security loss is acceptable since it would be hard for an imposter to present the user's decrypted mailbox in a credible way, and also because our primary goal is to obtain privacy with respect to a curious cloud service provider and not with respect to malicious third parties that are listening on connections.

5.1.3 Spamfilter

Spam filtering which is based on the content of email subjects and bodies will most likely not work when the content is encrypted since none of the checked words/patterns are found in the ciphertext. However, content-based filtering is only one part of contemporary spam filters, and other techniques such as black-/whitelisting based on email addresses, headers and domains or DNS-based techniques (SPF, DKIM) will continue to be effective.

A possible solution would be to add a content-based filter to the mailserver in our architecture. However, we feel that further analysis is required to estimate how much our system really impacts Gmail's filters, and so we leave this as future work.

5.1.4 Spellchecker

Gmail's spellchecking feature is affected by the same problem. Since spellchecking is done on the servers, it requires the cleartext to be sent to Google. If we insist on preventing any cleartext to be sent to Google at all times, then this feature will not be available to the user.

It is possible to argue that Google will probably not store users' spellcheck queries beyond computing the answer and that it would therefore be acceptable to keep using it. We leave this choice as a user setting. If set, it will encrypt the spellchecking request, leading to a server answer without any useful information. If not set, the request goes through in clear and will give results.

5.1.5 Conversation grouping

A very convenient feature of the Gmail web interface is that after using *Reply*, emails get grouped into conversations, saving a lot of space if there are many emails being sent back and forth between certain people. Thanks to the Gmail Support[12] we found that there are two conditions for this grouping to happen: the sender/receiver addresses must match (inversed) and the subject line must be equal apart from well-known prefixes like "Re:".

In light of our encryption scheme (refer to sections 5.2.1 and 7.1) this presents a challenge, because our scheme is designed in a probabilistic way such that two equal plaintexts will never lead to equal ciphertexts. The consequence of this is that on the servers, the first email and the replying email have subjects which differ in more than only the "Re:" prefix, and therefore Gmail will not group them into a conversation as this decision is made on the servers, before going through decryption.

To solve this we can for example use the encryption algorithm with a deterministic initialization vector rather than a random one. A hash value of the subject string could be such a deterministic IV, which would lead to equal cleartexts resulting in equal ciphertexts (per user), which is what we need. The consequence is a decreased theoretical security, which we discuss in section 7.1.

5.2 Encryption Scheme

5.2.1 Content Encryption

For securely preserving the content as well as its formatting (capitalization etc.) we need a cipher that allows reconstruction of every character as it was. We decided to use a symmetric encryption rather than a public key system because asymmetric encryption is computationally about 100 to 1000 times more expensive and because our middleware is the only point of encryption/decryption (no key transmission is necessary at any time).

Among symmetric ciphers it seems recommended[4][5] to use the *Advanced Encryption Standard* (AES), and we decided for a key length of 128 bits. To obtain a probabilistic encryption we run AES in *Cipher Block Chaining* (CBC) mode using a random *Initialization Vector* (IV) of 128 bits. The used IV is encrypted with a second key in *Electronic Cookbook Mode* (ECB) and then stored next to the ciphertext. The security of this scheme is described in section 7.1.

Encryption keys are unique among system users, every user has their own key. They are randomly generated when a new user is registered and will only be stored on the middleware (with backups).

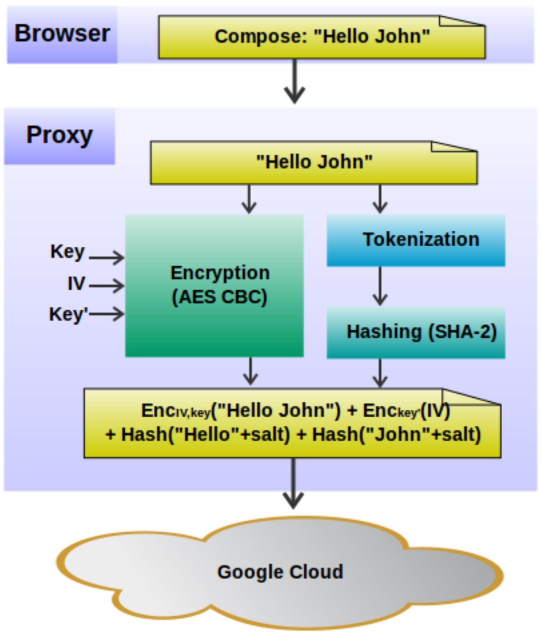


Figure 5.2: Illustration of the hybrid encryption scheme with hashing

5.2.2 Content Hashing

As mentioned before, we add hash values for all strings that should be searchable. For this we use the *Secure Hash Algorithm*, namely SHA-2 with a hash length of 256 bits.

When an email is composed, the middleware breaks the body content into tokens, optionally stop word elimination and stemming can be applied, and every resulting token is hashed. The hash values are then appended to the ciphertext and the IV. This process is illustrated in Figure 5.2.

When the user enters a search query it is intercepted and the same steps are applied to it. The resulting hash values are then sent to the Google servers to perform the search. This way, the emails which contain the relevant terms can still be identified and listed, and operators like AND/OR between the query terms will also keep working.

5.2.3 Address Replacement

As an additional privacy measure we can optionally replace all email addresses by random strings in incoming emails as described in the example in section 4.4. The goal is to make sure that, in addition to being unable to read our content, Google also does not get any information about who we converse with. This does not apply to outgoing emails because there the effective final recipients are encrypted in the body and thereby unaccessible to Google anyway.

To realize address replacement, the middleware needs to keep a database with the mapping of real addresses and their corresponding random strings. When an email comes in through the MTA, it will replace the address headers and generate new random strings for addresses that are not yet contained in the database. At the moment of rendering the inbox the ICAP server can then consult the database and do the inverse replacement so that the user will see the correct addresses.
We recommend to keep this feature optional because it breaks Gmail's grouping of replied emails into conversation threads. We initially believed that this would also make it impossible to use the address book feature (*Contacts*), but it turns out that is not the case.

Chapter 6

Implementation

In this chapter we will provide some information about which systems we used and what the involved challenges were. We are not going to present specific code but only high level descriptions. Everything is running on a server with Debian Linux.

6.1 Web Proxy

In order to be able to intercept the HTTP traffic between the user's browser and the Gmail servers we need a web proxy. For this we have used an open source proxy called *Squid*[7].
The proxy has three main responsibilities:

- **User authentication:** Different users have individual encryption keys, and so the middleware must know which key to use for every request. Squid can do this by requiring users to enter credentials before allowing usage.
- **SSL/TLS termination:** The proxy must terminate the SSL tunnel in order to be able to inspect and modify the HTTP data.
- **Redirect traffic through ICAP server:** The encryption/decryption is done by the ICAP server, and therefore the web proxy must pass every incoming request and answer to the ICAP module, which will send it back as soon as it's done.

Requiring users to authenticate with their credentials has the additional benefit of protecting the proxy from being abused as an open gateway. This method should be more effective than other options like restricting the IP range for clients or the destination domains for requests.

Probably the biggest challenge here is the SSL termination. As mentioned in sections 4.2 and 5.1.2, becoming the man in the middle has important consequences for security and trust. However, implementation-wise it is very simple to create a self-signed certificate for the *mail.google.com* domain.

6.2 ICAP Server

We have used an open source ICAP server called *Greasyspoon*[8] to implement our system. This integrates effortlessly with Squid and offers a convenient web administration interface. One can define scripts which will be executed on HTTP requests and responses, and from these scripts it is possible to make calls to imported Java code. The main application logic of the ICAP server is therefore written in Java.

We have already listed the numerous challenges for the ICAP server in section 4.3 so we're not going to repeat everything here. The most challenging part during the implementation was the guessing of the required character and HTML escaping methods as well as the correct modification of the length fields. They were difficult mainly because it is impossible to obtain any debugging information; if we make a mistake the only available feedback from the interface is that it freezes. In general we found Gmail's data formats to be very inconsistent: length fields may be present or not, word breaks may be inserted or not, characters may be unicode-escaped (\u00xx) or not, linebreaks may be encoded as "<br>" or as "\n", etc.

We would like to point out that this is the most fragile part of the implementation and the system's greatest weakness. While the mailserver is able to operate on well-defined standard protocols, the ICAP server is intercepting data in formats that are not meant to be understood by anyone apart from the Gmail servers and the javascripts of the web interface. In consequence, the data formats are not standardized and subject to change at any time. In fact we have observed such changes 4 times during the time of implementing the system, where 3 of those changes rendered our system unable to perform the decryption. In such cases the changes first need to be identified and then the necessary adaptations must be made, which results in a "downtime" for end users.

6.3 Mail Transfer Agent

As we described in section 4.4, the mail transfer agent performs encryption/decryption on the SMTP channel. For implementation we have used an open source mailserver system called *Postfix*[9].

Postfix allows us to specify content filters which can inspect and modify incoming emails before they are further processed. We make use of this by setting up a shell script which in turn will call different Java classes to make the decisions and perform the steps described in Figure 4.2. We use Java again so that we can reuse the relevant code from the ICAP server.

The biggest challenge in this part is the correct handling of all message format and character encoding requirements such as described in RFC 2822 as well as RFCs 2045 - 2049 [6]. For this purpose we use an open source package called *Javamail*[10], which handles most of that and allows us to access the relevant parts (subject, body and address headers) as Java strings.

6.3.1 Getting our MTA recognized

Setting up a mailserver brings a set of challenges on its own. Because of the widespread problematic of spam, well-known legitimate email providers such as Gmail will only accept email from servers which fulfill a list of requirements. The requirements all have to do with the *Domain Name System* (DNS), and explaining their precise meaning is beyond the scope of these implementation notes. The reader can consult the referenced RFCs[6] to find more information.

We have only experimented with Gmail, but these requirements probably differ among providers. We found the following conditions for our server to be sufficient:

- It is not an *open relay*, i.e. it must not forward email if it is not either responsible for forwarding the source domain or authoritative for the destination domain
- It is not on a blacklist (RFC 5782)
- It has a valid reverse DNS record (also called PTR record, see RFC 1912)
- It has a valid MX DNS record (RFCs 974, 1035)
- It has a valid SPF record (*Sender Policy Framework*[11])
- The TTL of all mentioned DNS records is not lower than 24 hours

In order to make sure that our server is not an open relay, we have Postfix check against the database that it is responsible for the recipient (see Figure 4.2). Violating messages are rejected immediately at the SMTP conversation level to prevent taking responsibility for non-delivery notifications (see section 3.3) to prevent backscatter (a form of spam which describes incorrectly sent bounce messages in the case of forged sender addresses).

6.4 Database

So far we have not put much emphasis on the database, but it should be clear that there needs to be a way of storing user's encryption keys, account names and settings. The database also enables the necessary "communication" between the proxy, ICAP server and MTA.

We have chosen a very basic MySQL database for this purpose.

Chapter 7

Evaluation

7.1 Security Evaluation

In the described system there are several aspects which need a security analysis: The used cipher, the hashing, the SSL termination and the particular case of deterministic encryption for subjects. We start off with the cipher.

There are several attacks that could be conducted to break a cipher, and in all our considerations we will assume that the main attacker is Google. The first scenario is a *ciphertext-only attack* (COA) where the attacker only sees ciphertext outputs $enc_k(plain)$ for unknown plaintexts. This attack on our system can easily be realized by the attacker. The literature[4] says that any secure cipher must have a key space which is not vulnerable to brute force search, and our 128 bit AES cipher satisfies this condition as the key is randomly generated when registering a new user.

The next stronger attack is the *Known-Plaintext attack* (KPA) where the attacker can see matching pairs of plaintext and ciphertext $(plain_i, enc_k(plain_i))$. AES was designed[5] to be secure against KPA, and that is important because this attack is also not difficult for our attacker. For our email system, the attacker could just send email to an encrypted account's address to obtain ciphertext corresponding to the email's plaintext, which actually represents an even stronger attack called CPA.

The *Chosen Plaintext Attack* (CPA) is the strongest attack that we need to be secure against. As indicated before, this means that the attacker gets access to an arbitrary number i of pairs $(plain_i, enc_k(plain_i))$ where he gets to choose $plain_i$. According to the literature[4], deterministic ciphers will never be secure against CPA, only probabilistic ciphers can be. For this reason we use AES in cipher block chaining mode with randomized initialization vectors to make it probabilistic, which makes it secure against CPA.

Next we analyze the security of the used hash function, which is important since we store hash values for every encrypted word. A cryptographically strong hash function needs needs to have two basic properties: *Preimage Resistance* and *Collision Resistance.* The first means that when given a hash h with $h = hash(m)$ it must not be feasible to find m, and the second means that it must be difficult for any probabilistic polynomial-time algorithm to find a collision ($h_1 = h_2$ for $h_1 = hash(m_1)$ and $h_2 = hash(m_2)$ where $m_1 \neq m_2$). The used *Secure Hashing Algorithm* (SHA) has these properties.

In addition to the mentioned aspects, other attacks can be launched against a hashing algorithm, most importantly *Rainbow Table* attacks. Rainbow tables are precomputed tables for reversing cryptographic hash functions. To be secure against this attack we need to make sure that our inputs are not contained in those Rainbow tables, and we do this by prepending a 128 bit random *salt* string which is also unique per user.

Next we look at the address replacement scheme. This is quite straightforward since the codes that replace cleartext addresses don't contain any information about the cleartext but are just randomly generated. It is simply a mapping which is unknown to an attacker, and so he can not learn anything from seeing the outputs.

As far as the modified subject encryption is concerned, we replaced random IVs by deterministic IVs to preserve the grouping into conversations. The consequence of this is that the cipher is no longer secure against chosen-plaintext attacks. As mentioned earlier in this section, the attacker is able to perform CPA by sending email with a carefully chosen subject to the encrypted account, so the security of the subject becomes compromised if the attacker builds himself a dictionary of $(plaintext, enc(plaintext))$ strings. This still involves a lot of work for the attacker but is theoretically feasible and the user should keep it in mind.

Some more security-related aspects should be mentioned. The main reason for a user to subscribe to this system is because he doesn't trust Google. Using our system requires the user to instead trust the operator of the middleware. However, if he is not willing to do that either there is always the possibility to himself become the operator of the middleware only for private usage.

We should also analyze the consequences of the proxy's SSL termination. By becoming the man in the middle on the user's HTTPS connection we now have two SSL tunnels rather than one. For the one between the proxy and Gmail there are no concerns, but the user is dealing with an increased risk on his end. SSL certificates have the purpose of guaranteeing authenticity of the server, but by accepting the proxy's fake self-signed certificate the user loses any certainty about who he is actually connected to. However, since the proxy should only be used to access the Gmail interface, there is an easy way for the user to verify that he is connected to the real proxy: only the real proxy should be able to decrypt the emails in his inbox. Therefore reloading the inbox would immediately reveal an imposter.

Note that it is not in our best interest to provide a certificate for *accounts.google.com* as well. The user must not authenticate towards Google through the proxy, as the security implications of that would be far greater (it would mean that we learn the user's Google account password, which is a responsibility that we do not want to assume). Trying to do this will fail exactly because our certificate is not valid for *accounts.google.com*. Instead we require the user to do the Google account login before turning on the proxy (or alternatively to use a browser functionality (plugin) that allows to specify for which URL patterns to use the proxy in a fine-grained way).

7.2 Usability Evaluation

7.2.1 General

We start the usability evaluation with some general comments that need consideration before deciding to use such a service. In chapter 2 we described that a similar proxy encryption approach has been done with Google's Calendar service. However, there is an important difference between an email service and a calendar service: While it is in the nature of calendar events that they can be modified at any time, the semantics of changing an email after it was sent or received are undefined. Emails are final in that way, they can only be deleted, and this has serious consequences.

For example, once a part of my Gmail inbox is encrypted, it can never be "permanently" decrypted again, and every future access to that part must go through a decrypting proxy. This makes it difficult to envision a scenario where a user decides to unsubscribe from the encryption service. In a calendar, the middleware could just fetch all encrypted events and write them back in clear, but this doesn't work for email. A possible workaround would be to send new clear emails with the previously encrypted content to oneself, but that would lose the date and recipients information associated with them.

Another consequence is that if the decryption key is lost and there is no backup, access to the mailbox is irreversibly lost.

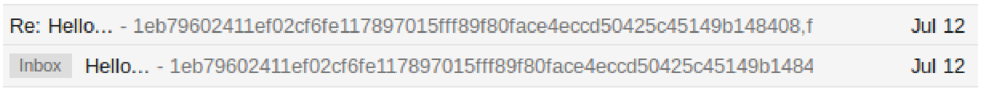
7.2.2 Setup

For new users who wish to start using this system there are two technical prerequisites: they must sign up for the proxy and they must have a Gmail account. However, there is a third, less technical prerequisite: they must start using a new email address. This may seem obvious, but we emphasize it because it is technically possible to use an existing Gmail account.

Either way, and this is very important to understand: using this service will lead to an inbox that is actually shared by two addresses. In the example from section 4.1 that would be the "encrypted" address *me@sgproject.inf.ethz.ch* and the "Gmail" address *my_ alias@gmail.com*. Both of these addresses are functional and can receive email, but only those emails which get sent towards *me@sgproject.inf.ethz.ch* will be encrypted.

Now, it is perfectly possible to subscribe with a Gmail account that has been in use before. The proxy is transparent enough so that it will just start encrypting new emails while leaving existing emails as they are. However, we strongly discourage users from doing this. If the address has been in use, that means that there are people who will keep sending directly to that address, while others will send to the newly created address in the non-gmail domain. Because of the automatic decryption though it is almost impossible for the user to distinguish encrypted emails from clear emails, which may lead to confusion and an illusion of security.

The necessary steps for registered users to get the service running on a new device are very simple thanks to the good portability of our approach. Only two simple steps are required: the user must configure his browser to use the service's proxy server, and he must also permanently store an SSL certificate exception in his browser (see sections 4.2, 5.1.2 and 7.1).



Re: Hello... - 1eb79602411ef02cf6fe117897015fff89f80face4eccd50425c45149b148408,f Jul 12

Inbox Hello... - 1eb79602411ef02cf6fe117897015fff89f80face4eccd50425c45149b1484 Jul 12

Figure 7.1: Screenshot of two search results of a search.

7.2.3 During Usage

After implementing the system we discovered several usability sacrifices that are necessary to guarantee the highest possible privacy. Because of this we decided to make several parts of the system optional so that the user can make his own decision on the inevitable trade-off between privacy and usability. Table 7.1 summarizes these trade-offs and the following bullets describe them in more detail.

Feature	*Benefit*	*Sacrifice*
Content Hashing	Enables Search function	-
Block Spellchecker (*optional*)	Prevents leaking the email content	Spellcheck function
Subject Hashing (*optional*)	Enable conversation grouping	Decrease in theoretical security
Address replacement (*optional*)	Prevents leaking addresses	Conversation grouping

Table 7.1: Overview of the privacy-usability trade-off

- **Content hashing:** As explained in section 5.2.2, we store hashes of every word in the email bodies, which offers the great benefit of being able to use the search function over the inbox which is extremely useful. We did not keep this optional because it doesn't have and real usability sacrifices. The only small point is that the inline previews of the search results in the result list by Gmail will show exactly what was found: a long hash value (illustrated in Figure 7.1). This is meaningless to the user but doesn't affect him much, and as soon as he clicks on a result, the email or conversation will be displayed normally.
- **Block Spellchecker** This feature is quite simple. Gmail offers a spellchecking functionality on the "Compose" screen which sends a request with the typed text to their servers and receives an answer with all markup information and correction suggestions. We offer to block (encrypt) this request in case the user doesn't want Google to receive his plaintext. Obviously, the sacrifice is that spellchecking is not available anymore.

- **Subject Hashing** This feature refers to the problem mentioned in section 5.1.5. If we want grouping into conversations or threads in the interface, the subject cipher need to be deterministic. We can achieve this by using a subject hash value as IV for the encryption. The sacrifice is a decrease in theoretical security (see section 7.1).

- **Address replacement** As discussed in section 5.2.3, we can further increase the offered privacy by replacing all recipient addresses in incoming emails by arbitrary strings to prevent Google from knowing who we are communicating with. However, this implies a usability sacrifice: we lose the grouping into conversations that we restored through subject hashing.

There is one more usability sacrifice that we have to endure in our implementation, and unfortunately it doesn't have any benefits. When we compose an email, the real recipients get encrypted in the body and replaced by our own external address to make sure the email is first sent to our middleware for decryption (see section 4.1). This leads to a problem when viewing the *Sent* folder, because there we don't see the real recipients but all sent emails appear to have been sent to our own address.

We suspect that undoing this transformation in the decryption step on the ICAP server is feasible, but it would require a deeper understanding (reverse-engineering) of the data formats as it involves inserting data in places that don't already carry a marker. We did not include it in table 7.1 because we believe that it can be fixed, but our current implementation can't do this and we will leave it as future work.

7.3 Performance Evaluation

When talking about performance we would like to address each of the subsystems separately because they are quite independent and could easily run on different machines.

For the proxy/ICAP combination we claim that there is only one performance-critical interface action: the initial loading of the inbox overview. However, the web interface is very smart in that regard. As we have mentioned before in section 4.3, it only loads preview snippets of constant length for every email to render the inbox overview (and the number of simultaneously displayed emails is limited to 100 in the settings, the default is 50). This means that the amount of data in the first request has a constant upper bound and will not increase with more or longer emails. It then immediately starts loading the complete contents of all displayed items in the background hidden from the user, so he will not notice if response time is a little longer than usual, unless he acts extremely quickly. The rest of the interface usage most of the time involves only this already preloaded data and is therefore very fast.

For the explained reasons we believe that the response times for individual users will depend only on the number of concurrent system users. We expect the sum of their interface actions to be approximatively uniformly distributed over time, so the response time of each individual action would barely be affected at first but then grow exponentially as the number of users grows large.

Rendering of the overview page subjectively always felt equally fast as without using the middleware. Because of this, and because it is difficult to simulate a big number of simultaneous web interface users for obtaining relevant measurements, we decided that the results of making scientifically sound performance measurements would not be worth the effort. Therefore, we cannot quantify the described effects, and further study would be necessary to determine how many users one ICAP server can handle.

As far as the MTA is concerned, we believe that compared to the ICAP server it is unlikely to become a bottleneck. As we just described, when loading an inbox the ICAP server will receive almost concurrent requests to decrypt 50 to 100 complete emails in a very short time, which is something that is unlikely to happen to the MTA (per user), so the load on the MTA should be more evenly distributed over time and therefore grow slower with increasing amounts of users. In addition, the user's expectations for the delivery time of an email are generally lower than his expectations on the responsiveness of a web interface, and also the delivery time of the MTA is only relevant while the user is manipulating the interface as otherwise he will not notice how long incoming email takes to get through.

Thanks to the ability of Postfix to reject "illegal" messages at the earliest point (during the SMTP conversation), even large amounts of incoming relay spam should not be a problem; the only concern might be if there are large amounts of spam which is directly directed at the addresses of our system. However, this is a problem that every mailserver faces and should be regarded separately.

In conclusion we believe that the ICAP server is most likely to become the system's bottleneck for a large number of users, but we also think that its load could easily be distributed over multiple machines if necessary. To quantify these beliefs further study would be required.

Chapter 8

Future Work

8.1 Status of Implementation

We have implemented most of the described features to check if they work. There are some exceptions, and in this section we list the restrictions of our implementation for which we are confident that the would have the know-how to solve them if given more time. Restrictions for which we are less confident to know how to address them will be listed in the *Open Questions* section.

- We have not yet implemented stemming and stop word elimination in the hashing process of the body contents. Doing this should decrease the amount of hash values to be stored and thereby decrease the response time for the user.

- There is currently no public interface to register a new user or enable/disable the features that we described as being optional. So far everything is done on request by the admin via direct database access. For putting the system into actual use, a web administration interface would need to be created.

- Email addresses often consist of a *personal* section containing the full name of the address owner and an *address* section containing the actual address. So far our implementation only takes care of the *address* section when handling and exchanging sender and recipient addresses; the *personal* part is left as future work.

- The current experimental implementation does not contain a mechanism to back up important parts of the database like encryption keys etc. If this project is pursued further it would be very important to include a backup strategy to be protected against hardware failure.

8.2 Open Questions

As mentioned in the last section, there are some remaining open questions and features which need further analysis. We will mention the most specific ones first and then progress to the more general ones.

Firstly, we are not sure if email attachments could also be encrypted. On the MTA side that should be straightforward, but we do not know how to route the user's file upload/download traffic through our ICAP server. For now, attachments remain in clear.

The second open issue is the one about not seeing the real recipients in the *Sent Mail* folder that we mentioned at the end of section 7.2. As stated there we suspect that it would be possible to fix this, but the complexity would probably be bigger than with our existing ICAP transformations.

Third, there is a small issue with the search function that would need further analysis. Even though we explicitly warn users not to use a Gmail address that has been in use before signing up to our service, some users might still decide to do so. In this case of a mixed inbox with clear as well as encrypted emails, a search query would only be effective on one kind of emails at a time, depending on if the query is hashed or not, which might confuse the user. On the other hand, if we decide to send both versions of the search query to the server then we leak information about the content of encrypted emails, which is also undesirable.

A fourth question concerns Gmail's address book feature *Contacts*. We have seen that *Contacts* can be used together with the email encryption, and the next question would be if we could also encrypt the address book.

As a fifth open question, a better performance analysis of our system should be carried out. We have made speculations about where we suspect the bottleneck to be but have not verified it, and we currently don't know for sure how many users our server can support or how their response times will be affected. Also, we don't have reliable information about how this system could be scaled to deal with big numbers of users.

The sixth point needing further analysis is the impact of our system on Gmail's spam filter as mentioned in section 5.1.3. This analysis should contain information about how many more messages are wrongly qualified when emails are encrypted (both false positives and false negatives).

The last remaining open question is very general and concerns portability of this system to mobile users. Since an increasing part of email usage happens on mobile devices it would be interesting to study the feasibility of a mobile version of this system. Obviously this would only affect the proxy/ICAP components since the MTA is independent from the user interface.

8.3 Final thoughts

On the last day before the submission deadline of this report we identified a serious vulnerability of the described system which had escaped our attention before. Unfortunately there was not enough time to adapt the previous chapters to reflect this new information, so we will briefly describe it here.

The problem lies in the hashes which enable the search function to work. We believed that the attacker could not precompute the hashes due to the *salt*, but we didn't realize that by sending email to encrypted accounts he can circumvent the effectiveness of the *salt*. By sending an email with only one word, the encrypted inbox will contain the result of the hashing **including** the salt value, and so by repeating this process Google could build an entire dictionary with relatively small effort.

There are some small steps which we can undertake to mitigate the damage; for example we could make sure that all duplicate hashes are eliminated and the unique hashes are shuffled. In consequence the attacker would only learn the set of words used in our emails, without knowing their order or multiplicity. However, even with such mitigation this is still a very serious privacy breach.

At the moment it is unclear if it is possible to solve this issue and which would be the best option. Two ideas come to mind:

- Instead of using a collision-free hash function like SHA we could purposefully use a function that produces many collisions, so that various words are mapped to the same hash value. The consequence of this is that a search for the hash value will produce many false positives. If the ICAP server could in an additional step filter them out after decrypting the contents, the user would not notice a difference. We believe that, similar to inserting the real recipients in the *Sent* folder such as described in section 8.1, it should be feasible but would require a lot more work for the reverse-engineering of the data formats.

- Rather than using hashes for search we could omit hashes completely and use only the ciphertext. Instead of generating a new random IV for every encryption the system could randomly select an IV from a bigger set of available IVs. For searching the proxy could issue a big search query where the search term is encrypted once with every available IV and all results are linked with the logical *OR* operator, which should make sure that the term is found regardless of which IV was used in the initial encryption.

Both described ideas are more than just small changes to the system and would need detailed study of their exact implications for the security as well as usability and performance.

Chapter 9

Conclusion

In this thesis we have presented a possible solution to the problem of privacy loss when using the Google Mail cloud service. We have introduced a middleware solution which can be inserted between the traditional client and server to encrypt or decrypt all passing data as necessary both for the HTTP and SMTP protocols.

The advantages of such a system, besides the fact that it now offers data privacy, are that it can use the original Gmail interface with its user-friendly functions such as search, conversation grouping and contacts, and also make use of the redundant storage, all while remaining almost as portable.

The main disadvantages are that Gmail's very high availability is reduced to the middleware's availability as well as a variable amount of usability sacrifices. Our detailed study of the privacy-usability trade-off in section 7.2 has shown how much privacy can be achieved while remaining (almost) transparent to the user, and we have identified several questions which deserve further study.

Along with the disadvantage of decreased availability we would like to again emphasize the fragile nature of the ICAP system (see section 6.2). It relies on unpublished data formats that Google can decide to change at any given time, which could lead to significantly decreased availability (downtimes) while the maintainer is reacting to the changes and adapting the implementation. For a production system nowadays this would most likely be unacceptable.

Bibliography

[1] Diallo M.H., Hore B., Chang E-C., Mehrotra S., Venkatasubramanian N.: *CloudProtect: Managing Data Privacy in Cloud Applications.* IEEE CLOUD 2012, 5th International Conference on Cloud Computing, June 2012.

[2] Sanamrad T., Widmer D., Kossmann D.: *My private Google calendar.* Technical report, January 2012.

[3] Harrington N.: *Integrate encryption into Google Calendar with Firefox extensions.* http://www.ibm.com/developerworks/linux/library/wa-googlecal/index.html, July 2008. Retrieved on July 3, 2012.

[4] J.Katz and Y.Lindell: *Introduction to Modern Cryptography.* Chapman & Hall/CRC, 2007.

[5] Daemen J., Rijmen V.: *The design of Rijndael.* Springer-Verlag New York Inc., 2002.

[6] Internet Engineering Task Force (IETF): *Request for Comments.* http://www.ietf.org/rfc.html. Retrieved July 12, 2012.

[7] Squid-Cache project website: http://www.squid-cache.org. Retrieved July 12, 2012.

[8] Greasyspoon project website: http://greasyspoon.sourceforge.net. Retrieved July 12, 2012.

[9] Postfix project website: http://www.postfix.org. Retrieved July 12, 2012.

[10] Javamail website: http://www.oracle.com/technetwork/java/javamail. Retrieved July 12, 2012.

[11] Sender Policy Framework website: http://www.openspf.org/. Retrieved July 12, 2012.

[12] Gmail Support website explaining conversations: http://support.google.com/mail/bin/answer.py?answer=5900. Retrieved July 12, 2012.