SEVENTH FRAMEWORK PROGRAMME

CloudSpaces

(FP7-ICT-2011-8)

Open Service Platform for the Next Generation of Personal Clouds

D4.3 Final release of privacy and interoperability components

Due date of deliverable: 30-09-2015
Actual submission date: 06-10-2015

Start date of project: 01-10-2012 Duration: 33 months

Summary of the document

Document Type	Deliverable
Dissemination level	Public
State	Final
Number of pages	50
WP/Task related to this document	WP6
WP/Task responsible	EPFL, URV
Author(s)	Rameez Rahman, Hamza Harkous, Marc Sánchez, Cristian Cotes
Partner(s) Contributing	EPFL, URV
Document ID	CLOUDSPACES_D2.4_141031_Public.pdf
Abstract	PrivySeal for risk mitigation of Personal Cloud users, Effective Risk Communication Models with Novel Risk Insights. It is also presented the privacy-aware data sharing with Attribute Based Encryption and its integration in StackSync.
Keywords	Cloud storage, synchronization, sharing, interoperability, Personal Cloud

Table of Contents

1 Executive summary

This deliverable is divided in two sections. In the first one, we present PrivySeal and in the second one, we detail Attribute Based Encryption and its integration with StackSync.

In the first section, we present novel rich communication models and a privacy enhancing technology that has been tested with over a thousand users. For risk communication models, as a case study, we consider the third party app ecosystem for the personal Cloud. Personal cloud services such as Google Drive and Dropbox allow their users to install a variety of attractive 3rd party apps. These 3rd party apps require access to the user's data in order to provide some functionality. Through an analysis of a hundred popular apps on Google Chrome store, we discover that the existing permission model is quite often misused: over *two thirds* of analyzed apps access more data than is needed for them to function. Further, we discover through user experiments that simply telling users that apps are accessing more data than they require does not discourage them from installing those apps. This poses two interesting challenges: a) can novel risk communication models be developed that can discourage users from installing privacy infringing apps? b) can we do a comprehensive analysis of different risk communication models to gain deeper insights into what influences users decisions in installing apps? To address these challenges, as our first contribution, we present an ensemble method that we call Personalized Insights. Models in Personalized Insights inform the users about the data-driven insights that apps can make about users (e.g., their sentiments towards entities, collaboration and activity patterns etc.). We compare the various models within Personalized Insights and discover several interesting properties of different risk communication models. We also find several models that are, on average, *twice* as effective as the current model in discouraging users from installing privacy infringing apps. We also integrate Personalized Insights in the scenario of user-to-user sharing privacy in **StackSync**. Based on the knowledge extracted from real users' data (over 112 gigabytes of Google Drive data from 1350 users), as our second contribution, we develop a privacy-oriented app store, PrivySeal, in which users can see privacy infringing Google Drive apps, and the far reaching insights these apps can make about them, and thus choose apps judiciously. PrivySeal can be accessed at this URL: **http://privyseal.epfl.ch**.

In the second part of this document, we discuss the problem of privately sharing folders with users, and how this challenge has been addressed by novel crypto primitives like Attribute-Based Encryption (ABE). With ABE, we have been able to provide fine-grained access control at the file level without sacrificing scalability. Concretely, we have associated each file with a set of attributes, which represents the file's encryption/decryption policy. A private key is associated with a monotonic access structure like a tree, which describes the user's permission (e.g., *Medical* **AND** (*Doctor* **OR** *Nurse*)). Then, a user can decrypt the file if and only if her access tree is satisfied by the file's attributes. Since the number of attributes is significantly lower than the number of shared files, the key management is simplified without impairing scalability. Next, we describe how ABE has been integrated into **StackSync**, including all the changes both at the metadata and storage backends. Finally, some preliminary results have been obtained that verify that the overhead is comparably small. The complete source code can be found at this URL: **http://github.com/stacksync**.

2 PrivySeal: Effective Risk Communication for Privacy Aware Sharing in the Cloud

Cloud services such as Google Drive, Dropbox, OneDrive etc., have become increasingly popular over recent years. At the same time, such services have raised privacy concerns about user's data. But the danger is graver than it appears. While such Cloud services are few in number, and at least have clearly defined privacy policies, they also serve as platforms and allow a myriad of 3rd party apps to work on top of users' data. These 3rd party apps provide certain functionalities to users, for which they require access to the users' data. Put simply, users sacrifice some of their privacy in order to get functionality that such apps provide. However, it appears that often such apps acquire more data than is needed for them to function.

As a case study, we did an analysis of a 100 third-party apps on one of the most popular personal Cloud services, namely Google Drive (240 million active users in 2014 [1]), and discovered that a staggering 68% of these apps require more permissions than they actually need for functioning. Furthermore, these apps usually have no clearly defined privacy policies. Thus, users often end up exposing more data than is needed to unaccountable apps. For instance, a user's favorite PDF converter is highly likely to get access to her music library and discover her taste in Mozart or obtain her geo-tagged photos and know where she went on the weekend. Throughout this document, we refer to such apps as *misbehaving apps*. As observed in other third-party apps ecosystems, giving such misbehaving apps superfluous access can potentially result in users' data being abused. This has recently been the case in the health apps market where the top 20 most visited apps were found to be sharing users' data with 70 analytics and advertising companies [2].

It can be discerned that whenever people share data, whether with 3rd party apps or with other people, it is the need of the hour to guide them about the risk that is posed to them because of their sharing activities. Towards that end we make the following specific contributions:

i. Popularity, rating and reputation are bad indicators of misbehavior: We review hundred third-party Google Drive apps and discover that more than two-third acquire more data than they need. We also learn that popularity and rating are unreliable indicators of app misbehavior. This has deep implications for all systems that depend on user or content reputation/rating. (Section 2.2).

ii. Telling users that they are at risk falls on deaf ears: We develop a simple risk communication model called *Simple Risk Notifications* that informs users about the unneeded permissions that misbehaving apps are using. Through user experiments, we discover that this model fails to deter users from installing misbehaving apps. Put bluntly, *telling users that their privacy is being infringed does not help.* This motivates us to ask whether **novel risk communication models can be devised that can inform users about misbehaving apps and discourage them from installing such apps?** Furthermore, we want to analyze the factors that can play a role in user's decision to install misbehaving apps and the influence different risk communication models can have on that decision (Section 2.3).

iii. Shocking them with intimate details, however, is another story: Towards that end, we present an ensemble method called Personalized Insights, inspired by the novel concept

of Inverse Privacy [3]. Inverse privacy refers to the situation when a user is not aware of the information that an external entity has on the user. Based on this definition, Personalized Insights communicate to users the immediate and far-reaching insights which can be inferred by the apps using superfluous permissions. These include but are not limited to: users sentiments towards people and things; user collaboration and activity patterns; the people, locations, and concepts that appear in users' photos, etc. Through user experiments, we show that Personalized Insights are *twice* as effective in deterring users from installing misbehaving apps as the current model. Our analysis reveals various factors that can deter users from installing misbehaving apps. For instance, we discover that insights which reveal users' *relations with other people* reduce by half the installation of misbehaving apps, as compared to insights that simply reveal information about the *users themselves* (Sections 2.4 and 2.5).

iv. PrivySeal helps users safeguard their data: We present PrivySeal a privacy aware app store that uses Personalized Insights to warn users about misbehaving apps. This store is available for public use at: `https://privyseal.epfl.ch` and has been used by over 1350 registered users. (Section 2.6).

2.1 Third-party Cloud Apps Ecosystem

2.1.1 General Model:

There are three entities that interact in the third-party Cloud app system: (1) a *developer* who programs and manages a *third-party application*, (2) a *user* who uses that application for achieving a certain service, and (3) a *cloud provider* at which the user's *data* is stored. Using the Cloud provider's API, the application gets access to a subset of the user's data after *user authorization*, which is based on the user accepting a list of *permissions* that determines this subset.

2.1.2 Threat Model

Upon using third-party Cloud apps that access their data, users sacrifice some of their privacy for getting some service(s). This tradeoff between privacy and services has been called the Privacy vs Services Dilemma in the literature [4]. However, as we will see in the next section, there are many apps that require more permissions than are needed for them to function. We call such apps *misbehaving apps*, as opposed to *well-behaving apps* that only request the permissions needed for their functionality.

Misbehaving apps might require more data for one of two reasons: (a) developers not having the option to ask for any less (from the Cloud provider's API); or (b) simply being data greedy and requesting for as much data as can be harvested. Regardless of the reason, these apps pose a *risk* which can be potentially exploited, e.g., by selling data to 3rd party advertising providers. Furthermore, the user cannot be held responsible for sacrificing their privacy in lieu of some functionality, as the same functionality could be provided by an app that requires less data. In this work, we seek to combat the risk posed by these misbehaving apps through improving the *risk indicators* that users are presented with during the authorization process.

Table 2.1a Permissions requested with the short name we use for reference

Permission	Short Name
View the **files** in your Google Drive.	DRIVE_READONLY
View and manage the **files** in your Google Drive.	DRIVE
View **metadata** for files in your Google Drive.	DRIVE_METADATA_READONLY
View and manage **metadata** of files in your Google Drive.	DRIVE_METADATA
View and manage Google Drive files that you have **opened or created with this app**.	DRIVE_FILE
View your Google Drive apps.	DRIVE_APPS_READONLY
Add itself to Google Drive.	ADD_DRIVE
View and manage its own configuration data in your Google Drive.	DRIVE_APPDATA

2.1.3 The Case of Google Drive

Towards that end, we have taken Google Drive as a case study, and we have anatomized this ecosystem in detail. Nevertheless, as we discuss later, the insights gained from our analysis are applicable to other Cloud platforms as well. To begin with, any developer can register an application that accesses Google Drive API at Google Developers Console[1] for free. She then receives a Client ID and Client Secret that need to be included in the application code to access Google APIs. The developer can then specify in the code a set of Google permissions (a.k.a. scopes) she wants to obtain.

The application itself can be hosted on any website the developer chooses; i.e., it is not hosted by Google itself. The developer can also submit a request for featuring the app on Google Chrome Web Store, which has a section for apps that work with Google Drive. In the store, apps are presented along with screenshots and descriptions of their functionality (provided by the developer). The store also allows users to rate and review applications. Apps can be also submitted to other web stores hosted by Google, such as the Add-ons Stores for Google Docs, Google Sheets, or Google Slides and the Google Apps Marketplace for enterprises.

An application can request permission to access Google Drive data at any time of its operation, and not necessarily at the beginning. For example, the user can be presented with a button in a side menu that reads "Import file from Google Drive", and clicking on this button redirects to a Google-hosted page that presents the set of permissions requested by the application, as shown in Figure 1. As we see later, the absence of a standard location and interface for hosting apps and triggering the permissions request is one of the reasons that makes the automated, large scale privacy analysis of apps infeasible. The main permissions pertinent to Google Drive are presented in Table 2.1a.

As far as files' data is concerned, an app can request access to all files (DRIVE and DRIVE_READONLY permissions) or on a per-file basis (DRIVE_FILE). In the latter case, the explicit approval for each new file(s) is mediated by an interface provided by Google. For example, the developer can present the user with a file picker popup (hosted by Google) so that she can select (and thus approve access to) the file. Alternatively, the file can be

[1] `https://console.developers.google.com`

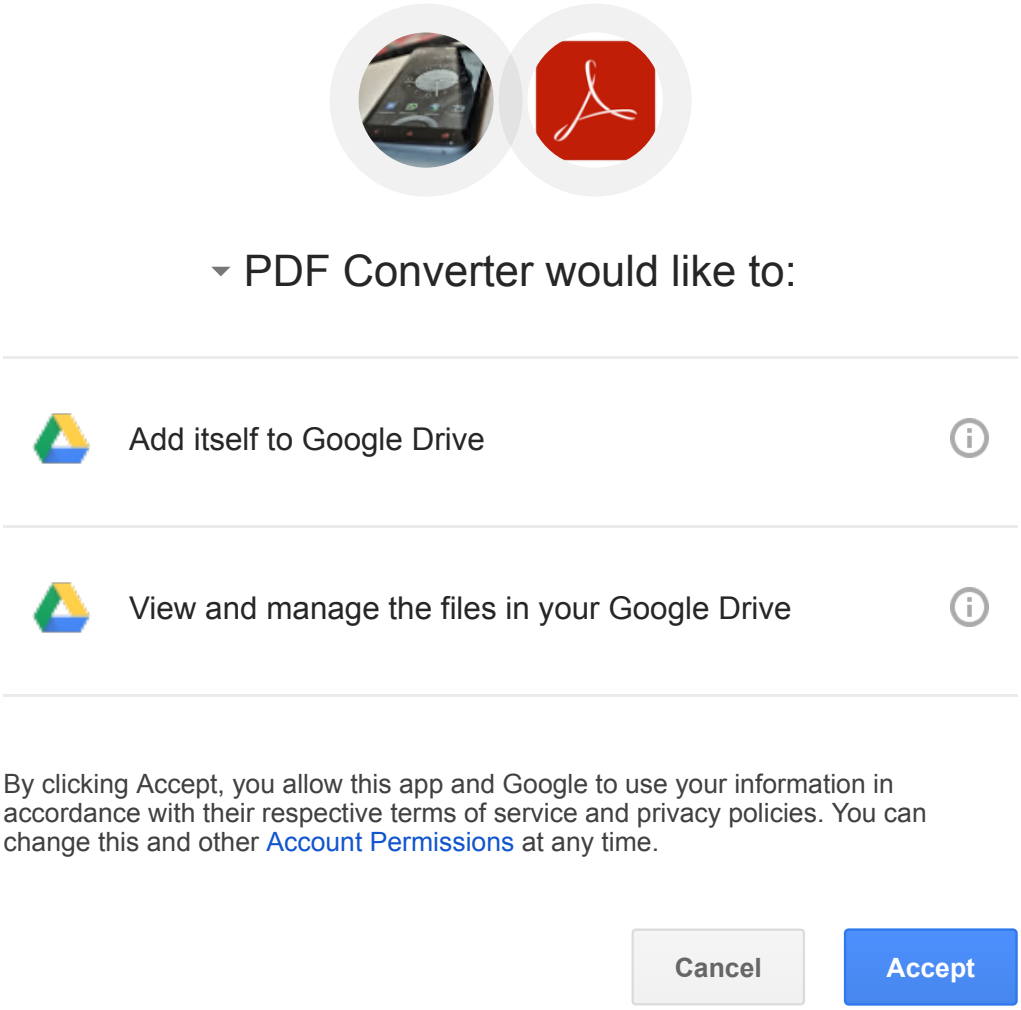


Figure 1: Example of the current permissions interface of Google Drive

opened from Google Drive's interface via the "Open with" option in the context menu of the file. In the case of full access, an app can access any file directly via Google Drive API without the need for user intervention. For example, this type of access enables an app to list all the user's files and download them in the background.

In addition to accessing file data, the developer can request access to file metadata (DRIVE_METADATA and DRIVE_METADATA_READONLY) or to the list of apps the user has authorized before (DRIVE_APPS_READONLY). It is worth noting that the permission list is not limited to Google Drive API and that it typically includes permissions from other Google APIs, such as access to user's profile information, email address, contacts list, etc.

2.2 Investigating the Privacy Risk posed by 3rd Party Google Drive Apps

As we can see from the previous section, there are no fine grained permissions that pertain to file contents between the two extremes of full access and per-file access. Hence, one could encounter a lot of applications that request more data than needed, e.g., by requesting full access, although their functionality might not necessitate such a level. Accordingly, the

access to user's files ends up in the hands of various parties, bearing the risk of intentional or unintentional disclosure. A lot of the highly used 3rd party applications do not have privacy policies or justifications of the requested permissions. Moreover, the users are not usually aware of the API details or the application functionality, especially before installing the apps. Thus, the choice of installation is not well-informed from a privacy perspective.

2.2.1 App Study

The question that comes next is: **"what is the extent of risk that actual users are exposed to?"** To answer this, we examine a sample of third-party Google Drive apps to determine the percentage of apps that requests extra permissions. We proceed to Google Chrome Web Store, which has a section for apps that work with Google Drive[2]. The store features apps on its main page, that change with time. We selected 100 featured applications from this page, and we manually reviewed them one by one.

2.2.2 Review Process

An app's review process starts by going to the application website, linked from the store, and testing the application manually. For each application, we first find the step where the app connects to Google Drive (if this is not upon the initial sign up). Then, we authorize the app to access a test Google Drive account created for this purpose.

We then record the following information:

1. all the requested permissions by the app
2. the permissions requested but not needed by the app
3. the alternative permissions provided by Google that the developer could have used instead of the ones in (2)
4. the alternative permissions not provided by Google that the developer could have used instead of the ones in (2)

The first step above is obvious. The second step requires (a) knowledge about Google Drive API and (b) investigating the ways in which the application accesses the user's Google Drive data, and the data it actually requires. As explained previously, when the approval for each file is always mediated by an interface provided by Google, then the per-file access level is sufficient for the functionality. If the application implements its own file browsing interface, as in the case of a custom photo browser, then the developer cannot use the per-file access. In that case, with the current Google Drive API, the only choice is to request full access. In the third step, we note down the alternative Google Drive permissions that the developer could have used. In the fourth step, we note down the permissions that the developer could theoretically use but is not able to since they are not currently provided by Google Drive. For instance, a photo editing app could use permissions that allow accessing photos only, if Google Drive were to implement this permission. During the review, we had

[2] `https://chrome.google.com/webstore/category/apps?_feature=drive)`

Table 2.2a Suggested permissions not offered by Google Drive

View photos in your Google Drive.
View and manage photos in your Google Drive.
View documents in your Google Drive.
View and manage documents in your Google Drive.
View files of specific type(s) in your Google Drive (other than images and documents).
View and manage files of specific type(s) in your Google Drive (other than images and documents).

a set of alternative permissions (presented in Table 2.2a) that are not currently provided by Google, and we report whether some of these permissions would be sufficient for replacing the actually requested permissions.

2.2.3 Review Results

Permission Usage:
Analyzing the application reviews, we found out that 68 out of 100 apps request unneeded permissions (Table 2.2b). In **64 of these 68 apps**, the developers could have requested less invasive permissions with the current API provided by Google. Had the more fine-grained permissions suggested in Table 2.2a been available, the 4 remaining apps could also have requested less invasive permissions. In total, 76 out of the 100 apps requested full access to the all the files in the user's Google Drive. Interestingly, **9 of every 10 apps** requesting full access are misbehaving.

As we show in Table 2.2c, the top permission that is needlessly requested is the full read and write access to Google Drive (in 55 apps), followed by the full read access permission (in 17 apps). This further increases the magnitude of data that can be exploited with the extra permissions. On the other hand, the per-file access permission is the top permission that is actually needed when requested. This happens in 41 of the apps. However, in **16 of these 41 apps**, we have found that the developer also requested full access to the user's data. We conjecture that this could be aimed at deceiving the user into installing such applications due to the presence of per-file access. Table 2.2d shows the distribution of alternative permissions that could replace the extra permissions needlessly requested by the apps. It is evident that the top alternative is the per-file access. This indicates that despite its limitations, simply the correct usage of the current Google Drive API (which does provide per-file access), can eliminate the major part of the privacy risk. Nevertheless, it is evident that developers are generally guilty of not doing this.

As we show in Table 2.2c, the top permission that is needlessly requested is the full read and write access to Google Drive (in 55 apps), followed by the full read access permission (in 17 apps). This further increases the magnitude of data that can be exploited with the extra permissions. On the other hand, the per-file access permission is the top permission that is actually needed when requested. This happens in 41 of the apps. However, in **16 of these 41 apps**, we have found that the developer also requested full access to the user's data. We conjecture that this could be aimed at deceiving the user into installing such applications due

Table 2.2b App Reviews Statistics

Metric	Value
Misbehaving apps	68/100 (68%)
Misbehaving apps due to Google Drive API limitations	4/68 (≈ 6%)
Misbehaving apps due to Developers' fault	64/68 (≈ 94%)
Apps with full access	76/100 (76%)
Misbehaving apps with full access	68/76 (≈ 89%)

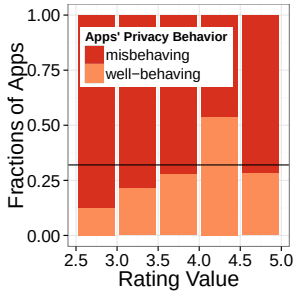


Figure 2: App misbehavior with different ratings values

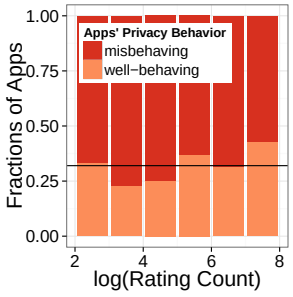


Figure 3: App misbehavior with number of ratings on a log scale

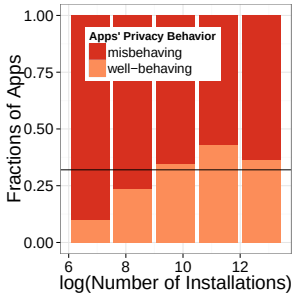


Figure 4: App misbehavior with number of installations on a log scale

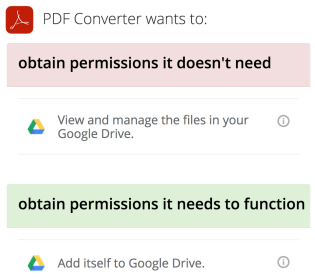


Figure 5: Example of Simple Risk Notifications interface

to the presence of per-file access. Table 2.2d shows the distribution of alternative permissions that could replace the extra permissions needlessly requested by the apps. It is evident that the top alternative is the per-file access. This indicates that despite its limitations, simply the correct usage of the current Google Drive API (which does provide per-file access), can eliminate the major part of the privacy risk. Nevertheless, it is evident that developers are generally guilty of not doing this.

Current Risk Indicators:
As we have reviewed apps from the Chrome Store, we were also able to extract, for each app, the *average rating*, the *total number of ratings*, and the *total number of installations*. These numbers are part of what is visible to the users when they are deciding on the app installation and can be perceived as indicators of trust. To determine the validity of this perception, we investigated the correlation between these metrics on one hand and the apps' (mis)behavior on the other.

For each of these metrics, we calculated the point biserial correlation coefficient[3] between the metric on one hand and a dichotomous variable B indicating whether the app is misbehaving. We found out that average rating metric is weakly correlated (0.189), indicating that the probability for the app to be well-behaving slightly increases with higher ratings. Moreover, the number of ratings has a weak positive correlation (0.256) with B. Finally, there is almost no correlation (0.082) between the number of app installations and B.

From this data, it is not immediately clear whether users can depend on a higher rating, for example, as a reliable indicator that the app does not request extra permissions. We dig

[3]This correlation coefficient is typically used to estimate the degree of relationship between a dichotomous nominal scale and a numeric scale.

Table 2.2c Permissions' Usage in Reviews

Permission	Unneeded	Needed
DRIVE_READONLY	17	1
DRIVE	55	7
DRIVE_METADATA_READONLY	2	1
DRIVE_FILE	0	41
DRIVE_APPS_READONLY	1	1

Table 2.2d Alternative Permissions Possibilities

Alternative Permission	Count
DRIVE_FILE.	48
DRIVE_METADATA_READONLY	1
View photos in your Google Drive.	1
View files of specific type(s) in your Google Drive.	2
View and manage files of specific type(s) in your Google Drive.	2

deeper into this issue by drawing the histogram showing the fraction of misbehaving apps at each rating interval (Figure 2). In this, as well as the next two graphs, we ignore the histogram bins in which we had less than 5 apps. The horizontal line at $B_{av} = 0.32$ in each of the graphs denotes the average proportion of misbehaving apps in the whole sample. In Figure 2, it can be observed that still nothing definitive can be said about whether or not an app is well-behaving through just taking its rating into account. In most cases, no matter what the rating, at most only 1 out of every 4 apps is well-behaving. Even in the best case scenario (of rating 4-4.5), the proportion of well-behaving apps is still only approximately 0.5. This suggests that even with this rating, it is a coin toss for the user to know if an app is well-behaving.

In Figure 3, we consider the case of ratings count for apps, represented on a natural log scale. The number of ratings has a weak correlation (0.256) with B. As in the case of the previous graph, it can be observed that the number of ratings also cannot help the user in determining whether or not an app is misbehaving. We can see that even at a very high number of ratings, the proportion of well-behaving apps remains below 0.5. In Figure 4, we show the same type of graph, with the natural log scale of the number of times an app has been installed on the x-axis. The correlation is almost absent as is evident from the fact that all but one of the bins are close to the general average fraction of good apps. While one can conclude that apps with low number of installations are generally misbehaving, the converse is not true: apps with high number of installations are not generally well-behaving. Therefore, as stated earlier, these **results call for new forms of risk indicators as users simply cannot know if an app is well-behaving either through its popularity or rating.**

2.3 The Case of Simple Risk Notification

Our app study shows that users' data is at risk and that the existing indicators fail to communicate the risk posed by the apps. The first approach that we investigate in this sec-

tion is whether explicitly telling the users that apps are misbehaving helps them take more privacy-aware decisions. We note here that we follow Google Drive's approach of requesting permissions "At Setup" [5] (i.e., at the first time of app authorization). This is unlike other ecosystems (e.g., iOS or Android M), which require a "Just in Time" approach (i.e., permissions are requested only when the actual functionality is needed), This is because, in Google Drive, many applications are supposed to work with the user's data even when she is offline. Hence, granting access in an interactive manner for individual permissions is not always feasible.

Our proposed scheme builds on the following hypothesis:

"When users are informed about the unneeded permissions being requested by applications, they are less likely to authorize such applications."

This scheme replaces the current permissions interface displayed in Figure 1 with a new interface, presented in Figure 5. We call this risk communication model *Simple Risk Notification* (SRN), and it reveals to the user the distinction between permissions that are needed for the application functionality, and those others that are unnecessarily requested.

2.3.1 Experimental Setup

In order to test our interface modifications, we designed a multi-stage experiment with actual users.

User Recruitment:
In order to recruit users, we primarily used our university's mailing list, in addition to spreading the invitation via friends and connections. The users were briefed about an application that is related to protecting the privacy of their data against third-party applications on Google Drive. The news about the app was also reported on the university's website and was picked up by several technology websites. The website itself described itself as an application for Google Drive that aims at exposing what 3rd party web apps can needlessly get about users. Via our website, the users can sign in to their Google account, and then grant full Google Drive access to our app. Next to the "sign in" button, we linked to our privacy policy, explaining what data the app gets and what it keeps.

Upon signing in, users were given two choices: (1) participating in our experiment or (2) continuing to the application (whose details will come later in Section 2.6.1). Only those users who had at least 10 files containing text or 20 images were allowed to continue. This is to ensure that they possess at least a minimal level of knowledge about Google Drive. Next, users who chose to participate in our experiment were assigned to one of the groups described below. As a motivation to complete the experiments, the users were enrolled in a lucky draw, where they could win one of five gift cards to a mobile app store of their choice.

Methodology:
A user experiment was divided into *tasks*. In each task, the user was requested to select an app with a specified *goal* and then determine if the app's permissions were too invasive for her to accept. For example, the goal would read "Select the app which allows you to extract the ZIP files on your Google Drive", and the corresponding app would be "ZIP Extractor". The user would choose this app among other apps that are listed with their descriptions in

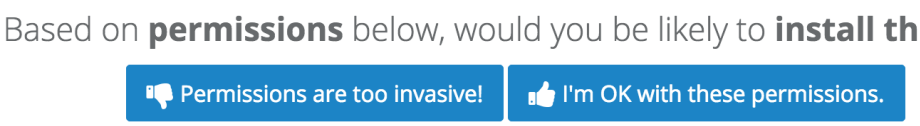


Figure 6: Decision prompt shown in each task

an interface similar to the actual Google Chrome Web Store. Only one app of those listed satisfies the given goal, and it is highlighted in the interface. This part of the setup only serves a gamification purpose in order to keep the user interested. Once the user selects the app, she is presented with the *Decision Prompt*, shown in Figure 6, positioned on top of a *Permissions Interface*. The decision prompt asks the user whether or not the permissions are acceptable. The permissions interface was varied according to the experimental group the user is assigned to.

The apps used in the experiment were obtained from the Google Drive section of the Chrome Web Store. Unlike in the store, we removed elements such as ratings, reviews, and screenshots and kept a minimal interface, allowing the users to focus solely on the app permissions. We also avoided using apps from popular vendors to avoid the bias resulting from users being influenced by famous brands. This is because we wanted to study the effect that the risk communication model has on the user's decisions, without the influence of extraneous factors[4]. Moreover, the apps were presented to the users in randomized order to compensate for the effects of learning and fatigue. For experimental purposes, we modified the permissions of these apps to be able to test various metrics. For reference, the permissions that each app requested are presented in Table 2.3a.

For the first part of this experiment, we had two groups:

1. **Baseline Group:** Users in this group were presented with a clone of the original interface that Google shows upon installing the app (shown in Figure 1). This group serves as the control group.
2. **SRN Group:** Users in this group were presented with the modified interface, previously shown in Figure 5.

2.3.2 Results

We got 97 users in total who successfully completed this part of the experiment. Out of them, 49 were in the Baseline Group and 48 in the SRN Group. In Figure 7, we plot, for the different apps, the *Acceptance Likelihood* AL defined as:

$$AL = \frac{\#(Accepts)}{\#(Accepts) + \#(Rejects)}, \tag{1}$$

where *Accepts* denotes the cases where users were fine with the permissions and *Rejects* denotes the cases where they found them too invasive. In order to compare the effect of

[4]Incidentally, the user might confront a scenario exactly as in the experiments, if she does not find the app from the store, but lands on a certain site that has the option of authenticating with Google and requires access to Google Drive.

Table 2.3a Permissions of apps in experiment, *R*: Requested, *N*: Needed, *U*: Unneeded, *NR*: Not Requested

App	DRIVE_PHOTOS	DRIVE	DRIVE_METADATA	DRIVE_FILE
ZIP Extractor		R, U		R,N
Xodo PDF Viewer & Editor			R,U	R,N
WhoHasAccess		R,U	R,N	
Video Converter		R,U		
Loupe Collage	N,NR	R,U		
Cloud Convert		R,U		N,NR

the group on AL for each app, we used Pairwise Fisher's Exact tests of significance, which allows testing the null hypothesis of no difference in any pair of proportions. Surprisingly, for each of the six apps, the tests did not show statistically significant differences between the two groups. Upon aggregating all the users' responses to all these apps, we obtained AL values of 0.385 and 0.392 for the Baseline and the SRN groups respectively. In this case, Fisher's exact test for the group-aggregated responses also did not show any statistically significant difference ($p - value = 0.928$). This indicates that **simply telling the users that an app requests extra permissions is not enough to deter them from installing the app.**

Data vs. Metadata Access: Comparing the fraction of installations of Xodo PDF Viewer and WhoHasAccess in the SRN Group in Figure 7, we can refer to Table 2.3a and observe that users are **more deterred when they discover that the unneeded permission involves full data access** (DRIVE) compared to metadata only access (DRIVE_METADATA) (Fisher's exact test p-value=0.002).

2.4 Designing Personalized Insights

Based on the findings of the first experiment, specifically the failure of SRN in deterring users from installing misbehaving apps, we decided to investigate other approaches to better communicate the risk of app installation to the users. Instead of only showing what the extra requested permissions are, we had the following hypothesis:

When the users are shown the possible data that can be extracted from the unneeded permissions granted to applications, they are less likely to authorize these applications

Accordingly, our next strategy is based on presenting users with insights that, in addition to indicating the misbehaving apps via SRN, give the users an idea on what the application

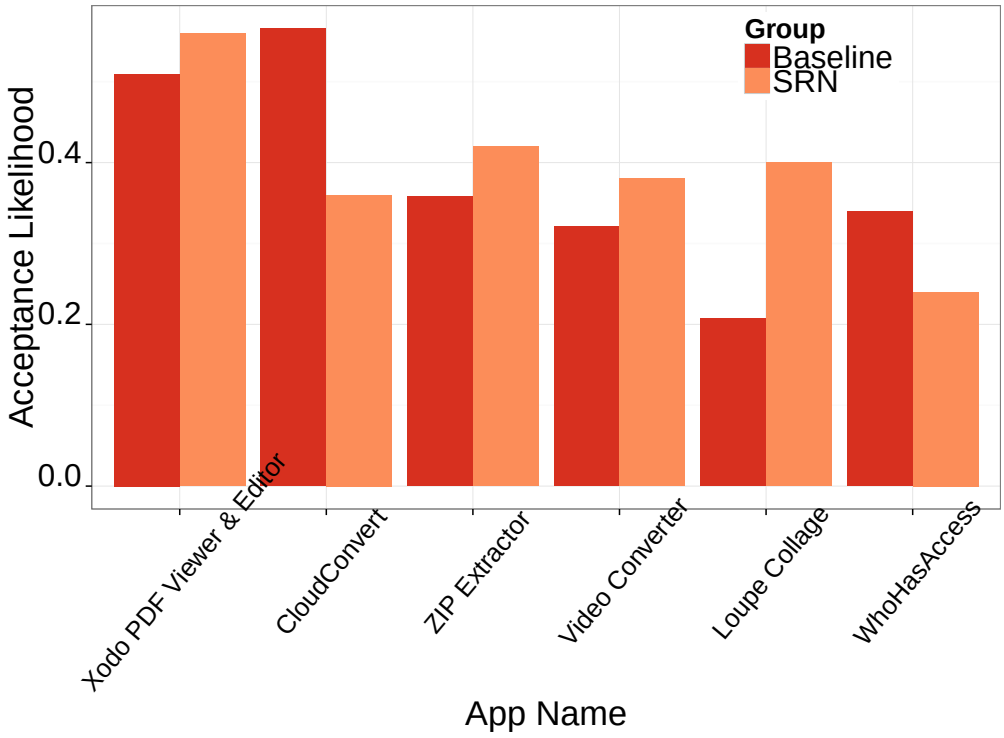


Figure 7: AL measured across apps in SRN and Baseline groups

developer can do with those extra permissions. Towards that aim, we differentiate between two types of insights:

1. **Immediate Insights:** These are randomly selected data examples, directly extracted from the user's Google Drive, such as excerpts of text or image files, photo locations, or people she collaborated with.

2. **Far-reaching Insights:** These are insights that go beyond examples and include what can be inferred by running more involved algorithms, such as sentiments towards entities, objects identified in photos, faces detected, etc.

An example of this new interface is shown in Figure 8. On the left, we have the same previous SRN interface. On the right, we have a question that says: "What do the **unneeded permissions** say about you?", followed by an answer in the form of a visual with short explanatory text.

In this section, we detail the different insights that were used as risk indicators and explain the algorithms used for generating each of them. Towards that goal, we highlight two file categories of interest: (1) textual files, such as PDF documents, word-processing documents, spreadsheets, presentations, text files, etc., and (2) image files, such as JPEG, PNG, TIFF, etc. We represent the set of textual files as $TF = TF_1, TF_2, .., TF_K$ and the set of image files as $IF_1, IF_2, ..., IF_L$.

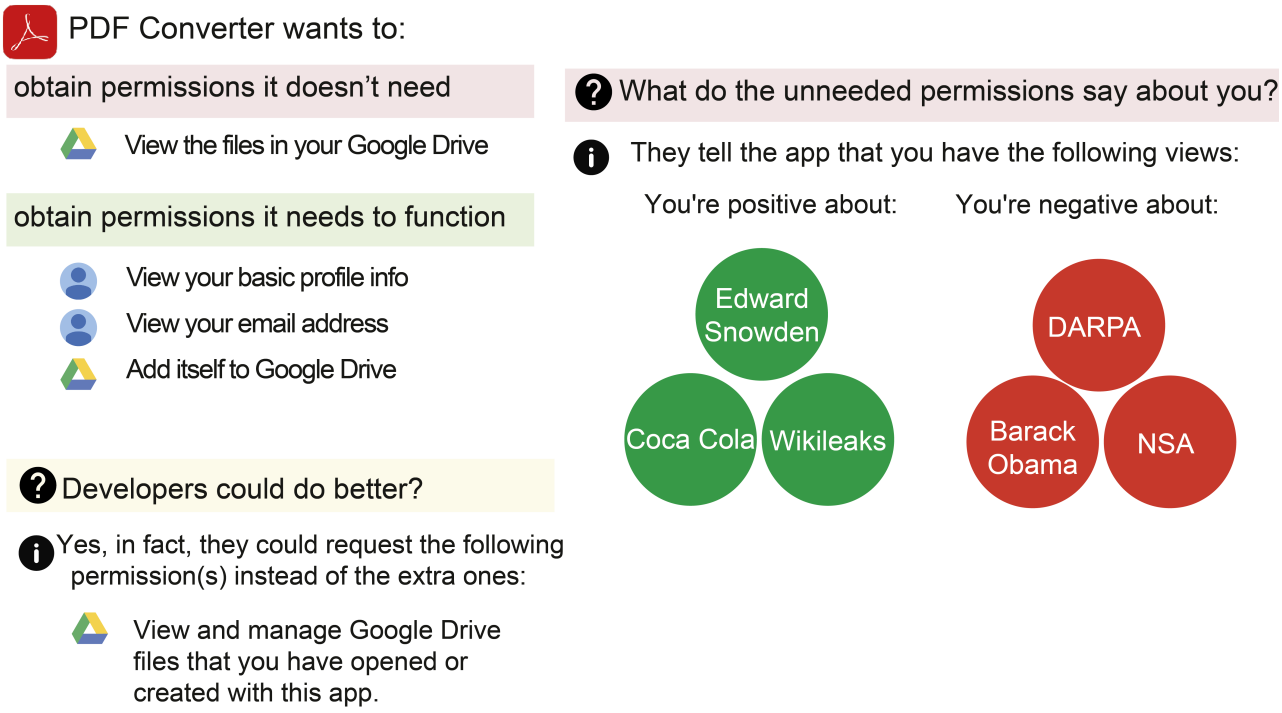


Figure 8: Modified interface with Personalized Insights

2.4.1 Far-reaching Insights

We begin by describing the algorithms for generating the far-reaching insights.

Entities, Concepts, and Topics (ECT):
The first type of insights we form is based on applying various NLP techniques to extract *named Entities (E)*, *Concepts (C)*, and *Topics (T)* from users' textual files. We combine these together due to the similar nature of these insights. Any one of E, T or C can be randomly displayed to the user in the interface.

i. Entities: We get the top named entities (e.g., people, places, companies, etc.) present in the user's textual files. Such entities are recognized using Named Entity Recognition (NER), which is a traditional problem in natural language processing that involves locating and classifying elements in text into pre-defined categories [6]. Our hypothesis was that displaying such information to the user might serve as a good indicator of risk as these entities might include people the user works with, companies she talks about, places she plans to visit, etc. For this task, we perform text extraction on each file, and we then pass the text to an external web service, namely AlchemyAPI. Given the text of file TF_j, this service returns a set of entities, along with the frequency of occurrence $f_{i,j}$ of each entity e_i in TF_j. We normalize this frequency for each entity by dividing it by $fmax_j$, which is the frequency of the most of recurrent entity in TF_j:

$$fnorm_{i,j} = \frac{f_{i,j}}{fmax_j} \tag{2}$$

Then, we compute an overall score for entity e_i across all the files in TF, by summing its

(a) Named *Entities* visual (b) *SharedInterests* visual (c) *FacesWithContext* visual (d) *FacesOnMap* visual

Figure 9: Example visualizations of Personalized Insights

individual normalized frequencies:

$$score(e_i) = \sum_{j=1}^{K} fnorm_{i,j} \tag{3}$$

As shown in Figure 9(a), we visualize the entities with the highest scores as a set of circles, each of a diameter proportional to the score of the corresponding entity. Different types of entities (e.g., people, places, etc.) have different circle color.

ii. Concepts: We also extract concept tags from users' documents. These concepts are high-level abstractions, not necessarily mentioned in the text. For example, the sentence "My favorite brands are BMW, Ferrari, and Porsche", would be tagged by the concept "Automotive Industry". Our rationale behind this insight type is that concepts can serve for profiling the users' interests. Hence, knowing that they can be potentially leaked can contribute to more user awareness. AlchemyAPI was again used for this task, returning, for each file TF_j, a set of concepts, each denoted as c_i along with a relevance score $r_i \in [0, 1]$. We used the following scoring method to rank the concepts across the user's documents:

$$score(c_i) = \sum_{j=1}^{K} r_{i,j} \tag{4}$$

Similar to the case of entities, we represent concepts by circles, each of a diameter proportional to the score of the concept.

iii. Topics: Topics are used to classify documents into high level categories, such as technology, art, business, etc. Our motivation behind displaying such information to the user is similar to the case of concepts, i.e., alerting her to profiling risks, although topics are at a higher level of abstraction than concepts. We used AlchemyAPI, which returns a maximum of 3 topics per file TF_j (each denoted as t_i), along with a relevance score $r_i \in [0, 1]$ for each of them. A topic comes in the form of "$a_1/a_2/\ldots/a_n$", representing a hierarchy among the labels (e.g., "/hobbies and interests/astrology" or "/finance/investing/venture capital"). In order to extract the top topics based on a user's documents, we use the same scoring method as that of concepts:

$$score(t_i) = \sum_{j=1}^{K} r_{i,j} \tag{5}$$

We represent topics by circles, similar to the case of entities, where the diameter of a circle is proportional to the score of the topic. Topics sharing the top level label are colored similarly.

Sentiments:
For each entity that occurs in TF, it is possible to also estimate whether the text relays a positive, neutral, or negative sentiment about that entity. We hypothesized that visualizing the most positive or most negative opinions that a user's documents relay would have a good alerting effect on the users' app installation choices. Towards that end, we use the sentiment analysis service of AlchemyAPI. For each TF_i, we select the entities labeled with positive or negative sentiments (each such entity also has a sentiment score $s_{i,j} \in [-1, 1]$ with 1 corresponding to the most positive sentiment and -1 to the most negative one.). We then compute the overall sentiment score s_i of entity e_i across the all the user documents TF:

$$s_i = \sum_{j=1}^{K} s_{i,j} \tag{6}$$

The sentiments with the highest positive and negative scores are then shown to the user, as was presented in Figure 8.

Top Collaborators:
The next insight we added was displaying the top collaborators a user has, based on the analyzed files. We define collaborators as people who share files with the user, regardless of who initiates the sharing operation. We hypothesize that top collaborators typically include close work colleagues, intimate friends, or people the user goes out with and shares pictures afterwards. Hence, showing such information to the user as data exposed to leakage would make her more alert about the potential risk. In the interface, this insight is visualized as a horizontal bar chart of the top collaborators with the bar lengths representing the relative frequency of the user's collaboration with each of them.

Shared Interests:
In this insight, we try to represent the user's mutual topics of interests with a group of people. Specifically, we show the user a list of people alongside different topics that have appeared in documents the user shared with these people. Towards that end, we perform the following steps:

- We determine the top topics as we have done in the previous insight.
- Then we select from these topics a subset S_t that only includes the ones which appeared in shared files.
- Via Google Drive API, we extract, for each topic t_i, a list $U(t_i)$ of collaborators (based on files it appeared in).
- We select from each $U(t_i)$ the most frequent collaborators (i.e., those appearing in most documents with this topic).

Users then get a visualization similar to Figure 9(b), where we show the three top topics from S_t along with the top collaborators for these topics.

Faces with Context:
We now come to the insights that are based on features inside the user's images. The first

insight in this category shows a group of faces, representing the most frequent people appearing in the user's images, alongside the concepts that appear in the same images. Figure 9(c) shows an example of this insight. The rationale behind this visualization is that it simulates a third-party trying to infer the people that the user is interested in or appears with, in addition to objects that are inside these photos. One can imagine that such information might be valuable, for example, to advertisers that aim to extract the user's interests in certain products and services. In order to achieve this visualization, we performed two steps:

i. Face clustering: It is evident that showing the user random faces detected in her photos will not create the same effect as when these faces are actually people she cares about. Our plan to achieve the latter case involves three steps:

- We use a face clustering algorithm in order to group together photos of the same person. As a result, we get a list of groups G, where each group $G_i \in G$ is comprised of the faces that belong to a person identified as p_i. The algorithm used is by Zhu et al., [7] implemented by the OpenBR framework. [8]
- From each group G_i, we exclude the faces with width (height) less than $\frac{1}{15}$ of the total image width (height).
- We exclude groups with less than 3 faces in total.
- We sort the groups by the number of faces in each of them.

ii. Image concept recognition: In order to identify the concepts inside each photo, we used a classifier from the Caffe library [9]. The classifier uses a pre-built deep learning network, that is based on the architecture used by Krizhevsky et al., [10] that won the Imagenet 2012 contest.

Based on the above, we show the user the top groups along with the top concepts associated with those groups.

Faces on Map:
In addition to the image content itself, image metadata can be also sensitive, especially the geographical location where the image is captured. Accordingly, our hypothesis was that showing people the places where their photos are taken, in addition to the faces and items in those photos, can serve as good indicators of risk. Figure 9(d) shows the visualization that we made to test this hypothesis. It consists of showing the faces of people overlaid on a map, centered at the geographical area where these faces appeared. Below the map is a list of the top concepts that appeared in the photos taken in that area. In our actual implementation, the visual is animated, moving between different areas to show the user the places that different photos were taken at. In order to construct this visualization, we had to cluster the images into different geographical areas. For that, we used the OPTICS algorithm (Ordering Points to Identify the Clustering Structure) by Ankerst et al.,[11]. OPTICS allows finding density-based clusters in spatial data and is tailored for detecting meaningful clusters with data of varying density. After getting the cluster results, the zoom level on the map is animated to show one cluster to the user at a time.

2.4.2 Immediate Insights

In the following, we describe the design of the immediate insights.

Image: We randomly show an image selected from the set IF of image files.

Location: We randomly choose a photo from IF, such that it includes a GPS location in its Exif data. Then we show that photo on a map centered at that location.

Text: We show the user an excerpt from the beginning of a randomly chosen textual file.

Collaborator: We show the profile picture and the name of a randomly chosen collaborator.

We note that the reasoning behind designing lightweight insights such as Immediate Insights was that we wanted to examine whether designing heavyweight insights such as Far-reaching Insights is worth the effort for us and the potential adopters of our approach, or do users respond equally favorably (or badly) to both the heavy and the lightweight approaches, in which case Far-reaching insights need not be adopted.

2.5 Evaluating Personalized Insights

2.5.1 Experimental Setup

In order to assess the effect of these different visuals, we conducted a user experiment based on the same methodology as in Section 2.3.1, with two new groups:

1. **Immediate Insights Group:** Users in this group were presented with the modified interface of Figure 8, based on the Immediate Insights of Section 2.4.2.
2. **Far-reaching Insights Group:** Users in this group were presented with the modified interface, of Figure 8, containing the insight described in Section 2.4.1.

In addition to the apps we had in Table 2.3a, we added new apps to these groups in order to further compare the effects of the different insights described above. We fixed the permissions of all the added apps to request both DRIVE and DRIVE_FILE access while needing only DRIVE_FILE. Hence, in addition to ZIP Extractor app in Table 2.3a (which has such permissions), users had 3 additional apps in the Immediate Insights group (to compare the four types of Immediate Insights). Similarly, users in the Far-reaching Insights group had 5 additional apps (to compare the 6 types of Far-reaching Insights).

It is worth mentioning at this point that after generating these insights from a user's files, these files are deleted immediately from our apps' servers. As per our displayed privacy policy, only the insights' data presented to the user is kept in the app database. Moreover, the user is given the option to delete her insights data at any time with a single click in the app's menu. We also involved our university's research ethics board in order to further ensure participants' privacy.

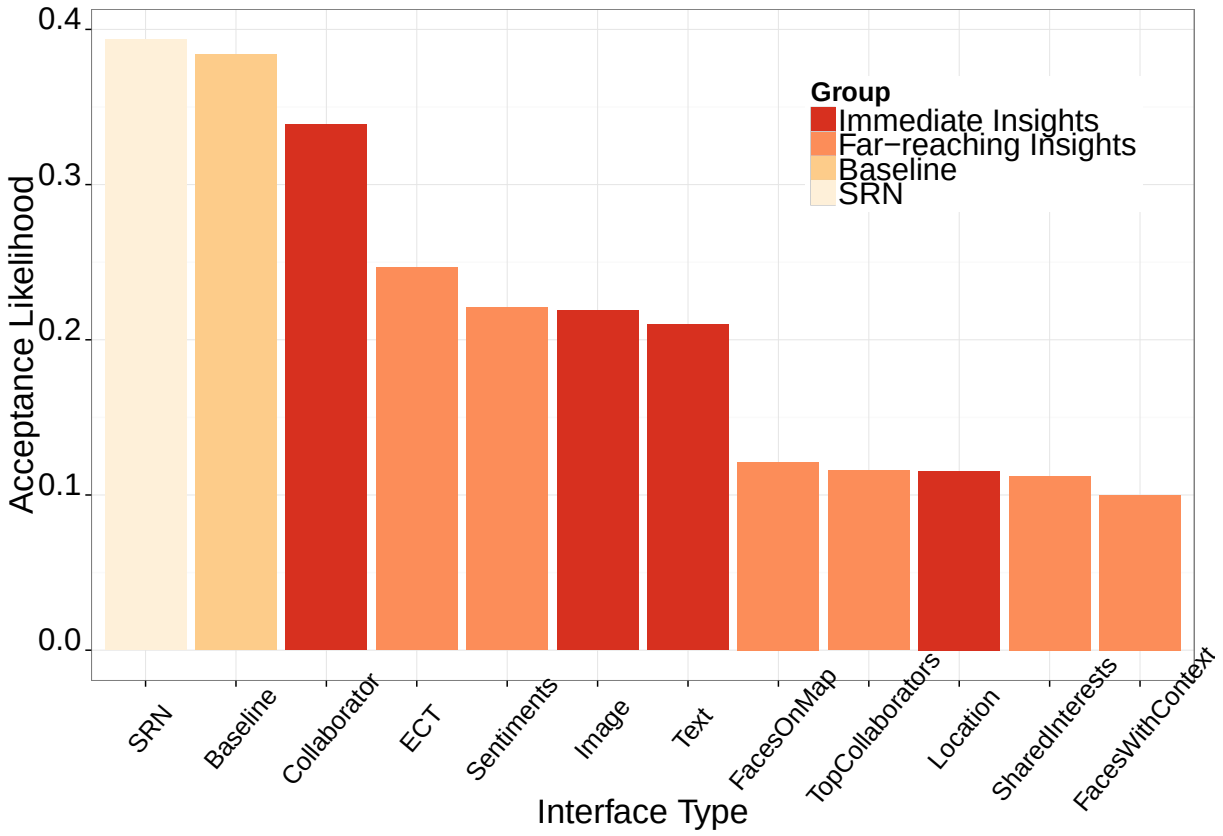


Figure 10: AL for the different types of interfaces

2.5.2 Results

We got 98 users in total who successfully completed this part of the experiment. Out of them, 48 were in the Immediate Insights group and 50 in the Far-reaching Insights group.

Efficacy of Insights:
We now present the first part of our results, where we investigate in detail the insights' effect on users' privacy decisions.

a) Insights' General Effect: We first found that, in general, the choice of the insight in the modified interface has a significant effect on the Acceptance Likelihood AL (Fisher's Exact test: $p - value = 2.2 * e^{-16}$). In order to compare the effect of different insights, we plotted in Figure 10 the Acceptance Likelihood for each insight and also for the Baseline and SRN groups from the first part of the experiment.

b) The Power of Relations-Based Insights: In order to compare these likelihoods, we used Pairwise Fisher's Exact test, which allows testing the null hypothesis of no difference in any pair of proportions. The results of the test are shown in Table 2.5a, where cells indicate the $p - value$ of the comparison test between the item defined by the row label and that defined by the column label.

The first interesting outcome from the comparison table and figure is that there is a category of insights (Category 1) composed of {*Image*, *Text*, *ECT*, and *Sentiments*} that are all

Table 2.5a Pairwise comparison (cells formatted in blue italics correspond to a statistically significant difference with $p - value < 0.05$)

	Collaborator	FacesOnMap	FacesWith-Context	Image	Location	Sentiments	Text	Top-Collaborators	Shared Interests	Baseline	SRN
FacesOnMap	*0.017*										
FacesWithContext	*0*	0.755									
Image	*0.03*	0.325	*0.017*								
Location	*0.032*	1	0.735	0.411							
Sentiments	*0.037*	0.333	*0.005*	0.882	0.298						
Text	*0.012*	0.261	*0.002*	1	0.315	1					
TopCollaborators	*0*	0.776	0.74	*0.035*	1	*0.011*	*0.005*				
SharedInterests	*0*	0.743	1	*0.038*	0.725	*0.022*	*0.014*	0.84			
Baseline	0.374	*0.002*	*0*	*0*	*0.005*	*0*	*0*	*0*	*0*		
SRN	0.265	*0.002*	*0*	*0*	*0.005*	*0*	*0*	*0*	*0*	0.803	
ECT	0.066	0.131	*0*	0.442	0.222	0.55	0.425	*0*	*0.003*	*0*	*0*

associated with a significantly higher acceptance likelihood than the category composed of {*FacesWithContext*, *TopCollaborators*, and *SharedInterests*} (Category 2). The insights within these two categories are adjacent to each other in Figure 10[5]. Since this is a very interesting result, we investigate further to analyze the defining characteristics of these two naturally clustered categories. The main feature of Category 1, which includes both Immediate and Far-reaching based insights, is that insights in this category are restricted to characterizing the user *herself*, such as showing text excerpts from her documents, topics appearing in them, or images she has in her files. On the other hand, the defining feature of Category 2 insights, which are all Far-reaching, is that they extend to characterizing the relationships of the user *with other people*. *FacesWithContext* shows the most important faces in user's photos along with the items appearing with them. *SharedInterests* shows the people who collaborate with the user and the type of topics they share. Also, *TopCollaborators* identifies the most frequent people the user interacts with. From this, one can conclude that users become especially alert when insights go beyond specifying interests to describing relations with others.

c) Impact of Face Recognition: Delving deeper into more results brought forth the comparison of different insights, one can notice that showing examples of user's images is significantly less alerting than showing the important faces and listing the concepts in the image ($p - value = 0.017$). This highlights the fact that users are sensitive towards the output of face detection and object recognition in photos. Given that services such as Google Photos, OneDrive, and Flickr already apply such techniques to facilitate search, the above result highlights that they can also be used by these companies to easily implement solutions such as ours for raising users' privacy awareness when sharing data.

d) Influence of High-Level Textual Insights: In the case of textual documents, showing the high-level entities or concepts extracted from the text does not seem to have a significant difference as compared to simply showing direct excerpts from the text ($p - value = 0.425$). Only when the relationship factor is introduced does the AL significantly decrease (as in the case of *SharedInterests*).

e) Inefficacy of Baseline and SRN: It is worth noting, however, that the SRN and Baseline approaches had a significantly higher AL than all the insights, except for the Collaborator insight. This highlights the fact that showing well-selected insights would almost always result in a significantly higher user alertness and deter them from installing misbehaving apps.

e) Superiority of Far-reaching Insights: By aggregating the results over all the experiments

[5]The number of users who had location-tagged photos was low; hence we could not obtain highly significant results in the case of *Location* and *FacesOnMap* insights. Nevertheless, their ranking in Figure 10 suggests they are both at the boundary between the two categories.

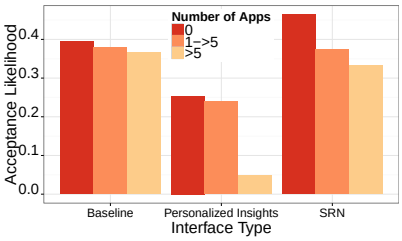


Figure 11: Effect of number of installed apps on AL

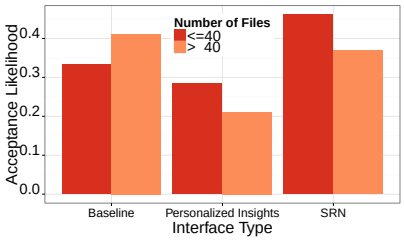


Figure 12: Effect of number of files on AL

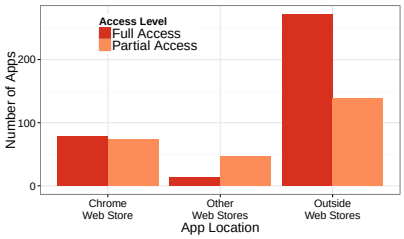


Figure 13: Change of access level with app location

with Far-reaching Insights, we found out that they result in a significantly lower AL than those with Immediate Insights ($AL = 0.165$ and 0.232 respectively with Fisher Exact's test giving a $p - value = 0.0017$). However, as stated above, Immediate Insights are significantly better than Baseline and SRN. Overall, these results corroborate the findings of [12], which showed the goodness of an approach similar to Immediate Insights, and at the same time, they also demonstrate the superiority of our novel approach of Far-reaching Insights.

Analyzing External Reasons behind User Decisions:
Aside from the general effect of insights which were a feature of our experiments, next we seek to explore whether other factors, external to our experiment design, affect the likelihood of acceptance.

a) Effect of User's App History: One factor that we test is the number of Google Drive apps that a user has installed in the past, apart from the ones that come by default (e.g., Google Docs, Google Sheets, etc.). By requesting the permission "DRIVE_APPS_READONLY", we were able to get the list of apps that the user has authorized at some earlier point in time. We represent the number of apps as a categorical variable of three possible values: $numberOfApps \in \{0, 0 \rightarrow 5, > 5\}$.

When analyzing the aggregated Acceptance Likelihood AL from the four groups of our two experiments, we found that *numberOfApps* does not have a significant effect on AL (Fisher's Exact Test: $p - value = 0.138$). However, we also investigated how *numberOfApps* affects the AL differently in the SRN and Baseline interfaces compared to the Personalized Insights interface (comprising both Immediate and Far-reaching Insights). Towards that end, we show in Figure 11 the AL for these interfaces for different values of *numberOfApps*. We calculated the pairwise Fisher exact test among the different proportions. We discovered that, in both the Baseline and SRN groups, there is no significant difference among the AL values for different number of apps (Fisher's Exact Test: $p - value > 0.05$). This lack of difference implies that users with high number of apps installed, contrary to what one might expect, do not install more misbehaving apps than other users. However, in the Personalized Insights case, users who installed more than 5 apps in the past show a significantly lower AL than those with less apps installed (Fisher's Exact Test: $p - value_{(>5 \text{ vs. } 1 \rightarrow 5)} = 0.0015$, $p - value_{(>5 \text{ vs. } 0)} = 0.0017$). Thus, we can conclude that Personalized Insights causes those "experienced users" to be drastically more alert given their previous usage of apps, while the effect on relatively inexperienced users is less extreme, but still significant (all $p - values < 0.05$) as compared to the Baseline and SRN Groups.

b) Effect of Amount of Data: Another factor that had a similar effect is the number of files a user has in her Google Drive (denoted as *numberOfFiles*). We transformed the number of files into a categorical variable by splitting the users into two categories: those with

numberOfFiles $\leq$ 40 files and those with *numberOfFiles* $>$ 40. We initially found that the effect of *numberOfFiles* on AL, when the four groups from the experiments are combined, is not significant (Fisher's Exact Test: $p - value = 0.0762$). As in the case of *numberOfApps*, we investigate in Figure 12 whether the Baseline, SRN, and the Personalized Insights are affected differently by *numberOfFiles*. In the case of Personalized Insights, we find a significant drop in the acceptance likelihood for users with of with *numberOfFiles* $>$ 40 (Fisher's Exact Test: $p - value = 0.0328$). In the other two cases, the change in the two directions was not statistically significant (Fisher's Exact Test: $p - value = 0.2755$ for Baseline and $p - value = 0.1351$ for SRN).

We conjecture that this is because users with a large number of files are more sensitive about granting access to applications when they become aware of the consequences of doing so via the Personalized Insights. Second, having more files also allows our algorithms to extract more information and infer more relations, and this can create a greater impact on users' decisions.

2.6 Solutions for Privacy Protection

2.6.1 PrivySeal: A Smart Privacy Assistant

Driven by the magnitude of the risk posed by misbehaving apps in Google Drive, we were motivated to bring the advantages of the Personalized Insights interface to the user community of this platform.

One approach to achieve that would be for Google itself to implement a scheme similar to ours and to integrate it within the app authorization process. However, we decided not to wait and chose an alternative approach, which is independent of the company's plans and is ready for user utilization immediately. We built the *PrivySeal* for Google Drive apps, which is readily available at `https://privyseal.epfl.ch`. The Privacy Store allows users to navigate a list of applications, click on those of interest, and check whether and how they can misbehave via our Personalized Insights interface. Users can filter applications by their (mis)behavior, and they can also search for apps. The component diagram for the Privacy Store is shown in Figure 14. Similar to the app reviews we conducted, we have included a "Review Wizard" inside the Privacy Store for indicating the requested, needed and unneeded permissions along with the alternative permissions the developer could have used. This responsibility is currently given to a small set of expert users and developers and is moderated by the store administrators. Developers who would like to object to existing reviews of their apps can submit rebuttals. Currently, the Privacy Store has more than 1350 registered users and 100 applications. We finally note that the Privacy Store gets access, as is the case with other apps, to users' data to generate insights. Hence, users are assumed to trust the provider of such a "Privacy-as-a-Service" solution. However, this assumption of trust will hold if a solution such as the Privacy Store is hosted by the Cloud provider itself (which already possesses the data), or an enterprise protecting its documents from third-party applications. The assumption of trust is also valid if the users choose to trust a *single* entity (such as the Privacy Store) to protect themselves from *multiple* other unaccountable misbehaving entities that they would otherwise be forced to trust.

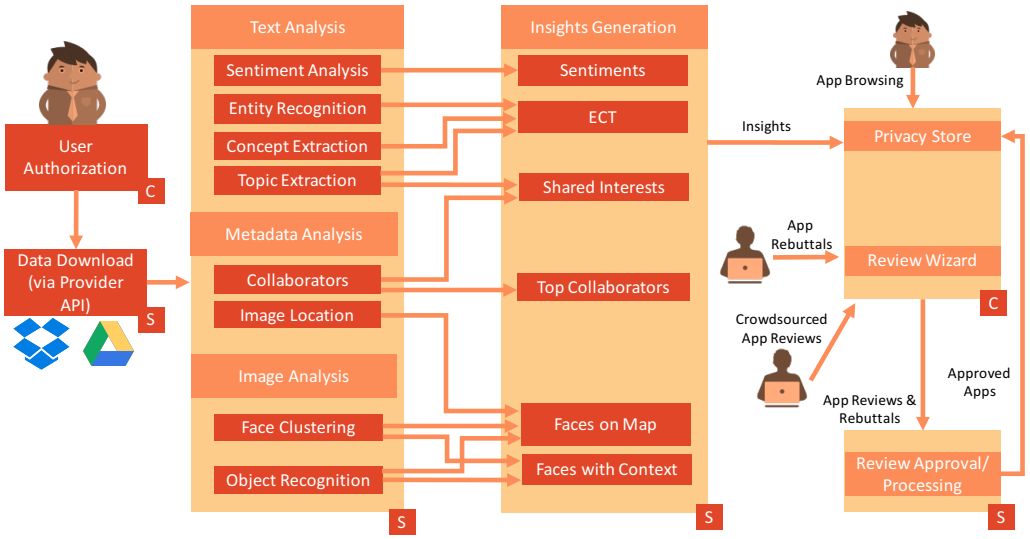


Figure 14: Component diagram of privyseal (components labeled by *S* are server-side and by *C* are client-side)

2.6.2 StackSync: Personalized Insights for user-to-user sharing

As our final contribution, we integrate Personalized Insights into one of this project's main deliverables, Stacksync. Now, Stacksync users when they want to share some data with others, can generate insights based on the data to be shared. They can then see what the sharee could potentially know about them given that data. Thus, they can share data judiciously on the Cloud and easily safeguard their privacy.

2.7 Related Work

To our knowledge, this is the first work that studies the problem of user privacy in the context of third-party applications on top of Cloud storage providers. Several works have previously studied the problem of direct information sharing to providers themselves (e.g., [13]).

2.7.1 Privacy in Other App Ecosystems

In the case of other ecosystems, there are related works that have studied the current state of privacy notices. For instance, Chia et al., conducted a large scale analysis of Facebook apps, Chrome extensions, and Android apps to study the effectiveness of user-consent permissions systems [14]. Similar to our findings, they also observed that the community ratings are not reliable indicators of app privacy in these ecosystems and showed evidence of attempts at misleading users into granting permissions via free apps or apps with mature content. Huber et al., developed *AppInspect*, a framework for automating the detection of malpractices in third-party apps within Facebook's ecosystem and used network traffic analysis to spot web trackers and identify leaks of sensitive information to other third parties [15]. The case of third-party apps in Google Drive differs from these platforms in that it is not possible to perform large scale analysis, firstly due to the absence of a standard application format and secondly due to the difficulty of automatically finding the triggering

button for permission requests in different applications. Aside from the above, client-side traffic analysis is not sufficient to detect all Cloud data leaks as the applications can send data to third parties after it arrives at the server side, to which outsiders do not have access.

2.7.2 Improving Current Privacy Notices

A few works have recently suggested improvements to the existing permissions schemes, with a special focus on the case of Google Play Store. Kelly et al., argued that the privacy information should be a part of the app decision making process and should not be left till after the user makes her decision [16]. Hence, they appended a list of "Privacy Facts" to the app description screen, textually indicating that the app, for example, collects contacts, location, photos, credit card details, etc., and found that it assisted users in choosing apps that request fewer permissions. Harbach et al., proposed to integrate examples from user's data in the permissions request screen to expose the data apps can get access to [12]. This involved showing random pictures, call logs, apps, location, and contacts from user's data that correspond to each permission. In this work, we go steps farther, and we show that well-crafted visuals showing far-reaching insights extracted from users' data can be more effective than randomly selected data. We also show through pairwise-comparisons among the insights themselves that the choice of the displayed risk indicators highly affects the interface's effectiveness. Furthermore, in both [16] and [12] the methodology involved users choosing between two apps where one requests a subset of the other's permissions. This does not tackle the general case where permissions are not necessarily subsets of each other, which was one of the reasons for adopting a different methodology in our experiments. It is also worth mentioning that, in our experiments the number of users who were involved with their personal accounts in the experiment was more than five times the number of users in [12] and [16]. Furthermore, we also provide a readily available solution for the public in the form of the Privacy Store.

Moreover, our work is in line with the best practices recommended by the recent work of Schaub et al., who developed a design space for privacy notices to assist researchers in increasing the impact of their schemes [5]. For instance, we implemented the multi-layered notice concept by showing data of textual and visual modalities. We also developed various visuals to ensure that the permissions dialogue is *polymorphic*, which was also shown recently to have an effect on reducing the habituation effect in the user's brain [17]. Personalizing warning notices, as we do in this deliverable, has been studied before in the context of LED signs [18] and was shown to significantly increase compliance compared to impersonal signs.

2.8 Conclusion and Future Work

In this document, we characterized the various factors that have an impact on user privacy in the ecosystem of third-party apps for the Cloud. We considered Google Drive as an example case study, and comprehensively anatomized the ecosystem from the viewpoint of users, developers and the Cloud provider. For users, we carefully devised a set of experiments and tested existing and novel risk communication models to analyze the factors that influence users' decisions in app installation. Our results provide interesting insights into how user

privacy can be improved and how Cloud providers can develop better risk indicators. We also present a privacy aware store for Cloud apps, which already has over 1350 registered users. From our store users and people who took part in our experiments, we had the unique and unprecedented opportunity to first hand study real users Cloud data. Based on this data, we were able to characterize the current behavior of third-party app developers and also point out avenues for developer misbehavior. Finally, based on our analysis, we provide several suggestions for Cloud providers that can help in safeguarding users' privacy and protecting their data from needless leakage and exploitation. In the future, we aim to build on the Privacy Store and develop a recommendation system that suggests apps of similar functionality but superior privacy. Finally, it would be interesting to study how our findings on the best risk indicators generalize to other ecosystems, such as Android or iOS.

3 Privacy-aware data sharing with Attribute Based Encryption

3.1 Introduction

In the context of a scalable Personal Cloud like that of StackSync, our approach for **privacy-aware data sharing** involves the design of a cryptographic component ready to be adapted to an existing architecture and able to work efficiently as an extension to our software. Data privacy in the Personal Cloud can easily get compromised, because clients typically delegate tasks such as protection and honest use of the files to remote storage servers that are out of their control. But, *what if their information is sensitive enough not to take the risk of trusting a third party?*

Filling this security gap is more challenging than it seems. As we try to avoid any unauthorized access to user information by adding extra complexity to our system, the proper encryption of such data implies a great challenge when it comes to generate, manage and distribute keys, particularly when data has to be shared among a large number of users. Thus, any attempt to keep the data reliably encrypted from the very first moment it leaves the user device (e.g., through its *storage* and *sharing*) is susceptible to impact the scalability and efficiency of our system in a relevant manner.

Our approach aims to overcome some of the greatest Personal Cloud sharing challenges, i.e., **privacy-aware data sharing**, whilst preserving efficient fine-grained access control over the outsourced data.

Achieving privacy-aware data sharing means designing a system able to gather user preferences on what data to share and with whom to share it, along with a trustworthy policy management infrastructure that ensures the proper and efficient enforcement of those preferences, without compromising the privacy of any of the users or data involved.

For this aim, we will adapt the proposal described by Yu et al. [19] to a Personal Cloud setting. Concretely, [19] introduces a fine-grained access control scheme that is very suited for the Personal Cloud, which combines *Key-Policy Attribute Based Encryption* (KP-ABE) with *Proxy Re-encryption* (PRE) and *Lazy Re-encryption*.

KP-ABE is a public key cryptography specially designed for data-sharing environments. By using its encryption technique, any data encrypted gets associated with a set of attributes. On the other hand, each user is provided with an access structure composed of a logical definition of attributes. The secret key of any user reflects the access structure in such a way that a user will be able to access certain content if and only if the data attributes satisfy his access structure.

While KP-ABE provides our system with a fully-fledged access control layer achieving both *fine-grainedness* and data *confidentiality*, some eventual operations must be delegated to the Cloud servers in order to offload users from some heavy crypto computations. In order to reduce the impact of these tasks, two complementary mechanisms are being implemented at the Cloud side:

- **Proxy Re-Encryption:** allows our Cloud Storage logic to manage specific changes or

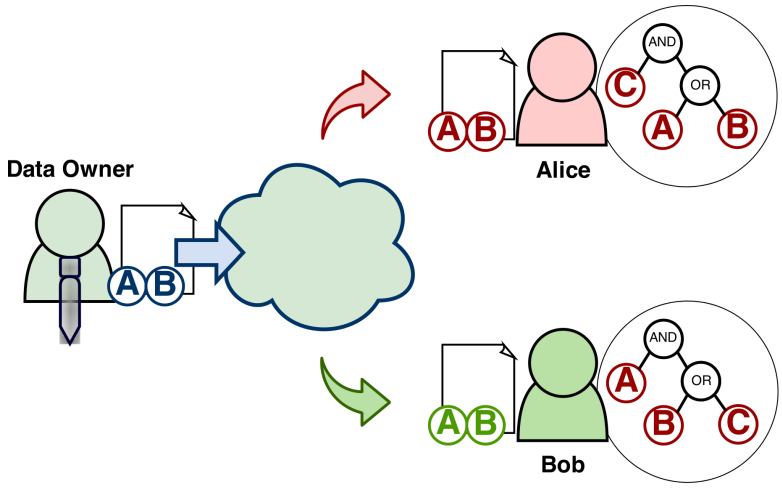


Figure 15: Data owner sharing a file between two different users with different access trees

updates over stored encrypted data efficiently while preserving data confidentiality, since the underlying plaintext remains withheld from the Cloud servers throughout the whole process; and

- **Lazy re-encryption**: alleviates the burden on Cloud servers by "aggregating" multiple successive re-encryption operations into one, thus statistically saving the computation overhead.

While at the time of this writing the basic access control logic has already been coded and tested, some parts are still missing, particularly concerning user revocation. Consequently, we mainly provide here an overview and complete specification of the operations required to integrate the approach of Yu et al. [19] with a Personal Cloud system, and to StackSync in particular. We will start with a description of the main actors participating in the system.

3.2 Models: Entities, System and Communication

The design of our component relies on the existence of three main entities or actors: the data owner, many data consumers or users, and the Personal Cloud server.

- In our access control model, every single user will play the role of **data owner** for his personal data and run a separate instance of the KP-ABE protocol. Consequently, as data owner, he will be responsible for setting up the system, generating keys and giving permissions to users. As its name suggests, the data owner will be able to create, store and share data.
- The **data consumers** will be the **users** of our system. They will access the files shared by the data owner, download data and decrypt it. In the original model described by Yu et al., users are assumed to have very limited access privileges since they can only read data. We will modify their model in order to enable users to have write permission, thus adapting it to a Personal Cloud model. It is worth pointing out that neither the data owner nor the regular users will be always online, as it happens in traditional Cloud models.

- The **Cloud server** will store all encrypted data remotely in a scalable and efficient way. As we already know, StackSync will also be responsible for management tasks like file synchronization, chunking or authentication. Moreover, the new component added to the system will store, update and distribute all metadata related to the new protocol. All issues raised by the adaptability of the new security concept to our original model will be widely discussed later.

Users will no longer have to worry about their information being compromised by any entity in the system. In our model, the Cloud server is considered honest but "curious", as it could try to find out as much secret information as possible based on the data provided by users. However, users will no longer have to blindly rely on the licit use of their data stored remotely since it will be protected anytime by cryptographic primitives.

Furthermore, all communication channels among nodes in the system are assumed to be properly protected under a suitable security protocol. StackSync already provides users with an authentication and authorization layer. Thus, privacy and confidentiality against malicious users will be preserved.

3.3 Preliminaries

This section will briefly describe the main concepts of our proposed scheme, most importantly the KP-ABE protocol. This chapter is intended as an overview of the relevant features in order for the deliverable to be self-contained; for further details please refer to [20] and [19].

3.3.1 KP-ABE

Key-Policy Attribute-Based Encryption (KP-ABE) is used to implement encryption to files and selectively restrict their access to a desired section or party within the system. In KP-ABE, the data owner defines a suitable set of available attributes and generates the corresponding public key (PK) and a KP-ABE master secret key (MK). The generated PK is composed of a set of Public Key Components (PKC_i) corresponding to each of the attributes defined beforehand.

In KP-ABE, each user must hold an access structure defining its privileges, which is a Boolean expression over attributes. This logical expression is often represented as a logical tree, where the leafs are attributes and the interior nodes are threshold gates. For instance, the data owner could choose to assign Alice an access structure. The data owner can define her access privileges as ("C" AND ("A" OR "B")), where "A", "B" and "C" are attributes (e.g. Accounting, Budgeting, Computing). The data owner would then generate and distribute to Alice her new secret key (USK) including implicitly those three attributes. The corresponding access tree would be represented as shown in Fig.15. Similarly, the data owner can also define an access structure to Bob as ("A" AND ("B" OR "C")) and provide him with his USK accordingly. From now on, Alice and Bob should be able to decrypt files according to their access policy.

On the other hand, KP-ABE encrypted files must specify a set of attributes in order to define in which context will be shared. For instance, the data owner can encrypt and upload a file with the attributes "A" and "B", as we can see in Fig. 15. After Alice and Bob download the file, Alice cannot see the underlying plaintext while Bob is able to correctly decrypt it. The KP-ABE construction ensures that only users with the proper access structure will be able to decrypt data encrypted under a certain set of attributes. In this case, Bob satisfies the condition ("A" AND "B"), where "A" and "B" are the attributes used in the encryption of the downloaded file. Alice will nevertheless not be able to decrypt the content since her access structure requires files to be encrypted under the attribute "C", in addition to "A" or "B".

Data must be accessible exclusively to the data owner and to those users who have been explicitly authorized by the owner. Data confidentiality will not be exposed to any risk anytime during storage and sharing. Therefore, neither unauthorized users nor Cloud servers themselves should be able to access any information in plaintext.

Another important issue is that of sharing flexibility, where our system should provide our sharing logic with efficient fine-grainedness. The finer the granularity, the greater the flexibility and control in access management. This is usually achieved by using Access Control Lists (ACLs) or similar methods. The problem here is that such access control concepts severely affect the scalability of our system.

Although our system already implements user authentication as a measure of access control and data protection, this cannot be the only access control logic since our goal is to provide complete confidentiality of data, regardless of how reputable our Cloud server is. We must take into consideration that any unknown vulnerability would compromise the privacy of all the stored data.

We also need to take into account the key features of KP-ABE scheme, noting the fact that our privacy model involves the exclusion of Cloud servers from any operation implying access to unencrypted information. Thus, data must be encrypted at any time throughout its lifecycle, except for its local use in authorized systems. This means that users should be responsible for an essential part of the key generation and distribution logic.

KP-ABE allows our system to store all data properly and unfailingly encrypted using a set of attributes. Only users who have been granted permission to access certain data will be able to manipulate the underlying information. Each user holds a key that implicitly contains his access policy. In this manner, software authentication and logging becomes a secondary access control layer, and data remains protected to third parties during its storage and sharing in lower levels.

We remind the reader that attributes represent a set of meaningful properties able to describe a diverse context. Attributes will be linked together by building logical expressions. In the KP-ABE model, each user is assigned an access structure, represented graphically as an access tree. Each user secret key reflects his access structure, and each file is encrypted using specific public key components that correspond to the attributes assigned.

Implementing this cryptography principle will enhance the original system in multiple ways:

First, we achieve fine granularity and great flexibility in our access control. By setting

a suitable universe of attributes, we are able to define a very complete access policy. The combination of these attributes in logical expressions can generate a wide range of access structures for the users. As mentioned earlier, data is encrypted by associating the desired set of attributes to a file. Consequently, the efficiency of our system does not lie with the number of users but with the number of attributes, hence its scalability.

On the other hand, we will make an important progress in terms of user privacy and data confidentiality. Data remain protected through its storage and sharing, even for authenticated users; they will not be able to access any readable information as long as do not have the right access permissions set in their access structure. As will be pointed out below, our Cloud server has access to neither file contents nor users' specific privileges, since essential information will not be disclosed.

Furthermore, both file management operations such as creation, update and deletion and giving access to new users are stand-alone tasks, which means that unconcerned files or users will remain unaffected.

One of the greatest challenges for the design of this new component is user revocation. The revocation process will inevitably involve the re-encryption of all data files accessible to the leaving user, using a new set of keys that should remain unknown to the revoked user. Furthermore, our system will be able to distribute the new set of public keys in accordance with the changes made.

Fortunately, users will be able to delegate most of the revocation complexity to the Cloud servers, taking advantage of Proxy Re-Encryption. Additionally, Lazy Re-Encryption and complementary design decisions will be considered in order to reduce the computational burden.

3.3.2 Proxy Re-Encryption (PRE)

Proxy Re-Encryption technique allows us to delegate most of the computational burden caused by user revocation to the Cloud. PRE scheme enables the Cloud server to efficiently translate encrypted data under specific public key components to data encrypted by an updated version of such components without disclosing the original plaintext. So, both data confidentiality and integrity are not compromised throughout the process. The Cloud server will also keep track of users' secret keys by storing most of their components in its database. Obviously, such personal data will remain encrypted and, most importantly, it is provably secure against any threat since users will always keep secretly an essential component for decryption. Secret keys will also be updated at the Cloud side following the same process. Users will be adequately notified and provided with the new up-to-date components when appropriate. Data owners will exclusively be responsible for generating a re-encryption key for the updated public key components and then provide it to the server. On receiving this data, our Cloud will be able to execute the described process of updating outdated stored data when appropriate.

3.3.3 Lazy Re-Encryption

Proxy Re-Encryption conveniently enhances user experience. However, it introduces a heavy computation burden on the Cloud servers. So, we must compensate somehow such workload intensification.

Lazy re-encryption alleviates this computation overhead on the server side. Our Cloud server will be able to "aggregate" multiple re-encryption tasks into one, reducing significantly the computation costs of their management operations. This is accomplished by executing PRE tasks only when needed. Re-encryption tasks are not executed whenever a data owner generates a new re-encryption key. Instead, re-encryption keys will be kept provisionally and used exclusively when an outdated file request from a user occurs. As a result, the re-encryption overhead will be amortized progressively. Further, multiple re-encryption keys can be recorded and later be merged into a single operation.

3.4 Architecture

The architecture of StackSync has been slightly modified to achieve security with a high granularity when sharing files. As can be seen in Fig. 16, CloudABE will be deployed as wrapper over the sync service.

Desktop clients will have the option to extend their services using this wrapper that will manage all ABE notifications. The interaction between a client and CloudABE will pass through the SyncService. All extra generated metadata flow will be exclusively managed by the brand-new wrapper. In this manner, we preserve most of the original design, attaching the new wrapper to the architecture in such a way that the key components of our system remain decoupled.

Lastly, the metadata database design has been modified so it can store all extra metadata related to the new protocol. Evidently, such information will be stored adjacent to the original design, and it will not have any impact on the database model managed by the synchronization service.

- **SyncService**: It is in charge of managing the metadata involved in file synchronization. As can be seen in the figure, it is using a database to store all the processed metadata. Desktop clients communicate with the SyncService for two main reasons: 1) to obtain the changes that took place when they were offline; and 2) to commit a new version of a file.
- **OpenStack Swift**: OpenStack Swift is a highly available, distributed, eventually consistent object store. The desktop client interacts with it to store and retrieve file chunks.
- **Desktop client**: The StackSync client is an application that monitors local folders and synchronizes them with the Cloud. It interacts with the synchronization, storage and encryption services. When a file change is detected, it first processes and encrypts the file with the user attributes, and then, it uploads the raw data to OpenStack Swift. Finally, the desktop client sends the associated metadata to the SyncService in order to commit the change and distribute the notification to the affected devices.

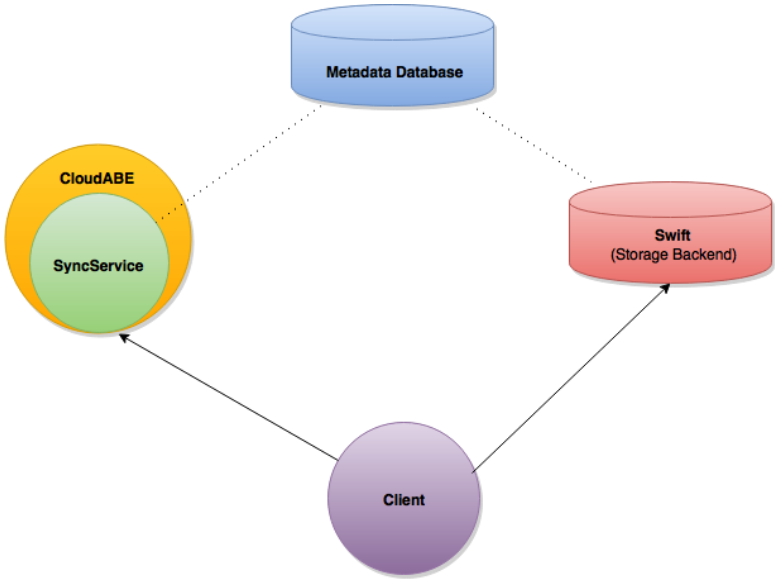


Figure 16: Integration of the CloudABE server with the StackSync architecture

- **Metadata database**: It is used to persist metadata related to users and files. The metadata database is currently a PostgreSQL database.
- **CloudABE wrapper**: This is the new component of the StackSync architecture. It will be responsible of the CloudABE encryption. Desktop clients will interact with it through the SyncService in order to exchange ABE metadata, such as attributes and public keys.

3.5 Operations

The new component enhances the basic operations of our system, allowing users to manage sharing preferences and access to resources. Most of the operations that will be described in this section are attached to those analogous to the traditional StackSync operations, coupling the KP-ABE scheme to the basic functionality.

In order to provide a high level vision of the implemented operations, we will avoid the need to describe in detail some cryptographic operations that are common knowledge like $ASetup$, $AKeyGen$, $AEncrypt$ and $ADecrypt$. For more details on these cryptographic algorithms, the reader is invited to look at [21] [20] [19].

Setting up the system

This operation is mainly executed by the data owner. First, a universe of attributes will be defined. Such universe of attributes will be a complete set of meaningful properties able to describe in proper detail the context in which the system will be deployed. This will allow the data owner to enjoy fine-grained user access control over all data shared.

Table 3.5a Notation used in the scheme description

Term	**Definition**
dow	Data Owner
PK	System Public Key
MK	System Master Key
PKC_i	Public Key Component for attribute i
USK	User Secret Key
$USKC_{dummy}$	Dummy User Secret Key Component
AU	Attribute Universe
AS	Access Structure
ATS	Attribute Set assigned to a file
M	Content of a file in plaintext
SYMK	Symmetric encryption key of a data file
E_{SYMK}	Symmetric encryption key of a file encrypted according to the KP-ABE protocol
$SYMEncrypt(SYMK, M)$	Encrypt a plaintext M with symmetric key SYMK using a secure symmetric encryption method
$SYMDecrypt(SYMK, E)$	Decrypt a ciphertext E with a symmetric key SYMK using a secure symmetric decryption method
$ASetup(k)$	Generate a system public key PK and master secret key MK
$AKeyGen(AS, MK, PK)$	Generate a user secret key SK from a given access structure AS
$AEncrypt(M, PK, ATS)$	Encrypt a plaintext with an attribute set ATS given and the system public key PK, using the KP-ABE protocol
$ADecrypt(E, ATS, USK, PK)$	Decrypt a ciphertext E encrypted under attribute set ATS, given the user secret key USK of a user and the system public key PK, using the KP-ABE protocol

System setup is based on a randomized algorithm that takes a security parameter k and the given universe of attributes AU. This operation will output the public key components PK corresponding to each attribute and a master key MK that will be kept in secret by the data owner. The data owner will finally send all generated public components to the Cloud.

Algorithm 1 SystemSetup(AU, k)

dow: $(PK, MK) \leftarrow ASetup(k)$
dow: $sendToCloud(PK)$

Example Usage: Let us recall the simple scenario shown in Fig. 15. The data owner must have set up the system before interacting with users and sharing files. The data owner first defines the universe of attributes as "A", "B" and "C" standing for "Accounting, "Budgeting" and "Computing", respectively. Then, it calls the algorithm $ASetup(k)$, which outputs PK and MK. The data owner then saves MK in order to keep it secretly and sends the generated PK to the Cloud, including its PKC_i for each of the attributes belonging to the defined AU.

Granting permission to users

The data owner will be responsible for choosing a suitable access structure AS for a new user wishing to join the system. As we already stated, an AS can be defined as an access tree over data attributes. Building the access policy for a new user becomes as simple as depicting its privileges as a combination of those attributes.

Secondly, the data owner calls the algorithm-level operation $AKeyGen$, which outputs the secret key USK for the new user. The data owner will then deliver the generated credentials (AS, USK) along with the system public key PK securely to the new user.

Lastly, the data owner will send all the cryptographic components comprising the generated USK except for the one corresponding to a complimentary "dummy attribute". This essential component is contextless, and most importantly, it will remain undisclosed to the Cloud server and kept as a secret exclusively by the new user. Such a design will allow the Cloud server to store and manage secret keys, while this fact will not imply any vulnerability to our system. These disclosed secret key elements will not give any extra advantage to either the Cloud server or third parties in the decryption of any data stored, since there still exists one undisclosed secret key component ($USKC_{dummy}$), essential in decryption tasks.

Algorithm 2 UserGrant($user, AS$)

dow: $USK \leftarrow (AS, MK, PK)$
dow: user.$deliverCredentials(AS, USK, PK)$
user: user.credentials.$put(AS, USK)$
dow: $sendToCloud(USK - \{USKC_{dummy}\}, PK)$

Example Usage: Once the data owner has executed the system setup as described in the previous section *Setting up the system*, it is ready for new users to join the system. The data owner builds the access structures for the users Alice and Bob as $("C''AND("A''OR"B''))$ and $("A''AND("B''OR"C''))$, respectively. The data owner will then execute the crypto algorithm $AKeyGen$ for each of the users in order to generate their secret keys as follows:

$$USK = AKeyGen(AS, MK, PK),$$

where AS is the defined access structure for each user. We note here that MK is only known and kept secret by the data owner. The generated credentials (AS, USK) will be delivered to each of the new users via a proper and secure out-of-band communication. On receiving and verifying the credentials received, Alice and Bob would respectively accept AS as their access structure and USK as their secret key for decryption tasks. The data owner will then also send the credentials to the Cloud, making sure that the component corresponding to the "dummy attribute" is not disclosed.

File upload

The following tasks will be performed by the data owner together with the basic StackSync calls for file uploading (SyncService and Swift interaction). This process is enhanced in the desktop client side in such a way that data owners will now be able to execute KP-ABE calls against our brand-new CloudABE module.

The data owner will firstly perform previous data processing such as data chunking, as executed in the traditional StackSync desktop client. The data owner will then select a symmetric encryption key $SYMK$ with which the data will be encrypted. Each chunk will be encrypted using this $SYMK$ and then sent to the Cloud along with the corresponding generated metadata. SyncService and Swift modules will be responsible for managing the storage of metadata and encrypted chunks, respectively.

The second stage of this operation is where the main KP-ABE encryption tasks are executed. The data owner first defines a set of attributes ATS for the file, reflecting the scope in which the file can be shared. This step is particularly important since it will determine the subset of users that will have granted permission to access the data uploaded.

The last step will be to encrypt $SYMK$ used in the encryption of data file chunks with the chosen ATS, following the KP-ABE scheme. The data owner will call the algorithm-level operation $AEncrypt$, which will output $ESYMK$ corresponding to the $SYMK$ encrypted under the KP-ABE protocol. The data owner will finally send all metadata generated by the described encryption tasks to the CloudABE module as follows:

FILE ID	ATS	E_{SYMK}

The following algorithm will briefly describe the second stage of the file upload operation, including mostly the KP-ABE file encryption procedure. We will therefore skip the earlier file processing tasks related to the SyncService and the back-end storage modules.

Algorithm 3 $FileUpload$(FILE ID, $SYMK$, PK, ATS)

dow: $E_{SYMK} \leftarrow AENcrypt(SYMK, PK, ATS)$
dow: $sendToCloud$(FILE ID, ATS, E_{SYMK})

Example Usage: Let us recall the case shown in Fig.15. The data owner wishes to share a new resource, such as a document or a report. The file is first processed by the native

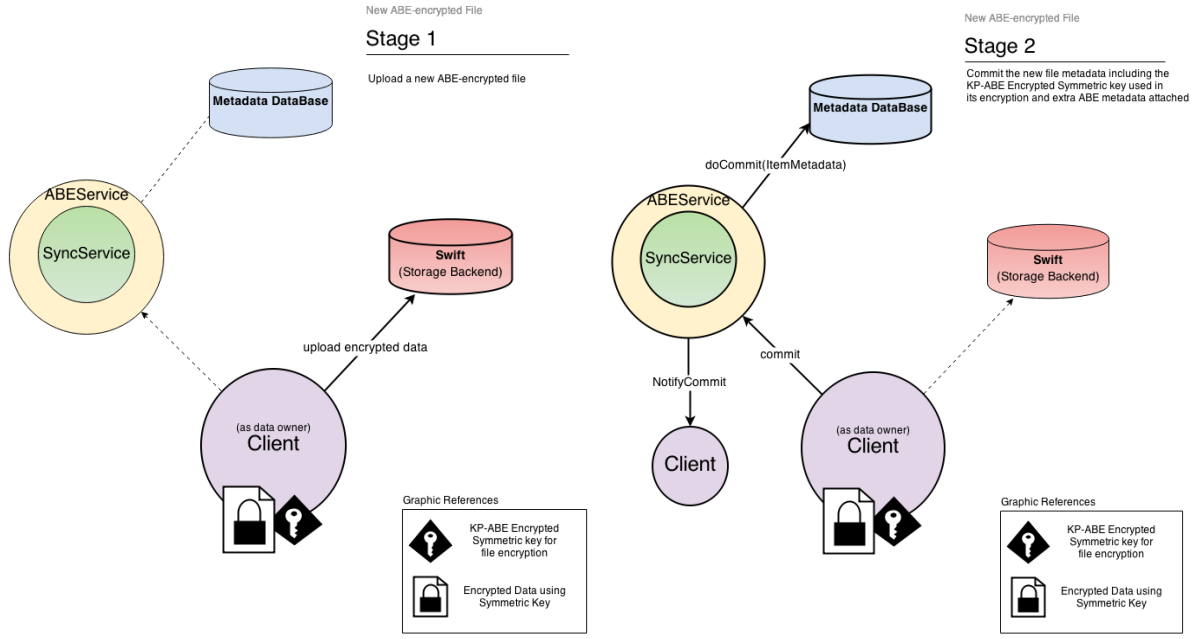
StackSync Client tasks, breaking it up into chunks, amongst other operations. In order to encrypt these chunks, the data owner first defines a $SYMK$ and then computes

$$SYMEncrypt(SYMK, M)$$

on each chunk, where M denotes the raw data in plaintext. The data owner sends all metadata generated from the previous file processing to the SyncService and then sends the encrypted chunks to the storage back-end. From this point onwards, the data owner will exclusively interact with the CloudABE module during the rest of the file upload process. The data owner decides that "Accounting" and "Budgeting" will define the context in which the uploaded document will be shared. Therefore, "A" and "B" conform the ATS for the new file. Then, the data owner computes

$$E_{SYMK} = AEncrypt(SYMK, PK, \{"A", "B"\})$$

and finally sends the ciphertext including the encrypted symmetric key (previously used in the encryption of chunks) along with the attached attributes to the Cloud server. This call will be directly served by the CloudABE module, where the KP-ABE metadata sent will be properly stored for further management.



(a) Data Owner generates the symmetric key and upload raw encrypted data to Swift.

(b) Data Owner uploads file metadata to the SyncService.

Figure 17

File Access

Analogously, the file access operation has been modified in order to append the KP-ABE functionality. StackSync will attend file download requests from authenticated users

in a similar way as before. After the requesting user has downloaded the file chunks, extra encryption metadata will be received from the new module as follows:

FILE ID	ATS	E_{SYMK}

The requesting user will then perform the suitable operations for the decryption of the received data. The client first decrypts data files by calling the algorithm-level operation $ADecrypt$ over $ESYMK$, using the secret key owned by the requesting user. Since the access structure is implicitly included in the USK, $ADecrypt$ operation will output the decrypted $SYMK$ if and only if the ATS related to the downloaded file satisfies the access structure assigned to the requesting user.

Once E_{SYMK} has been successfully decrypted according to the KP-ABE protocol, the user will be able to decrypt data chunks using the obtained $SYMK$ in order to access the data file in plaintext.

Algorithm 4 $FileAccess(chunks, ATS, E_{SYMK}, USK, PK)$

user: $SYMK \leftarrow ADecrypt(E_{SYMK}, ATS, PK)$
user: FILE $\leftarrow SYMDecrypt(SYMK, chunks)$

Example Usage: Let us suppose that Bob sends a request for the previously uploaded file by the data owner as shown in Fig.15. Bob receives the following information from the Cloud server:

FILE ID	encrypted chunks	Sync metadata	{"A", "B"}	E_{SYMK}

On receiving this response from the Cloud, Bob recovers his USK and calculates

$$ADecrypt(E_{SYMK}, \{\text{“}A\text{”}, \text{“}B\text{”}\}, USK_{BOB}, PK)$$

We remind the reader that the AS assigned to Bob is $(\text{“}A\text{”}AND(\text{“}B\text{”}OR\text{“}C\text{”}))$. Since the $(\text{“}A\text{”}AND\text{“}B\text{”})$ clause is satisfied in his access structure, Bob will be able to decrypt $SYMK$ correctly using his USK. For the decryption of the encrypted chunks received, Bob will compute

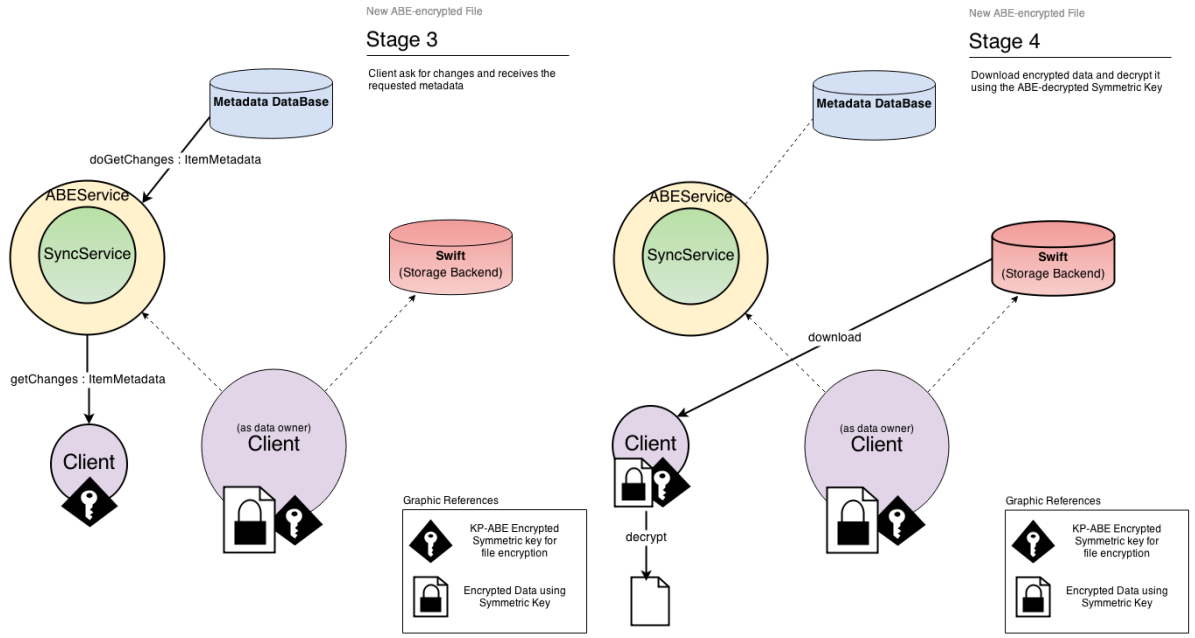
$$\text{FILE} \leftarrow SYMDecrypt(SYMK, chunks)$$

and finally obtain the plaintext corresponding to the requested file. Note that in case Alice requested the file she would not be able to decrypt $SYMK$ on the first stage, since her $AS = (\text{“}C\text{”}AND(\text{“}A\text{”}OR\text{“}B\text{”}))$ would not be satisfied by the attribute set assigned to the document.

User Revocation

One of the most challenging design factors is that of implementing the user revocation process. Fortunately, the KP-ABE scheme proposed in [19] includes a suitable and efficient approach for the implementation of this feature in our system.

Removing a user from the system inevitably requires the redefinition of keys. However, only a minimal set of key components will need to be re-computed. Besides, this process will be mainly carried out on the server side, enabling data owners to delegate most of the computation tasks to the Cloud server and go offline straight after executing a few simple operations. We must also emphasize that this fact will not suppose a heavy computation overhead for the Cloud, since re-keying and updating tasks will be executed in a "lazy" fashion, as the system requires minimizing the impact on efficiency.



(a) Client receives notification. (b) Download and decrypt data.

Figure 18

3.6 StackSync ABE Integration

Here we describe the modifications we have to make to **StackSync** in order to integrate the Attribute-Based Encryption into the synchronization protocol.

3.6.1 ABE Library

The first step in order to integrate our proposed KP-ABE protocol has been to develop an stand-alone library that implements all the internal logic of the protocol such as key generation, encryption or decryption.

The library has been developed in the most flexible way in order to be able to use it

without any condition such as an specific database, a determined file structure or file organization. In this way, integration will be much easier.

We differentiated between two user roles in order to make the library much more intuitive. The roles are the data owner and the invited users. The data owner has the required methods to do the setup of the system, encrypt or generate keys, which are: *ASetup(k), AKeyGen(AS, MK, PK), AEncrypt(M,PK,ATS)*. On the other hand, the invites user has only the required methods to decrypt, such as: *ASetup(k), ADecrypt(E, ATS, USK, PK)*.

Furthermore, the library contains auxiliary methods such as lexical analyzers to interpret access structure expressions, access tree builders or mathematical methods required for the different ABE operations.

Finally, the library includes a graphic interface in order to let the user draw an access tree that later will be interpreted by the library with the mentioned methods. An attribute selector graphic interface has also been implemented in the library in order to let the data owner choose which attributes will have the new shared file.

3.6.2 Commons, Communication and Contract

ABE introduces a set of new objects that must be included in the communication protocol between desktop client and SyncService. The source code for this can be found in the "commons" repository under StackSync's github main page.

The contract gets directly affected by this changes. We are talking about the information that our calls will now have to accept, transport and return. In order to ease the complexity that this enhancement would introduce to the existent contract, we take advantage of some of the most valuable techniques in Object Oriented Programming: inheritance, polymorphism and abstraction of classes.

Fortunately, our communication layer is completely tolerant to such techniques. Besides the improvement in the modularity of our code, these techniques will improve code readability and maintainability. The main changes introduced as well as the involved classes within commons in this process will be described in detail below.

3.6.3 ABE Classes

SyncMetadata

This introduced object will represent the abstraction of what originally was an ItemMetadata object, encapsulating as its name suggests all metadata related to the items to be synchronised in the system (files, workspaces...).

SyncMetadata is an abstract object that includes the common basic attributes for any existent metadata in the synchronisation process. This will allow us to abstract the original tight dependency between the common contract and the ItemMetadata object, whereas any object now inheriting from SyncMetadata will be valid to inject in our communication calls.

As a result, we will have two classes extending from this parent: the original ItemMetadata object, that will keep including the exact same type of information, and the new ABEItemMetadata object encapsulating all metadata related to the ABE protocol. At the same time, ABEItemMetadata will extend from ItemMetadata, so its inheritance relationship with SyncMetadata will be in this case implicit.

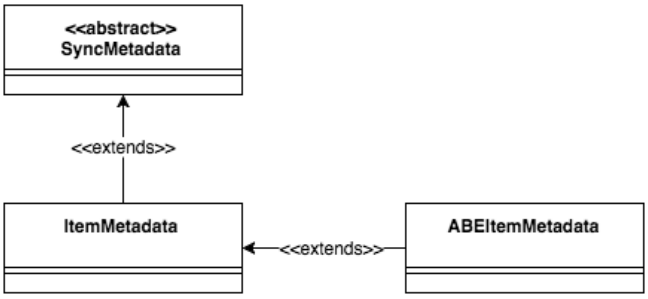


Figure 19: SyncMetadata class diagram.

ABEItem

Similarly to the relationship ABEItemMetadata - ItemMetadata, ABEItem will extend from the original Item object. In this case we will not require an abstraction of this object since our contract does not directly require neither of these objects but they will be accessed in client and server modules of our system.

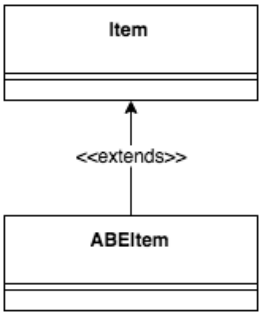


Figure 20: ABEItem class diagram.

ABEWorkspace

Similarly to the relationship between a traditional Item and an ABEItem, an ABEWorkspace will extend from Workspace.

ABEMetaComponent and Attribute

These will be objects strictly necessary for the integration of the ABE scheme and specific for it. These are mandatory to appear in commons since both ABEItemMetadata and ABEItem objects depend on them.

ABEMetaComponent defines each of the encrypted key components, whilst Attribute is the corresponding object for an attribute in ABE. Find more details about these in the figures below.

3.6.4 SyncService

The new elements introduced in the contract must now be handled by the synchronisation service side. Here, we will also make use of class polymorphism, abstraction and inheritance. However, the most challenging question in the SyncService side is, in which layers shall we abstract?

The answer is not trivial. The workflow begins at the boundary class SyncServiceImp, which implements the contract defined by ISyncService that can be found within commons. This last often forwards the synchronisation call to the handler, which receives the requested operation including the necessary objects. This layer is represented by the interface SyncHandler.

SQLSyncHandler, the class that implements the interface defined by SyncHandler, is responsible for handling the requested operations accordingly. It will mainly call the corresponding Data Access Object in order to perform a suitable set of operations: persisting, updating or retrieving metadata.

A particular Data Access Object (DAO) is defined for each of the models represented in persistence. For instance, the DAO that executes the tasks against the synchronisation database related to the persistence, update or access or Items will be ItemDAO. However, ItemDAO will be defined as an interface in order to abstract the logic of the handler from a specific database implementation.

We will therefore have an interface and at least a specific implementation for each of the objects to store in the SyncService persistence. In our case, the specific implementation is that of PostgreSQL database, therefore, PostgresqlItemDAO will be the class implementing all SQL calls to our database, and returning the requested objects accordingly if applies. This closes our information flow within the SyncService.

Now, we can not create an extra call for ABE in the boundary SyncServiceImp methods since it is tied to the contract defined, and we have already abstracted the dependency flow in that layer by our progress made in commons.

We could create an extra Handler implementation exclusively responsible for handling ABE requests, but that would introduce heavy replication in the lower level layers.

Information flow will be therefore abstracted until it hits the DAO layer. The DAO layer will be our extension point for ABE in the SyncService logic.

ABEItemDAO and PostgreSQLABEItemDAO classes

ABEItemDAO is our main extension point for ABE synchronisation metadata storage and retrieval in the server side.

ABEItemDAO is indeed an interface that extends from the existent ItemDAO interface, meaning that every implementation of this interface will have to implement the same methods as ItemDAO, handling all ABE metadata accordingly, plus its specific ones, in this case, the getABEItemsByWorkspaceId method.

PostgresqlABEItemDAO is for now our default implementation of the ABEItemDAO interface. This class will execute the operations against PostgreSQL database handling ABE

metadata accordingly.

3.6.5 PostgreSQL database

Minimum changes have been added to PostgreSQL by the SyncService in order to fit all ABE metadata for further management.

Table Attribute

Storage of all metadata related to the attribute concept in the ABE scheme. This includes a unique generated identifier (UUID), the name of the attribute, its latest version, its public key component and a json including the history list for versions referent to this attribute.

Table ABE Component

Storage of a component of metadata related to the ABE encryption of a file. This includes a unique generated identifier (UUID) as well as the identifier of the item the metadata references to, and the attribute ID used in this component encryption, the encrypted public key component and the version.

This constructs a simple relational dependency between tables as described in the Fig. 21.

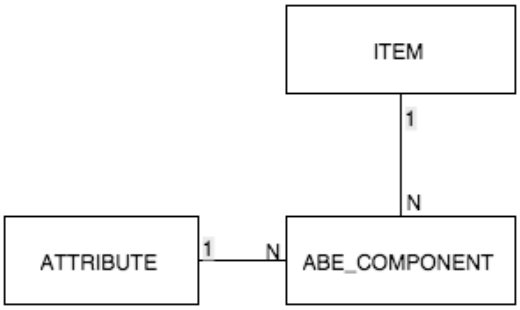


Figure 21: ABEItem class diagram.

3.6.6 Desktop client

The new privacy approach concentrates a great part of the logical complexity of its operations on the client side. Thus, it is imperative for us to keep the StackSync desktop as light as possible whilst preserving its original architecture and functionality.

All standalone logic related to the ABE, containing most client-side necessary algorithms implemented in Java language, has been tweaked and formatted in such a way that it is ready to be utilised as a client-side library within the StackSync Desktop Client as well as other future projects.

Obviously, some extra implementation has been added to the original desktop client. In this case, abstraction has been brought to the encryption layer, which will be explained in detail next.

The most relevant change introduced in addition to minor extensions in the desktop client core implementation is that of abstraction of the Encryption class. The original encryption technique implemented in the client side, based mainly on symmetrical encryption, will now become BasicEncryption, implementing a simple Encryption interface including the two basic operations within all encryption processes:

```
public interface Encryption {

    public CipherData encrypt(PlainData data);

    public byte[] decrypt(CipherData data);

}
```

Figure 22: Encryption interface.

Then, our new encryption implementation on the client side will be named ABEEncryption and will indeed implement the Encryption interface, hence the abstraction depicted in Fig. 23.

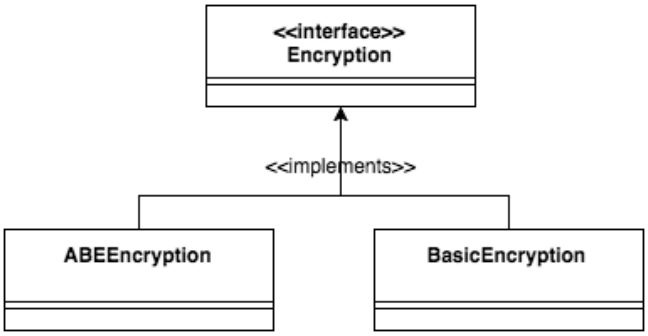


Figure 23: Encryption diagran.

We must although take into consideration that such abstraction contains certain dependent classes in the interface level. If we have a closer look at the Encryption Interface above, we can see at least two involved classes: CipherData and PlainData. Those are the classes encapsulating all necessary data and metadata necessary for the execution of the encryption tasks.

Now we will have two different extensions for each of these abstractions: BasicCipherData and ABECipherData for the CipherData abstract class; BasicPlainData and ABEPlainData for the PlainData abstract class. The class diagram is depicted in Fig. 24 .

3.7 Evaluation

3.7.1 Methodology

In this section we will explain how we correctly evaluate the cost that will assume a real system such as StackSync after integrating ABE as a privacy enhancement. In order to achieve

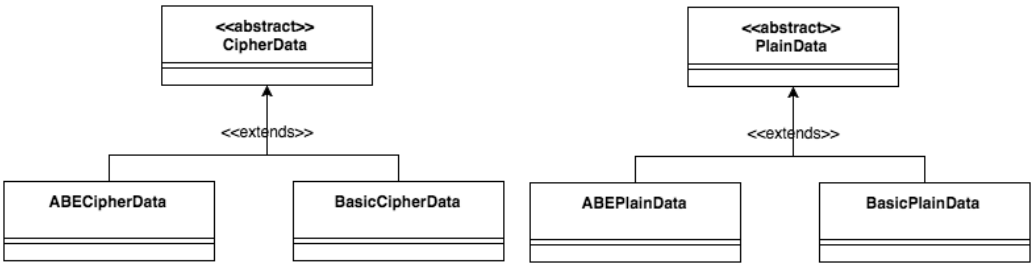


Figure 24: CipherData and PlainData class diagrams.

that, we first set up a simulated environment where we later executed a battery of monitorized stress tests using a wide range of configurations.

The benchmarking process is focused on the extraction and processing of useful data via monitorization from a set of tests executed in a controlled environment. Such data should be able to provide us with certain information, following the requirements that will be described later.

Hundreds of tests were executed, monitorized and processed in order to extract a meaningful set of values for the benchmarking of the results.

The first step is to create a specific target or set of requirements defining the criteria used for the benchmarking: First, we need to know how efficient is ABE data sharing compared to the baseline or original data sharing in StackSync. Secondly, we shall extract information about how costly is ABE in every stage of the workflow. Lastly, we want to know which part of the total cost represent each of the ABE features, breaking down the analytics into the different operations.

During the stress testing execution, we made sure all processes in the system had been shut down except for those essential ones to run the tests: system, SyncService and the two desktop clients.

A Wireshark process had also been added although the data gathered has not been plotted as it is not as meaningful as the rest of the benchmarking process. The evolution of the introduced data overhead by ABE can be easily deduced from the captions and has not been considered as interesting as its actual performance benchmarking, focusing instead on time resources and efficiency.

Tests are run by a shell script, which triggers certain repetitions for a set of actions with waiting intervals. The actions performed are basically to copy a file of defined size into the workspace of our client running as data owner. This, at the same time, will trigger all the synchronisation logic implemented, uploading the file and updates to the Cloud, which, in this case will be the SyncService and Swift running in local environment.

After this, the desktop client playing the role of a consumer will be notified of the changes in the shared workspace, downloading the raw data, and decrypting it using ABE thereafter.

After a defined cooldown period, this process is repeated by the bash script using a different set of parameters for a different set of features, that will be further described in the next section.

3.7.2 Cases and Scenarios

We have designed a suitable set of cases using different scenarios for our tests, in order for the output to be generated including data that we consider meaningful in our analysis.

First, we have defined a set of dynamic parameters, system properties or features for the execution of our tests.

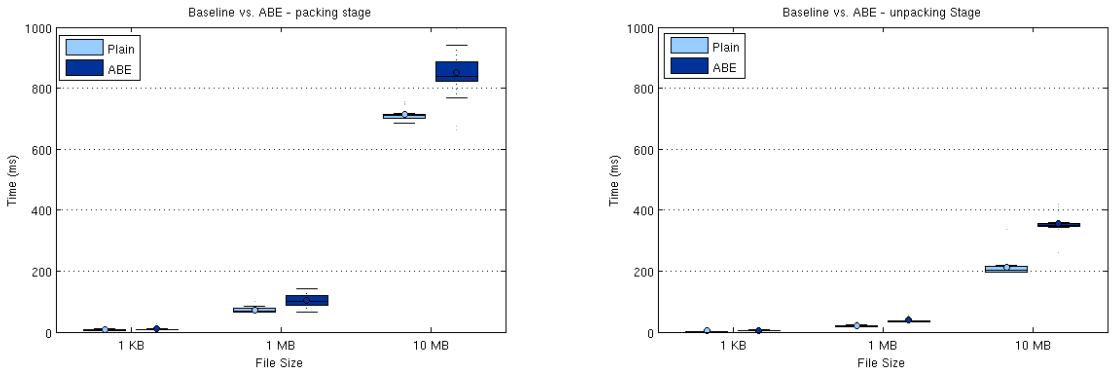
- Encryption type: None or ABE.
- File Size: 1KB, 1MB and 10MB.
- ABE attribute set: 3, 6 and 9.

A particular combination of these parameters will define a specific scenario for every test case, depending on the feature to be observed.

Generally, we will be observing the two most significant stages of the workflow of our system: packing and unpacking. Packing is the stage where data gets encrypted and compressed and thereafter sent to the Cloud. In the opposite way, unpacking is the stage where data gets uncompressed and decrypted, and thereafter sent to the Cloud.

3.7.3 Results

Encryption and Decryption Overhead



(a) Encryption stage. (b) Decryption stage.

Figure 25: Encryption and Decryption Overhead

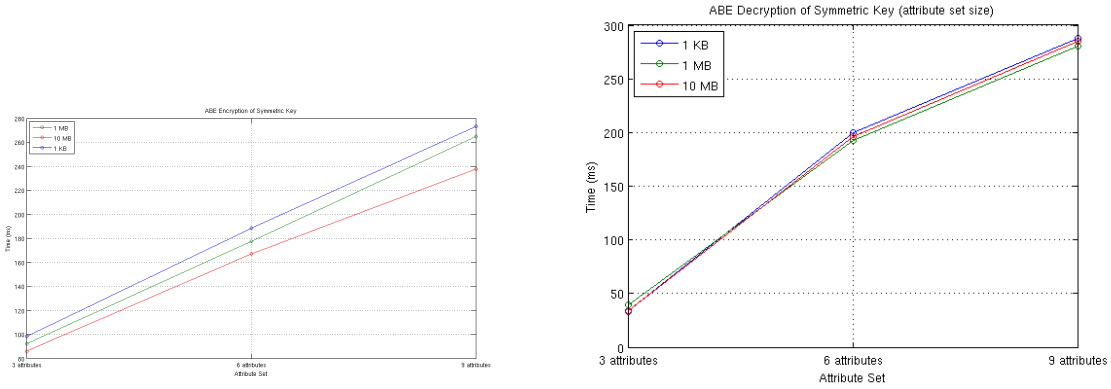
By observing both box plots from Fig. 25 we can see how, in general terms, computational cost has been introduced by the new privacy approach.

We can also observe how the unpacking stage (downstream) takes roughly half the time for the packing stage (upstream), and also the variance for the data collected for the packing stage is considerably higher. Nevertheless, this does not give us any meaningful information regarding the target of this study.

Impact of Attribute set size

In Fig. 26 we can see how both plots have clearly different tendencies: encryption has a linear evolution as we increase the attribute set used in the encryption, whereas decryption follows a logarithmic one. This confirms then the assumptions made in conceptual research.

The specific properties of these line regressions will be nevertheless subject of further research.



(a) Encryption stage. (b) Decryption stage.

Figure 26: Impact of attribute set size.

3.8 Use Manual

Share a folder

Sharing a folder is a very easy and simple process. We must look for our stacksync icon in the taskbar and right click over it to select share folder option.

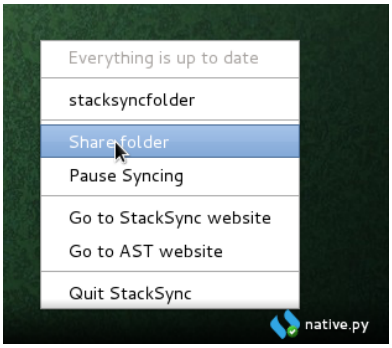


Figure 27: StackSync desktop client tray.

This will open a window that will ask us about the folder we want to share and the people we want to share the folder with. In this windows we will also be able to choose ABE encryption.

After filling the window, another one will appear to set the access structure for each written email.

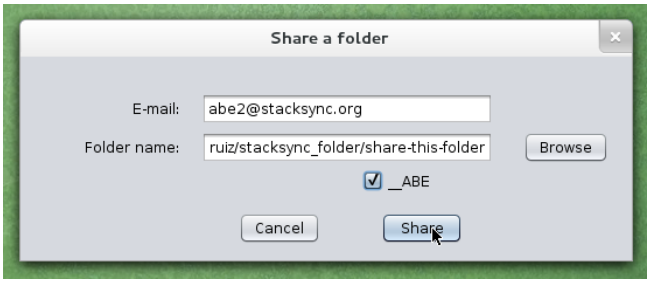


Figure 28: Panel to share a folder.

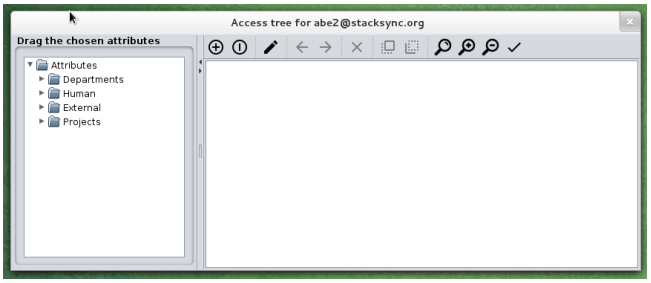


Figure 29: Access tree generation.

We can drag any attribute from the attribute tree to drop it into the drawing zone. Attributes can be connected with logical operators as the + and | that can be seen in the figure in order to construct an access policy, as the one in the following example:

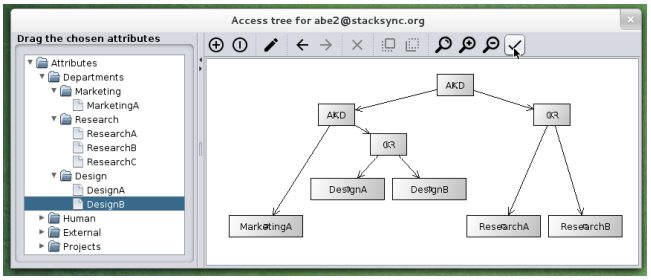


Figure 30: Access tree example.

Once it is done, access policy will be saved and the Cloud will do the rest.

Share a file

Sharing files in an ABE workspace is a very simple task. We only have to put a file into the shared folder and a windows with an attribute selector will appear.

As simple as clicking onto the different attributes we want to assign to a determined file. Once done, just click *OK*, and let the computer do the rest.

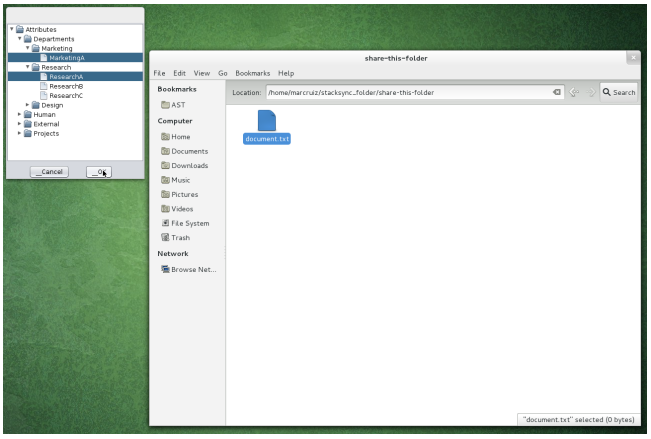


Figure 31: Example on how to select attributes to encrypt a file before synchronization.

3.9 Conclusions

In this section, we have presented the privacy-aware data sharing with Attribute Based Encryption in StackSync. With ABE and StackSync, users can share folders without worrying about the content of the files. Furthermore, it allows a fine-grained file encryption which can be useful for large companies where sharing data with many users is a must.

References

[1] "Google compute engine down by 10%, 240 million drive users," http://thenextweb.com/google/2014/10/01/google-announces-10-price-cut-compute-engine-instances-google-drive-passed-240m-active-use accessed: 2015-08-16.

[2] "Health apps run into privacy snags," http://www.ft.com/cms/s/0/b709cf4a-12dd-11e3-a05e-00144feabdc0.html, accessed: 2015-08-16.

[3] Y. Gurevich, E. Hudis, and J. M. Wing, "Inverse privacy (revised)," Tech. Rep. MSR-TR-2015-37, May 2015. [Online]. Available: http://research.microsoft.com/apps/pubs/default.aspx?id=245268

[4] S. B. Barnes, "A privacy paradox: Social networking in the united states," First Monday, vol. 11, no. 9, Sep. 2006. [Online]. Available: http://firstmonday.org/ojs/index.php/fm/article/view/1394

[5] F. Schaub, R. Balebako, A. L. Durity, and L. F. Cranor, "A design space for effective privacy notices," in Proceedings of the Eleventh Symposium on Usable Privacy and Security, 2015.

[6] D. Downey, M. Broadhead, and O. Etzioni, "Locating complex named entities in web text," in Proceedings of the 20th International Joint Conference on Artifical Intelligence, ser. IJCAI'07. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2007, pp. 2733–2739. [Online]. Available: http://dl.acm.org/citation.cfm?id=1625275.1625715

[7] C. Zhu, F. Wen, and J. Sun, "A rank-order distance based clustering algorithm for face tagging," in Proceedings of the 2011 IEEE Conference on Computer Vision and Pattern Recognition, ser. CVPR '11. Washington, DC, USA: IEEE Computer Society, 2011, pp. 481–488. [Online]. Available: http://dx.doi.org/10.1109/CVPR.2011.5995680

[8] J. Klontz, B. Klare, S. Klum, A. Jain, and M. Burge, "Open source biometric recognition," in Biometrics: Theory, Applications and Systems (BTAS), 2013 IEEE Sixth International Conference on, Sept 2013, pp. 1–8.

[9] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional architecture for fast feature embedding," arXiv preprint arXiv:1408.5093, 2014.

[10] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in Advances in neural information processing systems, 2012, pp. 1097–1105.

[11] M. Ankerst, M. M. Breunig, H.-P. Kriegel, and J. Sander, "Optics: Ordering points to identify the clustering structure," in Proceedings of the 1999 ACM SIGMOD International Conference on Management of Data, ser. SIGMOD '99. New York, NY, USA: ACM, 1999, pp. 49–60. [Online]. Available: http://doi.acm.org/10.1145/304182.304187

[12] M. Harbach, M. Hettig, S. Weber, and M. Smith, "Using personal examples to improve risk communication for security & privacy decisions," in Proceedings of the 32Nd Annual ACM Conference on Human Factors in Computing Systems, ser. CHI '14. New York, NY, USA: ACM, 2014, pp. 2647–2656. [Online]. Available: http://doi.acm.org/10.1145/2556288.2556978

[13] H. Harkous, R. Rahman, and K. Aberer, "C3p: Context-aware crowdsourced cloud privacy," in Privacy Enhancing Technologies. Springer, 2014, pp. 102–122.

[14] P. H. Chia, Y. Yamamoto, and N. Asokan, "Is this app safe?: A large scale study on application permissions and risk signals," in Proceedings of the 21st International Conference on World Wide Web, ser. WWW '12. New York, NY, USA: ACM, 2012, pp. 311–320. [Online]. Available: http://doi.acm.org/10.1145/2187836.2187879

[15] M. Huber, M. Mulazzani, S. Schrittwieser, and E. Weippl, "Appinspect: Large-scale evaluation of social networking apps," in Proceedings of the First ACM Conference on Online Social Networks, ser. COSN '13. New York, NY, USA: ACM, 2013, pp. 143–154. [Online]. Available: http://doi.acm.org/10.1145/2512938.2512942

[16] P. G. Kelley, L. F. Cranor, and N. Sadeh, "Privacy as part of the app decision-making process," in Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, ser. CHI '13. New York, NY, USA: ACM, 2013, pp. 3393–3402. [Online]. Available: http://doi.acm.org/10.1145/2470654.2466466

[17] B. B. Anderson, C. B. Kirwan, J. L. Jenkins, D. Eargle, S. Howard, and A. Vance, "How polymorphic warnings reduce habituation in the brain: Insights from an fmri study," in Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems, ser. CHI '15. New York, NY, USA: ACM, 2015, pp. 2883–2892. [Online]. Available: http://doi.acm.org/10.1145/2702123.2702322

[18] M. S. Wogalter, B. M. Racicot, M. J. Kalsher, and S. N. Simpson, "Personalization of warning signs: The role of perceived relevance on behavioral compliance," International Journal of Industrial Ergonomics, vol. 14, no. 3, pp. 233 – 242, 1994. [Online]. Available: http://www.sciencedirect.com/science/article/pii/016981419490099X

[19] S. Yu, C. Wang, K. Ren, and W. Lou, "Achieving secure, scalable, and fine-grained data access control in cloud computing," in Proceedings of IEEE International Conference on Computer Communications 10, 2010.

[20] V. Goyal, O. Pandey, A. Sahai, and B. Waters, "Attribute-based encryption for fine-grained access control of encrypted data," in Proceedings of ACM conference on Computer and Communications Security, 2006.

[21] A. Sahai and B. Waters, "Fuzzy identity-based encryption," Advances in Cryptology - Eurocrypt, vol. 3494, pp. 457–473, 2005.