EEXCESS

Enhancing Europe's eXchange in Cultural Educational and Scientific reSources

Deliverable D6.3

Second Security Proxy Prototype and Reputation Protocols

Identifier:	EEXCESS-D6.3.docx
Deliverable number:	D6.3
Author(s) and company:	Sonia Ben Mokhtar (INSA), Nadia Bennani (INSA), Lionel Brunie (INSA), Sylvie Calabretto (INSA), Thomas Cerqueus (INSA), Elöd Egyed-Zsigmond (INSA), Omar Hasan (INSA), Albin Petit (INSA), Pierre-Édouard Portier (INSA), Michael Granitzer (Uni Passau), Christin Seifert (Uni Passau), Jörg Schlötterer (Uni Passau)
Internal reviewers:	Mendely
Work package / task:	WP6 / Tasks 6.1, 6.2 and 6.3
Document status:	Final
Confidentiality:	Public
Version	30-10-2015

History

Version	Date	Reason of change
1	2015-10-19	Document created.
2	2015-10-26	Final version for the internal review.
3	2015-10-28	Inclusion of the comments from the internal review.
4	2015-10-29	Final version.

Impressum

Full project title: Enhancing Europe's eXchange in Cultural Educational and Scientific reSources
Grant Agreement No: 600601
Workpackage Leader: INSA
Project Co-ordinator: Silvia Russegger, Jr-DIG
Scientific Project Leader: Michael Granitzer, Uni-Passau

Acknowledgement: The research leading to these results has received funding from the European Union's Seventh Framework Programme (FP7/2007-2013) under grant agreement n° 600601.

Table of Contents

1 Executive Summary

1.1 Overview of the Deliverable

This deliverable presents the main results and achievements of the work package 6 in the last 12 months. It will be structured in 4 parts.

Design of a new attack: SimAttack is a new attack against solutions that aim at preserving users' privacy in web search. This attack has been used to exhibit the limitations of existing solutions, and demonstrate that existing solutions are not sufficient. This work led to the submission of a research paper to the Journal of Internet Services and Applications:

- Albin P., T. Cerqueus, S. Ben Mokhtar, D. Coquil, L. Brunie and H. Kosch. SimAttack: Private Web Search Under Fire.

A simplified presentation of SimAttack is given in section 3.

Design of a new protection solution: PEAS is a new solution to protect users' privacy when they search the web. This work is motivated by the lack of satisfying solutions, as it has been showed using SimAttack.

This work led to two publications:

- A. Petit, T. Cerqueus, S. Ben Mokhtar, L. Brunie and H. Kosch. PEAS: Private, Efficient and Accurate Web Search. In 14th IEEE International Conference on Trust, Security and Privacy in Computing and Communications (IEEE TrustCom 15).
- A. Petit, T. Cerqueus, S. Ben Mokhtar, L. Brunie and H. Kosch. How to Protect Your Privacy Using Search Engines?. In 10th European Conference on Computer Systems (EuroSys 2015), poster session.

The PEAS protocol and experimental results are presented in section 4.

Implementation of the privacy proxy: Section 5 of this deliverable includes a description of the implementation of the second privacy proxy prototype. It presents both the architecture of the system and the services that have been implemented and deployed. The services relate to the PEAS protocol (presented in section 4) and an interaction logging service.

Implementation of client-side components: Section 6 presents the implementation of client-side components. More precisely, it describes a Javascript component that allows client application developers to use the PEAS protocol to protect users' queries. It also describes the user profile interface designed to allow users to specify their personal data and their privacy settings.

1.2 Potential Risks

During this phase of the project, we identified potential risks and started to avoid or address them in the next phase. The potential risks are listed below.

- The project development is iterative. New specifications and requirements can be identified at each iteration. This leads to modifications that need to be integrated quickly by all the partners. For instance, the introduction of a specific call to get the details of a given recommendation has an important impact on the way the privacy-preserving solution is implemented in the system.
- In order to collect information on users interactions with the system, a logging service has been deployed. A solution must be devised to ensure users' privacy while retaining relevant information for the analysis. This problem will have to be addressed in the next phase of the project. Potential solutions relate to k-anonymity techniques.

1.3 Next Steps

In the next phase of the project, we plan to implement the unlinkability protocol of PEAS. It requires the deployment of a new version of the privacy proxy. Until now, the proxy is composed of only one entity (corresponding to the issuer). In the next version, the proxy will be composed of two entities: a requester and an issuer. It includes the adaptation of the implementation of the PEAS protocol to cope with the service providing details of a given recommendation. We also plan to implement the mechanism ensuring users' privacy regarding the logs that are stored on the privacy proxy. This work will be joined with the work package 5. A discussion on these aspects is presented in section 9.3 of deliverable D5.3).

2 Introduction

2.1 Purpose of this Document

This deliverable describes the second prototype for the functionality described in Task 6.1, 6.2 and 6.3. The source code for the prototype software components is available on the GitHub project repository (https://github.com/EEXCESS/privacy-proxy and https://github.com/EEXCESS/peas).

2.2 Scope of this Document

This document describes the second implementation of the privacy proxy. It covers the description of SimAttack (a new attack against solutions that aim at protection users privacy) and PEAS (a protocol that ensures private, efficient and accurate web search). It also covers the implementation of the privacy proxy services and client-side components.

2.3 Status of this Document

This is the final version of D6.3.

2.4 Related Documents

Before reading this document it is recommended to be familiar with the following documents:

- D3.3: Second Federated Recommender Prototype
- D5.3: Second Prototype on User Profile and Context Detection, Usage Analysis Methods and Services
- D6.1: Policy Model for Privacy and Feasibility Report
- D6.2: First Security Proxy Prototype and Reputation Protocols

3 SimAttack: Private Web Search Under Fire

This section presents SimAttack, a generic attack that targets popular Private Web Search solutions. Based on a similarity metric between a query and a user profile, the attack is able to de-anonymize or retrieve initial queries. Our analysis showed that no state-of-the-art solution protects the user in a satisfactory manner (as a high percentage of queries can be de-anonymized or retrieved). We studied different combinations of solutions and showed that combining them is not enough to ensure users privacy.

This section first presents state-of-the-art solutions to protect users' privacy. Then it describes SimAttack. Finally, it presents the evaluation results. The exhaustive list of the evaluation configurations is given but only the most interesting results are presented.

3.1 State-of-the-art Solutions to Protect Users' Privacy

3.1.1 Unlinkability Solutions

An approach to protect users privacy from a curious search engine is to prevent the latter from aggregating queries in a user profile. This can be achieved by generating fake user identities (e.g., taking the IP address of a third party). A basic solution to this end consists in using a Proxy or a VPN in which the distant server forwards user queries to the search engine on behalf of the user and returns results to the user. Unfortunately, this mechanism does not protect the user, as the distant server has the same knowledge as the search engine without any protection and is able to analyze queries. Anonymous networks (e.g., Onion Routing [Goldshlag, 1999], TOR [Dingledine, 2004]) represent a more complex approach to prevent a third party from knowing both the user identity and her query. Indeed, these solutions route user queries through multiple nodes before reaching the search engine. These solutions generate a high number of cryptographic operations, or all-to-all communication, and have a significant overhead (in terms of latency and network traffic), which makes them unpractical for interactive tasks such as Web search.

3.1.2 Indistinguishability Solutions

Indistinguishability solutions consist in making a potential user profile created by the search engine inaccurate. Consequently, the user's privacy is protected, as her interests cannot be discovered. The popular solution in this category, TrackMeNot (TMN) [Toubiana, 2004], periodically sends fake queries on behalf the user. In this approach, the challenge is to create fake queries that cannot be distinguished from the real ones. To do so, TMN based the generation of fake queries on RSS feeds. However, as these RSS feeds are set up by default or manually by the user, an adversary could distinguish between real queries and fake ones. Indeed, the adversary could identify fake queries by using the default RSS feeds or the distance to the user profile. Furthermore, this kind of solutions tends to overload the network by generating a large number of fake queries.

Another possibility to achieve indistinguishability is to modify the initial query. For instance, GooPIR [Domingo-Ferrer, 2009] adds to the initial query $k-1$ fake queries (generated using a dictionary). These sub-queries are separated by the logical *OR* operation in a new obfuscated query. As the authors of GooPIR consider that an adversary has no background knowledge about the user, this adversary can only guess which query was the initial one with a probability equal to $1/k$. However, we show in section 3.4 that SimAttack is able to retrieve the initial query with a high probability considering a user profile created with user's past queries.

3.2 Description of SimAttack

In this section, we present SimAttack, an attack against private Web search solutions. Compared to existing attacks, SimAttack is the first attack against all types of private Web search solutions, as using the similarity metric between a query and a user profile, an adversary is able to personalize SimAttack to any type of protections. The next sections present how SimAttack is able to break several types of protection mechanisms (e.g., anonymous network, adding fake queries or obfuscating a query with fake query).

3.2.1 Unlinkability Attack

The de-anonymization attack consists in linking a query to its requester profile, i.e., find its identity. For each user profile P_u previously collected by the adversary, it computes its similarity with the query q. It then returns the identity corresponding to the profile with the highest similarity. If the highest similarity equals 0 (i.e., all similarities equal 0), the identity of the requester remains unknown and the attack is unsuccessful. Otherwise, the attack considers the user identifier as the issuer of the query q.

3.2.2 Indistinguishability Attack

The attack against indistinguishability solutions identifies initial queries among obfuscated queries received by the search engine. Contrary to the previous attack, the adversary knows the identity of the user and thus tries to pinpoint fake queries by analyzing the similarity between queries and the user profile. The attack first determines which obfuscation mechanism is being used by checking if an obfuscated query $q+$ contains fakes queries, i.e., contains the logical or operator.

Let us consider the first case in which the query $q+$ is composed of $k+1$ queries (i.e., the initial query and k fake queries). The algorithm extracts each aggregated query qi from q+ and computes the similarity metric between these aggregated queries q_i and the user profile P_u. Then it stores the query with the highest similarity in the variable q'. If the similarity $sim(q', P_u)$ is different from 0, the attack returns q' as the initial request. Otherwise, the attack fails and the initial query is not retrieved, as $k+1$ queries are not similar to any user profile.

In the second case (i.e., the query does not contain fake queries), it distinguishes two cases: if the adversary has information about the generation of the fake queries or not. For instance, in TrackMeNot, the adversary is also able to compute fake queries using the pre-defined RSS feeds. These fake queries generated by the adversary are stored in a profile P_{FQ} (same structure as a user profile P_u). If we consider first that the adversary does not use this external knowledge, it evaluates if the similarity between the query $q+$ and the user profile P_u is greater than a given threshold δ. If it is the case, then $q+$ is considered as a real query, and is therefore returned. Otherwise, the query is considered to be a fake query. We now consider the situation in which the adversary knows the RSS feeds used by the user. Thus, she is able to use this external knowledge to distinguish fake queries.

The adversary first compares the similarity between the query $q+$ and the user profile P_u (i.e., $sim(q+, P_u)$) with the similarity metric between the query $q+$ and the profile of fake queries P_{FQ} (i.e., $sim(q+, P_{FQ})$). If $sim(q+, P_u)$ is greater than $sim(q+, P_{QF})$, $q+$ is closer to the user profile than the profile of fake queries. Consequently, $q+$ is considered as a real query, and is then returned. Otherwise, the query is considered to be a fake query.

3.3 Evaluations

3.3.1 Configurations

We conducted numerous experiments to show the limitations of existing state-of-the-art solutions. Table 1 shows the complete list of evaluations and their associated description. In this deliverable, we chose to report only the most interesting results. The other results are described in details in [Petit, 2015a].

Evaluation	ID	Description
Unlinkability solutions	E1-a	Impact of the method parameters (e.g., the smoothing factor)
	E1-b	Impact of the number of users in the system
	E1-c	Impact of considering k users with the higher
	E1-d	Impact of the number of user profiles
	E1-e	Impact of targeting a subgroup of users
	E1-f	Impact of the size of the user profile
	E1-f	Comparison of SimAttack with Machine Learning attack
TrackMeNot	E2-a	Privacy evaluation
	E2-b	Impact of the external knowledge
	E2-c	Impact of the number of fake queries
	E2-d	Impact of the size of the user profile
	E2-e	Comparison with Machine Learning attack
GooPIR	E3-a	Impact of the number of fake queries
	E3-b	Impact of the size of the user profile
TrackMeNot over an unlinkability solution	E4-a	Attacker without knowing user's RSS feeds
	E4-b	Adversary using user's RSS feeds
	E4-c	Impact of the number of fake queries
GooPIR over an unlinkability solution	E5	Impact of the number of fake queries

Table 1: List of evaluations conducted to show the limitations of state-of-the-art solutions.

3.3.2 Comparison with a Machine Learning Attack

In these experiments, we compare SimAttack to a machine learning attack. This attack uses as classifier Support Vector Machine (SVM). The choice of this classifier is based on previous study [Hearst, 1999] that shows that SVM (compared to other classifiers) gives better results for text classification. We reproduce the same condition of the attack: using Weka [Hall, 2009] (a popular open source software dedicated to machine learning) with LibSVM (an efficient implementation of SVM). We chose the same algorithm (C-SVC) with the same type of kernel (linear). We also let the parameter Epsilon (tolerance of termination criterion) to its default one (0.001). However, we optimize the parameter C to obtain the best results. Weka offers a specific option (CVParameterSelection) to test multiple values of the parameter and deduces the one that maximizes the performance of the classification. We found that the best value for C is 1.1.

3.4 Key Results

3.4.1 Impact of the Number of Users on Unlinkability Solutions (E1-b)

The probability to associate a query with its correct requester is inversely proportional to the number of users using the unlinkability solution. To study the impact of the number of users, we run SimAttack over 6 datasets: AOL100, AOL500, AOL1000, AOL5000, AOL10000 and AOL15000. Figure 1 shows that both recall and precision decrease when the number of users in the system increases. For instance, if we consider 100 users, SimAttack de-anonymizes 36.2% of queries with a precision of 42.2% while, for 15,000 users, it only de-anonymizes 11.8% of queries with a precision of 17.9%. Nevertheless, the decrease is not linear and tends to stabilize around 10% for the recall. This means that increasing the number of users using the unlinkability protection does not offer a better protection if this unlinkability protection is already used by a large number of users.

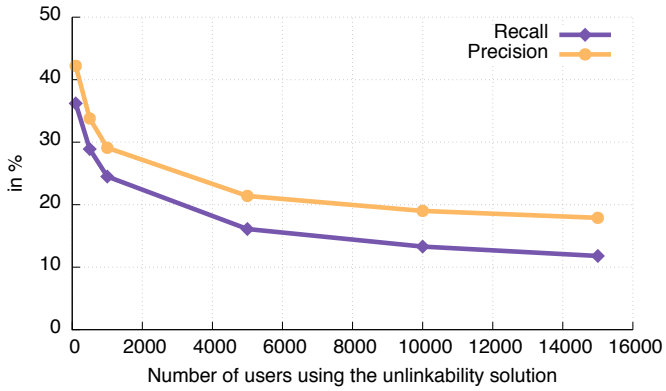

Figure 1: Results of SimAttack according to the number of users querying a search engine through an unlinkability solution.

3.4.2 Comparison of SimAttack with Machine Learning Attack (E1-f)

SimAttack and the Machine Learning attack both aim at linking a query to the user profile of its issuer. That is why, to compare their efficiency, we measure for both attacks the precision and the recall. As shown on Figure 2, we note that both attacks have a similar recall and a similar precision. For instance, if we consider 200 users, SimAttack has a recall of 36.3% and a precision of 41.1% while the machine learning has a recall of 35.6% and a precision of 40.6%. Nevertheless, overall the 5 datasets, SimAttack is slightly better than the machine learning attack especially when a large number of users are considered (on average, the F-Measure of SimAttack is 1.1% higher than the one obtained with the machine learning attack and, for 1,000 users, the F-Measure is 3.3% higher).

3.4.3 Impact of the Number of Fake Queries for TrackMeNot (E2-c)

In TrackMeNot, users can chose the number of fake queries that are issued. However, one can wonder if adding more fake queries gives a better protection to the user. To answer this question, we performed the previous analysis but with different numbers of fake queries: 30 fake queries per hour, 10 fake queries per hour and 1 fake query per hour. This setting consists in adding respectively 7,440 fake queries, 2,480 fake queries and 248 fake queries per user. For the record, the setting used in the two previous sections was 60 fake queries per hour. Figure 3 shows the results of SimAttack (with and without prior knowledge) according to the different settings aforementioned. We can note that the recall is identical for all settings (i.e., 48.0% or 35.7%). Changing the number of fake queries does not change either the user profiles or the users' queries. Thus, this is why changing the setting about the fake queries do not affect the recall. However, we note that the precision increases if we decrease the number of fake queries. Indeed, if there is a lower number of fake queries, the number of misclassified fake queries decreases, thus the precision increases. Consequently, sending too few fake queries helps the attacker to retrieve initial queries with a high precision (i.e., the precision is a bit lower than 100% for the setting 1 fake query per hour).

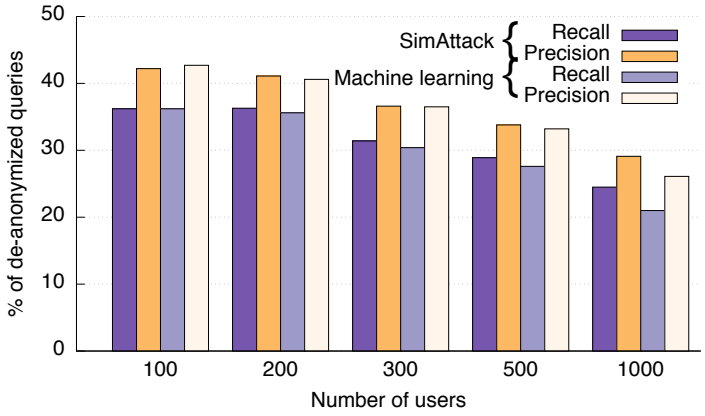

Figure 2: Comparison between SimAttack and a machine learning attack using Support Vector Machine.

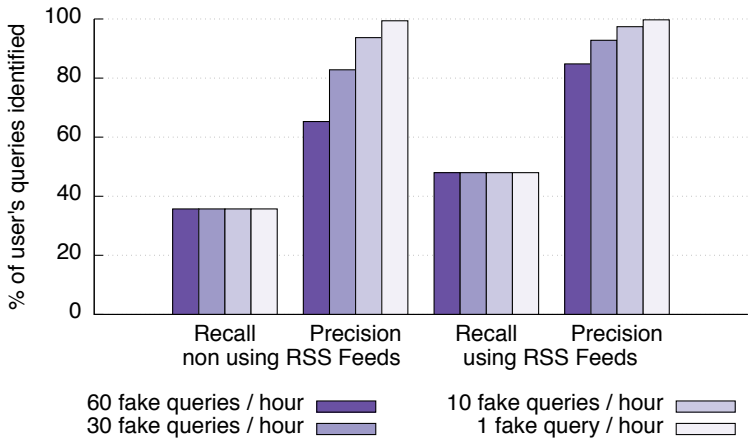

Figure 3: Results of SimAttack considering different numbers of fake query.

3.4.4 Impact of the Number of Fake Queries for GooPIR (E3-a)

Figure 4 shows the performance of SimAttack according to different numbers of fake queries (from 1 to 7). We present these results classified queries in three categories: *Identified*, *Misclassified* and *Unknown*. *Identified* means that the initial query is retrieved by the adversary (corresponds to the recall), *Miclassified* means that the initial query is classified as a fake query, and *Unknown* means that SimAttack could not classify the query, as all similarities between the queries (the initial query and the fake queries) and the user profile equals 0. We note that the number of fake queries has an impact on the protection of the user: when the user sends 1 fake query, SimAttack retrieves 61.2% of initial queries, while when she sends 7 fake queries, this number decreases to 51.4%. Nevertheless, this impact is limited, as adding 6 more fake queries helps to protect only 9.8% more initial queries (if we compare the results of adding 1 and 7 fake queries). Besides, apart from changing the number of fake queries, the percentage of queries retrieved by SimAttack is relatively high (more than half of initial queries are identified).

Furthermore, we study the reason why SimAttack does not identify some queries. We note that SimAttack cannot classify as initial query or fake query a lot of queries (i.e., queries marked as *Unknown* in Figure 4). For instance, when the user sends 1 fake query, 30.3% queries get the status *Unknown*. This is because the initial query gets a similarity with its issuer profile equals to 0. If these initial queries had been related to their issuer profiles, these queries would have identified as initial queries by SimAttack. Consequently, we can consider that GooPIR protects only queries classified as fake queries. For instance, when the user sends 1 fake query, GooPIR misleads the adversary for only 8.5% of queries. Besides, when the user sends more fake queries, GooPIR succeeds to mislead a higher percentage of queries (for 7 fake queries sent, 29.5% of queries are misclassified).

3.4.5 Impact of the Number of Fake Queries for TrackMeNot Over an Unlinkability Solution (E4-c)

In this section, we consider a hybrid solution formed by combining together TrackMeNot and an unlinkability solution. We show that the number of fake queries strongly impacts the efficiency of the current solution, as it necessarily increases the network traffic and thus increases the latency. In this section, we assess the protection offered by the hybrid solution according to the quantity of fake queries periodically created by TrackMeNot. Figure 5 shows the results of the attack for both version of SimAttack (i.e., using queries generated by the adversary or using the threshold δ equals to 0.8). We note that the quantity of fake queries does not impact the recall (i.e., the recalls are either 22.4% or 34.4% depending on the attack). However, the precision is strongly impacted (i.e., decreasing the number of fake queries increases the precision by 36.7% or 34.2%). Consequently, if the user wants to have a good protection (i.e., make the precision of the attack low), TrackMeNot has to issue a high number of fake queries. However, increasing the number of fake queries degrades the latency of all users in the system (as, by using an unlinkability solution, users have to share the same bandwidth). That is why users have to find and agree on a trade-off between level of protection and latency.

3.4.6 Impact of the number of fake queries GooPIR over an unlinkability solution (E5)

We consider a hybrid solution formed by combining together GooPIR and an unlinkability solution. We reproduce the same experiments as mentioned in section 3.4.5. However, because of the unlinkability solution, SimAttack needs to consider all user profiles to distinguish between fake queries and user's ones (as the adversary does not know the identity of the requester). Figure 18 shows that SimAttack is able to identify from 28.0% to 36.2% of initial queries depending on the number of fake queries the user used. In addition, if we compare this result to the one with GooPIR alone, the number of initial query identified decreases from 60.1% to 36.2% (when the user sends 7 fake queries). Consequently, adding the unlinkability solution helps to protect a high percentage of queries.

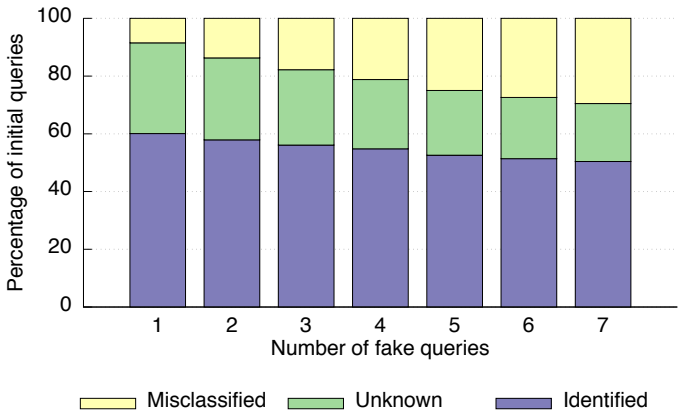

Figure 4: Results of SimAttack considering 100 users protected by GooPIR.

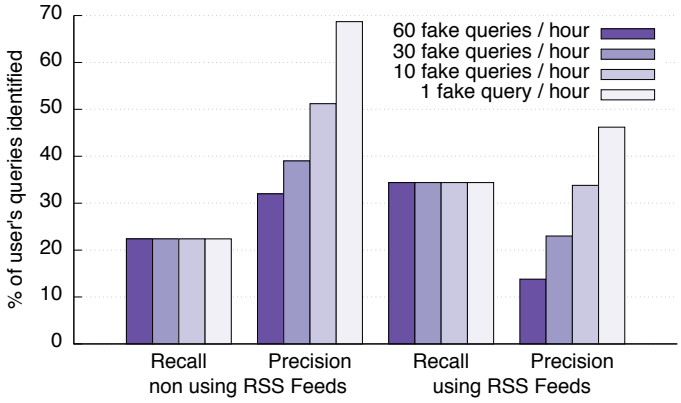

Figure 5: Results of SimAttack considering 100 users and protected by different numbers of fake query.

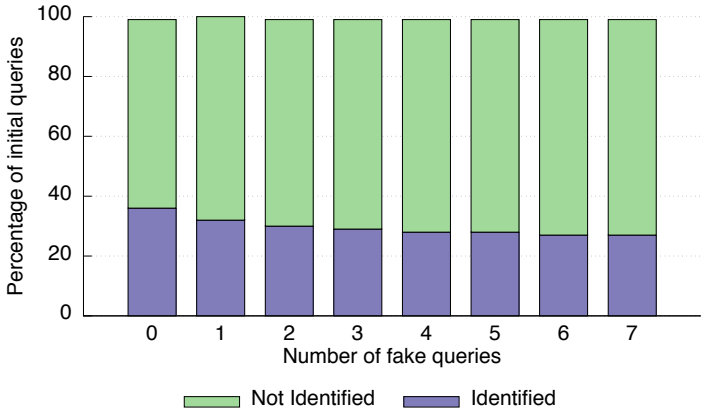

Figure 6: Percentage of queries protected by GooPIR over an unlinkability protocol and identified by SimAttack.

4 Private, Efficient and Accurate Web Search

This section introduces PEAS, a protocol ensuring Private Efficient and Accurate Search on the Web. It is designed in a generic way so it can be applied in multiple contexts. The next section describes how PEAS has been integrated in the EEXCESS framework.

PEAS aims at ensuring two properties: unlinkability between users and their queries, and indistinguishability between real and fake queries. Unlinkability techniques aim at protecting users' privacy by hiding their identity (e.g., their IP addresses). The most naive solution in this category consists in using a trusted proxy that forwards user queries to the search engine on behalf of the user [Shapiro, 1986]. Other techniques (e.g., [Castellà-Roca, 2009] and [Lindell, 2012], make users exchange their queries with each other: the queries issued by a user are forwarded to another user that eventually sends the query to the search engine. Consequently, this misleads the search engine, as it cannot infer the identity of the original requester.

To guarantee unlinkability, PEAS introduces an architectural solution: the integration of a privacy proxy in the system. To guarantee indistinguishability, PEAS offers a query obfuscation mechanism.

4.1 Privacy Proxy

The privacy proxy aims at ensuring unlinkability between the user and her queries. To achieve this goal and as we want an efficient protocol, the privacy proxy is split into two separated servers: one knows the identity of the users (e.g., their IP address), while the other one knows the content of their queries. They are respectively called the *requester* and the *issuer*. Figure 7 shows how the privacy proxy protocol achieves to hide information from one component or the other: when issuing a query q, the user creates a new symmetric cryptographic key K_q. Then, using the RSA encryption algorithm and the issuer's public key pk_1, she ciphers the aggregation of her query q and her key K_q. The cipher-text $E(q \bullet K_q)_{pk1}$ is then sent to the receiver (message 1). Since the receiver does not know the issuer's private key, it is not able to access the content of the initial query. As soon as the receiver receives the message, it assigns a unique identifier X_q to the query. Then, it forwards the message along with the identifier X_q to the issuer (message 2). The issuer receives the cipher-text from the receiver so it has no information about the user's identity. It uses its private key to retrieve the plaintext (concatenation of the user's query q and the user's symmetric key K_q), and then submits the initial query q to the search engine (message 3). Upon receiving the results (message 4), the issuer uses the user's symmetric key K_q to encrypt with the AES encryption algorithm the search engine's answer R and return the new cipher-text $\{R\}K_q$ along with the user's identifier X_q to the receiver (message 5). Finally, the receiver retrieves the requester identity using the identifier and sends the encrypted results $\{R\}K_q$ to the initial user (message 6). The latter uses the cryptographic key K_q to retrieve the results of her query.

For privacy reasons, the user needs to generate a new cryptographic key K_q every time she wants to issue a query. Otherwise, the issuer might use the key to link a set of consecutive queries issued by the same user and ultimately recover her identity as shown in [Barbaro, 2006].

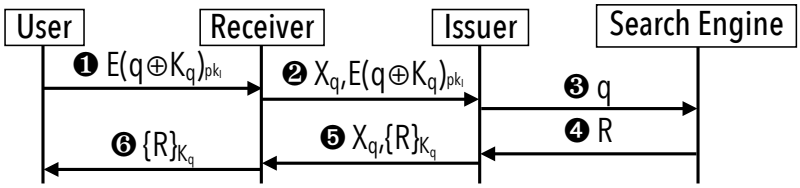

Figure 7: Privacy proxy protocol.

4.2 Obfuscation and Filtering Mechanism

An attack presented in [Peddinti, 2011] demonstrates that hiding users identity (i.e., using the privacy proxy alone) is not sufficient to protect their privacy. Therefore, it is necessary to make queries unlinkable to the user profiles. That is why we propose a new query obfuscation mechanism. Nevertheless, we advocate that as the user wants to retrieve the results that she would get without any protection mechanisms, the obfuscated query must contain the keywords of initial query. Consequently, our obfuscation mechanism aggregates the initial query with *k* fake queries separated with logical OR operators. This allows, in a post-processing phase, to retrieve results of the initial query by filtering out the irrelevant results introduced by the fake queries.

4.2.1 Fake Queries Requirements

A key challenge is to generate fake queries that are not distinguishable from the initial query. They must satisfy the following requirements:

1. Fake queries shall have similar structural properties as the initial one (e.g., number of keywords and usage frequency).
2. Fake queries shall be built from past queries. To address this key requirement, we propose to generate queries based on an aggregated history of past requests gathered from users of our system. The privacy proxy publishes this information, called the group profile. However, publishing users' past queries in a group profile may lead to information leakage. Consequently, it has to be done in privacy-preserving way.
3. Fake queries shall point at different sets of users; otherwise, the adversary can easily filter them out. That is why the group profile contains a set of aggregated histories built from disjoint users. Each aggregated history is used to generate one fake query at a time.

Given these requirements, the obfuscation mechanism consists in generating k queries that (i) have the same number of keywords than the initial queries, (ii) have a similar usage frequency that the initial queries, and (iii) have been issued by k other users. Besides, to mislead the adversary about the requester identity, these fake queries must not contain keywords that have already been used by the requester. In other words, the generated fake queries must be far from the user profile and close to other users.

4.2.2 Group Profile Exploitation

In order to generate realistic fake queries, we propose to exploit two types of information: the past queries of the user, and the past queries of other users. We consider that each user locally stores a history of her queries. Instead, as it has no direct access to the past queries of other users, the user relies on a group profile periodically published by the privacy proxy. This group profile is structured in multiple clusters of disjoint users. The past queries of a given user are aggregated in one cluster. Each cluster is associated with a co-occurrence matrix *M* such that the matrix's axes represent the vocabulary (i.e., keywords already used in past queries). Each element *M(a,b)* of the matrix represents the frequency of occurrence of the keywords a and b. The issuer performs the construction of the co-occurrence matrix for each cluster, as it is the only entity that has access to the content of the queries. Table 2 gives an example of such a co-occurrence matrix. The elements corresponding to the pair ("HIV", "Treatment") contain 1 because one query containing "HIV" and "Treatment" has been issued in the past.

	HIV	**Treatment**	**Depression**	**Test**	**Symptom**
HIV	0	1	0	10	11
Treatment	1	0	5	0	4
Depression	0	5	0	17	13
Test	10	0	17	0	0
Symptom	11	4	13	0	0

Table 2: Example of a co-occurrence matrix included in a group profile.

By construction of the co-occurrence matrix (i.e., aggregation of bi-occurrence frequencies), users have only access to an approximation of past queries. Thus, users' privacy is preserved. To generate fake queries using this matrix, users transpose a co-occurrence matrix to a co-occurrence graph, and compute all maximal complete graphs (also called *maximal cliques*). The completeness property is a necessary condition for already formulated queries, i.e., an already formulated query is necessarily a complete graph of keywords. Furthermore, the maximality property aims at maximizing the probability of selecting complete queries and not a subset of past queries. The computation of all the maximal cliques of a graph is an NP-complete problem, which we overcome using the Bron-Kerbosch algorithm [Bron, 1973]. Besides, this computation does not impact the latency as it is done once offline.

4.2.3 Fake Queries Generation

The generation of fake queries consists in selecting *n* words in the co-occurrence graph. This sub-graph of size n words is (i) a maximal clique, (ii) a non-maximal clique, or (iii) a non-complete graph. The choice is made randomly to mislead potential adversaries. Indeed, an adversary is not able to use the frequency or the number of keywords to determine if a query was issued by a user, or generated by the obfuscation mechanism.

4.2.4 Privacy-Preserving Group Profile Publication

To obfuscate their queries, users need to have access to the group profile. As mentioned previously, the issuer computes and releases this group profile periodically. The group profile is composed of *k+1* co-occurrence matrices (i.e., clusters) in which the issuer aggregates all queries. Nevertheless, the issuer has no information about which queries (among the obfuscated query) belongs to which cluster. To overcome this issue in a privacy-preserving way, the user gives this information by aggregating her query and the fake queries in the same order as the clusters (i.e., the first query belongs to the cluster 1, the second query belongs to the cluster 2, and so on). Besides, to generate the *k* fake queries and to aggregate queries in the correct order, the user needs to know which *k* clusters (among the *k+1*) she needs to use and which cluster is associated to her real queries. This is done by sending a special query to the receiver. When the receiver receives this special query, it checks if the user is already assigned to a cluster; otherwise, it randomly assigns the user to one of the *k+1* clusters. Finally, it answers the user with her cluster number.

4.2.5 Filtering Irrelevant Results

The filtering aims to remove the irrelevant results introduced by the fake queries. Indeed, search engines answer with a mix of results corresponding to *k+1* queries (*k* fake queries and the initial one). As we want the user to retrieve results that she would get without any protections (i.e., corresponding to the initial query), it is needed to filter out all the results. This filtering is done on the client side, as the user is the only one knowing which one is the initial query. For each result *r* from the result set, the mechanism determines if it has been retrieved because it relates to the initial query or to a fake query. A similarity score is assigned to each query. It considers meta-data (typically the title, description and URL). A result r is considered to relate to the initial query if and only if the initial query has the largest score.

4.3 Evaluation of PEAS

In this section, we present the evaluation of PEAS regarding the privacy, the performance and accuracy.

4.3.1 Privacy

The objective of this evaluation is to compare PEAS with GooPIR (an existing state-of-the-art solution) over an unlinkability solution. The de-anonymisation of queries is performed using two attacks: (i) an existing machine learning attack (ML attack) [Peddinti, 2011] and, (ii) SimAttack (presented in section 3). We adapted a machine learning attack to de-anonymize obfuscated queries. We first build a model using Support Vector Machine algorithm on the two thirds of the user queries. Then, using this model we are able to predict the user who issued a query. In our experiments, queries were obfuscated using *k* fake queries. Consequently, we evaluate

each of the *k+1* queries and we consider the one with the higher probability as the real one and its predicted user as the real user.

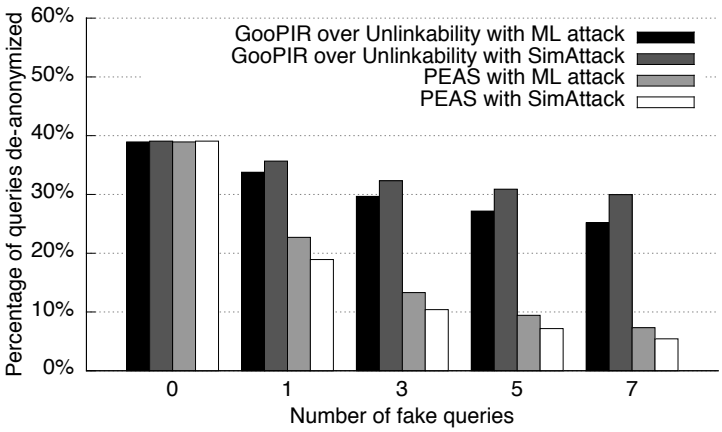

Figure 8: Comparison between PEAS obfuscation and GooPIR obfuscation over unlinkability depending on the number of fake queries.

Figure 8 shows the percentage of de-anonymized queries for both PEAS and GooPIR using the two attacks. The left bars of this figure (i.e., the bars at 0 on the x-axis), show results without any obfuscation mechanism; only unlinkability is used to protect user queries. Compared to this baseline, adding one fake query using PEAS decreases by half the number of linked queries (from 39.0% to 20.8%), while adding seven fake queries makes this number decrease by more than six times for both attacks (from 39.0% to 6.4%). From this experiment, we also note that PEAS clearly outperforms GooPIR over unlinkability in all configurations. Indeed, PEAS protects at least 13.9% and 20.9% more queries than GooPIR for the configuration with 1 and 7 fake queries respectively.

The results obtained by PEAS can be explained by the fact that the more fake queries we create, the higher the probability that one of the fake queries gets closer to a given user profile, than the original query is to its requester's profile. Note that, the obfuscation mechanism of PEAS does not protect all the queries as some of them are still linked to their originating user. We mitigate this result by showing in the following section that the confidence obtained by the attacker about these queries is actually very low.

4.3.2 Accuracy

In this section we evaluate the accuracy of PEAS by showing the impact of the obfuscation and filtering techniques on the information retrieval results. Practically, we study if the filtering mechanism is able to remove irrelevant results (i.e., those relating to fake queries) while keeping the relevant ones (i.e., those relating to the initial query).

To measure the accuracy, we consider two metrics. The first one, named *QM1*, measures the proportion of results effectively retrieved, while the second one, named *QM2*, takes into account the ordering of the results retrieved. They are defined as follows:

$$QM1 = \frac{card(R \cap R')}{card(R)})\times 100$$

$$QM2 = \frac{\sum_{r \in R \cap R'} |rank(R,r) - rank(R',r)|}{card(R \cap R')}$$

where *R* is the list of expected results, *R'* is the list returned by PEAS, and *rank(L, e)* represents the position of the element *e* in the list *L*. *QM1* equals 100 when *R* and *R'* contain the same elements. *QM2* compares the rankings of the expected list of results *R* with another list of results *R'*. It equals 0 if and only if the elements of $(R \cap R')$ appear at the same position in *R* and *R'*.

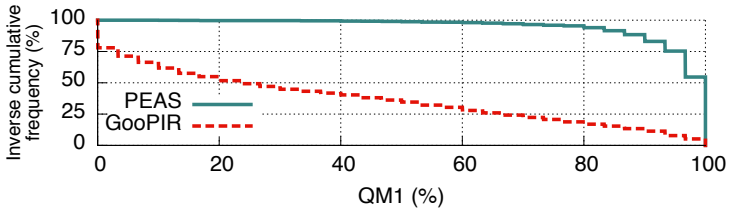

Figure 9: Percentage of expected results retrieved after filtering.

Figure 9 shows the inverse cumulative frequency distribution of the metric QM1. Results show that in more than 50% of cases, PEAS returns all the expected results (i.e., the results returned when no obfuscation technique is applied). In 95% of cases, PEAS returns more than 80% of the expected results, that is 24 correct results out of 30.

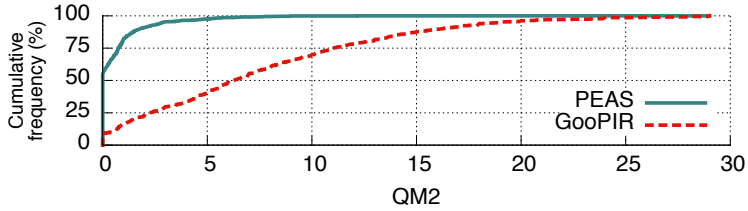

Figure 10: Comparison between the rankings of the filtered results and the expected ones.

Figure 10 shows the cumulative frequency distribution of the metric QM2. This figure shows that the rankings of the results returned are identical to the expected results in more than 50% of cases. Moreover, in 90% of cases, we find that $QM2 \leq 2$, which means that the rank of de-obfuscated results differ on average with the expected results by 2 units (that is, 2 positions in the ranked list). From these two experiments, we observe that PEAS filtering retrieves significantly more results than GooPIR (on average 95.3% for PEAS and 36.3% for GooPIR). Besides, the results returned by PEAS are more accurate than GooPIR as the ranking of results (compared to the expected results) differ on average by 0.6 unit for PEAS compared to 5.8 units for GooPIR.

The results presented on Figure 9 and Figure 10 demonstrate that PEAS preserves the accuracy of the results as both the retrieved results and their ordering are marginally impacted by the obfuscation technique.

4.3.3 Performance

We evaluated the performance of the privacy proxy protocol and compared it with the Onion Routing protocol [Goldschlag, 1999], as it is to the best of our knowledge the most efficient anonymous communication protocol. In order to have a fair comparison of both protocols, we implemented and configured the Onion Routing as follow: (i) we set up only two relays (as our privacy proxy is composed of two servers and using more relays would produce a higher overhead for Onion Routing), and (ii) we configure the protocol to re-negotiate the symmetric keys for each query (as it prevents the exit node from using the key to link queries coming from the same user). Finally, we assume that clients and servers of both protocols know all the necessary public keys in advance.

Theoretical evaluation: We distinguish three metrics: the number of symmetric encryption and decryption operations performed, the number of asymmetric encryption and decryption operations performed, and the traffic generated. We compute these metrics for the whole system. Table 3 shows the values of these metrics when using PEAS, Onion Routing and when directly accessing the search engine. These results demonstrate that PEAS requires less cryptographic operations than Onion Routing (twice as less RSA encryption/decryption and six times less AES encryption/decryption operations). In terms of network traffic, PEAS sends 38% less messages than Onion Routing. Indeed, querying a website using PEAS requires to send 15 messages (9 for the

TCP handshake and 6 for the protocol messages), while using Onion Routing requires to send 39 messages (9 for the TCP handshake, 18 for the TLS handshake and 12 for the protocol messages).

	Encryption		Decryption		Traffic
	RSA	AES	RSA	AES	
Direct access	0	0	0	0	5
PEAS	1	1	1	1	15
Onion Routing	2	6	2	6	39

Table 3: Number of cryptographic operations and traffic generated in the whole system.

Round trip time: We measured the time delay on the client side, from the submission of a query to the deciphering of the received response. To avoid uncertainty in the Internet network latency and in the search engine query processing, we simulate a search engine response on the last node (i.e., issuer or exit node). We consider one client using the architecture. We execute the PEAS implementation and the Onion Routing implementation on three identical servers (one as a client and two as a server). The average PEAS round trip time (measured on 50,000 queries) is 5.97 times lower than the Onion Routing protocol (2.761ms for PEAS vs 16.475ms for Onion Routing). This difference is mainly due to the number of cryptographic operations (as shown in Table 3) and to the re-negotiation of the symmetric keys for each query performed by the Onion Routing protocol.

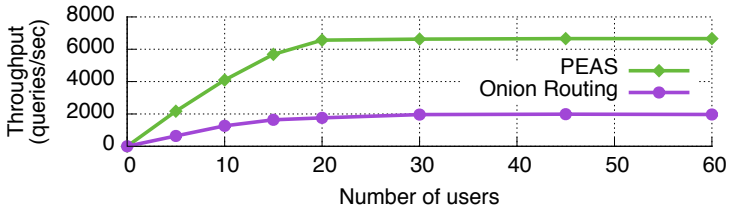

Figure 11: Comparison between the scalability of PEAS and Onion Routing.

Scalability: We evaluated the scalability of our architecture compared to Onion Routing. Figure 11 shows the throughput as a function of the number of clients using the two architectures. This figure shows that both architectures handle approximately the same number of users (i.e., 20) before saturation of the throughput. Nevertheless, PEAS architecture has a throughput three times higher than Onion Routing (i.e., 6,695 req/s for PEAS and 2,034 for Onion Routing).

5 Implementation of the Privacy Proxy

5.1 Architecture of the System

Before presenting the services offered by the privacy proxy, let us present the architecture of the system. Figure 12 depicts the different components of the system: the client application (e.g., the Google Chrome extension), the privacy proxy (composed of a requester and an issuer) and the federated recommender. The requester and the issuer are similar to those considered in the PEAS protocol (see section 4.1 and Figure 7). In the current implementation, the requester is not implemented. Thus, messages are directly sent from the client application to the issuer.

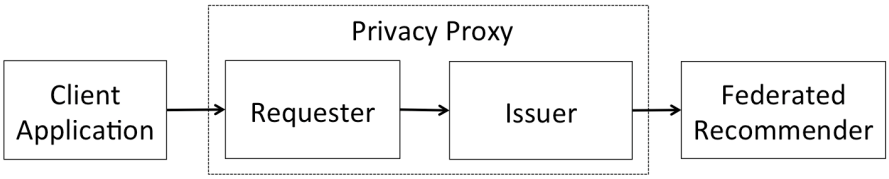

Figure 12: Simplified architecture of the EEXCESS framework.

5.2 Services Available on the Privacy Proxy

The main role of the privacy proxy is to act as an intermediary between client applications and the federated recommender (and other potential components). The services available on the privacy proxy are:

- **Recommend:** Forwards the request to the federated recommender or processes it according to the PEAS protocol (depending on the input format). The PEAS protocol is described in more details in section 5.2.1.
- **Get details:** Forwards the query to the Federated Recommender and returns the detailed results (see deliverable D3.3).
- **Get registered partners:** Forwards the request to the Federated Recommender and returns the list of partners currently registered with some statistics (see deliverable D3.3).
- **Get partner favicon:** Returns the favicon of a given partner (see deliverable D3.3).
- **Get preview image:** Returns a default image if the media is not available (see deliverable D3.3).
- **Suggests categories:** Forwards the request to the DoSER component (see D5.3, section 4.2).
- **Recognize Entity:** Forwards the request to the DoSER component (see D5.3, section 4.2).
- **Get co-occurrence graph:** Returns the co-occurrence graph of query terms. It must be used wisely, as the result is potentially voluminous (caching the result is recommended). It is described in more details in section 5.2.1.
- **Get maximal cliques:** Returns the set of maximal cliques extracted from the co-occurrence graph. Similarly to the previous service, it should be used wisely. It is described in more details in section 5.2.1.
- **Log interaction:** Logs a given interaction on the proxy. The logging service is described in more details in section 5.2.2.

5.2.1 PEAS Services

The PEAS protocol presented in the section 4 requires the deployment of software components on the client-side and on the server-side. Figure 12 presents the different components of the architecture: the client, the request, the issuer and the federated recommender. The privacy proxy is composed of both the requester and the issuer.

In the current implementation, only the indistinguishability protocol of PEAS has been deployed. The unlinkability protocol will be developed and deployed in the next phase of the project. As the requester is only used by the unlinkability protocol, it is not considered at this stage.

The source code and the documentation are accessible from http://purl.org/eexcess/components/privacy-proxy.

Recommendation: This service is used to return recommendations to users. It implements both the regular processing and the PEAS unlinkability protocol. The way to process a query depends on the query format itself: if the query is obfuscated (i.e., if it looks like *q* = (term1 term2) OR (term3 term4) OR (term5 term6)), the PEAS protocol is used; otherwise (i.e., if the query looks like *q* = term1 term2) the query is simply forwarded to the federated recommender. When an obfuscated query comes in, the following steps are executed:

1. The query is split in *k+1* sub-queries, where *k* is the number of fake queries. The input looks like *q* = (term1 term2) OR (term3 term4) OR (term5 term6)), and the output looks like {*q1* = (term1 term2), *q2* = (term3 term4), *q3* = (term5 term6)}. Each sub-query is a regular query.
2. Each sub-query is forwarded to the federated recommender.
3. All the result sets (one for each sub-query) is gathered and aggregated. The input looks like {*res1* = [r1, r2, r3], *res2* = [r4, r5, r6], *res3* = [r7, r8, r9]}, where *res1*, …, *res3* are result sets, and *r1*, …, *r9* are recommendations. The output looks like *res* = {[*r1*, *r2*, *r3*], [*r4*, *r5*, *r6*], [*r7*, *r8*, *r9*]}.
4. The result set res is sent to the client application that issued the original query.

Steps 1 and 4 are presented on Figure 13: they respectively correspond to the blocks *Split query* and *Merge results* in the issuer component.

The formats used to represent queries and the associated results are described here: http://purl.org/eexcess/documentation/request-and-response-format. The regular queries and the associated results are described in more details in section 5 of D3.3. The obfuscated query format is similar to the one of regular query, except for the *contextKeywords* attribute. Table 4 present the difference between the two formats. The associate result set format to obfuscated (also called *aggregated result set*), is presented in Table 5. As it can be seen, the *results* attribute contains several regular result sets.

Regardless of the nature of the query (regular or obfuscated), the list of keywords is stored in a query log. Each entry of the query log contains only the timestamp and the list of keywords. No information regarding the user is stored (identifier, origin, etc.). Therefore, the privacy of users is preserved. This query log is used in the generation of fake queries. It is discussed in the following paragraphs. In order not to overload the disk, and in order to be able to follow query trends, the query log only contains entries that have been issued recently. The time window is a parameter that can easily be adapted.

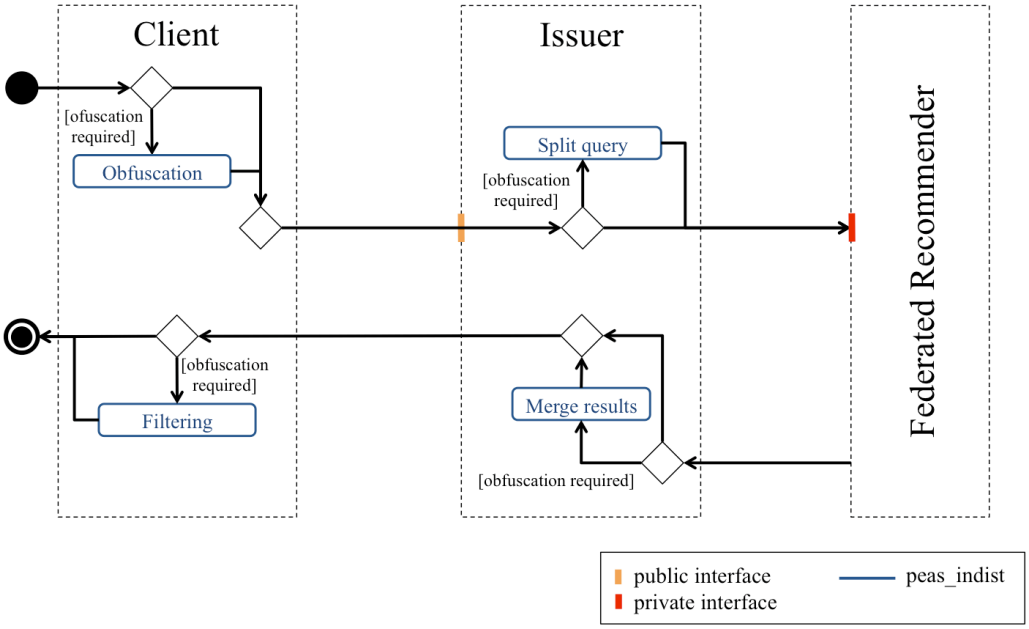

Figure 13: Activity diagram presenting the processing of a query (regular or obfuscated).

Attribute	Regular query	Obfuscated query

contextKeywords	<pre>[
	  {
	  "text":"women",
	  "type":"misc"
	  },{
	  "text":"Ada",
	  "type":"person"
	  }
	]</pre>
		<pre>[
		  [
		  {"text":"women", "type":"misc"},
		  {"text":"Ada", "type":"person"}
		  ],[
		  {"text":"art", "type":"misc"},
		  {"text":"graz", "type":"place"}
		  ],[
		  {"text":"cup", "type":"misc"},
		  {"text":"world", "type":"misc"}
		  ]
		]</pre>

Table 4: Comparison of regular and obfuscated query formats.

Regular result set	Aggregated result set

{
  …,
  "result":[…],
  …
}
	{
	  "results": [{
	  …,
	  "result":[…],
	  …
	  }, {
	  …,
	  "result":[…],
	  …
	  }, {
	  …,
	  "result":[…],
	  …
	  }]
	}

Table 5: Comparison of regular and aggregated result set formats.

Get co-occurrence graph: The indistinguishability protocol of PEAS uses a co-occurrence graph to generate fake queries on the client-side. As it would be too resource consuming to compute the graph on the client-side, we implemented a service on the privacy proxy. This service is then simply invoked from client applications. An example of the output of this service is given below:

```
[{
    "term": "car",
    "frequencies": [
        {"term": "vehicle", "frequency": 2},
        {"term": "blue", "frequency": 5}
    ]
},{
    "term": "vehicle",
    "frequencies": [
        {"term": "blue", "frequency": 8}
    ]
}]
```

Computing such a graph is costly (even on the server), as it requires reading the whole query log. The resources needed to compute the graph increases with the number of calls. Therefore, we implemented a caching mechanism. The co-occurrence graph is computed regularly (e.g., once a day) and the result is stored on disk. Thus, when the service is invoked, the graph is not computed but simply extracted from a file. It saves computation resources and reduces the response time of each call.

Get maximal cliques. The purpose of this service is similar to the one to get the co-occurrence graph. An example of the output is given below:

```
[
    [{
        "term": "car",
        "frequencies": [ ... ]
    },{
        "term": "vehicule",
        "frequencies": [ ... ]
    }]
]
```

The cache is also cached on the server.

5.2.2 Logging Service

A service to log interaction of users has been implemented on the privacy proxy. It stores information on users' activity on a given client application. This information is used exclusively to study users' experience in order to improve the whole system (or a specific application) or to introduce new features.

We distinguish two types of interactions:

- Implicit interactions: They can be logged automatically on the privacy proxy, as they relate to actions that call services hosted on the privacy proxy. From an application developer, not additional work is required to log these interactions.
- Explicit interactions: They relate to actions that are done on the client application. Therefore, in order to collect them, it is necessary to invoke a dedicated service. Application developers have to explicitly invoke the logging service hosted on the privacy proxy to collect a given interaction. To ease this task, an API has been developed on the client-side. This API is part of the C4 component and is described in more details in sections 9 and 10 of D5.3.

In total, 14 types of interaction are logged. They are presented in Table 6. The documentation is available at https://github.com/EEXCESS/eexcess/wiki/EEXCESS---Logging.

For each implicit interaction, the input data used when invoking the logging service must contains two mandatory attributes (an optional attribute can be added):

- origin: This attribute is used to identify the provenance of the interaction. It is composed of four attributes: clientType, clientVersion, module and userID. An example of this attribute is given below:

```
"origin": {
      "clientType": "EEXCESS - Google Chrome Extension",
      "clientVersion": "2.0",
      "module": "Facetscape",
      "userID": "E993A29B-A063-426D-896E-131F85193EB7"
}
```

- content: This attribute is used to encapsulate all the information related to a given interaction. Its format depends on the interaction itself (it described in the documentation).
- queryID (optional): This attribute represent the identifier of the user and is a string, e.g.,
  `"queryID": "A33B29B-BC67-426B-786D-322F85182DA6"`

When the service is called, it creates an entry in the current log. In order to ease the analysis of the logs, one log (i.e., one file) is created every day. A log entry looks like this:

```
{
    "interactionType": "...",
    "timestamp": “1234567890",
    "IP": "132.231.111.197",
    "origin": {
        "userID": "E993A29B-A063-426D-896E-131F85193EB7",
        "clientType": "EEXCESS - Google Chrome Extension",
        "clientVersion": "2beta",
        "module": "facetScape"
    },
    "content": { … },
    "queryID": "A33B29B-BC67-426B-786D-322F85182DA6"
}
```

Again, the content attribute format depends on the type of interaction.

Type	Interaction	Description
Implicit	query	When a query or a details query is received on the privacy proxy.
	detailsQuery	
	response	When a result set or a details response is sent back to the client application.
	detailsResponse	
Explicit	moduleOpened	When a module (e.g., the facetScape) is opened or closed by a user.
	moduleClosed	
	moduleStatisticsCollected	When statistics regarding a module have to be saved. Client application developers determine how to collect statistics and when they need to be stored.
	itemOpened	When an item (i.e., a recommendation) is opened or closed.
	itemClosed	
	itemCitedAsText	When an item (i.e., a recommendation) is cited in a document (e.g., a Wordpress blog post, a Moodle page) by a user.
	itemCitedAsImage	
	itemCitedAsHyperlink	
	itemRated	When an item (i.e., a recommendation) is rated by a user.
	itemBookmarked	When an item (i.e., a recommendation) is bookmarked by a user.

Table 6: Interactions logged on the privacy proxy.

6 Implementation on the Client-Side

6.1 PEAS Components

A Javascript component, called *peas_indist*, has been implemented to allow client applications developers to integrate the indistinguishability protocol of PEAS in their own applications. The code and the documentation are available on GitHub: https://github.com/EEXCESS/peas. This component offers two features:

- Query obfuscation: It takes as input a regular query and returns an obfuscated query. This task is represented by the *Obfuscation* block presented on Figure 13 (page 19).
- Result set filtering: It takes as input a set of regular result sets and returns an aggregated result set. This task is represented by the *Filtering* block presented on Figure 13 (page 19).

An example of how to use the obfuscation mechanism is given below:

```
require(["peas_indist"], function(peas_indist){
    var originalQuery = JSON.parse('{"numResults":3,"contextKeywords=[…]}');
    var k = 3; // Number of fake queries
    var obfuscatedQuery = peas_indist.obfuscateQuery(originalQuery, k);
});
```

The example below shows how to use the filtering mechanism:

```
require(["peas_indist"], function(peas_indist){
    var originalQuery = JSON.parse('{"contextKeywords":[…], …}');
    var obfuscatedQuery = JSON.parse('{"contextKeywords":[[…],[…],[…]], …}');
    var results = JSON.parse('{"results": […], …}');
    var filteredResults = peas_indist.filterResults(results, originalQuery);
});
```

The Javascript PEAS component has been integrated in the C4 component to ease its use by application developers (see deliverable D5.3).

The formats handled by this component are those presented in section 5.2.1. As mentioned in the previous section, the obfuscation mechanism uses a co-occurrence graph. Therefore, the Javascript component retrieves it from a service hosted on the privacy proxy (as described in section 5.2.1). Before the graph can be big, it is not optimal to retrieve it from the proxy every time it is needed. As a consequence, we implemented a caching mechanism on the client-side. The service is invoked only if the graph is not recent enough. A set of maximal cliques is also used in the obfuscation mechanism. Similarly to the co-occurrence graph, the result is cached locally to avoid unnecessary network traffic, and reduce the response time.

Another component, called *peas_adapt*, has been developed to adapt users' queries according to their privacy settings. Basically, this component is in charge of removing attributes that a given user does not agree to share. In other words, it filters the information sent to the federated recommender (through the privacy proxy). And example of how to use it is given below:

```
require(["peas_adapt"], function(peas_adapt){
    var originalQuery = JSON.parse('{…}');
    var policies = '[{"attribute": "ageRangePolicy", "level": 0},
      {"attribute": "genderPolicy", "level": 0},
      {"attribute": "locationPolicy", "level": 2},
      {"attribute": "interestPolicy0", "level": 1},
      {"attribute": "interestPolicy1", "level": 1},
      {"attribute": "languagePolicy0", "level": 1},
      {"attribute": "languagePolicy1", "level": 1}]';
    var adaptedQuery = peas_adapt.adaptQuery(originalQuery, policies);
});
```

The code and documentation are available on GitHub: https://github.com/EEXCESS/peas.

6.2 User Profile Interface

We refined the graphical user interface (already presented in D6.2) that allows a user to specify personal information and privacy settings. Figure 14 presents the current interface. The interface is integrated in the Google Chrome extension. In this version, the attributes of the user profile (user name, location, language skills, interests) and the privacy setting features are integrated in one interface. Compared to the previous version, multiple attributes (e.g., gender, first name and last name) have been discarded, as the federated recommender does not exploit them (and it was not planned to use them until before the end of the project). However, some attributes have been added (e.g., language skills) or modified (e.g., birth date was replaced by age range). The user profile is divided into four sections: General Privacy Settings, General Information, Languages and Interests. All the attributes are optional and can be left blank. The General Privacy Settings section contains only 2 parameters. The first one allows users to determine if they accept to have their interactions with the system logged (see section 5.2.2). The second parameter allows them to set a preferred level of protection for their queries. This level is used to determine to which extend queries must be protected with the PEAS solution (see sections 4, 5.2.1 and 6.1): *Low* (no obfuscation), *Medium* (queries are obfuscated with 2 fake queries) and *High* (queries are obfuscated with 4 fake queries). The General Information section contains three attributes: user name (only used to confirm that the user is logged on her own profile), location (country and city) and age range (child, young adult or adult). These last two attributes are used to personalise recommendations. The location can be used to provide recommendations close to the user. The age range can be used to recommend specific types of objects. For instance, a child will mainly be interested in pictures, videos or possibly short texts. The Languages section is used to capture the users' skills. Users can consider several languages and levels of proficiency. Depending on these skills, recommendations can be refined. Finally, the Interests section is used to capture users interests. Several sets of interests can be defined in order to have multiple profiles. For instance, a teach might want to have a professional profile (with teaching-related topics) and a personal profile (with hobbies-related topics). All the attributes can easily be hidden to the system (i.e., the privacy proxy and the federated recommender) or disclosed.

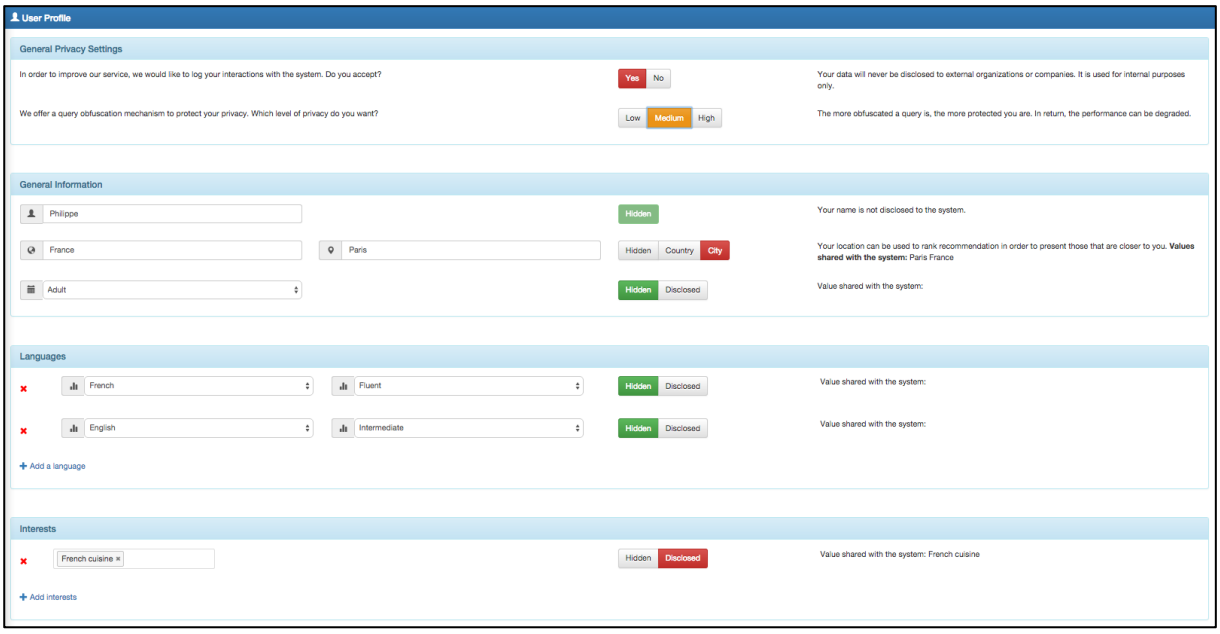

Figure 14: Graphical user interface of user profile.

7 Conclusions

This deliverable presented the main results achieved during the last 12 month of the project. It focused on four aspects. First, it presented SimAttack, a new attack against private web search solutions. SimAttack allowed us to show that existing solutions, even when they are combined, cannot guarantee users' privacy. Second, it presented PEAS, a new solution to protect users' privacy. Experimental results showed that it outperforms existing solutions. Third, this deliverable described the features that were implemented on the privacy proxy. Services and exchange formats are detailed. Finally, it described the client-side components implemented to ease the integration of PEAS on client applications.

As mentioned in the Executive Summary (more precisely in section 1.3), the next steps for the project are:

- Implementation of the unlinkability protocol of PEAS. It requires the deployment of a new version of the privacy proxy. Until now, the proxy is composed of only one entity (corresponding to the issuer). In the next version, the proxy will be composed of two entities: a requester and an issuer.
- Adaptation of the implementation of the PEAS protocol to cope with the service providing details of a given recommendation.
- Implement the mechanism ensuring users' privacy regarding the logs that are stored on the privacy proxy. This work will be joined with the work package 5. A discussion on these aspects is presented in section 9.3 of deliverable D5.3).

8 References

[Barbaro, 2006] M. Barbaro, T. Zeller, and S. Hansell. A face is exposed for AOL searcher no. 4417749. New York Times, 9(2008):8, 2006.

[Bron, 1973] C. Bron and J. Kerbosch. Algorithm 457: finding all cliques of an undirected graph. Communications of the ACM, 16(9):575–577, 1973.

[Castellà-Roca, 2009] J. Castellà-Roca, A. Viejo, and J. Herrera-Joancomartí. Preserving user's privacy in web search engines. Computer Communications, 32(13), 2009.

[Dingledine, 2004] R. Dingledine, N. Mathewson, and P. Syverson. TOR: The second generation onion router. In *Usenix Security Symposium*, 2004.

[Domingo-Ferrer, 2009] J. Domingo-Ferrer, A. Solanas, and J. Castellà-Roca. h(k)-private information retrieval from privacy-uncooperative queryable databases. *Online Information Review*, 33(4):720–744, 2009.

[Goldschlag, 1999] D. Goldschlag, M. Reed, P. Syverson: Onion Routing. Communications of the ACM 42(2) (1999)

[Hall, 2009] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, I.H. Witten: The Weka data mining software: an update. ACM SIGKDD explorations newsletter 11(1), 10–18 (2009)

[Hearst, 1998] M.A. Hearst, S.T. Dumais, E. Osman, J. Platt, B. Scholkopf: Support vector machines. Intelligent Systems and their Applications, IEEE 13(4), 18–28 (1998)

[Lindell 2010] Y. Lindell and E. Waisbard. Private web search with malicious adversaries. In Proceedings of PETS, pages 220–235. Springer, 2010.

[Peddinti, 2011] S. T. Peddinti and N. Saxena. On the effectiveness of anonymizing networks for web search privacy. In Proceedings of the 6th ACM ASIACCS, pages 483–489. ACM, 2011.

[Petit, 2015a] A. Petit, T. Cerqueus, S. Ben Mokhtar, L. Brunie and H. Kosch. PEAS: Private, Efficient and Accurate Web Search. In 14th IEEE International Conference on Trust, Security and Privacy in Computing and Communications (IEEE TrustCom 15).

[Petit, 2015b] A. Petit, T. Cerqueus, S. Ben Mokhtar, L. Brunie and H. Kosch. How to Protect Your Privacy Using Search Engines?. In 10th European Conference on Computer Systems (EuroSys 2015), poster session.

[Shapiro, 1986] M. Shapiro. Structure and Encapsulation in Distributed Systems: the Proxy Principle. In Proceedings of ICDCS, 1986.

[Toubiana, 2011] V. Toubiana, L. Subramanian, and H. Nissenbaum. Trackmenot: Enhancing the privacy of web search. arXiv preprint arXiv:1109.4677, 2011.

9 Glossary

Terms used within the EEXCESS project.

Partner Acronyms

JR-DIG	JOANNEUM RESEARCH Forschungsgesellschaft mbH, AT
Uni Passau	University of Passau, GE
Know	Know-Center - Kompetenzzentrum für Wissenschaftsbasierte Anwendungen und Systeme Forschungs- und Entwicklungs Center GmbH, AT
INSA	Institut National des Sciences Appliquées (INSA) de Lyon, FR
ZBW	German National Library of Economics, GE
BITM	BitMedia, AT
KBL-AMBL	Kanton Basel Land, CH
CT	Collection Trust, UK
MEN	Mendeley Ltd., UK
WM	wissenmedia, GE

Abbreviations

EC	European Commission
EEXCESS	Enhancing Europe's eXchange in Cultural Educational and Scientific resource

Acronyms

API	Application Programming Interface
GUI	Graphical User Interface
JSON	JavaScript Object Notation
REST	Representational State Transfer
WSDL	Web Services Description Language
SOAP	Simple Object Access protocol

Acknowledgement: The research leading to these results has received funding from the European Union's Seventh Framework Programme (FP7/2007-2013) under grant agreement n° 600601.