My Private Google Calendar and GMail

Tahmineh Sanamrad
ETH Zurich

Daniel Widmer
ETH Zurich

Lucas Braun
ETH Zurich

Patrick Nick
ETH Zurich

Donald Kossmann
ETH Zurich

Abstract

Although Cloud Applications provide users with highly available data services, they are missing privacy as a vital non-functional requirement. In this paper we leverage modern cryptography techniques to guarantee the user's privacy while the inherent functionality and portability of the cloud application remain intact. Our approach revolves on a transparent security middleware that sits between the user and the cloud service provider on a site trusted by the user. This layer accesses the request and response messages passed between the two parties in a fine-grained manner to preserve the functionalities. We implemented the methods and provide a middleware that allows users to keep their calendar information and E-Mail in an encrypted form using Google Calendar and GMail. Furthermore, we present the results of experiments with our middleware; these experiments show that the overhead to encrypt data on top of GMail and Google Calendar is negligible.

1 Introduction

Trust and privacy play a crucial role in today's applications, as more and more people and companies decide to outsource their data and IT services. There are plenty of cloud data services and web applications that help to organize and store data for free or at a low cost. Still, for some people high availability, up to date features, light-weight interfaces, backup, low maintenance costs and portability on the latest mobile devices seem all to be overshadowed by privacy doubts and suspicions.

This paper presents our experience on building a middleware to enforce privacy on top of two popular Web applications, namely Google Calendar and GMail. The goal is to use these two services without revealing any information to an attacker who has access to the Google Cloud (e.g., a Google system administrator) or who intercepts messages from or to the Google Cloud. We use a security middleware as a transparent encryption layer on a site trusted by the client. The key component in the security middleware is a proxy server that inspects and accesses the http message body, selectively encrypts/decrypts its content in a fine-grained manner, thereby preserving the original APIs. Recently in [9] a very similar approach to this paper has been suggested. However, the proxy server is installed on the client which restricts the portability but does not require an additional SSL termination step.

The main advantage of our architecture is transparency of the complicated encryption mechanism and key management for the end user. Also, given the variety of mobile devices that embed cloud data services, our

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

system assures a device-independent installation and thereby portability is guaranteed. As a proof of concept, we have developed a prototype of the calendar application for the most recent mobile devices.

To recover the missing functionality due to encryption, we compose a new scheme. The resulting scheme is a hybrid of a semantically secure encryption scheme and a keyword hashing scheme. The scheme has been analysed against certain adversary models. These models correspond to an honest but curious attacker that only by looking at the encrypted data residing on the cloud tries to infer the plaintext. The performance benchmarks show that generally the latency cost of a hybrid encryption mechanism is negligible.

Enforcing privacy and compensating the missing functionality could not have been done without a carefully planned component orchestration on the middleware. Along the way, many interesting problems have been addressed and new techniques have been devised. For example in order to add sender/receiver anonymization in the GMail and Google Calendar Invitation, an additional component namely a mail transfer agent has been added to the middleware, to offer additional privacy.

The remainder of this paper is organized as follows: In section 2 the architecture of our approach is being discussed and compared against other possible approaches. In section 3 and 4 the methods invented to tackle the privacy/functionality seesaw are discussed. Section 5 looks at the war stories that are still unresolved or need further treatments. In section 6, benchmark results will show an initial comparison between different possible encryption techniques. Finally, we conclude the paper in section 7 while discussing the related work and the future prospects of the project.

2 Proxy Architecture

In this section we will first explore different possible approaches to provide security for the end user given our case studies, Google calendar and GMail. The main idea is to replace the plain text content entered into the Google calendar's web interface with a ciphertext before submitting the request to Google. In order to accomplish this, there are two possible approaches as shown in Figure 1:

1. Having a rich client that takes care of the replacement
2. Adding a level of indirection between the client and the service provider

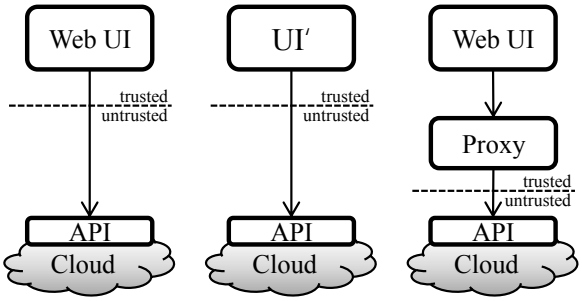
2.1 Rich Client vs. Proxy Middleware

As shown in Figure 1(a), currently the users directly use the API of the cloud service provider and submit their requests in plaintext. Although an SSL connection is established between the user and the cloud service provider, the data stored and processed on the cloud side is all in plaintext.

To have the data encrypted before submission, there are two possible solutions. The first solution is shown in Figure 1(b). In this approach, a new user interface is implemented on the client side. In addition to the new user interface, the security procedures are to be taken care of on the client. In our case study, Google actually provides a clean *Calendar Data API*that enables the users to access its calendar methods, such as `create()`, `edit()`, and `invite()`, while giving them a lot of flexibility to design their own desired user interfaces. A considerable amount of documentation has been written and dedicated to support developers who use this API. On the other hand, given the variety of the devices and hardware architectures, implementing such a user interface for each and every device out there is a cumbersome task with a high development cost. Additionally, the current user interfaces already provided by the cloud service providers are efficient and friendly enough. Another disadvantage is that the end user has to undergo a lot of installation and set up efforts on every single device she has.

The second approach is shown in Figure 1(c). This solution adds a level of indirection between the user and the cloud service provider. This level of indirection resides on the user's trusted site and is a proxy server that acts as an intermediary for requests from clients seeking resources from the cloud service provider. The first advantage of our approach is that the client accesses the proxy server using the same API as the one provided by the cloud service provider; this assures the transparency of the security mechanism to the end user, so the user

should not even be aware of the encryption process going on behind the scenes. The second advantage is the low installation cost for the user; all that the user needs to do is to route the relevant traffic through the proxy middleware. The third advantage is the portability of our approach. This way all the user's devices can keep their current well provided and maintained interface of the cloud service provider and only route their traffic through the proxy middleware.



(a) Traditional (b) Rich Client (c) Proxy MW

Figure 1: Comparing different approaches

2.2 Make it Work

As already discussed in the previous subsection, there are several advantages associated with the proxy architecture. Having a security middleware has been previously discussed in the database encryption research area in [3], or in some works related to the Internet data storage security such as [8]. However our security middleware consists of a proxy server that catches relevant http traffic, modifies the message content in a fine-grained manner with the help of a content adaptation server, and sends it further. The components to build such a system are as follows:

- *Proxy server*. As shown in Figure 2, the client connects to the proxy server on the middleware, requesting some service from the cloud service provider. The proxy server evaluates the request according to its filtering rules. If the request is validated by the filter, the proxy provides the resource by connecting to the cloud service provider and requesting the service on behalf of the client.
- *ICAP Server*. ICAP stands for Internet Content Adaptation Protocol(RFC3507) which is used to extend transparent proxy servers, as it is shown in Figure 2. This component adapts the content of the http messages by performing the particular value added service (content encryption/decryption) for the associated client request/response.
- *Mail Transfer Agent*. The MTA mainly acts as a relay for email and operates independently from the proxy and ICAP server. It will receive messages, encrypt/decrypt the message and send it further.

As mentioned in the introduction, the privacy of the users is guaranteed through encryption. However, privacy comes in cost of functionality. Therefore, the effect of every decision in one dimension (Privacy or Functionality) must be thoroughly examined in its counteracting dimension. In the next section, we look at Google Calendar and GMail scenario separately. For each scenario we name the main functions, and what does privacy imply. We devise a hybrid encryption scheme to deal with the defined privacy requirements. Two Adversary models will be introduced to analyse the advantage of an attacker that only has access to the encrypted data. Moreover in the GMail section we show how our approach tackles the sender/receiver anonymization problem.

3 Google Calendar

In this section we focus on the Google Calendar case study. First, we state what functionality of the calendar we are interested in. Then, we describe what does privacy imply in our system. After the preliminary definitions,

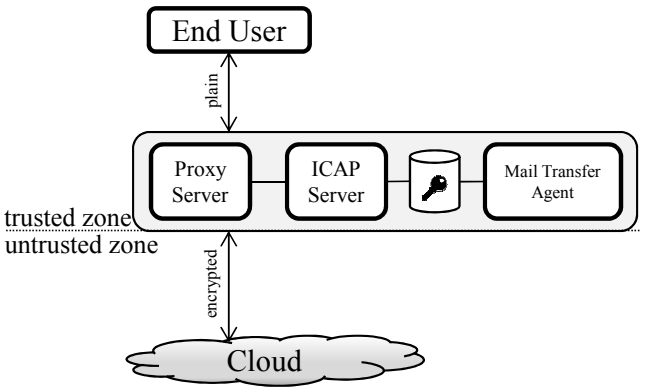


Figure 2: An overview of the architecture and components in our system

we show how the privacy is implemented and functionality is preserved.

Functionality. In the Google Calendar model the users interact with the cloud application mainly through `create()`, `display()` and `search()` functions. The users can also interact with each other through `share()` and `invite()` functions.

Privacy is defined as the inability of the adversary (e.g., an operator who has access to the Google cloud) to infer information about the events just by looking at the encrypted data. In our model with some probability the attacker will be able to guess some information, which is captured as the attacker's advantage (Section 3.2). The remainder of this section elaborates on the encryption scheme that is used to achieve this privacy and implement the Google Calendar functionality at the same time.

3.1 Hybrid Encryption Scheme

Assume we have a user Alice who wants to create and store a calendar event using an untrusted calendar application. The proxy server inspects Alice's calendar traffic (with her permission) and extracts the plain text pieces. The safest approach from this point on would be to replace the plaintext with a ciphertext generated by a semantically secure encryption scheme, such as AES operating in CBC mode [7]. Since the proxy is trusted by Alice, it will generate and store a key to be used for encryption. The randomized element of the encryption, namely the initialization vector (IV) is stored along with the ciphertext on the untrusted server to enable decryption at some point. However, if a probabilistic encryption scheme is used, one of the main calendar functions, namely search will be disabled, since the randomized element (IV) is missing to reconstruct a query that matches an entry in the cloud. Another challenge is that assume Alice has created an event called *"Bob Birthday"*, and she wants to search for *"bob"*. Even if we were using a deterministic encryption scheme, we were unable to retrieve the event because obviously Enc_K*("Bob Birthday")*<>* Enc_K*("bob")*.* This simple example shows the need of tokenizing and normalizing the user input plaintext as well as the search query. Therefore, to guarantee both search and exact retrieval of Alice's entries, we concatenate the list of normalized keywords of the event entry to the encrypted message. However, the keywords should also not leak any information and at the same time be searchable. There are different approaches on searchable encryption[1, 2, 4, 5], but the experiments we have performed in Section 6 shows that hashing is more efficient than encryption. Thus, we devise an idealized smoothing hash function. This hash function maps a keyword to a hash value. Based on a frequency histogram of the keywords kept in the main memory of the middleware, the hash function dynamically adjusts its assignments to perfectly smooth out the frequency of the hash values produced. Based on the frequency distribution of the input keywords, the degree of collision and multi-hashed values (one value has multiple hashes) in the system will be determined. Note that in order to create an idealized smoothing hash

function, we need a histogram that is built on the plaintext domain, $\mathcal{D}$. Construction 3.1.1 and figure 3 best describe our hybrid encryption scheme.

Construction 3.1.1: Let $\mathcal{SSE} = (\mathcal{IV}, \mathcal{E}nc', \mathcal{D}ec')$ be a semantically secure encryption scheme and $\mathcal{ISHS} = (\mathcal{T}ok, \mathcal{N}orm, \mathcal{H}ash, \mathcal{H}ist)$ be an idealized smoothing hash scheme on Domain, $\mathcal{D}$. We define our Hybrid scheme, $\mathcal{HS} = (\mathcal{K}, \mathcal{E}nc, \mathcal{D}ec)$ to be the following:

- $\mathcal{K}$ is a random function that generates a 128-bit key for the proxy, K_p.
- $\mathcal{IV}$ is a random function that generates a 128-bit initialization vector for the each message, $iv = \mathcal{IV}()$.
- $\mathcal{E}$nc gives the plaintext message m, to the $\mathcal{E}nc'$ function of SSE to generate the ciphertext, $c = \mathcal{E}nc'(K_p, iv, m)$. The random initialization value is first concatenated to the encrypted message, then the hashed keyword list, kl, generated by $ISHS$, $kl = \mathcal{H}ash(\mathcal{N}orm(\mathcal{T}ok(m)), \mathcal{H}ist(\mathcal{D}))$ is concatenated to the encrypted message as well. The encrypted message will be $c_m = c \| iv \| kl$.
- $\mathcal{D}$ec takes the proxy key, K_p, the ciphertext and the randomized part of the encrypted message, c_m; and, by using the $\mathcal{D}ec'$ function it revives the original message $m = \mathcal{D}ec'(K_p, iv, c)$.

Although a semantically secure encryption scheme encrypts the messages, the keyword list generated by the hash function weakens the security of our hybrid scheme. In the next section we look at two adversary models that analyse the advantage of the adversary given our definition of privacy in the beginning of Section 3.

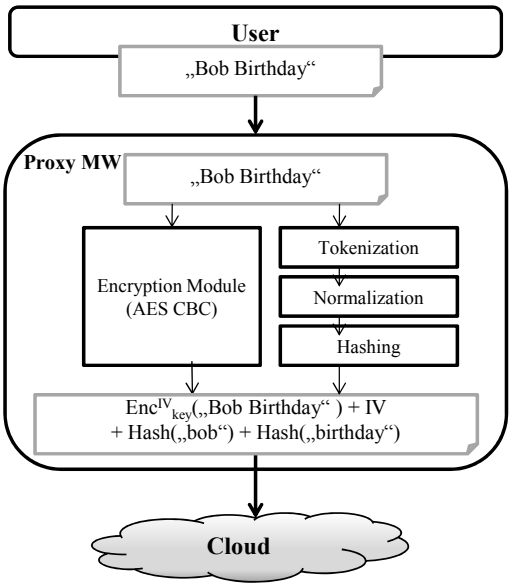


Figure 3: Hybrid Encryption Scheme

3.2 Adversary Models

In this section we introduce two security definitions, *Frequency Indistinguishability* and *Event Uncertainty*. We then analyse the advantage of the adversary in each model.

3.2.1 Frequency Indistinguishability

A deterministic scheme leaks the frequency distribution of the underlying plaintext which is not desirable. In order to show the resistance of our scheme against the frequency analysis of the keywords, we introduce a new security definition called *Frequency Indistinguishability*.
Frequency Indistinguishability: The advantage of an adversary in distinguishing pairs of ciphertexts just by looking at the frequency distribution of the messages they encrypt should be negligible. Let $\mathcal{HS}$ be our hybrid encryption scheme from Construction 3.1.1. For an adversary $\mathcal{A} = (A_1, A_2)$, we define its IND-Freq advantage as:

$$Adv_{\text{HS}}^{\text{ind-freq}}(\mathcal{A}) = Pr[Exp_{\text{HS}}^{\text{ind-freq-1}}(\mathcal{A}) = 1] - Pr[Exp_{\text{HS}}^{\text{ind-freq-0}}(\mathcal{A}) = 1]$$

For $b \in \{0, 1\}$ the experiments $Exp_{HS}^{ind-freq-b}(A)$ can be viewed in Experiment 1 and We say that HS is ind-freq secure if the ind-freq advantage of any adversary against HS is small.
Proof: The proof relies on two important procedures in the Experiment 1. First, $\mathcal{R}ebalance$, assures that M_0 and M_1 have identical histograms, i.e. the histograms are formed with the same number of buckets and the same frequency distribution. Second, the $\mathcal{S}ort$ sorts the result of our hashing scheme based on their frequency. Applying $\mathcal{S}ort$ on the hashes will map the output of $\mathcal{S}ort$ to the output of $\mathcal{R}ebalance$ in terms of frequency distribution. By definition the $\mathcal{R}ebalance$ procedure assures identical frequency distribution between the two

chosen frequency distributions by adversary. Hence, we can say that deciding to which frequency distribution the output of $\mathcal{S}ort$ belongs, is not better than a random guess. Therefore, the advantage of adversary $\mathcal{A}$, in experiment 1 is negligible.

Corollary 1: Given the above model we can conclude that in order to be safe against frequency analysis on our domain, $\mathcal{D}$, we need to $\mathcal{R}ebalance$ the histogram of our domain with a uniform frequency distribution. In other words,
$hist_0 \leftarrow \mathcal{H}ist(\mathcal{D});\ hist_1 \leftarrow \mathcal{H}ist(UNIFORM); \mathcal{R}ebalance(hist_0, hist_1)$

Algorithm 1 :$Exp_{\text{HS}}^{\text{ind-freq-b}}(A)$

$(\mathcal{M}_0, \mathcal{M}_1) \xleftarrow{\$} \mathcal{A}_1$
if $|\mathcal{M}_0| \neq |\mathcal{M}_1|$ **then return** $\perp$
$hist_0 \leftarrow \mathcal{H}ist(\mathcal{M}_0)$
$hist_1 \leftarrow \mathcal{H}ist(\mathcal{M}_1)$
$\mathcal{R}ebalance(hist_0, hist_1)$
let $m_1^j, m_2^j, ..., m_l^j$ be the elements of M_j for $j \in \{0, 1\}$
if $\exists i : 1 \leq i \leq l$ and $|m_i^0| \neq |m_i^1|$ **return** $\perp$
for $j = 1$ **to** l
| $h_j \leftarrow \mathcal{H}ash(m_j^b, hist_b)$
| $H_j^b \leftarrow h_j$
$H^b \leftarrow \mathcal{S}ort(H^b)$
$d \leftarrow \mathcal{A}_2(h_1, h_2, ..., h_l)$
return d

3.2.2 Event Uncertainty

We define another adversary model that is only applicable to calendar data. This adversary takes advantage of the repetitive pattern or length of certain event. By applying additional background knowledge, the adversary might be able to make strong guesses about certain events that the user is likely to attend. For example a yearly event is most likely to be an anniversary (e.g. birthdays) or adversary knows that the user is a professor and is most likely to attend a certain conference on certain days. In order to decrease the strength of the adversary's guesses we add noise to our system. Therefore, we introduce a new security definition called *Event Uncertainty*.
Event Uncertainty. Given a time interval what is the advantage of an adversary in guessing whether an event is real or not. Let I be the interval, I_{all} be the set of all events in I, and I_{real} be the set of real events in I. Let $e \in I_{all}$ be an event, we define a function $\tau(e)$ to return the duration of an event. Hence, the advantage of the adversary in the interval of I will be:

$$Adv^I(\mathcal{A}) = \frac{\sum_{e \in I_{real}} \tau(e)}{\sum_{e \in I_{all}} \tau(e)} \tag{1}$$

The naive way of adding noise is by random. We believe, however that there are more clever ways to add noise to the encrypted data to break repetition patterns or fuzzify the duration of certain events. For example in case of a birthday event, changing it to a monthly event would conceal its yearly pattern. Nevertheless, we have not developed a concrete model to optimally add noise to the calendar data. Creating event uncertainty could also be done by changing date and time of an event. Unfortunately, this approach is hard to implement, because of the prefetching procedure going on in the background of the calendar page.

4 GMail

In addition to Google Calendar, we studied the proxy architecture to achieve privacy on top of GMail. GMail also provides a sophisticated API and a Web interface and it involves confidential information that we would like to protect from *honest and curious* adversary that has access to the Google cloud.

Functionality. In GMail, the users interact with the cloud application mainly through `compose()`, `send()`, `search()`, and `receive()` functions. In addition, GMail provides functionality to filter spam, group conversations, and to spell-check.

Privacy is again defined as the inability of the adversary to infer information just by looking at the encrypted emails. The information we want to hide is the sender, recipient, title and message body. The message body is encrypted using the same hybrid encryption scheme discussed in Section 3.1. Concealing sender and recipient from the Google Mail Server undermines the main functionality of a mail server. In the next section we will show how to resolve this issue.

4.1 Sender and Recipient Anonymization

In this section, we suggest a sender/recipient anonymization technique that reduces GMail to be solely a storage engine and an email management interface. Our method is explained through an example. Assume Alice is a user of our proxy's mail service. Her gmail account is *alice@gmail.com*. The proxy, assigns another email address to Alice, called *alice@proxy.ethz.ch*. This email address is in fact the address which Alice can be contacted by other people. However, in order to access and operate her email account:*alice@proxy.ethz.ch* she needs to login to her gmail account, *alice@gmail.com*. Now assume Alice wants to send an email to Bob. Bob is not a proxy user, thus he cannot read encrypted contents. As shown in figure 4, Alice logs into her GMail account, composes an email with Bob's address in the recipient field and presses send. What happens in the proxy is that the email content and recipients are extracted, the content and actual recipients are encrypted and stored in the message body. Moreover, the recipients are replaced by a random recipient residing on the proxy's mail server. The message will be stored on Google servers, but now it is sent back to the proxy instead of Bob. This time the mail transfer agent on the proxy receives the message and decides what to do with the content. In this case since Bob is not a proxy user, the message will be decrypted. Bob's address is extracted from the message body and the message is sent by *alice@proxy.ethz.ch*. This approach guarantees recipient anonymization.

Now let us walk through a scenario in which a plaintext message is sent by Bob to Alice. In order for Bob to send a message to Alice, he needs to use *alice@proxy.ethz.ch* as her address. The mail server on the proxy receives the message, encrypts it, and sends it to Alice's gmail account, *alice@gmail.com*. Alice can simply access her inbox by logging in to her gmail account, and the proxy guarantees that Alice sees all her emails in plain text. This approach guarantees sender anonymization [10].

5 War Stories

In this section we look at the challenges that are still unresolved or can be more gracefully done in our system as future work.

GMail Spam Filter. The spam filter feature of GMail inspects the sender, content and subject of an email. Encrypting emails will disable it. A possible solution is to implement a Spam-filter by adding a content-based filter to the mail server on the proxy middleware[10].

GMail Conversation Grouping. A very convenient feature of the Gmail web interface is that emails get grouped into conversations if there are many emails being sent back and forth between certain people. This feature improves the inbox organization of the user. There are two conditions for this grouping to happen: the sender/recipient addresses must match and the subject line must be equal apart from the well-known prefixes

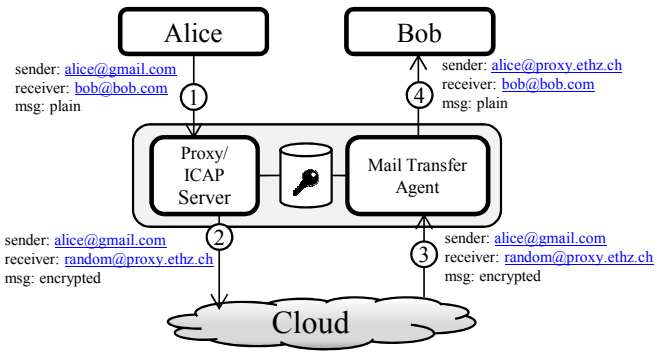


Figure 4: Sending an Email

like ”Re:”Our encryption 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 different subjects; therefore, Gmail is unable to group them into a conversation. To solve this we need to use a deterministic encryption for the title and recipients. Using a deterministic encryption has its own pitfalls and is prone to frequency analysis.

SSL Interception. Google, like other secure web applications, uses SSL (Secure Socket Layer) protocol which encrypts the segments of network connections above the Transport Layer, using asymmetric cryptography for privacy and a keyed message authentication code for message reliability. Normally, a proxy server should not read the content of the message as an intermediary between the Google and end user. Nevertheless, some proxy servers offer options to decrypt SSL traffic and allow transparent SSL traffic redirection; thus, instead of having an encrypted SSL tunnel between the end user's browser and Google's server, our proxy server terminates the Google's SSL traffic at the proxy level. Sequentially, the ICAP server extracts the content to be encrypted and reconstructs the HTTP message on its way to the Google server. The adapted content is then sent to the end-user, while presenting a forged SSL certificate to the user's browser. In other words, our middleware basically performs something similar to a Man in the middle attack, but the big difference is that the user agrees to give our middleware the permission to access its contents. The user can give the permission by either confirming a certificate exception on the browser for the first time visiting the Google domain, or installing our root CA Certificate in the trusted root Certificate Authority list of the browser.

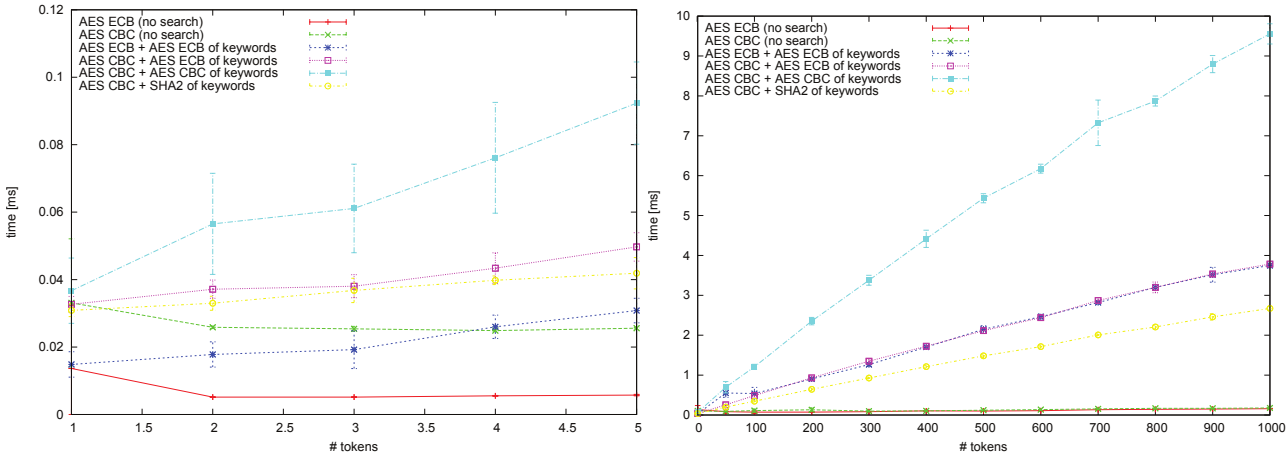
Query Log Attack To perform search and at the same time be safe against frequency attacks and have event uncertainty, we have added collision, multi-hash values and noise to our encrypted messages. Each of them have its own consequences. Adding collision, will cause the search result to retrieve more than expected, but our system easily eliminates false positives on the security middleware. Having multi-hash values, however will cause the proxy to submit a disjunctive search request, which reveals the connection between the keywords and eventually leaking the frequency distribution. Last but not least, noise added to the encrypted data will never be searched for; thus, it also leaks information about what events are fake. Solving these problems remains a future challenge.

Chosen Plaintext Attack. So far we have only analysed an honest but curious attacker, that only tries to infer the plaintext by looking at the ciphertext. A well-known adversary model that has been neglected so far, is an attacker that can use the proxy server and has access to the encrypted data on the untrusted cloud. This strong adversary is able to perform adaptive chosen plaintext attacks on the proxy. However, it can be easily stopped by assigning each user its own key and hash function, instead of using a proxy-wide one.

Key Management. Using multiple keys per user adds security, but on the other hand complicates the searchability of the calendar and again is vulnerable against query log attacks because of submitting a disjunctive search query. In case of sharing a calendar, the middleware needs to be able to deal with giving and revoking keys to and from other users.

6 Experiments

In the previous section we have shown that some level of privacy can be achieved while preserving the key functionality of the cloud application. However, security comes with a cost. In this section we will look at the encryption cost using different encryption and hashing methods.[1] The goal of these experiments is to show the cost of keyword extraction vs. no keyword extraction, and also hashing vs. encryption. As a baseline we use a deterministic encryption scheme (AES ECB) and a probabilistic encryption scheme (AES CBC) without keyword extraction. We then add the keyword extraction phase and measure the cost of hashing vs. encryption. In several previous work such as [9, 6] a symmetric encryption of the keywords has been proposed. Therefore, we have also included AES ECB + AES ECB encryption of the keywords to represent a pure deterministic encryption, AES CBC + AES ECB encryption of keywords to represent the probabilistic encryption of messages and deterministic encryption of keywords, and AES CBC + AES CBC of the keywords to represent probabilistic encryption of both messages and keywords, to cover all the schemes proposed by previous work.



(a) Encryption Time of Events

(b) Encryption Time of Documents

Figure 5: Comparison of different encryption methods

The following conclusions are to be drawn from our experiments.

Conclusion 1) By looking at figure 5 we can see that generally the latency introduced by applying security modules is considered to be small, in order of milliseconds.

Conclusion 2) In graph 5(a) we can see that in most of the cases the latency does not scale with the small number of tokens, whereas in graph 5(b), we clearly see that the latency increases linearly with the message size. This fact shows that in very small documents the cost of message encryption dominates the cost of keyword extraction and hashing (or encryption), but in bigger documents, the cost of keywords extraction overshadows the cost of the message encryption.

Conclusion 3) In graph 5(b) we can see that deterministically encrypting the keywords in ECB mode is much faster than having a probabilistic encryption scheme for the keywords.

Conclusion 4) Finally, in graph 5(b) we can see that hashing the keywords is much faster than symmetrically encrypting them in any way (ECB or CBC). The security implications introduced by hashing have been discussed in section 3.1.

Please note that these experiments are not showing the latency of our system. Given that google has unknown flow control mechanisms, performing scalability benchmarks is a challenge and is left for future work.

[1] The experiments were conducted on a Lenovo Thinkpad T400 having an Intel Core Duo cpu clocked at 2.80 GHz, 4 GB of RAM and ubuntu 12.04 as operating system.

7 Conclusion

This paper describes the system we have implemented to solve privacy issues in web applications by having a transparent encryption layer. The goal is to preserve the advantages of cloud-based web services (i.e., low cost, no administration, great user experience) without sacrificing privacy, performance, functionality and portability. The paper showed how this goal could be achieved for the Google Calendar and GMail service. A proxy architecture was devised and a number of new techniques were implemented in order to preserve the Google Calendar and GMail functionality on encrypted data. In particular, a new encryption scheme was presented that allows to search on the encrypted data. Experiments also support the fact that the proposed security scheme is more efficient among the other suggested schemes from previous works. Additionally, Performance experiments showed that the latency impact is tolerable.

There are several avenues for future research. First, we would like to apply our approach to other Web Applications such as Google Contacts and Google Docs and the services provided by other providers (e.g., Microsoft, Yahoo, Amazon, etc.). A number of new technical challenges need to be addressed to support all the features of these services, but the general approach and the proxy architecture should still be applicable.

7.1 Related Work

The proxy architecture has been also recently proposed by [9]. However, in their approach the proxy resides on the client machine; therefore limiting the portability. Also in our paper we have devised a new way to perform search on encrypted data. This topic is not new. There has been several related work in this area. In [4] they have a very similar use case, in which the user wants to search for a keyword in her encrypted emails. Their solution suggests that sender encrypts every keyword in her mail with the user's public key. Another work [1] suggests several encryption schemes to enable search on encrypted documents. In[2] they came up with secure indexes to enable search on encrypted data, but these approaches [1, 2, 4, 5, 6] are based on the fact that the untrusted server is programmable and can implement our search mechanism and data structure, whereas in our case we know almost nothing about how search is implemented on the Google servers.

Another set of related work is iDataGuard [8] which is an interoperable security middleware for untrusted Internet data storage. Their main goal is to adapt to heterogeneity of interfaces of Internet data providers and enforce security constraints. They also allow search on encrypted data using a special indexing technique. However, they search at a file-level granularity whereas we provide fine-grained encryption to preserve privacy of more complicated web application than just pure data storage.

References

[1] Dawn X. Song et al., *Practical Techniques for Searches on Encrypted Data* 2000: IEEE Symposium on Security and Privacy.

[2] Eu-Jin Goh., *Secure Indexes* 2003: IACR Cryptology ePrint Archive.

[3] Ernesto Damiani et al., *Balancing confidentiality and efficiency in untrusted relational DBMSs* 2003: CCS.

[4] Dan Boneh et al., *Public Key Encryption with keyword Search* 2004: EUROCRYPT.

[5] Yan-Cheng Chang and Michael Mitzenmacher *Privacy Preserving Keyword Searches on Remote Encrypted Data* 2005: ACNS.

[6] Reza Curtmola et al., *Searchable symmetric encryption* 2006: CCS.

[7] Jonathan Katz and Yehuda Lindell *Introduction to Modern Cryptography* 2007: Chapman & Hall/CRC.

[8] Ravi Jammalamadaka et al., *iDataGuard: an interoperable security middleware for untrusted internet data storage* 2008: USENIX.

[9] Mamadou H. Diallo et al.,*CloudProtect:Managing Data Privacy in Cloud Applications* 2012:IEEE Cloud.

[10] Patrick Nick *Encrypting Gmail* 2012: ETH Zuerich Master Thesis.