Doctoral Thesis
in partial fulfillment of obtaining the degree of Doctor in Computer Science

This thesis was prepared in collaboration between the Laboratoire d'Informatique en Image et Systèmes d'Information (LIRIS) at INSA Lyon *and the chair of Medieninformatik at* Universität Passau.

INTRODUCING PRIVACY IN CURRENT WEB SEARCH ENGINES

Albin PETIT

submitted in cotutelle-procedure to

Institut national des sciences appliquées de Lyon
Ecole doctorale d'Informatique et de Mathématiques (InfoMaths)
&
Universität Passau
Fakultät für Informatik und Mathematik

defended on 15/03/2017 in front of a jury composed of

Prof. Claude Castelluccia
INRIA Grenoble, France
Reviewer

Prof. Rüdiger Kapitza
TU Braunschweig, Germany
Reviewer

Prof. Harald Kosch
Universität Passau, Germany
Examiner

Prof. Catherine Berrut
Université Grenoble Alpes, France
Examiner

Dr. Mathias Lux
Universität Klagenfurt, Austria
Examiner

Prof. Lionel Brunie
INSA Lyon, France
Adviser

Prof. Michael Granitzer
Universität Passau, Germany
Adviser

Dr. Sonia Ben Mokhtar
INSA Lyon, France
Co-Adviser

Publication No. 2017LYSEI016 | Year 2017

"Privacy isn't about hiding something. It's about being able to control how we present ourselves to the world."

— Bruce Schneier,
Cryptography and Security Specialist

"You have to fight for your privacy or you lose it."

— Eric Schmidt,
Executive Chairman of Google Inc.

Acknowledgements

I would like to express my deep and sincere gratitude to my advisors Prof. Lionel Brunie, and Prof. Michael Granitzer, who introduced me to the fascinating world of scientific research. Their scientific expertise as well as their valuable remarks have contributed to the success of my PhD thesis. I could not have imagined having better advisors for my PhD study.

I am also very grateful to Sonia Ben Mokhtar for her patience, guidance and encouragement over the last years. It would never have been possible for me to accomplish this thesis without her incredible support. I am really thankful to her for all the invested time she gave to the supervision of my thesis especially to our numerous scientific discussions.

Besides my advisors, I would like to warmly thank the members of my PhD defense committee: Prof. Claude Castelluccia, Prof. Rüdiger Kaptiza, Prof. Catherine Berrut, and Dr. Mathias Lux for their interest in my work, their efforts in reviewing the manuscript, and their insightful comments.

I would like to thank Prof. Harald Kosch for his administrative support and his guidance during the last three years. This cotutelle would never have been possible without his involvement. I am also grateful to Ingrid Winter and Annemarie Gertler-Weber for all the support and the assistance they gave me during my time in Passau.

The MDPS doctoral college was an incredible place for scientific discussions and valuable feedback. Many thanks to all MDPS members especially to Nadia Bennani, David Coquil, and Morwenna Joubin who have been coordinated it. I would also like to thank the European Union for funding this thesis through the European Project EEXCESS (Seventh Framework Program), the French-German University (DFH/UFA) and the German Academic Exchange Service (DAAD) for their financial support to the cotutelle, and the "Welcome Services for international academics" at the University of Passau for their generous help with administrative tasks.

My thanks also go to Thomas Cerqueus and Antoine Boutet who helped me enormously with meaningful discussions and insightful comments. I also do not forget Gabriele Gianini who opened new perspectives into my PhD topic and gave me valuable feedbacks.

I also want to thank my fellow labmates at the University of Passau (Germany) and at INSA Lyon (France) for their support: Christin, Johannes, Jörg, Konstantin, Stefan, Torben, Amadou, Diana, Guido, Loïc, Lyes, Manel, Mazen, Merza, Pierre-Louis, Tarek, Tobias, Vincent (B), Vincent (P), Yulian, Zeina.

Finally, I want to address my special thanks to Marielle for her continuous support during this journey full of unexpected turbulences.

Abstract

During the last few years, the technological progress in collecting, storing and processing a large quantity of data for a reasonable cost has raised serious privacy issues. Many online services (e.g., Facebook, Google) now have the ability to profile their customers to offer them targeted advertising. The revelation by Edward Snowden in May 2013 of a massive global surveillance program run by the NSA demonstrates that intelligence services are also collecting and exploiting a large quantity of personal data. Therefore, on a daily basis, many entities spy on users interactions, but most Internet users do not measure the extent of the collection. The main reason is that online services do not offer their users the possibility to access the collected data. In addition, their terms of service do not precisely inform users about what data is collected, and the purpose of the collection.

Privacy concerns many areas, but is especially important in frequently used websites like search engines (e.g., Google, Bing, Yahoo!). These services allow users to retrieve relevant content from an increasing amount of data published on the Internet. The good quality of their results comes from the exploitation of user personal data. As a direct consequence, search engines are likely to gather and store sensitive information about individual users (e.g., interests, political and religious orientations, health condition). In this context, developing solutions to enable users to query these search engines in a privacy-preserving way is becoming increasingly important.

In this thesis, we introduce SimAttack an attack against existing solutions to query a search engine in a privacy-preserving way. This attack aims at retrieving the original user query by exploiting unprotected user queries previously collected by an adversary. We use SimAttack to assess the robustness of three representative state-of-the-art privacy-preserving solutions. We show that these solutions are not satisfactory to protect the user privacy.

We therefore develop PEAS a new protection mechanism that better protects the user privacy (according to SimAttack). This solution leverages two types of protection: hiding the identity of the user and masking her queries. The former is achieved by ciphering and sending queries through a succession of two nodes, while the latter hides queries by combining them with several fake queries. The main challenge in our approach is to generate realistic fake queries. We solve it by generating queries that could have been sent by other users in the system.

Finally, we present mechanisms to identify sensitive queries. Our goal is to adapt existing protection mechanisms to protect sensitive queries only, and thus save resources (e.g., CPU, RAM). Indeed, a common query on a cake recipe does not need the same protection as a query on an HIV infection. We design two modules to identify sensitive queries and deploy them on real protection mechanisms. We establish empirically that adapting existing protection mechanisms dramatically improves their performance.

Résumé

Au cours des dernières années les progrès technologiques permettant de collecter, stocker et traiter d'importantes quantités de données pour un faible coût, ont soulevés de sérieux problèmes concernant la vie privée. De nombreux services web (ex. : Facebook, Google) ont maintenant la possibilité de créer un profil de leurs clients pour par exemple, leur offrir de la publicité ciblée. La révélation par Edward Snowden en mai 2013 d'un programme de surveillance massif et généralisé opéré par la NSA démontre que les services secrets collectent et exploitent massivement les données personnelles. Ainsi, de nombreuses entités espionnent quotidiennement les interactions des utilisateurs, alors qu'une majorité de ces utilisateurs n'en a toujours pas conscience. Cette ignorance s'explique en partie par le fait que ces services web ne permettent pas à leurs utilisateurs de consulter les données qu'ils collectent. De plus, les conditions générales d'utilisation n'informent pas précisément l'utilisateur sur la nature des données collectées ainsi que sur l'objet de leur collecte.

La protection de la vie privée concerne de nombreux domaines, en particulier les sites internet fréquemment utilisés comme les moteurs de recherche (ex. : Google, Bing, Yahoo!). Ces services permettent aux utilisateurs de retrouver efficacement du contenu parmi une quantité grandissante de données publiées sur internet. La pertinence de leurs recommandations s'explique par l'exploitation des données personnelles des utilisateurs. Par conséquent, les moteurs de recherche sont susceptibles de collecter et stocker des données sensibles d'utilisateurs (ex. : leurs centres d'intérêt, leurs opinions politiques et religieuses, leurs conditions de santé). Dans ce contexte, développer des solutions pour permettre aux utilisateurs d'utiliser ces moteurs de recherche tout en protégeant leurs vies privées est devenu très primordial.

Dans cette thèse, nous introduirons SimAttack, une attaque contre les solutions protégeant la vie privée de l'utilisateur dans ses interactions avec les moteurs de recherche. Cette attaque vise à retrouver les requêtes initialement envoyées par l'utilisateur en exploitant d'anciennes requêtes non protégées précédemment envoyées par l'utilisateur. Nous utiliserons cette attaque pour analyser la robustesse de trois mécanismes de protection représentatifs des solutions existantes. Nous avons montré que ces trois mécanismes ne sont pas satisfaisants pour protéger la vie privée des utilisateurs.

Par conséquent, nous avons développé PEAS, un nouveau mécanisme de protection qui améliore la protection de la vie privée de l'utilisateur (en particulier par rapport à SimAttack). Cette solution repose sur deux types de protection : cacher l'identité de l'utilisateur et masquer sa requête. La première protection chiffre et envoie les requêtes via une succession de deux serveurs tandis que la deuxième protection masque la requête de l'utilisateur en la combinant avec des fausses requêtes. La difficulté majeure de notre approche est de générer des fausses requêtes réalistes. Nous avons résolu ce problème en générant des requêtes qui auraient pu être envoyées par d'autres utilisateurs du système.

Pour finir, nous présenterons des mécanismes permettant d'identifier la sensibilité des requêtes. Notre objectif est d'adapter les mécanismes de protection existants pour protéger uniquement les requêtes sensibles, et ainsi économiser des ressources (ex. : CPU, mémoire vive). En effet, une requête banale sur une recette de gâteau n'a pas besoin de la même protection qu'une requête sur une infection liée au VIH. Nous avons développé deux modules pour identifier les requêtes sensibles et nous avons déployé ces modules sur des mécanismes de protection. Nous avons établi qu'adapter des mécanismes de protection améliore considérablement leurs performances.

Zusammenfassung

In den letzten Jahren sind durch den technologischen Fortschritt im Bereich des Sammelns, Speicherns und Verarbeitens von großen Datenmengen mit überschaubaren Kosten schwerwiegende Datenschutzbedenken entstanden. Viele Online-Dienste (z.B. Facebook, Google) besitzen nun die Möglichkeit detaillierte Profile ihrer Nutzer zu erstellen und mit gezielter Werbung anzusprechen. Die Veröffentlichungen von Edward Snowden im Mai 2013 brachten die tiefgreifende globale Überwachung durch die NSA ans Licht die zeigten wie Geheimdienste ebenfalls große Mengen an persönlichen Daten sammeln und nutzen. Tagtäglich werden die Aktivitäten von Nutzern von vielen Seiten ausspioniert, aber die meisten Nutzer sind sich des Ausmaßes der Überwachung nicht bewußt. Der Hauptgrund liegt darin, dass Online-Dienste ihren Nutzern nicht die Möglichkeit geben auf ihre gesammelten Daten zuzugreifen. Darüberhinaus informieren die Nutzungsbedingungen nur ungenügend über die Art der gesammelten Daten und den Zweck der Sammlung.

Datenschutz betrifft viele Bereiche, ist aber besonders wichtig für häufig besuchte Webseiten wie Suchmaschinen (z.B. Google, Bing, Yahoo!). Diese Dienste ermöglichen es Nutzern relevante Inhalte aus dem stetig wachsenden "Datensee" im Internet herauszuholen. Die gute Qualität ihrer Ergebnisse beruht auf der Nutzung persönlicher Daten. Als direkte Konsequenz ist es wahrscheinlich, dass Suchmaschinen sensible Informationen über einzelne Nutzer sammeln und speichern (z.B., Interessen, politische und religiöse Orientierung, Gesundheitszustände). In diesem Zusammenhang gewinnt die Entwicklung von Privatsphäre-schützenden Lösungen, die es Nutzern erlauben Suchmaschinen zu befragen, an Wichtigkeit.

In dieser Arbeit, präsentieren wir SimAttack als Angriff gegen existerende Lösungen um eine Suchmaschine in Privatsphäre-erhaltender Weise anzufragen. Dieser Angriff zielt darauf ab die ursprüngliche Nutzeranfrage wiederherzustellen, indem man ungeschützte Nutzeranfragen verwendet die der Angreifer zuvor gesammelt hat. Wir nutzen SimAttack um die Robustheit von drei repräsentativen State-of-the-Art Privatsphäre-erhaltenden Lösungen zu evaluieren. Wir zeigen, dass diese Lösungen die Privatsphäre der Nutzer nicht zufriedenstellend schützt.

Deshalb entwickeln wir PEAS als neuen Mechanismus, um den Schutz in Bezug auf SimAttack zu erhöhen. Dazu verfolgt PEAS zwei Ansätze: verbergen die Identität der Nutzer und maskieren der Suchanfragen. Ersteres wird erreicht durch Verschlüsseln und Versenden der Anfragen durch eine Folge über zwei Knoten, während letzteres Anfragen verschleiert indem sie mit mehreren imitierten Anfragen vermischt. Die Hauptherausforderung unseres Ansatzes ist es realistische, imitierte Anfragen zu erzeugen. Dazu werden Anfragen erzeugt, die von einem anderen Nutzer des Systems geschickt hätten werden können.

Letztlich präsentieren wir ein Verfahren um sensible Anfragen zu iden-

tifizieren. Unser Ziel ist es existierende Schutzmechanismen anzupassen um nur sensible Anfragen zu schützen und damit Ressourcen zu schonen (z.B. CPU, RAM). In der Tat braucht eine gewöhnliche Anfrage nach einem Kuchen-Rezept nicht das gleiche Ausmaß an Schutz wie eine Anfrage nach einer HIV-Infektion. Wir entwerfen zwei Module um sensible Anfragen zu identifizieren und kombinieren sie mit real-existierenden Schutzmechanismen. Wir zeigen empirisch, dass diese Modifikation die Leistung existerender Schutzmechanismus dramatisch verbessert.

Contents

Acknowledgements iii
Abstract v
Résumé vii
Zusammenfassung ix
List of Figures xiv
List of Tables xvii
List of Algorithms xviii

Chapter 1 Introduction 1

1.1 Motivation . . . 1
1.2 Problem Statement . . . 4
1.3 Research Objective . . . 4
1.4 Contributions . . . 5
1.5 Thesis Organization . . . 6

Chapter 2 State of the Art 9

2.1 Private Web Search Solutions . . . 9
2.1.1 Unlinkability Protocols . . . 9
2.1.2 Indistinguishability Mechanisms . . . 16
2.1.3 Private Information Retrieval . . . 19
2.1.4 Private Web Search Metrics . . . 19
2.1.5 Discussion . . . 27
2.2 Sensitive Query Detection . . . 32
2.2.1 Data Sensitivity . . . 33
2.2.2 Query Categorization . . . 34
2.2.3 Discussion . . . 35

Chapter 3 SimAttack, an Efficient and Scalable Attack Against Private Web Search Solutions 37

3.1 Objective . . . 37
3.2 Design of the Attack . . . 37
3.2.1 Similarity Metric Between a Query and a User Profile . 38
3.2.2 Unlinkability Attack . . . 39
3.2.3 Indistinguishability Attack . . . 39
3.2.4 Indistinguishability Over an Unlinkability Solution Attack . . . 41

3.3 Experimental Set-up . . . 43
3.3.1 Datasets . . . 43
3.3.2 Representative Private Web Search Solutions . . . 46
3.3.3 Concurrent Attacks . . . 47
3.3.4 Evaluation Metrics . . . 50
3.3.5 Query Similarity Metric Selection . . . 51
3.3.6 Smoothing Factor α . . . 52
3.4 Evaluation . . . 55
3.4.1 Privacy Protection . . . 55
3.4.2 Execution Time . . . 64
3.5 Conclusion . . . 66

Chapter 4 PEAS, a New Robust and Efficient private Web Search Solution 69

4.1 Objective . . . 69
4.2 PEAS in a Nutshell . . . 69
4.3 PEAS Detailed Description . . . 70
4.3.1 Privacy Proxy . . . 72
4.3.2 Obfuscation Mechanism . . . 72
4.3.3 Filtering Irrelevant Results . . . 76
4.4 PEAS Distributed Deployment . . . 77
4.4.1 Receiver and Issuer Selection . . . 78
4.4.2 Group Profile Management . . . 78
4.5 Security Analysis . . . 79
4.5.1 Cryptographic Algorithms Used by PEAS . . . 79
4.5.2 Unlinkability Guarantee of the Privacy Proxy . . . 79
4.5.3 Non-Collusion Assumption between the Receiver and the Issuer . . . 80
4.5.4 Publication of the Group Profile . . . 80
4.5.5 Indistinguishability of the Initial Queries . . . 81
4.5.6 Number of Fake Queries k . . . 81
4.6 Experimental set-up . . . 82
4.6.1 Dataset . . . 82
4.6.2 Methodology and Metrics . . . 82
4.7 PEAS Experimental Evaluation . . . 85
4.7.1 Privacy . . . 86
4.7.2 Accuracy . . . 91
4.7.3 Performance . . . 93
4.8 Conclusion . . . 98

Chapter 5 SAM and LAM, Identifying Sensitive Queries to Improve the Efficiency of Private Web Search Solutions 99

5.1 Objective . . . 99
5.2 Overhead Made by Private Web Search Solutions . . . 100
5.2.1 Overhead Due to the Protection of Semantically Non-sensitive Queries . . . 100
5.2.2 Overhead Due to the Protection of Non-Linkable Queries 102
5.3 Sensitivity Assessment . . . 103
5.3.1 Overview . . . 103
5.3.2 User Centric Sensitivity Modeling . . . 103
5.3.3 Semantic Assessment Module (SAM) . . . 104

5.3.4 Linkability Assessment Module (LAM) 105
5.4 Deployment 106
5.5 Experimental Set-up 106
5.5.1 Datasets 107
5.5.2 Embarrassing Categories 107
5.5.3 Evaluation Metrics 108
5.5.4 Creating the dictionary D used by SAM 108
5.6 Evaluation 109
5.6.1 Evaluation of the Semantic Assessment Module 109
5.6.2 Evaluation of the Linkability Assessment Module . . . 110
5.6.3 Impact of Using SAM and LAM on Existing Private Web Search Solutions 112
5.7 Discussion 114
5.7.1 On the Identification of Users' Identity 114
5.7.2 On the Advantages of the Assessment Modules 115
5.7.3 On the Adaptation of the Number of Fake Queries . . 115
5.8 Conclusion 117

Chapter 6 Conclusion and Perspectives **119**

6.1 Conclusion 119
6.1.1 Robustness of Existing Private Web Search Solutions . 119
6.1.2 A New Efficient Private Web Search Solution 120
6.1.3 Adapting the Protection to the Query Sensitivity . . . 121
6.2 Perspectives 121
6.2.1 Dataset Employed and Adversary Model 121
6.2.2 Privacy Attacks 122
6.2.3 Private Web Search Solutions 123
6.2.4 Adaptation of Existing Solutions 124

Appendix A Stopwords **125**

Appendix B Complementary Experiments on SimAttack **127**

Appendix C Proverif code of PEAS **133**

Appendix D Instructions Given in the Crowd-sourcing Task Conducted with Crowdflower **137**

Appendix E List of Categories Available in the Semantic Assessment Module (SAM) **139**

Bibliography **141**

List of Figures

Figure 1 Our three contributions presented in a single scenario. . . 6
Figure 2 Sending a query to a search engine through the Tor protocol. 12
Figure 3 Distribution of the number of queries issued per user in the filtered dataset. 44
Figure 4 Distribution of the number of queries issued per user for the different dataset. 45
Figure 5 More queries in the requester user profile are similar to the protected query than the ones contained in other profiles. 53
Figure 6 The smoothing factor α does not have a strong influence on the similarity metric computed by SimAttack. 54
Figure 7 The smoothing factor α does not have a significant impact on the F1-score of SimAttack. 55
Figure 8 Increasing the number of users in the system decreases both the recall and the precision of SimAttack. 56
Figure 9 SimAttack and ML Attack have a similar recall and precision. 56
Figure 10 The value of δ significantly impacts the performance of SimAttack. 57
Figure 11 The number of fake queries sent by TrackMeNot only impacts the precision of SimAttack. 59
Figure 12 The number of fake queries generated by GooPIR has a limited impact on the privacy protection of the user. . . . 60
Figure 13 The number of fake queries generated by GooPIR has a limited impact on the privacy protection of the user. . . . 60
Figure 14 The value of δ significantly impacts the performance of SimAttack against TMN over an unlinkability solution. . . 61
Figure 15 The number of fake queries sent by TrackMeNot over an unlinkability solution does not change the recall but strongly impacts the precision of SimAttack. 62
Figure 16 The number of fake queries has a limited impact on the protection of queries. 63
Figure 17 SimAttack is faster than the machine learning attack, especially for a high number of users. 64
Figure 18 Considering TMN, SimAttack identifies user queries faster than the machine learning approach for both classifiers. . . 65
Figure 19 Considering GooPIR, SimAttack performs much faster than the machine learning attack. 66
Figure 20 PEAS combines an unlinkability solution provided by the privacy proxy and an indistinguishability solution performed by the client that obfuscates the user queries. 69
Figure 21 Sequence diagram of PEAS. 71
Figure 22 A partial view of the operations performed by the issuer. . 73
Figure 23 Example of a co-occurrence matrix included in a group profile. 74

Figure 24 Example of a co-occurrence graph extracted from a co-occurrence matrix. 74
Figure 25 The two possible PEAS distributed architectures: a centralized group profile and a decentralized group profile. . . 78
Figure 26 The robustness of PEAS increases according to the number of fake queries. 86
Figure 27 PEAS outperforms GooPIR regardless of the number of fake queries. 87
Figure 28 Taking the confidence level into consideration decreases the recall of the privacy attacks but increase the precision . 88
Figure 29 PEAS fake queries are close to previous queries sent by other users in the system. 89
Figure 30 Increasing the size of the user profile improves the quality of fake queries. 90
Figure 31 The distributed deployment does not really impact the privacy protection. 91
Figure 32 Comparison between the filtering performed by PEAS and the filtering performed by GooPIR. 92
Figure 33 The word order of the query has a non-negligible impact on the results returned by the search engine. 93
Figure 34 The maximal clique extraction performed by the client follows an exponential running time. 93
Figure 35 The latency is impacted by the number of fake queries. . . 94
Figure 36 The size of the messages sent over the network increases according to the number of fake queries. 95
Figure 37 The scalability of the receiver and the issuer are similar, both serve queries up to a certain load where they become saturated and are no longer able to answer clients straightaway. 96
Figure 38 Comparison between the scalability of PEAS and Onion Routing. 97
Figure 39 Comparison between the scalability of PEAS for a different number of servers (*PEAS x-x* means that the PEAS protocol was deployed with x receivers and x issuers). . . . 97
Figure 40 Percentage of linkable queries per user. 102
Figure 41 Deployment of SAM and LAM for the three existing private Web search solutions: Tor, GooPIR, and PEAS. . . . 106
Figure 42 The recall and the fall-out obtained with SAM according to the maximum number of synsets per keyword s and the maximum number of categories per synsets c. 109
Figure 43 The recall and the fall-out obtained with LAM according to the smoothing factor α and the threshold λ (considering 100 users in the system). 111
Figure 44 Impact of the number of users on the recall and the fall-out obtained using the Linkability Assessment Module. . . 112
Figure 45 Mapping of the similarity metric between the query and its requester user profile to a number of fake queries. . . . 115
Figure 46 The efficiency of the attack decreases according to the proportion of queries considered in user profiles. 128
Figure 47 Exploiting smaller user profiles makes it harder for SimAttack to identify user queries. 128
Figure 48 GooPIR ensures a better protection if the adversary has only a smaller and less accurate user profile. 129

Figure 49 Exploiting smaller user profiles makes it harder for SimAttack to identify user queries. 130
Figure 50 Exploiting smaller user profiles make it harder for SimAttack to identify user queries. 130
Figure 51 Increasing the number of user profiles, with respect to the number of users in the system, decreases the precision while slightly reducing the recall. 131
Figure 52 The number of de-anonymized queries increases according to the number of probable users returned by SimAttack. 132
Figure 53 The recall slightly increases according to the number of pairs $(query, user)$ returned by SimAttack. 132

List of Tables

Table 1 A summary of metrics to assess the privacy protection, the performance and the accuracy of private Web search techniques. 28
Table 2 Comparison between private Web search solutions. 30
Table 3 Comparison between the existing query classifiers 35
Table 4 Dice's coefficient maximizes the efficiency of the SimAttack on the dataset `AOL100`. 52
Table 5 SimAttack performs better considering an adversary with prior knowledge about RSS feeds. 58
Table 6 Performance of the machine learning classifiers on queries protected by TrackMeNot. 58
Table 7 Performance of the privacy attack considering TMN over an unlinkability solution and the exploitation of the RSS feeds. 62
Table 8 Number of cryptographic operations and traffic generated in the whole system. 96
Table 9 Categories considered as embarrassing categories. 108
Table 10 Impact of the number of synsets and categories on the J_3 score (in %) obtained with SAM. 110
Table 11 Impact of the thresholds λ and α on the J_3 score (in %) obtained with LAM (considering 100 users in the system). . 111
Table 12 Comparison between the privacy protection offered by Tor and Tor+SAM. 112
Table 13 Comparison between the protection offered by GooPIR and GooPIR+SAM. 113
Table 14 Comparison between the protection offered by PEAS and PEAS+SAM+LAM. 114
Table 15 Impact on the robustness of PEAS considering the new version of LAM that adapts the number of fake queries. 116

List of Algorithms

Algorithm 1	Similarity metric between a query and a user profile.	38
Algorithm 2	De-anonymization Attack.	39
Algorithm 3	Indistinguishability Solutions Attack.	40
Algorithm 4	Indistinguishability over unlinkability Attack.	41
Algorithm 5	ML Attack against GooPIR.	48
Algorithm 6	ML Attack against GooPIR over an unlinkability solution.	50
Algorithm 7	Machine learning attack against TMN over an unlinkability solution.	50
Algorithm 8	Generation of the fake queries.	75
Algorithm 9	Results filtering.	77
Algorithm 10	Simulation of the execution of the OR operator.	84
Algorithm 11	Semantic Assessment Module.	105

Chapter 1

Introduction

1.1 Motivation

The advent of social networks in the 2000s has been followed by several controversies about personal information [1, 2]. At that time, users did not realize that making their private life publicly available could have serious consequences, in particular be used many years later against them. The exploitation of personal data has become important for marketing purposes. In recent years, the technological progress in collecting, storing and processing a large quantity of data for a reasonable cost has enabled Web providers to profile their users. For instance, Facebook has based its business model on targeted advertising. It delivers relevant advertisements to Facebook users by exploiting their user profiles. These profiles are constructed from the interactions made by users (e.g., likes or posts on Facebook) and contain all topics users are interested in (e.g., football, politics). Nevertheless, this is not respectful of the user's privacy as users are not aware which data is precisely collected by the platform and what type of mining techniques is applied. Indeed, terms of service are usually vague: they do not clearly indicate which data is collected and which operations are performed. For instance, Google specifies that it stores data that "*you upload, submit, store, send or receive to or through our Services*" and their "*automated systems analyze your content*" [3]. Private data exploitation does not only concern online providers. The revelation by Edward Snowden in May 2013 of a massive global surveillance program run by the NSA [4], convinced even the most skeptical people that the exploitation of personal data is a reality. To prevent such an exploitation, users should control information that they explicitly put online (e.g., tweets, posts) but also information they share with online services. For that reason, most major online services have been developing settings to enable users to configure their own privacy policy. Nevertheless, as shown by a study on Facebook [5], there is often a gap between what the user wants for her privacy and what she achieves in practice. It is explained by two facts: either users do not correctly set up their privacy settings or there is no option to configure what they desire. Regarding the last reason, the European Union is pushing Web providers to implement some privacy settings. For instance, Article 12(b) of the EU Directive 95/46/EC [6] gives a person the right to ask for her personal data to

be deleted, if her data is incomplete or inaccurate. In practice, some Web providers ignore such a regulation. For instance, Google was condemned in May 2014 by the Court of Justice of the European Union (CJEU) because it refuses to withdraw some data from its index [7]. Therefore, Google was forced to introduce a *right to be forgotten* in its search results. Since then, Google has already received 639,271 requests to remove 1,794,862 URLs [8]. In 2016, the European Union modernizes its data protection rules through the Regulation 2016/679 [9]. The noticeable changes concern non-European Web providers that have to respect European protection rules. This regulation also inverses the burden of proof (i.e., Web providers have to prove the relevance of the data; not the individuals), forces Web providers to inform third parties that a person wants her data to be deleted, and imposes fines to companies that do not respect regulations (up to 2% of the annual worldwide turnover). Nevertheless, these privacy regulations are only available in Europe and Web providers chose not to generalize them globally [10, 11].

Search engines (e.g., Google, Bing, Yahoo!) have become an incontrovertible way for finding content on the Internet. In the 90s, retrieving specific content was not an easy task even if the quantity of accessible websites was much lower. Web directories (e.g., the WWW Virtual Library [12], LookSmart [13]), maintained by human editors, were at that time the main solution for users to discover new websites. Because of the increasing amount of data populating the Web, search engines have become the natural way to access information on the Web. In twenty years, they have made incredible progress to retrieve accurate information among this large quantity of data. This is mainly due to the collection and the exploitation of usage data [14, 15]: inferring user interests from their query logs, refining recommendations based on the popularity of a result or on the time a user spent on the result. That is, a better understanding of users and their interactions allows search engines to personalize their answers and improve their recommendations.

While gathering all this data has been a key factor in improving the quality of the results returned by search engines, it has also raised serious issues regarding user privacy. With the huge number of queries stored by a search engine, the latter knows better than anyone the interests of its users. By using a search engine, users do not imagine that they expose themselves to a massive leak of information. In 2006, AOL published a set of queries previously issued by their users [16]. Its objective was to give real data to the research community. These logs contain approximately 21 million queries formulated by 650,000 users over three months (March, April, and May 2006). Even though these logs are pseudo-anonymized (meaning that explicit user identifiers, e.g., name, email address, are removed; instead, users are identified by a unique number), the identities of a few users could have been retrieved based on information contained in their queries [17]. Beyond the scandal of publishing real search queries without asking for the user consent, this release has allowed many studies on real search queries [18, 19, 20, 21]. In terms of privacy, these studies mainly focus on understanding what personal information is extractable from these logs (e.g., [22]). Indeed, these queries contain explicit and implicit information about their requesters. Explicit information is contained in the information embedded in the query itself. For instance, the person behind the user id #4417749 in the AOL log dataset was identified as Thelma Arnold. Indeed, it was deduced from her queries (and other public information) that she was a sixty years old single woman living in Lilburn with three dogs. Explicit information either reveals straightforward facts to infer the identity of the user (e.g., names, social security numbers, address) or discloses personal behavior about the user. For instance, in the query "*my*

head feels sensitive to the touch after doing cocaine" sent by the user #1879967, we understand that this user consumes illegal drugs and put her health in danger. On the other hand, implicit information is extracted from the whole set of logs using advanced techniques (e.g., data mining methods [23, 22]). For instance, it is possible to infer the age of a user by analyzing her queries. Indeed, it can be assumed that queries about video games are made by a young person while queries about retirement pensions are made by an older person. The same reasoning can be made for other categories (e.g., wealth, sex, location). As a consequence, query logs offer a wealth of information as explicit and implicit data reveals many details about users' lives.

One might argue that inferring and disclosing information about herself is not a problem as "*she has nothing to hide*". But, as mentioned by Solove in [24], the term "privacy" is not reduced to the disclosure of secret information. It also includes the exploitation of personal data, which can be seen as watching all the comings and goings of individuals. Nevertheless, as it is hard for users to assess the outcome of such a data exploitation, Marx [25] defines privacy as "*a value which may only be appreciated once it is lost*". Indeed, individuals often underestimate the impact of their data. This is pointed out by Schneier in an article entitled "The Eternal Value of Privacy" where he denounces potential abuses of the exploitation of private data. He illustrated his speech by quoting Cardinal Richelieu: "*If one would give me six lines written by the hand of the most honest man, I would find something in them to have him hanged.*" [26]. Schneier wants to emphasis that the exploitation of personal data can have disastrous consequences. For instance, in the context of Web search, a malevolent search engine could sell information on the wealth or diseases of their users to banks or insurance companies.

As the current search engines lack dedicated privacy-preserving features and do not fulfill people's expectations in terms of privacy, alternative search engines have emerged: metasearch engines (e.g., DuckDuckGo [27], StartPage [28] and search engines (e.g., Qwant [29]). The former enhance existing search engines by focusing on the privacy-protection of their users, while the latter develops a search engine that does not exploit users' information. Nevertheless, these alternatives do not implement any specific privacy-preserving mechanisms. Instead, they claim, in their terms of service, that they do not collect personal information about individuals. For instance, DuckDuckGo affirms that they "*also save searches, but again, not in a personally identifiable way, as we do not store IP addresses or unique User agent strings*" [30]. However, as we have seen in the AOL scandal, pseudo-anonymized logs are not sufficient to correctly protect users. Besides, as their implementations are not publicly available, and as they do not explicitly provide the data they log, users cannot be confident on the privacy protection obtained by these solutions. Users can only trust these services and hope that their data is saved in a privacy-preserving way.

To overcome this issue, researchers have been investigating solutions to create search engines that ensures a privacy-protection by design. Under these schemes, users can query a search engine without revealing the query content. In practice, such solutions rely most of the time on homomorphic encryption and specific recommendation algorithms (e.g., [31, 32]). These approaches are promising as they ensure a full-privacy protection (i.e., search engines do not manipulate data in an intelligible format). Nevertheless, cryptographic operations increase the computation time on both clients and search engines [33] and impact the quality of the results (as several information retrieval techniques, e.g., query expansion, cannot be applied to encrypted queries). Apart from these technical reasons, search engines have no interest to deploy such

algorithms. They would lose the ability to exploit personal data and thus to make money with targeted advertising. Therefore, as their deployment seems unrealistic in practice, we do not consider in this thesis such a type of solutions.

1.2 Problem Statement

In this thesis, we aim to find solutions to preserve the user privacy in their interactions with Web search engines. Solutions must be compatible[1] with existing search engines (i.e., no modification is required on search engines). Regarding the privacy requirements, we consider an *adversary* that aims to collect personal data about users. In practice, the adversary can be seen as a curious search engine. Therefore, private solutions must prevent search engines from collecting accurate data on users and thus, deducing their interests. Furthermore, we assume that search engines received from their users non-protecting queries (i.e., before their users use any protection mechanisms). Consequently, search engines own a query history about their users that they can use to bypass privacy-preserving mechanisms (e.g., de-anonymized anonymous queries). Therefore, private solutions must ensure that they are resilient to an exploitation of these queries. We also assume that the adversary has access to all additional public knowledge (typically, information available on the Internet).

Private solutions also have requirements in terms of efficiency. These techniques introduce mechanisms that have a non-negligible cost for users. They might require extra computation time (e.g., to generate fake queries) or increase the latency (i.e., the time waited by users to obtain their results) due, for instance, to query anonymization. Consequently, a first requirement is to have a reasonable overhead such that it does not degrade the user experience.

A third set of requirements concerns the quality of the results returned by search engines. Private Web search solutions often degrade the quality of the results (due to the protection applied to queries). Protecting the user privacy, and at the same time obtaining irrelevant results, is not satisfactory and makes protection mechanisms useless. Due to this reason, protection mechanisms should preserve the quality of the results as much as possible.

1.3 Research Objective

Solutions to tackle this problem already exist in the literature. They are classified into two categories. The first one, called *unlinkability*, consists in hiding the user's identity from the search engine (typically her IP address[2]). Unlinkability solutions are mainly ensured by anonymous communication protocols (e.g., Onion Routing [38], Tor [39], Dissent [40, 41], RAC [42]). There are also peer-to-peer solutions (e.g., UPIR [43], Crowds [44]) in which users, grouped in a community, send their queries to a search engine on behalf of each other. The second type of solutions, called *indistinguishability*, aims at either altering the user's queries or hiding the user's interests. In practice they add extra queries to the original query (e.g., GooPIR [45]) or send periodically fake queries to the search engine (e.g., TrackMeNot [46]). Nevertheless,

[1] The term "compatible" refers to technical aspects. Privacy-preserving techniques might not be conformed to the terms of service provided by search engines or the legislation in force in the country of use. Nevertheless, these legal aspects remain outside the scope of this thesis.

[2] In the thesis, we assume that users employ solutions (e.g., [34, 35, 36, 37]) to (i) remove all quasi-identifiers in her queries (e.g., cookies, HTTP header, set of plugins) and (ii) block all tracking systems that could lead to the identification of the user.

attacks against unlinkability and indistinguishability techniques [47, 48, 49] have shown that they are not satisfactory to protect all queries. As these two types of solutions are complementary (i.e., the former modifies the IP address of the requester while the latter modifies the content of the query), a natural question that arises is whether a combination of these approaches would solve the problem. In the literature, no solution combining both unlinkability and indistinguishability solutions has been proposed. Therefore, we attempt to address the two following research questions through this thesis:

- **Q1**: *Are current state-of-the-art solutions (e.g., GooPIR, Tor, TrackMeNot) satisfactory to query a search engine in a privacy-preserving way? What kind of protection is the best for users?*
- **Q2**: *Can the combination of an indistinguishability mechanism over an unlinkability protocol improve the privacy preservation of users' data? Can new mechanisms give a better user privacy protection?*

Furthermore, solutions from the literature give the same protection to queries. They do not adapt their protection to the sensitivity of a query: a query about HIV infection is protected as a common query about a cake recipe. Consequently, current protection mechanisms waste resources to protect non-sensitive queries. Therefore, we also address the following research question:

- **Q3**: *Is it possible to assess the sensitivity of a query? If yes, can the efficiency of private Web search solutions be improved by adapting their protection mechanism to the sensitivity of a query?*

1.4 Contributions

Regarding the first research question **Q1**, previous studies [47, 48] established that an adversary with a user query history is able to break the protection achieved by both unlinkability and indistinguishability solutions. Nevertheless, the impact of the attack is limited as it does not scale regarding both the number of queries and the number of users. For that purpose, we design a new attack, SimAttack, that requires less computation time and succeeds in breaking a larger proportion of queries. As a result, we show that today's state-of-the-art methods are not satisfactory to protect users.

For the second research question **Q2**, we show empirically that running an existing indistinguishability solution over an unlinkability protocol improves the user protection but not enough to significantly boost the user protection. It is the reason why we design a protocol called PEAS (Privacy, Efficient and Accurate Web Search). PEAS relies on new unlinkability and indistinguishability solutions. Our approach has a better tolerance against an adversary that owns a user query history. In addition to a better privacy preservation, the unlinkability part of PEAS is designed for a more suitable efficiency (compared to state-of-the-art solutions) while the indistinguishability part of PEAS aims at obfuscating user queries without a significant degradation of the quality of the results.

Concerning the last research question **Q3**, current private Web search solutions can be improved by adapting their protection mechanisms to the query sensitivity. Instead of protecting all queries similarly, private Web search mechanisms can focus on sensitive queries. This would significantly improve their efficiency while the privacy protection would remain barely unchanged

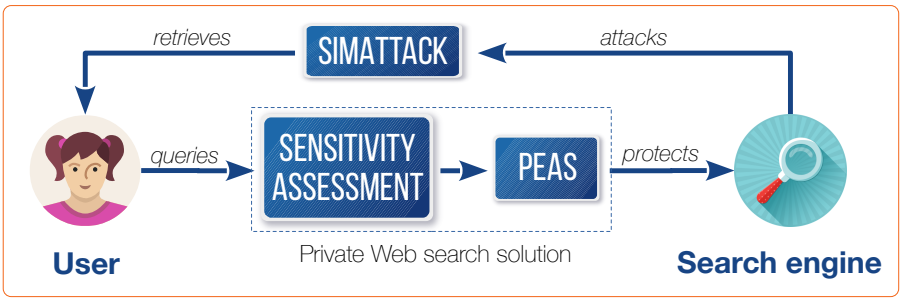


Figure 1
Our three contributions presented in a single scenario.

for users. We identify two types of sensitivity: (i) queries related to an embarrassing topic (semantic) and (ii) queries close to its requester profile (linkability). Our third contribution identifies sensitive queries with two modules: SAM (Semantic Assessment Module) and LAM (Linkability Assessment Module). We deploy these modules on existing solutions and show that they can significantly improve their performance.

We summarize the three contributions presented in this thesis through the scenario presented in Figure 1. A user sends a query through a private Web search solution. The sensitivity of the query is assessed and PEAS (the private Web search solution) adjusts its query protection accordingly. Finally, an adversary (e.g., the search engine) uses SimAttack to break the query protection and retrieves both the user identity and the user query.

1.5 Thesis Organization

The remainder of this thesis is organized as follows. Chapter 2 focuses on some relative work. Chapters 3 to 5 describe the three contributions developed in this thesis. Chapter 6 concludes the thesis and presents its perspectives.

A more detailed overview of the content of each chapter follows:

- **Chapter 2** — We present in this chapter the related work that has been reported in the literature. In particular we focus on two aspects: the protection of the user privacy and the identification of sensitive data. Regarding the first aspect, we mention several solutions that allow users to send their queries to search engines in a privacy-preserving way. For each of them, we discuss their shortcomings. Regarding the identification of sensitive data, we first exhibit different definitions of sensitive data and then review solutions that identify sensitive queries.
- **Chapter 3** — The third chapter presents SimAttack, an efficient attack against private Web search solutions. SimAttack uses a query history of users (i.e., non-protected queries previously collected by the adversary) to bypass most privacy Web search protections. SimAttack bases its decision on a similarity metric between a protected query and the query history. We assess the robustness of three representative solutions with SimAttack and then analyze properties that influence the results of the attack.
- **Chapter 4** — The fourth chapter presents PEAS, a new private Web search solution. PEAS combines an unlinkability mechanism with an indistinguishability mechanism. The former hides the identity of the requester by sending queries through a succession of two nodes (i.e., the privacy proxy), while the latter hides user queries by merging them with

realistic fake queries (i.e., queries that could have been sent by other users in the system).

- **Chapter 5** — The fifth chapter introduces SAM and LAM, two modules to identify semantically sensitive and linkable queries. Then, these modules are used to adapt three representatives solutions and only protect sensitive queries. As a result, SAM and LAM decrease the cost of protecting queries without significantly reducing the user protection.
- **Chapter 6** — The last chapter summarizes and discusses the three contributions detailed in the thesis. It also focuses on the possible future work regarding these three contributions.

CHAPTER 2

State of the Art

We overview in this chapter related works about solutions that enable users to query search engines in a privacy-preserving way. We also examine metrics to assess these solutions in terms of privacy protection, accuracy of their results, and performance. Finally, we exhibit the existing methods to detect sensitive queries.

2.1 Private Web Search Solutions

Solutions to privately query search engines can be classified in two categories: (i) systems ensuring unlinkability between requesters and their queries, and (ii) systems guaranteeing indistinguishability of user interests. Privacy-aware mechanisms can also be directly implemented on the search engine through Private Information Retrieval (PIR) protocols.

2.1.1 Unlinkability Protocols

One approach to protect user privacy from a too curious search engine is to prevent the latter from identifying her real identity. As mentioned in [50, 51], the identity of a user can be tracked through multiple identifiers: IP address, cookies, HTTP headers, set of browser plugins, etc. Separately, each identifier might not be sufficient to uniquely identify a user (in that case, they are called *quasi-identifiers*) but they can often be combined to create a unique identifier. To avoid an adversary from creating such a unique identifier, quasi-identifiers have to be removed (e.g., [34, 35, 36]) or changed with fake values (e.g., [52]). Nevertheless, these techniques cannot be applied on IP addresses. To mask this specific identifier, we can distinguish two approaches: (i) a proxy-based approach, in which distant servers hide user IP addresses, and (ii) a user-based scheme, in which users make their queries sent by other users in a peer-to-peer community.

2.1.1.1 Proxy-based Approaches

Basic techniques to hide the IP address leverage a Proxy [53] or a VPN [54] server as a relay. This distant server forwards user queries to the search engine on behalf of the user and then returns its results to the user. Unfortunately,

these mechanisms only shift the privacy problem from the search engine to the relay, which can collect and analyze queries sent by users. Nevertheless, as users might have more trust in proxies than in search engines, these solutions still improve the data protection.

Similarly, commercial services (e.g., DuckDuckGo [27], StartPage [28]) provide a search tool based on other search engines (e.g., Google, Wikipedia). These services claim that they do not collect personal information about individuals. But, to monitor their platform and identify issues, they collect anonymized user interactions. For instance, DuckDuckGo affirms that they "*also save searches, but again, not in a personally identifiable way, as we do not store IP addresses or unique User agent strings*" [30]. Nevertheless, having "*no unique User agent string*" is not enough to protect users. For instance, multiple strings can be combined to create a unique identifier. As shown by the AOL scandal, protecting logs is not trivial. For that reason, users cannot be confident about the privacy protection brought by these platforms.

To overcome these issues, several anonymous techniques have been developed to prevent a third party from identifying the requester behind an anonymous query. For instance, Chaum suggested in 1981 a routing protocol named Mix network [55] to exchange untraceable electronic mail. This technique uses a chain of proxy servers called *mix* to hide the participants of an exchange. The protocol works as follows for a user U that wants to send a message M to the destination A. The user U ciphers first her message M and a random string R_0 with pk_A, the public key of A. Then, she ciphers the previous cipher text and a random string R_1 with pk_1, the public key of the mix. This double encrypted message is sent to the mix (left-hand side of Equation 1) that deciphers it and obtains the right-hand side of Equation 1 (the random string R_1 is removed).

$$\{R_1, \{R_0, M\}_{pk_A}\}_{pk_1} \rightarrow \{R_0, M\}_{pk_A} \tag{1}$$

Finally, the message $\{R_0, M\}_{pk_A}$ is sent to its final destination A which deciphers it with its private key to retrieve the message M. For a better protection, Chaum suggests in his paper to use multiple mixes (also called *cascade*). The use of a cascade requires small changes in the protocol. Instead of ciphering the message with one public key (pk_1 in the previous example), the message is ciphered successively with the public keys of the n selected mixes:

$$\{R_n, \{R_{n-1}, \ldots, \{R_2, \{R_1, \{R_0, M\}_{pk_A}\}_{pk_1}\}_{pk_2} \ldots\}_{pk_{n-1}}\}_{pk_n}$$

Furthermore, to prevent an eavesdropper from analyzing the order of messages received and sent by each mix, a mix reorders the traffic to change the messages' order of arrival.

Web Mixes [56] is based on the idea of Mix network introduced by Chaum. The core of the protocol remains unchanged (i.e., queries are sent through multiple mixes before reaching the search engines), but Berthold et al. suggest several improvements: (i) the protocol prevents an adversary from knowing the sender and the receiver of a query (i.e., sender anonymity and receiver anonymity), (ii) the protocol requires an authentification that makes flooding attacks impossible or very expensive, (iii) the protocol includes a feedback system to inform users about their current level of protection, and (iv) the protocol incorporates a cache proxy for better performance. Regarding the first aspect, sender anonymity and receiver anonymity are achieved by sending dummy queries and dummy answers. The quantity of dummy queries (answers) is adjusted such that all senders (receivers) in the protocol send (receive) the same amount of data during each period of time. This

prevents an adversary from identifying a sender and a requester using traffic analysis techniques. Regarding the second aspect, before sending queries, users have to authenticate themselves to each mix and generate a ticket. To do so, they individually contact each mix that verifies their certificate. Then, users have to provide the mix with a blinded message. The message is signed by the mix and returned to the users who in turn verify the signature. As detailed in Mix network, each encryption step requires the use of a random string (R_1, R_2, ..., R_n in the example above). Web Mixes replaces random strings by the unblinded messages generated by users (called tickets in the protocol). Regarding the third aspect, users are informed about their protection level. Indeed, considering a dynamic group of users (i.e., new users are continuously joining the protocol and some users are leaving the protocol), an adversary could deduce possible senders with an intersection attack. The protection level indicates the potentiality of such an attack. It corresponds to the number of users in the anonymity group (i.e., the group of possible senders obtained with the intersection attack). This group is computed using information periodically published by each mix: the number of active users and the logout time for each user who leaves the anonymity group. If the group becomes too small, users are alerted. Regarding the last aspect, Web Mixes introduces a cache-proxy between the Mix network and the distant server. The cache-proxy receives queries from the last Mix and sends them to the search engine. Then, it sends answers received from the search engine back to the user using the same mixes in a reverse order. The cache-proxy is also responsible to generate dummy answers.

West et al. [57] find a vulnerability in the authentication process. They pointed out that a mix has no guarantee that the ticket sent with messages was created in the current session. An attacker can use the ticket generated for an older message and thus bypass the authentication process. The authentication process has for objective to limit flooding attacks (as they would have been too expensive). By exploiting this flaw, an adversary is able to perform a flooding attack at a low cost.

Syverson et al. proposes a new technique called Onion Routing [38]. This technique establishes an anonymous connection between a user and a search engine using a sequence of intermediate nodes called *onion routers*. Onion routers are connected by a permanent connection. The user defines a route through the onion routing network, prepares an *onion* to establish the anonymous connection and sends it to the first onion router. An onion contains (i) information about the next hop in the route, (ii) a key seed for generating symmetric keys and (iii) an embedded onion. It is ciphered with the public key of the onion router to which the onion is sent. An onion router that receives an onion, removes its outer later using its private key, generates symmetric keys from the key seed and retrieves the next onion router. Then, it forwards the embedded onion to this onion router. As soon as the onion reaches the last onion router, the anonymous connection is established and thus the user can communicate with the search engine anonymously. To send data through the anonymous connection, the user encrypts successively her data for each onion router in the route (using the symmetric keys generated from the key seeds sent in the onion). Each onion router, receiving a message, removes (or adds, if data is sent backwards by the search engines) a layer of encryption and forwards it to next onion router. Finally, the last onion router retrieves the plaintext and transmits it to the search engine.

For instance, if the user wants to establish the anonymous connection involving the onion routers OR_1, OR_2, and OR_3, she generates the following

onion:

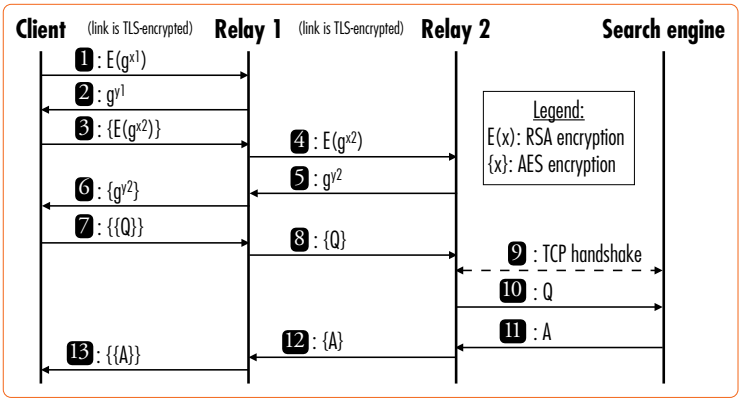
$$\{OR_1, seed_1, \{OR_2, seed_2, \{\varnothing, seed_3\}_{pk_3}\}_{pk_2}\}_{pk_1}$$

where pk_1, pk_2, and pk_3 are respectively the public keys of OR_1, OR_2, and OR_3; $seed_1$, $seed_2$, $seed_3$ are the key seeds for generating symmetric keys; and $\varnothing$ indicates to OR_3 that it is the last router. Then, when the anonymous connection is established, the user ciphers her query q successively with K_3, K_2, and K_1; where K_1, K_2, and K_3 are the symmetric keys established respectively with OR_1, OR_2, and OR_3. Then, she sends the cipher text to the first onion router that applies the protocol. When the cipher text reaches OR_3 (the last onion router in the path), it deciphers the cipher text and sends the query q to the search engine.

Tor [39] is the most famous onion routing technique (more than 3 million daily active users [58]). Tor is a second implementation of Onion Routing that addresses several of its limitations: congestion control, integrity checking, perfect forward secrecy, etc. Similarly to Onion Routing, Tor sends each query through multiple nodes using a cryptographic protocol. Queries are encrypted multiple times (creating different layers) and routed through randomly selected nodes. Each node deciphers the cipher text (removing each layer) and forwards it to the next node until it reaches the exit node. The exit node retrieves the query and sends it to the search engine. For simplicity, we detail the protocol using only two onion routers OR_1 and OR_2 (by default, Tor established a connection through three onion routers). As depicted by Figure 2, the user starts by negotiating a symmetric key with each onion router (OR_1 and OR_2). She sends the first half of the Diffie-Hellman handshake (g^{x1}) to the first router (OR_1) and in return gets the other part (g^{y1}), i.e., messages 1 and 2 in the figure. Using the two parts of the handshake, the user and OR_1 are able to compute a symmetric key $K_1 = (g^{x1})^{y1} = (g^{y1})^{x1} = g^{x1y1}$. Similarly, the user performs the same exchanges with the second router (OR_2) through a ciphered connection with OR_1 (using the symmetric key K_1) , i.e., messages 3 – 6 in the figure. We denote by $K_2 = g^{x2y2}$ the second symmetric key between the user and OR_2. Then, the user ciphers her query successively with K_2 and K_1. She sends the cipher text to OR_1 (messages 7) which deciphers the first layer using the symmetric key K_1. The onion router OR_1 forwards the remaining message to OR_2 (messages 8). Then, the router OR_2 deciphers the message it received from OR_1 using the symmetric key K_2. Finally, OR_2 sends the initial query to the search engine. As soon as it receives the results from the search engine, the router OR_2 ciphers the results with the symmetric key K_2, forwards the message to OR_1 that ciphers it with the symmetric key K_1, i.e., messages 9 –

Figure 2

Sending a query to a search engine through the Tor protocol.



12. Finally, the router OR_1 forwards the message to the user that deciphers it with successively the symmetric keys K_1 and K_2 to retrieve the results of her query.

The Tor protocol presents a major drawback when it is used to protect search queries. Symmetric keys (in the example above K_1 and K_2) can be used to protect multiple queries, as they are regenerated only when the path changes. In practice, Tor changes the path every 10 minutes meaning that, during this period of time, the exit node can use its symmetric key (in the example above K_2) to identify all queries coming from a unique requester. As shown by the AOL scandal [17], linking several queries together reveals information that can lead to the identity of the user.

Moreover, Tor does not prevent freerider nodes from misbehaving. Freeriders have no interest in acting as a relay. They will drop messages whenever they can. To overcome this issue, nodes should be monitored to verify that they behave correctly. RAC [42] enhances Tor with such a feature. Under this protocol, nodes are organized on several virtual rings such that, for a given ring, a node has a predecessor node and a successor node. A node might be part of several rings and thus have multiple predecessors and successors. To ensure that no message is dropped by a freerider, nodes have to broadcast all messages they relay. Broadcast messages have to circulate through all nodes in the ring such that if a node does not receive a message from one of its predecessors, it considers this predecessor as a freerider. Regarding the performance, the modifications made by RAC dramatically decrease its throughput, as RAC has a throughput 20,000 times lower than Tor.

Notwithstanding, all the aforementioned techniques [56, 38, 39, 42] rely on a non-collusion assumption. No router in the network has all the information about the query and its requester: the initial node knows the identity of the requester (i.e., her IP address) but has no information about her query (as it is ciphered); the last router knows the content of the query but has no knowledge about the identity of the requester; the intermediate nodes have no clear information about either the query or the user identity. If all routers in the path collude, they can break the anonymity property and associate the query with its requester. To decrease the probability of such an event, Tor offers the possibility to increase the number of nodes used to forward messages.

2.1.1.2 User-Based Schemes

User-based schemes also aim at hiding the user's IP address but using a fully decentralized architecture. They consider that a group of users who want to send queries can cooperate to send queries on behalf of each other. For instance, UPIR [43] proposes a protocol that allows users to get their queries sent by another user in the group. Users do not contact each other directly. They share memory sectors on a simple wiki-like collaborative environment. Users save their queries on an encrypted sector and then, another user sends them to the search engines. Answers are stored in the encrypted sector as well. The collaborative environment does not need to be trusted as queries and answers are stored in an encrypted format. Nevertheless, this solution is not satisfactory in practice as its response time (i.e., time required to receive the results) is relatively high. If we consider a system of 993 users, the best and the worst response time achieves by the solution (depending on the settings) are respectively 5.13 seconds and 30.74 seconds (without including the network latency).

Reiter and Rubin suggest a more efficient protocol with Crowds [44]. The protocol organizes users in a community such that they are able to send

queries on behalf of each other. Before sending queries to the search engine or to another user, users flip a coin to decide if they either forward the query to another user in the group or submit the query to the search engine. In Crowds, intermediate users know the content of the query, but as they are not sure about the identity of requesters (depending on the results of the coin flipping, the number of forwards may be long or short), they cannot aggregate queries sent by a single user. Viejo et al. [59, 60] propose some modifications to this protocol. The forward decision is no longer chosen by flipping a coin but users automatically forward queries that they consider harmful for them (decision based using the Profile Exposure Level, metric further detailed in Section 2.1.4.1). Besides, the protocol allows users to refuse to forward queries they receive from another user (for selfish reasons for instance). In that case, the user can ask another user to forward her query. If ultimately all users in the community refuse to forward the query, the user has to send it by herself. This means that her query is not protected. Viejo et al. have designed this protocol for high performances. They show that on average the response times is 0.73 seconds (much lower than 5.13 seconds required by [43]). Nevertheless, this protocol presents a major drawback. If all users refuse to forward queries, the user cannot get her query protected.

For that reason, Castellà-Roca et al. propose UUP [61], a more secure solution to exchange n queries in a privacy-preserving way. UUP uses cryptographic primitives to implement a private shuffling algorithm. All users in the group cipher their queries q_i with a shared key y and send them to each other. UUP gives a predefined order to each user in the group. One after the other, users shuffle queries, re-mask them (to obtain re-encrypt queries) and forward them to the next user. When the last user has re-masked and permuted the queries, she broadcasts the results $\{c_1, ..., c_n\}$ to all the group members, where $c_1, ..., c_n$ are the initial ciphered queries $\{q_1\}_y, ..., \{q_n\}_y$ permuted and re-masked. Then, each user deciphers her assigned query (according to her position in the protocol). Due to the n re-masking steps, the decryption requires from the n users to send their shares. They correspond to the initial cipher queries $\{q_i\}_y$ re-ciphered with their private key α_i. As soon as they obtain queries, users forward them to the search engine. Finally, as users do not know the identity of the initial requester, they cannot forward them the results of their queries. For that reason, upon receiving the results from the search engine, users broadcast them to all the n users. The initial requesters analyze the broadcast messages to identify the results corresponding to their initial queries.

Romero-Tris et al. [62] point out that UUP introduces a delay of 6.8 seconds. This is due to the cryptographic primitives and the broadcast messages. Romero-Tris et al. suggest modifications to decrease the response time. Firstly, they propose to decipher queries before the last broadcast. This modification has two advantages: (i) decrease the broadcast time as messages are shorter and (ii) users do not need to exchange their shares. Secondly, as broadcasting all query answers produces the most significant overhead, Romero-Tris et al. remove this step. Instead, all users query the search engines with all queries (including her own query). These two modifications reduce the delay from 6.8 seconds for UUP to 3.2 seconds for the new protocol.

Moreover, Lindell et al. [63] identify that UUP is not resistant to malicious adversaries. The shuffling algorithm proposed by UUP does not ensure that the n users correctly follow the protocol. For that reason, Lindell et al. introduce a verification step after the shuffling protocol to ensure that no user cheats during the shuffling stage. The verification step proves that all users involved in the protocol shuffle queries correctly and do not modify

messages. Then, when it is established that no malicious user takes part in the exchange, queries are deciphered and sent to the search engine. Due to a higher number of cryptographic operations, authors claim that their protocol is twice as expensive as UUP. An empirical evaluation conducted by Romero-Tris et al. [64] indicates that, for three users, the protocol is about seven times more expensive than UPP. Knowing that UUP introduces a delay of 6.8 seconds [62], the solution proposed by Lindell et al. seems impractical.

Due to this reason, Romero-Tris et al. propose a new protocol with similar guarantees [64]. For better efficiency, they create a new shuffling algorithm based on Optimize Arbitrary Size (OAS) Benes network [65]. It enables the permutation of n inputs with 2×2 switches (2 inputs and 2 outputs). To have a multi-party version of the OAS Benes network, Romero-Tris et al. use t OAS Benes network, where t depends on the minimum number of honest users in the system. They also improve the verification step by using zero-knowledge proof protocols based on ElGamal [66] (i.e, PEP [67] and DISPEP [67]). By improving the shuffling step and the verification step, Romero-Tris et al. decrease the number of messages sent on average (i.e., $4n - 4$ messages for their proposal and $4n - 2 - 2/n$ for Lindell et al. [63], where n is the number of users). They also analyze the computation time of their protocol and note that it is about two times lower than then one obtained with [63] (considering either three, four, or five users in the system).

A recent approach [68] points out that the solution proposed by Lindell et al. [63] suffers from a malicious user. They claim that malicious users are able to modify the query sent to search engines. To fix the protocol against such misbehaving users, they propose (similarly to [62]) that users send the n queries to the search engine (instead of sending only one query). Sending all queries does not disclose more information but increases dramatically the computational time and the number of messages sent on the network.

Furthermore, Lindell et al. [63] detects if a malicious user does not execute the protocol correctly, but it does not identify which user is the malicious one. Another solution called Dissent [40, 41] increases the user protection by identifying the malicious user. Compared to Lindell et al. [63], Dissent implements another shuffling algorithm. Each user encrypts her query q_i with all members' secondary public keys $z_1, ..., z_n$ (the ciphered queries are denoted C'_i). Users store their own C'_i for later use, and encrypt C'_i with all members' primary public keys $y_1, ..., y_n$ (the cipher text is denoted by C_i). Then, each user sends C_i to the first member. One after another, users permute the order of $C_1, ..., C_n$, re-mask queries by deciphering cipher texts with their private key, and forward the message to the next user. Finally, the last user obtains a permutation of $C'_0, ..., C'_n$ and broadcasts it to all users. Users can verify the correctness of the protocol by identifying that her own C'_i is included in the results. If the protocol is executed correctly, users broadcast their secondary private key such that they have *in fine* all keys to decipher all C'_i and retrieve the queries q_i. Otherwise, users start a blame phase identifying the malicious user. They reveal their own C'_i allowing each user to replay the protocol. By comparing, the messages effectively sent and the ones replayed, users are able to point out which user misbehaves. However, it was shown by [42] that Dissent does not scale when the number of users increases. According to this paper, the node throughput drops down to (almost) zero when the number of nodes is higher than 50.

In addition, the work presented in [69] models a situation in which a user gets part of her queries submitted on her behalf by other users and the remaining part is sent by herself to the search engine. The authors illustrate with a mathematical study how users can mutually benefit from exchanging

their queries. They model user behavior by a probability distribution and assess the privacy leakage with the Shannon entropy [70].

2.1.2 Indistinguishability Mechanisms

With indistinguishability solutions, search engines are only able to collect inaccurate users queries and interests. Consequently, as the users' interests cannot be truly discovered, the user privacy is preserved. To distort user profiles, most of these solutions send fake queries (in addition to users' queries) but other approaches rely on specific techniques: splitting the user profile, not sending the original query, etc.

2.1.2.1 Indistinguishability Based on Fake Queries

The main challenge for indistinguishability solutions based on fake queries is to generate plausible and realistic fake queries. Indeed, fake queries must not be distinguishable from real ones (otherwise the protection is broken). Most solutions ensured this by generating fake queries close to users' interests. In that case, by computing a user profile, an adversary has only access to generic information about the user but she cannot infer their specific topics of interest. PRAW [71, 72, 73] is a tool that generates dummy Web transactions to confuse eavesdroppers about the real user's interests. In the context of Web search, these dummy transactions are fake queries sent to the search engines. PRAW sends on average T_r fake queries for each real user query. These fake queries are constructed from two different sources: an internal user profile (describing exact users' interests) and an internal database of terms (containing terms related to the general interests of the user). These resources are collected from the Web pages accessed by users. Furthermore, a similar approach called Plausibly Deniable Search (PDS) [74] differs in the generation of fake queries. PDS aims at generating fake queries which could have been sent by the user on her regular basis. As a consequence, PDS allows the user to deny to any organizations having one of her queries sent as she can pretend that one of the fake queries was her original query and thus claim that the problematic query was only a fake query generated by PDS. The PDS fake queries are generated from a large collection of Web pages. It exploits the relation between the terms contained in Web pages to generate realistic fake queries. Another recent approach presented by Viejo et al. [75, 76] generates fake queries based on a user profile extracted from social networks. They argue that user profiles extracted from social networks are more accurate than constructing them from Web pages accessed by the user. Consequently, fake queries derived from these user profiles are more realistic.

GooPIR [45] (Google Private Information Retrieval) is a Java program to query Google in a privacy-preserving way. This protection mechanism can also be used with other search engines but only Google is supported by the application. GooPIR generates k fake queries for each user query but, contrary to PRAW, GooPIR introduces fake queries inside real user queries. Specifically, it generates a new query composed of the initial query and the fake queries; all these queries are separated by the logical OR operator. Fake queries are generated by randomly selected keywords in a dictionary. Any type of dictionary can be exploited. In the current implementation, the dictionary is computed using news articles from WikiNews [77], but GooPIR's authors mention in [45] that query logs can also be used. In practice, to generate fake queries, GooPIR selects first, for each keyword contained in the initial query, k words that have a similar usage frequency. Then, from the $k \times n$

words selected keywords (considering that the initial query is composed of n keywords), GooPIR creates k fake queries of n words. By default, GooPIR generates three fake queries (i.e., $k = 3$) but users can manually modify this value. Finally, as Google returns documents related to the initial query and the fake ones, GooPIR implements a filtering step to remove the irrelevant results introduced by fake queries. The algorithm tests for each result if its title or its description contains all keywords of the initial query. If so, the result is displayed, otherwise it is discarded.

TrackMeNot [78, 46] (from now on referred to as TMN) is a Firefox plugin which periodically generates fake queries to hide user queries in a stream of related queries. Fake queries are sent independently of real user queries. TMN aims, with the use of fake queries, to ensure that user profiles created by the search engine do not reflect users' real interests. After the installation of TMN, users can define different settings according to their desired level of protection. Two main parameters impact the user protection: the RSS feed lists and the delay between two fake queries. The RSS feeds list is composed by default of four RSS feeds coming from: *cnn.com*, *nytimes.com*, *msnbc.com* and *theregister.co.uk*. Users can modify this list to remove or add extra RSS feeds. Modifying this setting is crucial, as keeping the initial list might help an adversary to distinguish between real queries and fake ones. However, it is not trivial for users to find good RSS feeds, as they should find RSS feeds that cover all their ever-changing interests. Moreover, TMN gives users the possibility to customize the protection by choosing the time between two fake queries. They can choose between several possibilities: from 10 fake queries per minute to 1 fake query per hour. Consequently, TMN allows users to decide the quantity of noise they want to introduce in their user profile created by the search engine. Also, users have the possibility to activate the "burst mode". In that case, when a real query is issued, TMN sends at the same time multiple fake queries to cover it. To generate fake queries, TMN transforms titles of articles listed in RSS feeds into queries. To do so, it randomly selects a title and extracts its keywords. Then, from all available keywords, it randomly selects between one and six keywords to form the fake query. The randomness prevents TMN from generating deterministic fake queries. A given title can produce several different fake queries. Consequently, two TMN users using the same RSS feeds do not in general create the same fake queries.

More recently, Viejo et al. [79, 80] address the trade-off between privacy and the accuracy of the results. They argue that as GooPIR or TMN fake queries do not share the same interests as the original queries, these two solutions decrease the accuracy of the results. They propose to generate semantically-related fake queries in such a way that they can control the distance between fake interests and real ones. Their solution works as follows: (i) it assigns to each term contained in the query a predefined category using an ODP taxonomy [81] or Wordnet [82]; (ii) for each category, it selects related categories in the taxonomy (called fake categories); (iii) for each fake category, it retrieves fake terms; (iv) by combining fake terms, it generates fake queries. Nevertheless, the authors do not show that their approach achieves better results than GooPIR or TMN. They only validate the fact that, if search engines extract their user profiles using the same ODP taxonomy, their approach preserves about 57% of categories in the user profile. Moreover, the proposed solution partially protects users, as it only generalizes users' interests. For instance, instead of knowing that a user practices sailing, the search engine knows that the user is interested in water sports. As a result, the solution does not give any guarantee about the user protection. For in-

stance, in the case of a disease, the solution hides the name of the disease but does not prevent search engines from knowing that the user is infected by a disease.

Other works [83, 84] focus on a theoretical analysis of fake queries. They define criteria that fake queries should follow to protect users correctly in their interactions with search engines. For instance, Optimized Query Forgery (OQF) [83] suggests that fake queries should be generated such that the Kullback-Leibler divergence between the user keyword distribution and the population keyword distribution is low. Nevertheless, OQF mathematically solves the problem but does not indicate in practice how to generate these fake queries. Similarly, Noise Injection for Search Privacy Protection (NISSP) [84] conducts a theoretical study on noise injections. They obtain criteria for generating the optimal noise. But the authors do not mention if such noise is obtainable in practice with real data.

2.1.2.2 Indistinguishability Based on Query Transformation

In this section, we address approaches that do not rely on fake queries. To obfuscate user profiles, they all use different techniques. For instance, Query Scrambler (QS) [85, 86] protects users by not sending their queries but, rather, by issuing similar queries. More precisely, for each user query, it generates a set of related queries by generalizing the concepts used in the initial query and then, by merging and filtering all the results obtained with these related queries, QS retrieves the potential results for the initial query. However, this solution suffers from a low accuracy as there is a low overlap between the results obtained with the related queries and the ones obtained with the initial query.

Another approach, Dispa [87, 88, 89], tries to find a trade-off between user privacy and user profile exploitation. It assumes that search engines identify users with a cookie. Therefore by generating multiple cookies for a single user, Dispa creates different user profiles on the search engine. The search engine does not know that these different user profiles refer to the same user. Besides, Dispa uses the cookies such that each user profile contains a specific trend of user interest. As a result, Dispa reduces the information disclosure without introducing noise in user profiles. Nevertheless, this approach relies on the strong assumption that search engines identify users with their cookies; other elements such as the IP address, or HTTP headers are not considered.

2.1.2.3 Indistinguishability Based on User Profile Obfuscation

To obtain personalized results while keeping the user's privacy protected, solutions [90, 91] have been proposed to send an obfuscated user profile with the query. The exploitation of these user profiles allows search engines to personalize the results. Nevertheless, these solutions require cooperative search engines as they need to modify their personalization algorithm to accept obfuscated user profiles.

In [90], the authors suggest constructing a hierarchical user profile from all users' personal documents (e.g., browsing history, emails). The user profile contains the frequent terms used in these documents, as they indicate a possible user interest. The construction of a hierarchical user profile is based on two metrics: one to detect similar terms and another one to find parent-child terms. Then, the solution protects the user privacy by removing specific domains from the user profile (i.e., the bottom of the hierarchy). This generic

user profile contains enough data to personalize the results but not enough to compromise the user privacy.

In UPS [91], contrary to the previous solution, they generalize the user profile according to the query. Indeed, topics non-related to the user query are not relevant to personalize the results; they only disclose more user interests. Besides, as the user profile is based on an existing hierarchy (e.g., ODP, Wikipedia), UPS ensures that an adversary cannot infer topics that have been removed from the hierarchy. For instance, considering a given node in the hierarchy, depending on the number of its children, it is possible to infer a more specific interest. UPS uses such information to generalize the user profile.

2.1.3 Private Information Retrieval

Private Information Retrieval (PIR) could be used by search engines to offer a service that enables a privacy protection by design (e.g., [31, 32]). Under such protocols, users access information stored on the distant server without revealing to the search engine what information they access. The only information known by the search engine is that the user has sent a query. In general, PIR protocols consist in three algorithms: constructing protected queries (keywords are at least encrypted), performing the information retrieval (in such a way that the search engine has no access to the query and the results), and finally reconstructing the result list on the client side.

For instance, Pang et al. [31] present a system that protects queries by decomposing them into multiple buckets of words. These buckets are then ciphered with a homomorphic encryption scheme. Encrypted buckets are not a problem for the search engine as it uses the homomorphic properties[1] to manipulate initial data. Therefore, with the homomorphic properties, the search engine gives a relevance score to each document indexed in its database. Relevance scores are not accessible by the search engine as they are still ciphered with the homomorphic encryption scheme. Finally, the search engine returns the list of relevance scores to the user. The user deciphers this list and identifies the documents with the highest relevance scores.

PIR schemes ensure a privacy-protection by design: no third party can access the content of the query. Nevertheless, in practice, such protocols suffer from many limitations. Homomorphic encryption schemes are costly [33]. They engender extra computation time on the search engines (due to the ciphered buckets). Besides, as the plain text is not accessible by the search engine, several information retrieval techniques cannot be applied (e.g., query expansion). Finally, such schemes require specific recommendation algorithms on the search engine. But, as search engines are interested in personal data, they have no interest in deploying such protocols.

2.1.4 Private Web Search Metrics

Private Web search solutions contain approaches that rely on a large number of methods (e.g., anonymous network, group exchange, fake queries). This diversity of solutions made researchers evaluate their work differently. In this section, we introduce the metrics used to quantify the user protection (privacy). We also review metrics to estimate their cost (performance) and the quality of the results they return (accuracy).

[1] Example of a homomorphic property: the product of two cipher texts $E(x)$ and $E(y)$ is equal to ciphering the product of the two plain texts $E(x \times y)$, where x and y are the plain texts, and $E(.)$ the ciphering function.

2.1.4.1 Privacy Protection Metrics

To obtain a measure about the user privacy protection, existing solutions protect real search queries with their solution and try to retrieve the initial information. They consider whether a static adversary (i.e., comparing the data collected by the adversary with and without protection) or a dynamic adversary (i.e., running privacy attacks to bypass the protection). Furthermore, other approaches perform a theoretical evaluation to show that their protocols have specific guarantees.

- **Privacy Quantification** — Privacy quantification methods consider an adversary that collects user queries and quantify the knowledge that the adversary has obtained if a protection mechanism is used or not. Several methods have been proposed in the literature:
  - **Search engines** — TrackMeNot [46] uses Yahoo! to quantify the data leakage. Contrary to the other well-known search engines, Yahoo! makes public the lists of interests it infers from the user queries. As a result, the evaluation of TrackMeNot studies how the list of interests evolves with the introduction of fake queries.
  - **Entropy** — Another metric uses the entropy to quantify the user privacy. Indeed, the entropy [70] estimates the quantity of information represented by a discrete random variable. Applied to our context, the entropy measures the quantity of information an adversary has on a given user. The discrete random variable models the user. For instance, it takes the set of keywords used by the user as the sample space. If we denote by X the random variable with the keywords $\{x_1, ..., x_n\}$, the entropy $H(X)$ is expressed as follows:

    $$H(X) = E[-\log_2(p(X))] = -\sum_{i=1}^{n} p(x_i) \cdot \log_2 p(x_i), \quad (2)$$

    where p is the probability mass function of the discrete random variable X (i.e., the probability that a given keyword was used by the user).
  - **Profile Exposure Level** — The entropy measures the knowledge that an adversary has obtained on a user. Nevertheless, this metric needs to be compared with the one obtained without protection. For that reason, Erola et al. [59, 60] define the Profile Exposure Level (PEL). If we consider two random variables: X representing the queries which are originally sent by the user (i.e., without any protection mechanism) and Y representing the queries issued with the protection mechanism, the Profile Exposure Level equals the normalized version of the mutual information between X and Y. More precisely, considering $H(X)$ the entropy of the original set and $I(X, Y)$ the mutual information between X and Y, the PEL is defined by:

    $$PEL = \frac{I(X, Y)}{H(X)} \cdot 100.$$

    This metric measures the percentage of user information that is exposed when Y is disclosed.
  - **Degree of anonymity** — The entropy is also used in [92] to define the degree of anonymity provided by an unlinkability solution. It

quantifies the ability of an adversary to link the query back to its requester by an attack. Let X be the discrete random variable representing the potential requesters in the system. The entropy of X is expressed by Equation 2 where $p(x_i)$ represents the probability that the user x_i is the original requester. We denote by H_M the maximum entropy of the system:

$$H_M = \log_2(n),$$

where n is the number of users in the system. Finally, we define the degree of anonymity d as follows:

$$d = 1 - \frac{H_M - H(X)}{H_M} = \frac{H(X)}{H_M}.$$

In other words, the degree of anonymity considers the quantity of information the attacker has learned by collecting queries (i.e., $H_M - H(X)$) normalized by the maximum entropy H_M. This metric d varies between zero and one: a value of 0 means that one user is identified as the initial requester with a probability one while a value of 1 means that all users are identified as the potential requester with the probability $1/n$.

- **Privacy Attacks** — The literature contains several attacks against private Web search solutions. We can distinguish between active attacks [93, 94, 95] and passive attacks with machine learning algorithms [47, 48, 96, 49]. A first type of active attacks is a timing attack. As detailed in [93, 96], measuring the time taken by users to access a particular page discloses information. Indeed, most browsers implement a caching system to reduce the bandwidth consumption. Nevertheless, the caching system also reduces the time needed for users to access information. Consequently, according to the access time, an adversary deduces if the user has already accessed such information. By exploiting this, an adversary can identify several interactions made by a given user. It is also possible to de-anonymize an interaction if a user accesses a service with and without anonymizing her queries. Another type of attacks [95] observes changes in traffic patterns to de-anonymized queries. They show that they can follow messages through anonymous networks by causing congestion in the network. If they follow messages from its sender to its destination, the anonymity provided by the network is broken.

  On the other hand, machine learning attacks are also used to bypass privacy-preserving solutions. These techniques use either supervised machine learning (e.g., Support Vector Machine [97], Logistic Regression [98]), or clustering algorithms (e.g., k-means [99]).

  - **Support Vector Machine** [97] — For a binary classification, a support vector machine finds the hyperplane that separates the two classes with the largest distance between the nearest data point of the two classes. More formally, if we denote by $\vec{w} \cdot \vec{x} - b = 0$ the equation of the hyperplane, where $\vec{w}$ is the normal vector and $b/||\vec{w}||$ the distance to the origin, the problem consists in finding $\vec{w}$ and b that minimize

    $$\frac{1}{2}||\vec{w}||^2,$$

subject to the constraint

$$y_i(\vec{w} \cdot \vec{x_i} - b) \geq 1, \quad \forall i \in [\![1, n]\!],$$

where n is the number of points and $\vec{x_i}$ is a data point belonging to the class y_i (either 1 or -1). Nevertheless, as the two classes might not be linearly separable, Vapnik and Cortes imagine the concept of *soft margin* [97]. They introduce slack variables ξ_i to tolerate errors in the training classification. The optimization problem is modified as follows: minimize

$$\frac{1}{2}||\vec{w}||^2 + C\sum_i \xi_i,$$

subject to the constraints

$$y_i(\vec{w} \cdot \vec{x_i} - b) \geq 1 - \xi_i \text{ and } \xi_i > 0, \quad \forall i \in [\![1, n]\!],$$

where C is a constant to control the trade-off between minimizing the number of errors in the training set and maximizing the margin. By combining multiple binary SVM classifiers, it is possible to obtain a multi-class SVM classifier. It has been shown in [100] that the one-against-one method [101] is more suitable for practical use than other aggregation methods (e.g., one-against-all method [101], DAGSVM [102]).

– **Logistic Regression** [98] — For a binary classification problem, logistic regression models the probability that an observation $\vec{x}$ belongs to the positive class $\mathscr{C}_+$:

$$Pr(Y = \mathscr{C}_+ | X = \vec{x}) \quad \text{(also denoted by } p(\mathscr{C}_+|\vec{x})),$$

where $Y = \{\mathscr{C}_+, \mathscr{C}_-\}$ is the binary output variable (i.e., the predicted label), and X is the input variable (i.e., the vector of features). The simplest model defines $p(\mathscr{C}_+|\vec{x})$ as a linear function of the covariates $\vec{x}$. Nevertheless, as a linear function is not bounded between zero and one, it is not possible to model such a probability with a linear function of $\vec{x}$. To remove the range restrictions, the logistic regression operates on the logistic function of $p(\mathscr{C}_+|\vec{x})$:

$$logit\big(p(\mathscr{C}_+|\vec{x})\big) = \log\Big(\frac{p(\mathscr{C}_+|\vec{x})}{1 - p(\mathscr{C}_+|\vec{x})}\Big).$$

The logistic function maps probabilities from a range of $]0, 1[$ to the full range of real numbers. Then, by modeling the logistic function of $p(\mathscr{C}_+|\vec{x})$ as a linear function of the feature vector $\vec{x}$, we obtain:

$$\log\Big(\frac{p(\mathscr{C}_+|\vec{x})}{1 - p(\mathscr{C}_+|\vec{x})}\Big) = \theta_0 + \vec{x} \cdot \vec{\theta},$$

where $\vec{\theta}$ is the regression coefficients and θ_0 is the error term. As a result, the probability $p(\mathscr{C}_+|\vec{x})$ is expressed by:

$$p(\mathscr{C}_+|\vec{x}) = \frac{1}{1 + e^{-(\theta_0 + \vec{x} \cdot \vec{\theta})}}.$$

The decision rule can be determined with a threshold of 0.5. If $p(\mathscr{C}_+|\vec{x}) \geq 0.5$, $\vec{x}$ belongs to the positive class. Otherwise, it belongs to the negative class. The algorithm can be extended to a

multi-class problem with the multinomial logistic regression [103]. Considering that the output variable Y has n possible outcomes (instead of two), a multinomial logistic regression runs $n-1$ independent binary logistic regression models (one outcome is chosen as a "pivot").

– **k-Means** [99] — The k-means clustering algorithm attempts to split n observations into a fixed number k of clusters (each observation belongs to the cluster with the nearest mean). Specifically, given a set of observations $(\vec{x_1}, ..., \vec{x_n})$, k-means clustering splits this data in k sets $S_1, ..., S_k$ such that it minimizes the within-cluster sum squares. In other words, minimize over all k sets $S_1, ..., S_k$ the quantity:

$$\sum_{i=1}^{k} \sum_{\vec{x} \in S_i} ||\vec{x} - \vec{\mu_i}||^2,$$

where $\vec{\mu_i}$ is the mean of the observations contained in the set S_i. In practice, a common method to initialize the clusters (and define each $\vec{\mu_i}$) is the Forgy method [104]. It randomly chooses k observation from the n observations and uses them as initial means. Then, the remaining observations are associated with their nearest mean. The algorithm computes the k new means, as the centroid of each of the k clusters. Finally, the two previous steps are repeated (i.e., assignment of the observations and means calculation) until a convergence is reached.

Attacks based on supervised machine learning algorithms [47, 48] aim to (i) identify the issuer of an anonymous query or (ii) retrieve the initial query from an obfuscated query. In the first case, they consider a system with n users and want to determine in which class $\mathscr{C}_1, ..., \mathscr{C}_n$ an anonymous query q belongs. In the second problem, the attacks consider two classes, one for real queries and one for fake queries. They want to determine in which class the query q belongs (in other words, if the query q is a real query or a fake one). Supervised machine learning algorithms work in two phases. They first learn a model from prior knowledge and then classify data according to this model. In our context, the learning phase extracts from existing queries discriminant criteria to characterize users or fake queries. Then, the attack applies these criteria on queries to retrieve their requester (in the case of anonymous queries) or retrieve the real queries (in the case of indistinguishability mechanisms).

Similarly, clustering algorithms [48, 96, 49] distinguish between real queries and fake ones. Nevertheless, they do not require any prior knowledge on fake queries. Considering a set of queries $q_1, ..., q_n$, the clustering algorithm splits queries in two clusters: one for fake queries and one for real ones.

The success of these attacks is measured by the precision and the recall (two classical metrics for information retrieval problems). The computation of these measures can be done using two averaging operations [105]: *macro-averaging* and *micro-averaging*. The former computes the precision and the recall per class and then aggregates values by computing their means while the latter computes the precision and the recall without distinguishing between classes. For instance, if we consider an attack against anonymous solutions, the precision and the recall for the

macro-averaging are defined by:

$$\text{precision}_1 = \frac{1}{|U|}\sum_{u\in U}\frac{TP_u}{TP_u + FP_u},$$
$$\text{recall}_1 = \frac{1}{|U|}\sum_{u\in U}\frac{TP_u}{TP_u + FN_u},$$

where U is the set of users in the system, TP_u is the number of true positives for the user u (i.e., the number of queries issued by user u and successfully retrieved by the adversary), FP_u is the number of false positives for the user u (i.e., the number of queries issued by a user v but considered by the attack as sent by $u \neq v$), and FN_u is the number of false negative (i.e., the number of queries sent by u but considered by the attack as sent by $v \neq u$). On the other hand, the precision and the recall for the micro-averaging are defined by:

$$\text{precision}_2 = \frac{\sum_{u\in U} TP_u}{\sum_{u\in U} TP_u + FP_u},$$
$$\text{recall}_2 = \frac{\sum_{u\in U} TP_u}{\sum_{u\in U} TP_u + FN_u}.$$

We note that the micro-averaging approach gives the same weight to all queries while the macro-averaging approach balances the weight according to the quantity of queries sent by each user. For our problem, the macro averaging metrics make more sense, as we are studying the user privacy protection individually. Besides, in the case of an unbalanced number of queries, the micro averaging metrics reflect only the protection of the most active users. Due to these reasons, approaches in the literature also use the macro-averaging metrics (e.g., [47, 49]).

- **Theoretical Analysis** — Some private Web search solutions aim to guarantee theoretical properties (e.g., unlinkability between the requester and her queries). As presented by Guha et al. [106], these properties can be theoretically proved by a protocol verifier based on the Dolev-Yao model (e.g., ProVerif [107], Avispa [108]). These verifiers take as input (i) the protocol transposed in a domain specific language or π-calculus (a mathematical formalism) and (ii) the information that an adversary wants to access. For instance, in anonymous networks, an adversary wants to link a query with its requester or more generally, in a cryptographic protocol, an adversary wants to access the plain text contained in a cipher text. To perform such a verification, the verifier derives all possible states in the protocol (e.g., by forging new messages or replaying existing ones) and ensures that, in all possible states, the adversary cannot obtain the information she is interested in. The major drawback of these verifiers is the transcription of the protocol in a domain specific language, any difference with its real implementation might hide a potential flaw.

2.1.4.2 Performance Criteria

Protecting queries engenders an overhead for the entities involved in the protocol. Firstly, on the client's computer, the generation of fake queries or the cryptographic primitives have a non-negligible cost. Secondly, on search engines, fake queries produce extra computation, as they have to compute and

return results for these fake queries. Lastly, some private web search solutions require the use of extra entities (e.g., in anonymous protocols). Therefore, the multiple query forwardings impact the latency. Consequently, an evaluation of the performance considers the impact that private Web search solutions have on these entities. Their efficiency can be measured with the following metrics:

- **Network traffic** — Using extra relays or sending fake queries impacts the network. There are more messages sent over the network or the packet size is greater. A basic evaluation of the performance compares the number of extra messages exchanged or the extra packet size sent.
- **Latency** [62] — The latency is the time a user needs to wait for her answer. More formally, it corresponds to the time between the generation of the query and the reception of the results. A comparison between the latency with and without protection gives an idea about how the protection impacts the user experience.
- **Throughput** [109] — The throughput is the number of queries the search engine or the private solution is able to deal with per second. This measure is bounded as all systems are limited either by the network or the computing power. If the maximum throughput is reached, new queries are put on hold and thus the latency increases (more times to get the results). A low maximum throughput makes the solution impractical as few users can use the solution at the same time.
- **CPU cycles** [110] — The CPU cycles are the number of instructions a processor executes to complete the execution of an algorithm. By comparing the number of cycles required for sending queries with and without using a protection mechanism, this metric measures precisely the extra computation introduced by the protection mechanism.
- **Energy consumption** [111] — By mesuring the energy consumption required to create, send and process queries, it is possible to quantify the cost due to private Web search mechanisms.

2.1.4.3 Quality of the Results

The results returned through a private Web search solution differ from the ones returned for a non-protected query. Firstly, to protect the user privacy, most private solutions introduce noise in users' queries and thus two different queries produce two different sets of results. Secondly, private Web search solutions are not compatible with personalization, as their use prevents search engines from collecting personal data about users.

To assess the quality of the results obtained with a private Web search solution, the evaluation compares the results obtained with a non-protected query and the ones obtained with a query issued through a private Web search solution. We denote by S_1 the original results, by S_2 the results obtained with the private Web search solution and by $\text{rank}_L(r)$ the rank of a result r in a set L. Ideally, the two sets of results S_1 and S_2 should contain the same results in the same order, indicating that the private solution does not degrade the quality of the results. As it is not the case generally, we present several metrics to precisely compare S_1 and S_2:

- **Proportion of results in S_1 contained in S_2** — GooPIR [45] evaluates the accuracy with QM_1. This metric assesses the proportion of results

retrieved with the original query (S_1) that are contained in the results retrieved with the obfuscated queries (S_2):

$$QM_1 = \frac{|S_1 \cap S_2|}{|S_1|}$$

This metric can be seen as a recall. If we define all results in S_1 as relevant results, QM_1 is the proportion of relevant results retrieved in S_2. GooPIR asks search engines to return 25 results for each query (i.e., $|S_1| = |S_2| = 25$) but QM_1 can be computed with a different number of results.

- **Spearman's Footrule** — Spearman's Footrule [112] computes the average displacement of all results between S_1 and S_2. It is defined by:

$$F = \frac{\sum_{r \in S_1 \cap S_2} |\text{rank}_{S_1}(r) - \text{rank}_{S_2}(r)|}{|S_1 \cap S_2|}$$

- **Kendall's Tau** — Kendall's Tau [113] is a metric to evaluate the correlation between two rankings. It counts the total number of inversions between the two rankings. More formally, Kendall's Tau τ is defined by:

$$\tau = \sum_{\substack{(r_1,r_2) \in (S_1 \cap S_2)^2 \\ \text{rank}_{S_1}(r_1) < \text{rank}_{S_1}(r_2)}} \mathbb{1}_{\text{rank}_{S_2}(r_1) > \text{rank}_{S_1}(r_2)}$$

where $\mathbb{1}_X$ is equal to 1 if the condition X is true. Otherwise, it equals 0.

- **Normalized discounted cumulative gain (NDCG)** — NDCG [114] evaluates the usefulness of the ranking based on the graded relevance of the results. The $NDCG$ is defined by:

$$NDCG = \frac{DCG}{IDCG},$$

where $IDCG$ is the ideal discounted cumulative gain (i.e., the DCG obtained with the results contained in S_1) and DCG is the discounted cumulative gain defined as:

$$DCG = \text{rel}(r_1) + \sum_{i=2}^{|S_2|} \frac{\text{rel}(r_i)}{\log_2(i)} \quad \text{where} \quad r_i \in S_2,$$

where $\text{rel}(r)$ is the relevance value assigned to each result r in S_1. For instance, if S_1 contains 15 results, we can define the relevance of these results by assigning to the first five results the relevance of 3; to the five following results the relevance of 2; and to the remaining five results the relevance of 1. Otherwise, the results have the relevance of 0. More formally, the function rel presented in this example is defined by:

$$rel(r) = \begin{cases} 3 & r \in S_1 \text{ and } \text{rank}_{S_1}(r) \in [\![1, 5]\!] \\ 2 & r \in S_1 \text{ and } \text{rank}_{S_1}(r) \in [\![6, 10]\!] \\ 1 & r \in S_1 \text{ and } \text{rank}_{S_1}(r) \in [\![11, 15]\!] \\ 0 & \text{otherwise} \end{cases}$$

- **Mean Average Precision (MAP)** — MAP expresses the mean over multiple queries of the average precision. For one query, we define AP the average precision by:

$$AP = \frac{1}{|S_1|} \times \sum_{r \in S_1 \cap S_2} P(r),$$

where $P(r)$ is the precision at cut-off r in the set S_2. In other words, P(r) is the number of relevant results contained in the first $\text{rank}_{S_2}(r)$ results in S_2 divided by $\text{rank}_{S_2}(r)$. If we consider that $S_1 = \{a, b, c\}$ and $S_2 = \{b, c, d\}$ where a, b, c, d are results, the average precision is:

$$AP = \frac{1}{3} \times [\frac{\overbrace{1}^{b \in S_1}}{\underbrace{1}_{\text{rank}_{S_2}(b)}} + \frac{\overbrace{2}^{(b,c) \in S_1 \times S_1}}{\underbrace{2}_{\text{rank}_{S_2}(c)}}] = \frac{2}{3}.$$

If we consider now that $S_2 = \{b, d, c\}$, the average precision is:

$$AP = \frac{1}{3} \times [\frac{1}{1} + \frac{2}{3}] = \frac{5}{9}.$$

2.1.5 Discussion

In this section, we summarize the aforementioned privacy-preserving solutions and compare them according to our objectives. We first recall the features they provide. Then, we summarize their required hypotheses and precise the criteria we use to compare these solutions (the retained criteria are based on the metrics previously presented). Finally, considering their features, hypotheses and criteria, we draw a comparison between all these solutions.

2.1.5.1 Feature Provided

The unlinkability solutions (i.e., solutions that hide the user identity by sending queries through multiple nodes) give different level of protection. The basic one is the *unlinkability* between the sender and her message. Several protocols (e.g., [56, 42]) guarantee a further protection: *sender anonymity*, and *receiver anonymity*. The former property ensures that the sender cannot be distinguished among all users in the system, while the latter ensures that the receiver cannot be determined. Other unlinkability solution (e.g., [46, 45]) generate *fake queries* to introduce some noise in the user profile created by the search engine or make the user query indistinguishable among a set of queries. We characterize each of these solutions according to the sources from which they generate fake queries: Web Pages (W), Social Networks (SN), Dictionary (D), RSS Feeds (R) and ODP Taxonomy / WordNet (O). Lastly, we distinguish another type of indistinguishability solutions. Such solutions transform the user query: sending similar queries, modifying her headers, etc. We denote these solutions by *query transformation*.

2.1.5.2 Required Hypotheses

Private Web search solutions have made several considerations for the deployment of their solution. Some of them consider first that the nodes employed in their protocol do not collude. For instance, unlinkability solutions send queries through multiple nodes. If all the nodes in the path collude, the unlinkability protection is bypassed as the nodes can associate the query with

its requester. The *non-collusion* assumption ensures that such a misbehavior is not permitted. Nevertheless, this hypothesis does not prevent nodes from free-riding. They might not be interesting in following the protocol (e.g., not forwarding queries to the next node). Some protocols implement mechanisms to detect free-riders and exclude them from their protocol. Otherwise, the remaining protocols assume that *no free-riders* are deployed in their protocol. Nevertheless, these hypotheses do not protect against a node that modifies the user query. As mentioned in [68], such a guarantee is obtained only if the user sends her queries directly to the search engines. Finally, privacy attacks [47, 48, 49] have shown that an adversary that has previously-collected queries can bypass protection mechanisms. Most of these protections do not consider such an attack. Therefore, they assume that the adversary was not able to collect *prior knowledge* on users.

Table 1

A summary of metrics to assess the privacy protection, the performance and the accuracy of private Web search techniques.

	Metrics	Range	Description
Privacy	Entropy [70]	$[0, 1]$	Quantity of information contained in a user profile.
	Profile Exposure Level [60]	$[0, 100]$	Percentage of user information disclosed.
	Degree of anonymity [92]	$[0, 1]$	Quantify if an adversary is able to identify the requester of an anonymous query.
	Recall & precision of privacy attacks [47, 48, 49]	$[0, 100]$	Attacks that try to bypass protected queries and retrieve its requester or its initial content.
	Theoretical Analysis [107, 108]	$\{0, 1\}$	Prover that indicates if there is a flaw in a protocol.
Performance	Network traffic	$[0, \infty[$	Number of messages sent on the network.
	Latency [62]	$[0, \infty[$	Time needed to receive the results from the search engine.
	Throughput [109]	$[0, \infty[$	Number of queries sent per second (on average).
	CPU cycles [110]	$[0, \infty[$	Number of instructions taken by the processor to send a query.
	Energy consumption [111]	$[0, \infty[$	Energy uses to send queries to a search engine
Accuracy	GooPIR (QM_1) [45]	$[0, 1]$	Proportion of results retrieved without protection that are also retrieved with a protection mechanism.
	Spearman's Footrule [112]	$[0, n/2]$	Compute the average displacement between two rankings.
	Kendall's Tau [113]	$[0, n]$	Evaluate the correlation between two rankings.
	NDCG [114]	$[0, 1]$	Evaluate the usefulness of a ranking.
	Mean Average Precision	$[0, 1]$	Evaluate the quality of a ranking.

2.1.5.3 Evaluation Criteria

Private Web search solutions have to protect users without significantly impacting the performance and the accuracy of the results. As we have seen in Section 2.1.4, private Web search solutions have been assessed with a large variety of metrics. We summarize them in Table 1. As we can notice these metrics complement each other. For that reason, no unique metric has been used in the literature to assess private Web search solutions. In addition, no existing evaluation in the literature makes a comparison between the aforementioned private Web search solutions. To overcome this issue, we establish a qualitative comparison between solutions. This comparison is performed according to the following criteria:

- **Privacy** — As previously mentioned, we consider a system in which the adversary has been able to collect user queries (i.e., before using any protection mechanism, users sent their queries in clear texts). Most solutions in the literature (e.g., [46, 45, 39, 62]) do not consider such hypothesis. Therefore, by considering an adversary with prior knowledge on users, privacy attacks [47, 48, 49] show disparities in the privacy protection. Our qualitative comparison tries to emphasis this disparity. Furthermore, privacy guarantees offered by private solutions differ. For instance, some solutions tolerate free-riders while others do not. Our qualitative evaluation considers such different levels of guarantees.

- **Performance** — The performance established for private solutions are usually not suitable for comparison. The authors evaluate their solutions on different architecture with different settings (e.g., number of users). To establish a comparison in terms of performance, we exploit how the authors position their papers with each other.

- **Accuracy** — Few solutions evaluate the impact that protecting queries has on the quality of the results returned to the user. The accuracy of the results is difficult to assess as it is search engine dependent. Search engines implement their own algorithm to exploit user data, and therefore, from one implementation to another, the quality of the results obtained with the private Web search solutions might differ. Therefore, we focus our evaluation on the search engine personalization. In particular, we study the accuracy of the user profiles created by search engines.

2.1.5.4 Comparison Between Private Web Search Solutions

Using criteria established in the previous sections, we compare the private Web search solutions presented in Sections 2.1.1 and 2.1.2. We exclude from this comparison PIR solutions [31, 32], as they are not compatible with our requirements (i.e., they require from search engines specific recommendation algorithm). We summarize in Table 2 the aforementioned private Web search solutions. "✔" is used if a given solution implements a feature or requires a hypothesis, "✖" indicates that a feature is not provided or a hypothesis is not required for a given solution. Regarding the evaluation criteria, our qualitative comparison is displayed through the five balls: "○" indicates a very bad score, "◔" a bad score, "◑" a neutral score, "◕" a good score, and "●" a very good score.

Regarding the privacy protection ensured by private Web search solutions, we note from Table 2 a large disparity. Firstly, solutions like Viejo et al. [79],

Table 2

Comparison between private Web search solutions.

		Features					Hypotheses			Evaluation		
		Sender Anonymity	Receiver Anonymity	Unlinkability	Fake Queries[2]	Query Transformation	Non-Collusion	No free-riders	No prior knowledge	Privacy	Performance	Accuracy/Personalization
Unlinkability	Proxy [53]	✖	✖	✔	-	-	-	✔	✔	◔	◑	◔
	VPN [54]	✖	✖	✔	-	-	-	✔	✔	◔	◑	◔
	Mix network [55]	✖	✖	✔	-	-	✔	✔	✔	◕	◔	◔
	Web Mixes [56]	✔	✔	✔	-	-	✔	✔	✔	◕	◔	◔
	Tor [39]	✖	✖	✔	-	-	✔	✔	✔	◕	◔	◔
	RAC [42]	✔	✔	✔	-	-	✔	✖	✔	●	○	◔
	UPIR [43]	✖	✖	✔	-	-	✔	✔	✔	◕	◔	◔
	Crowds [44]	✖	✖	✔	-	-	✔	✔	✔	◕	◑	◔
	Erola et al. [60]	✖	✖	✔	-	-	✔	✔	✔	◕	◑	◔
	UUP [61]	✖	✖	✔	-	-	✔	✔	✔	◕	◔	◔
	Romero-Tris et al. [62]	✖	✖	✔	-	-	✔	✔	✔	◕	◔	◔
	Lindell et al. [63]	✖	✖	✔	-	-	✔	✖	✔	●	○	◔
	Romero-Tris et al. [64]	✖	✖	✔	-	-	✔	✔	✔	◕	◔	◔
	Dissent [40]	✖	✖	✔	-		✔	✖	✔	●	○	◔
Indistinguishability	PRAW [73]	-	-	-	W	-	-	-	✔	◑	◕	◑
	PDS [74]	-	-	-	W	-	-	-	✔	◑	◕	◑
	Viejo et al. [76]	-	-	-	SN	-	-	-	✔	◑	◕	◑
	GooPIR [45]	-	-	-	D	-	-	-	✔	◑	●	◔
	TrackMeNot [46]	-	-	-	R	-	-	-	✔	◑	●	◑
	Viejo et al. [79]	-	-	-	O	-	-	-	✔	◔	◕	◑
	Query Scrambler [86]	-	-	-	-	✔	-	-	✔	◔	◕	○
	Dispa [89]	-	-	-	-	✔	-	-	✔	◔	◕	◕

[2] We precise the sources used to generate fake queries: W = Web Pages; SN = Social Networks; D = Dictionary; R = RSS Feeds; and O = ODP Taxonomy / WordNet.

QS [86] and Dispa [89] give a partial protection to users. They let search engines create user profiles but, *in fine*, these profiles contain only generic information about users. Such solutions hide very specific information (e.g., the name of a disease) but still disclose some user private information (e.g., the user has a disease). Therefore, the privacy protection obtained with these solutions is relatively low. A similar protection is obtained with a Proxy [53] and a VPN [54], as these solutions have to be trusted (i.e., they have access to both the query content and the user identity). Indistinguishability solutions (e.g., GooPIR [45], TrackMeNot [46]) present a better protection. Fake queries mislead the adversary about real user interests. But, such protections suffer from re-identification attacks [48, 49]. Indeed, an adversary that has previously-collected non-protected queries[3] is able to identify fake queries and retrieve user queries. A more advanced protection is ensured by unlink-

[3] Google, Bing, and Yahoo! collect user queries, especially the non-protected queries previously sent by users.

ability solutions. They prevent search engines from aggregating queries per user and thus, creating their user profile. Among these solutions, RAC [42], Lindell et al. [63] and Dissent [40] ensure a more robust protection by detecting free-riders. When such protocols detect misbehaving nodes, they are excluded from the protocol. Nevertheless, the protection obtained with unlinkability solutions is not optimal as they suffer from de-anonymization attacks [47]. An adversary who had previously-collected queries about the users is able to retrieve the requester of an anonymous query.

Furthermore, Table 2 indicates that the type of protection influences the performance, especially the time waited by users to obtain their query results. Unlinkability techniques (e.g., Tor [39], UUP [61], Dissent [40]) send queries to search engines through multiple nodes and thus, the multiple query forwardings introduce a higher latency. In addition, these protocols execute cryptographic primitives to secure communications. Ciphering and deciphering messages engender extra computation time. Nevertheless, these solutions have different performance. Those which ensure sender and receiver anonymity or detects free-riders (i.e., RAC [42], Lindell et al. [63], Dissent [40]) require much more computation. Others that exchange n queries in a privacy-preserving way (e.g., UPP [61]) or ensure anonymous communication (e.g., Tor [39]) need fewer resources. Finally, the most efficient unlinkability solutions forward queries from users to users (e.g., Crowds [44]) or implement a Proxy [53] or a VPN [54]. Nevertheless, these protocols still introduce more delay than indistinguishability solutions, as the latter interacts with search engines directly. Indistinguishability solutions implement different methods to generate fake queries. We note that the fake query generation made by GooPIR [45] and TrackMeNot [46] is relatively straightforward (i.e, randomly selecting keywords in a dictionary, and extracting keywords from titles of newspaper articles contained in RSS feeds). Others indistinguishability solutions use more advance techniques (e.g., crawling Web pages [73, 74], using ODP taxonomies [89], exploiting social networks [76]) that engender extra computation.

In addition, private Web search solutions alter the quality of the results, as they impact the personalization performed by search engines. Due to the protection, search engines are not able to collect accurate information about a specific user. As a result, they create inaccurate user profiles and therefore, by personalizing the results with these inaccurate user profiles, they decrease the quality of the results. Nevertheless, as search engines implement their own personalization algorithms and do not make them publicly available, it is hard to assess precisely the influence of this incorrect personalization. Due to these reasons, we focus on the accuracy of the user profiles. Intuitively a more accurate user profile will lead to a better personalization. We distinguish three groups of solutions that disrupt the user profiles: (i) solutions that hide the user identity [53, 39, 44, 62]; (ii) solutions that obfuscate user queries [45, 46]; (iii) all other solutions [86, 89]. The accuracy obtained with the first type of solution is relatively low. Due to the unlinkability solutions, search engines create user profiles from queries issued by multiple users. Therefore, these profiles do not represent any users correctly. The second type of solutions protects users with fake queries. These user profiles contain information about a single user and some fake queries. Therefore, compared to the first group of solutions, these profiles better represent users. Even though the accuracy of these profiles depends on the quantity of fake queries generated by the private solution. Besides, for specific solutions like GooPIR [45], the accuracy of the results is degraded by an obfuscation mechanism. It introduces non-relevant results in the result list (i.e., results returned for the fake

queries), but they can partially be removed by a filtering algorithm. The last type of solution modifies the initial query. As Dispa [89] dispatches queries among multiple profiles (according to their topic), user profiles do not contain wrong information, and therefore the personalization is not impacted by Dispa. QS [86] does not disturb the user profiles, but as it does not send the initial query (only related queries are sent to the search engine), the results obtained through this solution are not very accurate.

Several works [90, 91] focus on improving the personalization by sending a lightweight user profile with the user query. In that case, user profiles are managed on the user side. To avoid any disclosure of information, they are protected with several privacy-preserving techniques (e.g., generalization) before being sent to the search engine. Nevertheless, as all well-known search engines (e.g., Google, Bing, Yahoo!) do not offer the possibly to send a user profile with the query, these solutions are not suitable for our problem. Due to this reason, it has been advised in [115] to perform a client-side personalization. For instance, UCAIR [116] learns user interests to improve user queries (through query reformulation) and to enhance their results (through a re-ranking).

2.1.5.5 Representative Private Web Search Solutions

Further in the thesis, we evaluate our own private Web search solution. This evaluation should contain a comparison with the state of the art. As we have seen, the literature contains two types of private Web search solutions: unlinkability solutions and indistinguishability solutions. Regarding the first type of solutions, we do not need to focus on a specific solution, as unlinkability solutions consider an adversary that cannot link a query with its requester. Therefore, we model unlinkability solutions by masking the query requester. Regarding indistinguishability solutions, we discard solutions that transforms the user query, as they do not provide a strong protection. Moreover, PRAW [73] and PDS [74] cannot be employed because they require previous Web Pages consulted by the users (our dataset of real search queries – see Section 3.3.1 – contains only URLs). Similarly, Viejo et al. [76] that require user social networks cannot be used. Among the three remaining solutions, we select TrackMeNot [46] and GooPIR [45], as they provide an open-source implementation. Besides, such solutions employ fake queries differently: TrackMeNot sends fake queries periodically while GooPIR include fake queries inside the user query. Therefore, these two solutions give a good representation of indistinguishability solutions.

2.2 Sensitive Query Detection

In this section, we overview solutions to detect sensitive queries, as part of this thesis focuses on adapting the protection provided by private Web search solutions to query sensitivity. Firstly, we exhibit several definitions of sensitive data mostly based on the notion of embarrassing topics (e.g., health, sex, politics). Then, we present solutions to classify queries into categories. No existing work has been proposed to detect sensitive queries. But, if we identify in a taxonomy tolerate (according to the definition that a user gives to sensitive data), a query categorizer is able to identify sensitive queries (i.e., queries related to a sensitive category).

2.2.1 Data Sensitivity

No formal definition about data sensitivity exists in the literature. Indeed, as sensitivity is a subjective notion (i.e., people have their own judgements about what is sensitive), it is not possible to create one generic definition about sensitive content. Nevertheless, by sensitive content, people often refer to a similar content. For instance, Google's privacy policy [3] describes sensitive personal information as "*confidential medical facts, racial or ethnic origins, political or religious beliefs or sexuality*". Microsoft does not explicitly define sensitive content but makes restrictions in its advertising platform for the following categories: "*adult content; controversial content; dating and relationships; drugs and related paraphernalia; firearms and weapons; health care; political; religious content and suffering; violence and exploitation gambling; surveillance equipment*" [117]. Similarly, the following categories are listed as unacceptable or restricted products and services in Yahoo!'s ads platform [118]: *adult sexual products & services, questionable legality products & services (e.g., Fake IDs & Diplomas), recreational drugs, tobacco products, alcohol products, credit repair services, dating sites, family planning, financial services, gambling, prescription drugs & pharmacies, religious, weight loss products & services*. Another point of view comes from the Digital Advertising Alliance (DAA) that defines sensitive data as "*personal information [...] from children [...] under the age of 13*" and "*financial account numbers, Social Security numbers, pharmaceutical prescriptions or medical records about a specific individual*" [119].

Administrative authorities or national laws also give a definition to sensitive contents. For instance, in 1978, the French Republic published a law stating that "*It is prohibited to collect and store in databases [...] nominative data that reveal, directly or indirectly, racial origins or politic, philosophical or religious origins, or trade-union membership of people*"[4] [120]. The European Union completed this law by a directive in 1995 [121] saying that "*Member States shall prohibit the processing of personal data revealing racial or ethnic origin, political opinions, religious or philosophical beliefs, trade-union membership, and the processing of data concerning health or sex life*". The United Kingdom transposes this directive in its national law [122] by adding *criminal records* to the definition. More recently, the European Union mandated a working group to refine the notion of sensitive data given in the directive published in 1995. In an advice paper [123], they consider the terms "*philosophical beliefs*" and "*health data*" as too broad, the term *racial* as unclear and suggest the addition of new categories such as "*genetic data, biometric data, geo-location and financial situation*". The US makes several laws [124, 125, 126] in which they consider "*financial data*", "*health information*" and "*electronic communication record*" as non-disclosable data. The state of California defines in the California Civil Code [127], personal information, with a list of 27 data types such as "*height; weight; religion; occupation; education; political party affiliation; medical condition; drugs and therapies; products purchased, leased, or rented; or information pertaining to creditworthiness, assets, income, or liabilities*".

Researchers also tackle this problem of defining sensitive data. They suggest defining sensitive data by studying users' behaviors. A first approach [128] uses Quora to identify sensitive topics and words. Quora is a question-and-answer website where users can post their messages anonymously. In [128], the authors made the assumption that Quora anonymous messages are related

[4] Translation from the French act n°78-17 published on January 6, 1978: "*Il est interdit de mettre ou conserver en mémoire informatisée [...] des données nominatives qui, directement ou indirectement, font apparaître les origines raciales ou les opinions politiques, philosophiques ou religieuses ou les appartenances syndicales des personnes.*"

to at least one sensitive topic. As a consequence, by analyzing the distinction between anonymous and non-anonymous messages posted on Quora, they obtained a notion of data sensitivity. They found out that, in addition to the general consensus (e.g., the definition given by Google), other types of data might be sensitive. They take the example of criticisms against prestigious institutions (e.g., Harvard College). Most Quora users criticize these institutions anonymously. Nevertheless, as search queries and public posts are not the same type of texts, the definition of sensitive data found by Peddinti et al. is not entirely valid in the context of Web search. For instance, sending positive or negative queries about prestigious institutions does not reveal any opinion about the requester: A supporter of a given institution could be interested in consulting negative criticisms against it.

2.2.2 Query Categorization

We overview solutions to classify queries into a predefined set of categories, as we are interested to identify sensitive queries. By identifying sensitive categories in a taxonomy, a query categorizer is able to identify sensitive queries. Query categorization has been studied for many years. Initially, query categorizers were designed to improve search engines. Indeed, by extracting the categories of a query, search engines could remove the results that do not belong in those categories. Recent query categorizers [129, 87] are based on a local search engine and an Open Directory Project (e.g., [81]). An ODP uses a hierarchical ontology scheme under which webpages are categorized. The classification is performed manually by a community of users. Originally, an ODP was used to find webpages corresponding to a specific category. Nevertheless, it is also possible to extract from an ODP a list of web pages corresponding to a given category. The query categorizer indexes all web pages contained in an ODP such that, for a given query, it returns the most relevant web pages. Then, using the ontology provided by the ODP, it maps the list of web pages to a list of categories. The categories are finally ranked according to their numbers of occurrences. Another solution [130] uses existing search engines to categorize queries. For each category in a predefined set of categories, it sends to the search engine a new request composed of the initial query and the category. Then, by comparing the number of results obtained with each request, the solution is able to create a ranked list of categories.

In [131], Alemzadeh et al. propose a different approach based on Wikipedia articles. Their solution maps each keyword of a given query to a Wikipedia page. Then, it extracts all categories associated with the Wikipedia pages and using a density metric, the solution retrieves the most relevant categories. In [132, 133, 134], the authors propose a query classifier by combining a machine learning approach (i.e., perceptron with margins algorithm) with a linguistic model (to disambiguate and interpret semantic). These classifiers are trained on a huge set of manually classified queries such that they are able to predict a category for a given query.

In 2005, SIGKDD organized a competition entitled "*Internet user search query categorization*" in which researchers were invited to propose a method to classify queries into 67 given categories. The winner of the competition suggested a solution [135] based on two classifiers: the first classifier is a Support Vector Machine classifier to associate a category to a snippet (i.e., a result in the search engine result list) and the second classifier maps categories returned by the search engine to the 67 predefined categories. By applying these two classifiers on the results returned by a search engine, this solution obtains a ranked list of categories (ranked by relevance). Two other contribu-

Table 3
Comparison between the existing query classifiers

	Ullegaddi et al. [129]	Juarez et al. [87]	Li et al. [130]	Alemzadeh et al. [131]	Beitzel et al. [134]	Shen et al. [135]	Kardkovacs et al. [136]	Vogel et al. [137]
Techniques employed								
Local Search Engines	✔	✔	-	-	-	-	-	-
Machine learning	-	-	-	-	✔	✔	-	-
Other	-	-	✔	-	-	-	-	-
Data used								
Wikipedia	-	-	-	✔	-	-	-	-
ODP	✔	✔	-	-	-	-	✔	✔
Web Search Engines	-	-	✔	-	-	✔	-	-
Categorized queries	-	-	-	-	✔	-	-	-

tors involved in this contest published their solutions. The first one [136] uses a neural network learning algorithm to classify queries into predefined categories. The authors also combine the machine learning approach with two search engines (LookSmart and Zeal) from which they obtain categories of a given query. They developed an algorithm to map these categories to the ones provided in the KDD contest. Finally, their solution retrieves the top five categories returned by the neural network or the mapping of categories obtained with the two search engines. Other researchers that participate in the KDD contest published their solution [137]. They propose a similar approach that send queries to a web directory in order to obtain a list of categories. This list is mapped to the predefined KDD taxonomy and then, ranked to obtain the final categories. Furthermore, the second KDD contributors who make public their solution.

2.2.3 Discussion

We summarized in Table 3 the aforementioned query classifiers. We detail for each solution the type of techniques they employ and the data they use. We note that most query categorizers are based on local search engines or machine learning algorithms. These algorithms are computationally expensive and thus not compatible with our requirements (i.e., personal computers have limited resources). Moreover, query categorizers mainly use four types of data: Wikipedia, ODP, online search engines, and manually categorized queries. To obtain a good accuracy, query categorizers necessitate a large collection of data. But large databases are too big to be stored on a personal computer. To overcome these issues, query categorizers could rely on online services, but the query sensitivity needs to be assessed before sending queries to any distant servers (otherwise, it would disclose personal information). Furthermore, data employed by query categorizers is based on a static context. They cannot automatically adapt themselves to the evolution of the Internet. Any update requires a human intervention on data. This represents a non-negligible effort, in particular if a large new set of queries need to be manually categorized. Besides, each update either generates a new index (in the case of a local search engine) or trains a new classifier (in the case of a machine learning algorithm) that engender heavy computations. Finally, the recall achieved by the existing query categorizers is relatively low, indicating

that classifying queries into a predefined set of categories is a difficult problem. For instance, the recall obtained by the winner of the KDD competition [135] is only 47.9%; meaning that on a dataset of 800 queries, only 47.9% of queries were correctly classified compared to a human-based classification.

CHAPTER 3

SimAttack, an Efficient and Scalable Attack Against Private Web Search Solutions

3.1 Objective

This chapter presents SimAttack, a generic attack against all types of private Web search solutions. Compared to existing attacks [47, 48, 96, 49], we propose an approach that targets both unlinkability solutions (e.g., Tor [39]) and indistinguishability solutions (e.g., TMN [46], GooPIR [45]). To the best of our knowledge, no existing attacking has been published against GooPIR-like solutions. Besides, as unlinkability solutions and indistinguishability solutions are independent of each other, users can protect their queries by running an indistinguishability solution on top of an unlinkability solution (even though no solution in the literature considers this possibility). Therefore, we designed SimAttack to target as well this type of private Web search solutions.

Existing attacks require a lot of resources that limit their potentiality. Due to this reason, previous studies limit their evaluation to 60 or 100 users. We design SimAttack such that it processes a large number of queries from a large number of users in a reasonable amount of time.

3.2 Design of the Attack

In this section, we present SimAttack, an attack against all types of private Web search solutions. SimAttack is based on a similarity metric between an incoming query and a user profile (i.e., queries previously collected by the adversary). Using this metric, SimAttack is able to break the different types of private Web search solutions. Indeed, the metric indicates if the query is related or not to a given user profile. As a consequence, SimAttack is able to de-anonymize an anonymous query (by finding the user profile which maximizes the similarity metric) or differentiate the fake queries from real ones (by finding the query which has the highest similarity with the user

profile). SimAttack is a user-centric attack which tries to compromise the privacy of each user independently.

The next sections explain the computation of the similarity metric between a user profile and a query, and detail how SimAttack takes advantage of the similarity metric to break the targeted protection mechanisms: unlinkability solutions, indistinguishability solutions, and indistinguishability solutions over unlinkability solutions.

3.2.1 Similarity Metric Between a Query and a User Profile

We create a similarity metric $\text{sim}(q, P_u)$ to characterize the proximity between a query q and a user profile P_u related to a user u. The query q is a short text represented by a binary vector:

$$q = [v_{a_1}, v_{a_2}, ..., v_{a_k}],$$

where $v_{a_1}, v_{a_2}, ..., v_{a_k}$ are binary values to indicate if the terms $a_1, a_2, ..., a_k$ are used or not in the query q. As mentioned in Section 1.2, the adversary was able to collect non-protected queries issued by the users in the system. These queries are aggregated for each user u in a user profile P_u. We define a user profile P_u as a non-sorted sequence of queries:

$$P_u = \{q_1, q_2, ..., q_j\},$$

where $q_1, q_2, ..., q_j$ are the non-protected queries sent by the user u and collected by the adversary. The similarity metric $\text{sim}(q, P_u)$ returns a value between zero and one where greater values indicate that the query q is close to the user profile P_u. Algorithm 1 presents in detail the computation of the similarity metric.

Algorithm 1: Similarity metric between a query and a user profile.

input : q : a query,
P_u : profile of user u (history of query issued by u),
α : a smoothing factor.

`/* Compute a similarity between` q `and all queries contained in` P_u `*/`

1 **for** $q_i \in P_u$ **do**
2 $a_i \leftarrow \text{similarity}(q, q_i)$;

`/* Sort the previous values in ascending order */`

3 $(x_0, ..., x_{|P_u|-1}) \leftarrow \text{sort}(\{a_i\}_{i \in [\![0, |P_u|-1]\!]})$;

`/* Compute the exponential smoothing */`

4 $s \leftarrow x_0$;
5 **for** $i \in [\![1, |P_u| - 1]\!]$ **do**
6 $s \leftarrow \alpha \cdot x_i + (1 - \alpha) \cdot s$
7 **return** s ;

It first computes the value a_i corresponding to the similarity between two vectors: the query q and the query q_i stored in P_u (line 2). Several well-known metrics can be employed to compute the similarity between two vectors, e.g., Dice's coefficient [138], cosine similarity [139] and Jaccard index [140]. We discuss its choice in Section 3.3.5. Then, Algorithm 1 ranks all the values $\{a_i\}_{i \in [\![0, |P_u|-1]\!]}$ in ascending order (line 3). The similarity metric $\text{sim}(q, P_u)$ is finally computed as the exponential smoothing of these values (lines 4 to 6). Indeed, the exponential smoothing corresponds to a weighted mean with a

higher weight on the highest values. The highest values are the values that discriminate a user. Indeed, a high value indicates that the user already issued the query (or part of the query). The exponential smoothing uses a smoothing factor α to establish the weight given to the values. This parameter α varies between 0 and 1. In practice, the value of α does not strongly impact the results of SimAttack as presented later in Section 3.3.6. Regarding the complexity of this algorithm, computing the similarity between two vectors is $O(n)$, where n is the size of the two vectors (i.e., the size of the vector space). Therefore, the overall complexity of Algorithm 1 is $O(|P_u| \cdot [n + \log(|P_u|)])$.

3.2.2 Unlinkability Attack

The de-anonymization attack identifies the requester of an anonymous query. Algorithm 2 describes this attack. For each user profile P_u previously collected by the adversary, it computes its similarity with the query q (line 5). Then, it returns the identity id corresponding to the profile with the highest similarity (line 6). If the highest similarity equals zero, all similarities equal zero and therefore the identity of the requester remains unknown and the attack is unsuccessful (line 7). Otherwise, the algorithm considers the user id as the initial requester of the query q.

The complexity of Algorithm 2 is $O(|U| \cdot m \cdot [n + \log(m)])$, where m is the maximum number of queries in the user profiles $\{P_u\}_{u \in U}$, and n is the size of the vector space.

Algorithm 2: De-anonymization Attack.

```
   input: q : a query,
          U : set of users.
   /* Retrieve the user id which maximizes sim(q, P_id)        */
1  (id, sim) ← (u_0, sim(q, P_{u_0})) ;
2  for u_i ∈ U \ {u_0} do
3      sim_u ← sim(q, P_{u_i}) ;
4      if sim_u > sim then
5          (id, sim) ← (u_i, sim_u) ;
   /* Check that sim is different from 0, otherwise the
      attack is unsuccessful                                   */
6  if sim > 0 then return id;
7  else return ∅ ;
```

3.2.3 Indistinguishability Attack

The attack against indistinguishability solutions aims to identify initial queries among faked or 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 detailed in Algorithm 3 proceeds as follows. It first determines which obfuscation mechanism is being used. More precisely, it checks if the obfuscated query q^+ contains several fake queries separated by the logical OR operator (line 1) (i.e., behavior of GooPIR). It might appear that the logical OR operator was introduced by the user in her query (and not by the obfuscation mechanism). Nevertheless, as the user query and all fake queries have the same number of keywords, it is easy to detect if the logical OR was introduced by the user or the obfuscation mechanism.

Algorithm 3: Indistinguishability Solutions Attack.

input : q^+: a query,
P_u: a user profile,
δ: a threshold,
P_{FQ}: a profile of fake queries.

1 **if** $q^+ = q_0$ OR ... OR q_k **then** /* q^+ contains fake queries separated by the logical OR operator (i.e., GooPIR) */
  /* q^+ contains fake queries separated by the logical OR operator (i.e., GooPIR) */
2 $(q, sim) \leftarrow (q_0, \text{sim}(q_0, P_u))$;
3 **for** $q_i \in q^+ \setminus \{q_0\}$ **do**
4 $sim_i \leftarrow \text{sim}(q_i, P_u)$;
5 **if** $sim_i > sim$ **then** $(q, sim) \leftarrow (q_i, sim_i)$;
  /* Check that sim is different from 0, otherwise the attack is unsuccessful. */
6 **if** $sim > 0$ **then return** q;
7 **else return** $\emptyset$;
8 **else**
  /* q^+ is a fake query or a real query (i.e., TMN) */
9 **if** $P_{FQ} = \emptyset$ **then** /* Do not exploit the RSS feeds */
10 **if** $sim(q^+, P_u) > \delta$ **then return** q^+;
11 **else** /* Exploit the RSS feeds */
12 **if** $sim(q^+, P_u) > sim(q^+, P_{FQ})$ **then return** q^+ ;
13 **return** $\emptyset$; // q^+ is considered to be a fake query

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 q_i from q^+ and computes the similarity metric between these aggregated queries q_i and the user profile P_u (lines 3 and 5). Then it stores the query with the highest similarity in the variable q. Finally, the algorithm checks if the similarity $\text{sim}(q, P_u)$ is different from zero (line 6). If not, it means that the $(k+1)$ queries are not similar to any user profile and thus the attack fails (line 7). Otherwise, the algorithm returns q as the initial request (line 6).

In the second case (i.e., the query does not contain the logical OR operator), it distinguishes two cases: if the adversary has a prior knowledge about RSS feeds used by the user to generate the fake queries or not. If we consider first that the adversary does not have 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 so, then q^+ is considered as a real query, and is therefore returned (line 10). Otherwise, the query is considered to be a fake query (line 13).

Conversely, if we consider the situation where the adversary knows the RSS feeds used by the user to generate fake queries, the adversary generates fake queries using these predefined RSS feeds. These fake queries are stored in a profile P_{FQ} (same structure as a user profile P_u). Then, the adversary uses this external knowledge to distinguish fake queries (line 12). It first compares the similarity between the query q^+ and the user profile P_u (i.e., $\text{sim}(q^+, P_u)$) against the similarity between the query q^+ and the profile of fake queries P_{FQ} (i.e., $\text{sim}(q^+, P_{FQ})$). If $\text{sim}(q^+, P_u)$ is greater than $\text{sim}(q^+, P_{FQ})$, 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 (line 13).

The complexity of Algorithm 3 is $O(k \cdot |P_u| \cdot [n + \log(|P_u|)])$, where k is the number of fake queries, and n the size of the vector space.

3.2.4 Indistinguishability Over an Unlinkability Solution Attack

To break solutions that uses an indistinguishability solution on top of an unlinkability solution, the attack combines the two previous attacks. As depicted in Algorithm 4, its objective is to identify both the initial requester and the initial query. Similarly to the attack presented in Algorithm 3, Algorithm 4 first determines which obfuscation mechanism is being used by looking for logical OR operators (line 1). If the protected query q^+ contains

Algorithm 4: Indistinguishability over unlinkability Attack.

input : q^+: a query,
U: set of users,
δ: a threshold,
P_{FQ}: a profile of fake queries.

1 **if** $q^+ = q_0$ OR ... OR q_k **then**
  /* q^+ contains fake queries separated by a logical OR operator (i.e., GooPIR) */
2 **for** $q_i \in q^+$ **do**
    /* Retrieve the most probable user for each sub-query */
3 $id[i] \leftarrow$ Algorithm_2(q_i, U) ;
4 $I \leftarrow [\![0, k]\!]$;
5 **for** $i \in [\![0, k]\!]$ **do**
6 **if** $id[i] = \emptyset$ **then** $I \leftarrow I \backslash \{i\}$;
  /* Retrieve the most probable sub-query q_a */
7 **if** $I = \{a\}$ **then**
8 **return** $(q_a, id[a])$;
9 **else if** $|I| > 1$ **then**
10 $a \leftarrow$ index s.t. sim$(q_a, P_{id[a]})$ is maximal over I;
11 $b \leftarrow$ index s.t. sim$(q_b, P_{id[b]})$ is maximal over $I \backslash \{a\}$;
12 **if** $sim(q_a, P_{id[a]}) - sim(q_b, P_{id[b]}) > cf$ **then**
13 **return** $(q_a, id[a])$;
14 **return** $\emptyset$; // The attack is unsuccessful
15 **else**
  /* q^+ is a fake query or a real query (i.e., TMN) */
16 $id \leftarrow$ Algorithm_2(q^+, U) ; // Retrieve the most probable user
17 **if** $id \neq \emptyset$ **then**
18 **if** $P_{FQ} = \emptyset$ **then** /* Do not exploit the RSS feeds */
19 **if** $sim(q^+, P_{id}) > \delta$ **then return** (q^+, id);
20 **else** /* Exploit the RSS feeds */
21 **if** $sim(q^+, P_{id}) > sim(q^+, P_{FQ})$ **then return** (q^+, id) ;
22 **return** $\emptyset$; // q^+ is a fake query
23 **return** $\emptyset$; // The attack is unsuccessful

$(k+1)$ fake queries q_i, it first extracts them and then retrieves, for each of them, its potential requester $id[i]$ by invoking Algorithm 2 (lines 2 to 3). Then, it removes queries which are not associated with a potential requester (lines 5 to 6), i.e., queries for which Algorithm 2 was unsuccessful. We denote the set of indexes corresponding to the remaining queries by I. Finally, if I contains one single element a (i.e., only one query is associated with a potential requester), it returns the pair $(q_a, id[a])$ such as the query q_a is considered as the initial query and the user $id[a]$ as the initial requester (lines 7 to 8).

However, if I contains at least two elements (line 9), Algorithm 4 retrieves the pairs $(q_a, id[a])$ and $(q_b, id[b])$ which have the highest similarity over I, where a and b are their indexes in I, and evaluates the difference between them (line 12):

$$\text{sim}(q_a, P_{id[a]}) - \text{sim}(q_b, P_{id[b]}).$$

To ensure a certain confidence in the results, if this difference is too small, the attack is considered as unsuccessful (i.e., the pair $(q_a, id[a])$ is not returned). Indeed, from two pairs that have approximately the same similarity, it is not possible to clearly identify the real one. If the difference is greater than a threshold cf (initialized at 0.01 by default), the algorithm returns the pair $(q_a, id[a])$ such as the query q_a is considered as the initial query and the user $id[a]$ as the initial requester (line 13). Otherwise, the attack is unsuccessful (line 14).

When queries do not contain OR operators, the algorithm first retrieves the potential requester id by calling the Algorithm 2 (line 16). If this id is not empty (i.e., if the attack made by the Algorithm 2 is successful), it distinguishes two cases depending if the adversary has a prior knowledge about RSS feeds used by the user. As mentioned in the previous section, if the adversary is able to generate fake queries from the RSS feeds used by the user, she creates a profile P_{FQ} (similar to a user profile P_u) that contains a set of fake queries. Let us consider the first case in which the adversary does not have this knowledge (lines 18 to 19). We then consider that the adversary only knows prior queries sent by the user. Therefore, the adversary is able to distinguish between fake queries and real ones by comparing the similarity between the query q^+ and the user profile P_{id}, $\text{sim}(q^+, P_{id})$, with the threshold δ. If $\text{sim}(q^+, P_{id})$ is greater than δ, the query is considered as a real query sent by the user id and thus, the pair (q^+, id) is returned (line 19). Indeed, a similarity higher than δ means that the query is close to the user profile and thus is likely to be a real user query. Otherwise, the query q^+ is considered as a fake query (line 22)

On the other hand, if we consider that the adversary is able to generate a set of fake queries with the RSS feeds (lines 20 to 21), the algorithm determines if the similarity distance between the query q^+ and the user profile P_{id}, $\text{sim}(q^+, P_{id})$, is greater than the similarity metric between the query q^+ and the profile of fake queries P_{FQ}, $\text{sim}(q^+, P_{FQ})$. In that case, the pair (q^+, id) is respectively considered as the initial query and the initial requester and returned by the algorithm (line 21). Otherwise, as no pair has been returned, the query q^+ is considered as a fake query (line 22).

The complexity of Algorithm 4 is $O(k \cdot |U| \cdot m \cdot [n + \log(m)])$, where k is the number of fake queries, m the maximum number of queries in the user profiles $\{P_u\}_{u \in U}$, and n the size of the vector space.

3.3 Experimental Set-up

Our experiments are conducted on a commodity desktop workstation with a 2.2 GHz quad-core processor with 8 GB of memory. In this section, we present the dataset of real queries we used to evaluate SimAttack. We also introduce the three state-of-art solutions against which we evaluate SimAttack: unlinkability solutions, GooPIR, and TMN. Furthermore, we detail the machine learning attack against which we compare the performance of SimAttack. Then, we explain the metric we use to assess the efficiency of the attacks: the recall, the precision and the F1-score. Finally, we precise how we chose the similarity metric and the smoothing factor α used by SimAttack.

3.3.1 Datasets

In this section, we present the dataset of real queries used in all our evaluations. This dataset is based on query logs released by the search engine AOL in 2006. In the following section, we outline all steps we followed to build our dataset.

3.3.1.1 AOL Search Queries Logs

The AOL query log dataset was published in August 2006. It contains queries issued by approximately 650,000 users over a three-month period (March, April and May 2006). AOL published the dataset to give useful real world data to help researchers in conducting their research. As users were not aware that their queries would be publicly available, they could not modify their queries accordingly or restrain themselves to send them (especially their sensitive queries). Queries in the dataset are pseudo-anonymized meaning that users are identified by a unique number and all other personal information (e.g., name, email address) was removed. All queries in this dataset include five attributes:

- **AnonID** — a number that identifies a user uniquely.
- **Query** — the query issued by the user.
- **QueryTime** — a timestamp indicating at what time the query was issued.
- **ItemRank** — if the user clicked on a page in the result list, the rank of the page the user selected.
- **ClickURL** — if the user clicked on a page in the result list, the URL of the page the user selected.

In the following we only use two of these attributes: **AnonID** and **Query**.

3.3.1.2 Pre-Processing

The attribute **Query** is not directly exploitable from the dataset as it is a short text. We pre-process it with two Natural Language Processing (NLP) techniques. First, the *tokenization* splits the short text into a set of words by identifying whitespace or more complex regular expressions. Then, the *stemming* replaces each word with its stem. A stem is a part of a word to which prefixes or suffixes can be attached. For instance, the word "friendship" is transformed into "friend" (the suffix *-ship* is removed). In our experiments, we use the tokenizer provided by the Stanford CoreNLP library [141] to split queries into a set of words. Then, we stem each word by eliminating or replacing the suffix using the Porter algorithm [142].

We also filter irrelevant keywords by applying three other NLP techniques: named-entity recognition, part-of-speech tagging and stop words removal. The *named-entity recognition* identifies the named entities (e.g., person, localisation), the numerical entities (e.g., money, percent), and the temporal entities (e.g., date, duration) using specific rules and dictionaries. The *part-of-speech tagging* is a process to identify the role of a word in a sentence (e.g., noun, verb, adjective). The *stop word removal* filters out specific keywords; usually the most common words of a language (e.g., the, it, what, in English) as they do not provide any meaningful information. In our experiments, we remove all stop words from the set of stemmed words; the list of stop words is accessible in the Appendix A. We also categorize each stemmed word with the *named-entity recognition* provided by the Stanford CoreNLP library to remove words identified as number, date, duration, pronouns, interrogative words, money, and time. These words are not relevant in characterizing users. Finally, we also use the *part-of-speech tagging* provided by Stanford CoreNLP library as a library presented in Chapter 5 requires it.

3.3.1.3 Filtering Users

The AOL dataset contains a majority of users that issued few queries during the three months. For instance, 56.1% of the users issued less than 15 queries during the dataset period. However, non-active users are not suitable for our evaluations as they issued too few queries. Indeed, as explained later in this chapter, some of the queries are employed to create user profiles. These profiles are used by the adversary to break private Web search solutions. A profile with too few queries does not reflect its user and thus is not useful to break any protection mechanisms. For that reason, we are interested in the most active users. These users are difficult to protect as their huge number of requests reveal a lot of personal information.

We consider that an active user is a user that sent queries during at least two-thirds of the dataset period (i.e., 61 days) with a consecutive period of half of the dataset period (i.e., 45 days). We filter the whole dataset to target active users and thus reduce the dataset from approximately 650,000 users to 18,164 users. A better overview of the selected users is given by Figure 3. It presents the number of queries sent by each user. They issued between 62 queries and 3,156 queries over the dataset period. Besides, we note that 91.2% of users issued less than 500 queries.

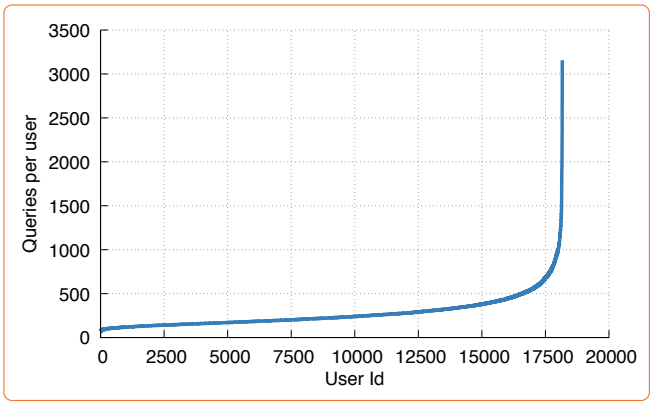


Figure 3
Distribution of the number of queries issued per user in the filtered dataset.

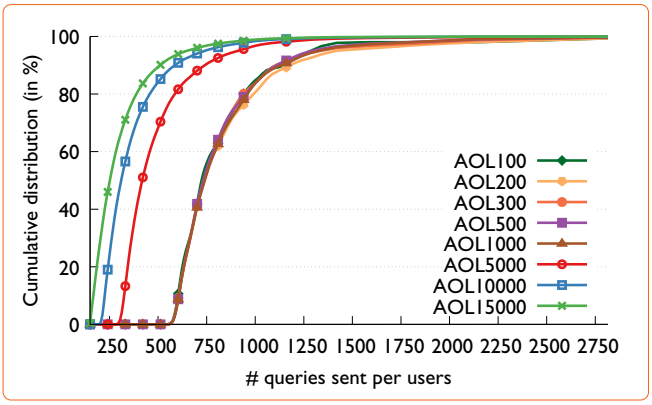


Figure 4
Distribution of the number of queries issued per user for the different dataset.

3.3.1.4 Dataset Creation

We created five datasets containing different number of active users (from 100 to 1,000 users): `AOL100`, `AOL200`, `AOL300`, `AOL500`, `AOL1000`. To do that, we ordered the 18,164 users according to the number of queries they issued and then selected the top 1,000 users to create the dataset `AOL1000`. We then generated the four other datasets as a subset of `AOL1000`. To retain similar statistical properties and ensure that users issued a significant number of queries, we selected users according to the number of queries they issued. The selection was made randomly to obtain the desired number of users (e.g., 100 users for `AOL100`). Figure 4 depicts the Cumulative Distribution Function (CDF) of the number of queries issued by the users in these five datasets. We note that each of these five datasets follows approximately the same distribution. Besides, these CDF show that users in these datasets issued at least 500 queries.

In addition, we also created three extra datasets with a higher number of users. They contain, respectively, the 5,000; 10,000; and 15,000 most active users. We name these datasets `AOL5000`, `AOL10000` and `AOL15000`. Figure 4 shows that these three datasets do not follow the previous distribution of queries per user due to the lack of highly active users in the AOL dataset. However, the results obtained with these datasets give a minimum value, as considering an adversary has been collected more queries would likely increase the efficiency of the privacy attacks.

Moreover, we note that evaluating our work with up to 15,000 users is by far lower than the number of users that query current search engines. For instance, our three-month dataset of 15,000 users contains 4,621,545 queries while Google [3] receives every day more than 3 billion queries [143]. But, due to the lack of available data, it is not possible to perform a more realistic evaluation. Besides, performing an evaluation with a lower number of users advantage an adversary, as having fewer users generates less traffic and thus make a re-identification attack easier.

As mentioned in Section 1.2, we considered that the adversary was able to collect queries about her users (i.e., before they use any protection mechanism). Consequently, we split each dataset in two parts: a training set used to build user profiles (i.e., the previously-collected queries obtained by the adversary), and a testing set used to assess the robustness of the considered privacy-preserving mechanisms. In the literature, previous works [47, 48] used the two thirds of user queries for the training set and the remaining third of queries for the testing set. Therefore, we used the same proportion for our experiments.

Furthermore, TrackMeNot and GooPIR (i.e., two protection mechanisms assessed in the evaluation section) require the generation of fake queries. The number of users involved does not impact the user protection, as the adversary knows the identity of the requester, and thus attacks each user separately. Therefore, for TMN and GooPIR, we only considered the dataset of 100 users, `AOL100`. This dataset was chosen as it contains enough users to perform our experiments; a larger number of users would require more computational time. To generate the appropriate number of fake queries for TrackMeNot, we considered that users used their computer 8 hours a day and have set up TMN to send a maximum of *60 queries per hour* (the default value). As the testing set contains queries issued during 31 days, we generated 14,880 fake queries per users (i.e., 60 queries $\times$ 8 hours $\times$ 31 days). We denote by `TMN100`, the `AOL100` dataset with the 1,488,000 fake queries (i.e., 100 users $\times$ 14,880 fake queries per users). We also generated datasets with a lower number of fake queries (corresponding to the TMN setting *30 queries per hour*, *10 queries per hour*, *1 queries per hour*). These datasets contained respectively 7,440; 2,480; 248 fake queries per user. For GooPIR, we generate fake queries using the algorithm described in Section 3.3.2.3. Each user query is protected with up to seven fake queries.

Finally, in the case of TMN, we also generated a set of fake queries for the adversary. As mentioned in Section 3.2.3, RSS feeds are publicly available on the Internet, and as it is likely that some users do not modify the default TMN RSS feeds, the adversary is able to generate fake queries similar to the ones used by these users. Therefore, these fake queries might help the adversary from distinguishing fake ones from real ones. These fake queries are stored in a user profile called P_{FQ}. To generate a fair amount of queries for the adversary, we considered in our evaluations that the adversary generates the same number of fake queries as the users did. As a result, P_{FQ} contains 14,880 fake queries.

3.3.2 Representative Private Web Search Solutions

As mentioned in Section 2.1.5.5, unlinkability solutions, TrackMeNot, and GooPIR are three solutions that represent private Web search solutions. In this section, we detail how we set up these solutions for our experiments.

3.3.2.1 Unlinkability Solutions

There are many private Web search solutions that enable the unlinkability between the query and the user behind it (see Section 2.1.1). In this dissertation, we denote by unlinkability solutions all these solutions. In terms of privacy protection, they all hide the identity of the requester and thus do not need to be distinguished. We model these solutions by considering that the adversary receives queries without knowing the identity of the requester.

3.3.2.2 TrackMeNot (TMN)

To ensure that the generated fake queries (built from RSS feeds captured in 2014) are compatible with terms that users cared to look for in 2006 (year of capture of the AOL dataset), we compute the overlap between the words used in fake queries and the words used in the whole AOL dataset. We found that 85.6% of words used in fake queries are also contained in the AOL dataset (6,918 words out 8,082). In addition, the remaining 14.4% (i.e., words in fake queries that are not used in the AOL dataset) is partly due to a specific vocabulary employed in fake queries, as the titles of newspaper articles for which

they are generated, often contained a very specific vocabulary. Consequently, as the two different dates of capture do not strongly influence the usage of the terms, it justifies the joint use of these two datasets.

3.3.2.3 GooPIR

To generate fake queries with GooPIR, we created a dictionary from the AOL dataset. More precisely, we extracted all keywords and their usage frequency from the 20 million AOL Web search queries. Then, based on this dictionary, the GooPIR algorithm created up to seven fake queries for each original query.

3.3.3 Concurrent Attacks

Similar attacks to SimAttack have been published in the literature [47, 48]. In this section, we present our implementations of the two machine learning attacks: one against unlinkability solutions (ML Attack against Unlinkability) and another one against TrackMeNot (ML Attack against TMN). Nevertheless, to the best of our knowledge, no previous attack has been published against GooPIR or indistinguishability solutions run on top of an unlinkability protocol. For that reason and to establish a baseline, we extend the existing attacks to target GooPIR and indistinguishability solutions run on top of an unlinkability protocol.

The four attacks use Weka [144], a popular open-source machine learning framework. Weka contains all well-known classifiers (e.g., SVM, Random forests, Logistic regression). As mentioned in Section 3.3.1, our datasets are split in two parts: The first ⅔ of queries are used to train the classifiers while the remaining ⅓ of queries composed the test set.

3.3.3.1 ML Attack Against Unlinkability

The implementation of the first attack uses the Support Vector Machine (SVM) classifier. Two different algorithms are available in Weka: SMO [145] and LibSVM [146]. We selected LibSVM as it is more efficient than SMO to perform an SVM classification [147]. The current attack is a multi-class problem. It associates one class to one user. Several solutions expand the original binary SVM algorithm to a multi-class SVM algorithm. LibSVM implements the one-against-one method [101]. This method has been shown to be the most suitable for practical use [100]. We reproduced the attack as it was published in [47]: the same type of SVM (i.e., C-SVC), the same type of kernel (i.e., linear), and the same tolerance of termination criterion Epsilon (i.e., 0.001 that corresponds to its default value). However, for parameter C, that controls the trade-off between minimizing the number of errors in the training set and maximizing the margin between two classes, we optimize its value to obtain the best performance. A small value will not tolerate errors in the training set while a large value will maximize the margin between classes. Weka offers a specific option (`CVParameterSelection`) to find the parameter C that maximizes the performance of the classification. Using this option, we found out that the best value for C is 1.1 (the default value is 1). The SVM classifier is trained on previously-collected queries by the adversary (i.e., queries previously sent without any protection).

3.3.3.2 ML Attack Against TMN

The attack against TMN has been published by Peddinti et al. [48]. Distinguishing fake queries from real ones is a binary classification problem: a first class contains fake queries and a second class real queries. The attack requires the adversary to generate a set of fake queries using the same RSS feeds used by the users. It is reasonable to consider this list known by the adversary, as most of the users might not modify the default RSS feed list. Furthermore, as TMN does not hide the identity of the requester, each user is processed separately. Therefore, by considering a system with n different users, the attack consists in n distinct binary classification problems. Each classifier is trained using real queries contained in the user profile of the considered user, and the set of fake queries generated by the adversary. As a result, the n classifiers learn the difference between real queries and fake queries. Peddinti et al. showed in their paper that logistic regression is the best classifier to identify fake queries. Consequently, we implement the attack using `Logistic`, the default logistic regression implementation available in Weka. This algorithm builds a logistic regression model with a ridge estimator [148] and uses the Quasi-Newton method [149] to fit the model to the data. The Quasi-Netwon method gives a good trade-off between fitting accuracy and speed. Logistic regression does not require the setting-up of any parameters.

Moreover, Peddinti et al. did not investigate the SVM classifier in their studies. They mentioned in [150] that they "*neglect other better classifiers like SVM, so as to estimate the lowest accuracies that can be achieved*". Consequently, we also include in our experimentations an attack based on the SVM classifier. Similarly to the logistic regression attack, the SVM attack distinguishes fake queries using k binary classification problems (one classifier per user). We parametrize the SVM classifier with the same settings we used for the attack against unlinkability solutions.

3.3.3.3 ML Attack Against GooPIR

The previous attacks suppose an adversary with a prior knowledge on fake queries (otherwise, the adversary is not able to train the SVM or the logistic regression classifier). Therefore, the previous attack cannot target GooPIR, as the latter generates fake queries by randomly selecting words in a dictionary. To obtain a fair competitor to compare the results of SimAttack, we modified the attack against unlinkability techniques to create ML Attack against

Algorithm 5: ML Attack against GooPIR.

input : q^+ : a protected query,
SVM: SVM model obtained after the training phase,
u: identity of the requester

```
1 q ← null ; // Query identified as the initial one
2 p_max ← −1 ; // Probability that q belongs to the user u
3 for q_i ∈ q^+ do
4     p̂ = SVM(q_i, u) ; // Probability that q_i belongs to the
        user u
5     if p̂ > p_max then
6         q ← q_i ;
7         p_max ← p̂ ;
8 return q;
```

GooPIR. GooPIR obfuscates a user query by generating a protected query composed of $k+1$ sub-queries: the original query and k fake queries; all sub-queries are separated by the logical OR operator. The attack aims to retrieve, from a protected query, the sub-query that appears to be the most likely one. ML Attack against GooPIR is explained through Algorithm 5. It first retrieves each sub-query in the protected query by identifying logical OR operators. There is no issue if a sub-query contains a boolean operator, as the user query and all fake queries have the same number of keywords. Consequently, it is easy to distinguish the logical operator introduced by the user from the ones introduced by GooPIR. Then, the algorithm reuses the SVM classifier trained for the unlinkability attack (as the training step is not specific to unlinkability solutions). The algorithm returns, for each sub-query sent by the user u, the probability that the sub-query was issued by u (line 4). Finally, the algorithm identifies the query with the highest probability and considers it as the initial query. Consequently, without any knowledge on the generation of fake queries, Algorithm 5 distinguishes fake queries from real ones.

The prediction complexity for the linear SVM is $O(|U| \cdot n)$, where $|U|$ is the number of users in the system and n is the number of features. Indeed, our SVM classifier uses $|U|-1$ binary classifiers that compute a linear equation in a vector space of size n. Therefore, the overall complexity of Algorithm 5 is $O(k \cdot |U| \cdot n)$, where k is the number of fake queries.

3.3.3.4 ML Attack Against Indistinguishability Solutions Over an Unlinkability Solution

Users are able set-up on their computer an indistinguishability solution together with an unlinkability solution. Therefore, it corresponds to an indistinguishability solution that runs on top of an unlinkability protocol. As a result, the aforementioned machine learning attacks are not designed against such protection mechanisms. Nevertheless, it is possible to adapt the de-anonymization attack against them. We present the adaptation we made to attack GooPIR and TrackMeNot over an unlinkability solution.

- **GooPIR over an Unlinkability Solution** — If we consider that queries are protected by GooPIR run on top of an unlinkability solution, protected queries are obfuscated with k fake queries and then anonymized. We adapt the machine learning attack against unlinkability solutions to de-anonymize the query and identify the query requester. Algorithm 6 retrieves from a protected query (i.e., $k+1$ anonymous queries), the most probable pair (query, user) behind the protected query. In line 5, we apply the SVM classifier on each $k+1$ sub-queries contained in the protected query q^+. For each sub-query, the SVM classifier returns the most likely requester $\hat{u}$ with a probability $\hat{p}$. Then, the algorithm returns the pair (q,u) that obtained the highest probability p_{max} (line 6 to line 9): q is considered as the initial query and u as the initial requester.

  The complexity of Algorithm 6 is $O(k \cdot |U| \cdot n)$, where k is the number of fake queries, $|U|$ is the number of users, and n the number of features in the SVM classifier.

- **TrackMeNot over an Unlinkability Solution** — If we consider queries protected by TMN over an unlinkability solution, the protection results in sending anonymously user queries and periodically generated fake queries. The goal of the attack is to correctly identify real queries and

Algorithm 6: ML Attack against GooPIR over an unlinkability solution.

input : q^+: a protected query,
SVM: SVM model obtained after the training phase.

$q \leftarrow null$; `// Query identified as the initial one`
$u \leftarrow null$; `// User identified as the initial one`
$p_{max} \leftarrow -1$; `// Probability that` q `belongs to the user` u
for $q_i \in q^+$ **do**
    $\hat{u}, \hat{p} = \text{SVM}(q_i)$;
    if $\hat{p} > p_{max}$ **then**
        $q \leftarrow q_i$;
        $u \leftarrow \hat{u}$;
        $p_{max} \leftarrow \hat{p}$;
return (q, u);

their requester. For that purpose, we adapt the unlinkability attack by adding an extra class for fake queries. As mentioned in Section 3.3.3, the adversary might be able to generate a set of fake queries. This set of fake queries is added to the training set to train the SVM classifier. The SVM classifier is then able to identify if a given query q_i belongs to a valid user $\hat{u}$ or is a fake query. Therefore, the attack retrieves the most probable class corresponding to the protected query. If the class corresponds to a user $\hat{u}$, the query is considered as a user query sent by the user $\hat{u}$. Otherwise, the query is considered as a fake query. We summarize the attack with Algorithm 7. The complexity of this algorithm is $O(n)$, where n is the number of features in the SVM classifier.

Algorithm 7: Machine learning attack against TMN over an unlinkability solution.

input : q^+: a protected query,
SVM: SVM model obtained after the training phase.

$\hat{u} = \text{SVM}(q^+)$;
if $\hat{u} \neq \mathscr{C}_{fakeQuery}$ **then**
    return $\hat{u}$;
else
    return $\emptyset$; `//` q^+ `is a fake query`

3.3.4 Evaluation Metrics

To measure the efficiency of the previous attacks, we consider the precision and the recall. The precision is the fraction of retrieved instances that are relevant, while the recall is the fraction of relevant instances that are actually retrieved. For instance, considering the attack against anonymous solutions, the precision is the fraction of de-anonymized queries that are correctly de-anonymized by the attack while the recall is the fraction of queries correctly

de-anonymized by the attack. In our context, they are defined as follows:

$$\text{precision} = \frac{1}{|U|} \sum_{u \in U} \frac{TP_u}{TP_u + FP_u},$$
$$\text{recall} = \frac{1}{|U|} \sum_{u \in U} \frac{TP_u}{TP_u + FN_u},$$

where U is the set of users in the system, TP_u is the number of true positives for the user u (i.e., the number of queries issued and successfully retrieved by the adversary), FP_u is the number of false positives for the user u, and FN_u is the number of false negatives for the user u. The interpretation of FP_u and FN_u is specific to each solution:

- for unlinkability solutions, FP_u is the number of queries issued by a user v but considered by the attack as sent by $u \neq v$ and FN_u is the number of queries sent by u but considered by the attack as sent by the user $v \neq u$.
- for TMN, FP_u is the number of fake queries sent by u but identified by the attack as real queries, and FN_u is the number of real queries sent by u but identified by the attack as fake queries.
- for GooPIR, FP_u and FN_u are the number of queries sent by u for which the attack identifies a fake query as a real one. In a unary classification problem, if a real query is identified as a fake one, necessarily a fake one is identified as a real one. Therefore, FP_u equals FN_u.
- for TMN over an unlinkability solution, FP_u is the number of fake queries issued by the user u but identified by the attack as real queries sent by u plus the number of queries issued by $v \neq u$ but considered by the attack as real queries sent by u, and FN_u is the number of real queries sent by u but identified by the attack as fake queries or as issued by $v \neq u$.
- for GooPIR over an unlinkability solution, FP_u is the number of queries sent by u for which the attack identifies a fake query as a real one plus the number of queries issued by $v \neq u$ but considered by the attack as real queries sent by u, and FN_u is the number of queries sent by u for which the attack identifies a real query as a fake query or as issued by $v \neq u$.

Following the discussion about macro-averaging and micro-averaging made in Section 2.1.4.1, we define the precision and the recall with the macro-averaging version. In our context, it makes more sense to aggregate these metrics per user and analyze the robustness per user (macro-averaging), than consider all queries independently and obtain a result that gives more weight on highly active users (micro-averaging). The precision and the recall can be combined in a single metric, the F1-score, defined as the harmonic mean of the precision and the recall:

$$\text{F1-score} = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$$

3.3.5 Query Similarity Metric Selection

As presented in Section 3.2.1 (see Algorithm 1), SimAttack computes the similarity between two queries: a protected query and a query in the user profile.

Table 4
Dice's coefficient maximizes the efficiency of the SimAttack on the dataset `AOL100`.

Metric	Recall	Precision	F1-score
Jaccard distance	31.4%	36.9%	33.9%
Dice's coefficient	36.7%	42.9%	39.6%
Cosine similarity	36.2%	42.4%	39.0%

A query is represented in a Vector Space Model. Therefore, computing the similarity between two queries is computing the similarity between two vectors. We denote by $\boldsymbol{a}$ the binary vector corresponding the protected query and by $\boldsymbol{b}$ the binary vector corresponding to a query stored in the user profile. We present three well-known metrics used in the literature to compute similarity between two vectors:

- **Jaccard index** [140] — The botanist Jaccard introduced the Jaccard index to compare species in the vegetation of the Alps (France, Germany, Switzerland). This index, also known as Tanimoto similarity, is expressed by:
$$J(\boldsymbol{a}, \boldsymbol{b}) = \frac{\boldsymbol{a} \cdot \boldsymbol{b}}{||\boldsymbol{a}||^2 + ||\boldsymbol{b}||^2 - \boldsymbol{a} \cdot \boldsymbol{b}}.$$
- **Dice's coefficient** [138] — Another botanist Dice introduced a coefficient to evaluate the degree to which two different species are associated in nature. To avoid a coefficient that uses one of the two species as a base, Dice defined the following coefficient:
$$D(\boldsymbol{a}, \boldsymbol{b}) = 2 \cdot \frac{\boldsymbol{a} \cdot \boldsymbol{b}}{||\boldsymbol{a}||^2 + ||\boldsymbol{b}||^2}.$$
- **Cosine similarity** [139] — The cosine similarity measures the angle between the two vectors $\boldsymbol{a}$ and $\boldsymbol{b}$. It is defined as:
$$C(\boldsymbol{a}, \boldsymbol{b}) = \frac{\boldsymbol{a} \cdot \boldsymbol{b}}{||\boldsymbol{a}|| \cdot ||\boldsymbol{b}||}.$$

As we assume that users do not consider the word order in their queries, we omit in our experiments other advanced metrics (e.g., the Hamming distance, the Levenshtein distance).

We execute SimAttack to analyze which metric gives the best performance. For the evaluation, we consider a smoothing factor α equal to 0.5 (in the next section, we show that the smoothing factor does not strongly influence the results). We summarize in Table 4, the recall, the precision, and the F1-score for queries contained in the dataset `AOL100`. We note that the maximum recall and the maximum precision are obtained with the Dice's coefficient (respectively, 36.7% and 42.9%). Similar results have been obtained on the seven other datasets (i.e., `AOL200` to `AOL15000`). Therefore, we configure SimAttack in the remaining evaluations with the Dice's coefficient.

3.3.6 Smoothing Factor α

SimAttack uses a smoothing factor to compute a similarity metric between a query and a user profile. We first discuss the relevance of the exponential smoothing to compute this metric. Then, we analyze the impact of the smoothing factor α on the similarity metric. Finally, we study the influence of the smoothing factor α on the results of SimAttack.

3.3.6.1 Relevance of the Exponential Smoothing

To break private Web search solutions, SimAttack computes a similarity metric between a query and all queries contained in a user profile. As established in the previous section, the similarity metric is the Dice similarity. Therefore, we obtained a Dice similarity for each query contained in the user profile. These Dice similarities are then aggregated in a single similarity value through the exponential smoothing. This technique computes an average over all the Dice similarities by giving to each of them a different weight:

$$expSmooth = \alpha\big[\sum_{i=0}^{t-1}(1-\alpha)^i x_{t-i}\big] + (1-\alpha)^t x_0,$$

where $\{x_i\}_{i\in[\![0,t]\!]}$ are the similarity values and t the number of queries contained in the user profile. The weights are parametrized with the smoothing factor α.

To better understand the relevance of the exponential smoothing, we depict in Figure 5 the Dice similarities before computing the exponential smoothing (considering AOL100). More precisely, we depict the mean over all queries in the testing set of the 30 highest Dice similarity obtained with their user profile P_u and with the other users profiles $P_v \neq P_u$. We represent the distribution of values obtained for the other user profiles through its maximum, mean, and median. Figure 5 shows that few queries have a Dice similarity different from 0. They correspond to queries previously issued for which at least one keyword is also used in a protected query. Compared to queries that do not share any keywords with the protected query, these queries are highly relevant as they convey information on the proximity between the protected query and the user profile. Therefore, to obtain a single relevant similarity metric out of all Dice similarities, these queries should be weighted differently in the average computation. For that reason, we employ the exponential smoothing. It gives greater weights on higher similarity values than lower ones. These weights are parametrized with the smoothing factor α.

Furthermore, Figure 5 indicates that the requester profile contains a greater proportion of queries with a high Dice similarity than the other user profiles. This gives the intuition why SimAttack is able to retrieve the correct user profiles among all available user profiles. In particular, we note the clear distinction between the correct user profile and the average over all other user profiles.

From this figure, we also notice a huge disparity between user profiles. The large difference between the median and the best of all other profiles points

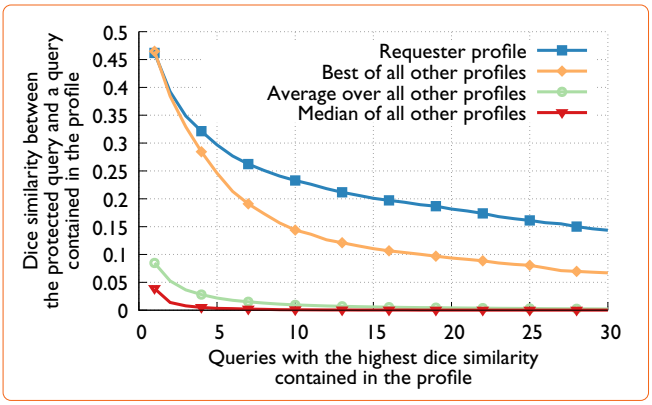


Figure 5

More queries in the requester user profile are similar to the protected query than the ones contained in other profiles.

out this disparity. In addition, the median indicates that, for half of the user profiles, almost no query has a keyword in common with the protected query. It explains why the similarity metric with a user profile is in general close to 0.

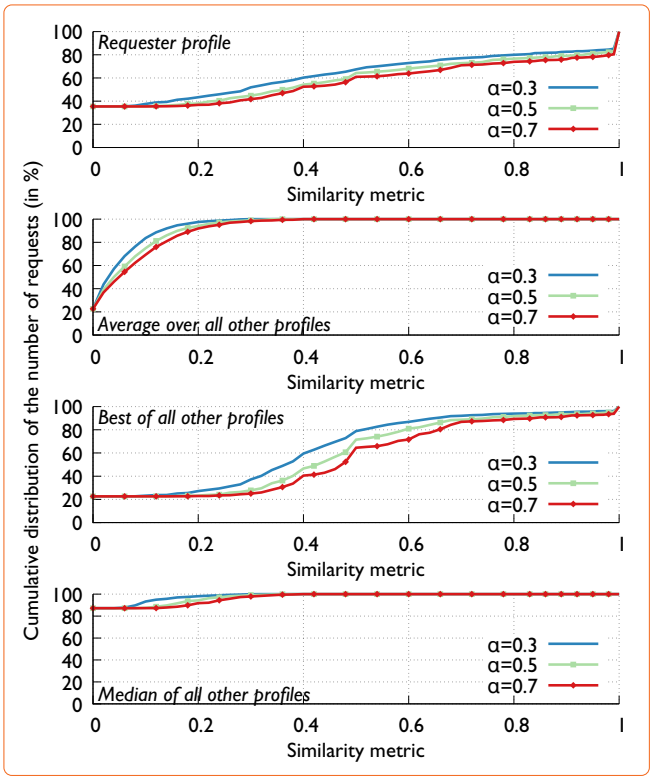
3.3.6.2 Impact of the Smoothing Factor α on the Similarity Metric

The exponential smoothing weighs the Dice similarity values differently according to the smoothing factor α. In this section, we study the impact of α on the similarity metric computed by SimAttack. Figure 6 gives the cumulative distribution of the number of queries according to the similarity metric computed by SimAttack. We show the results for different values of α (i.e., 0.3, 0.5, and 0.7). We consider in our experiments the requester user profile, and the mean, the best and the median over all other user profiles. We note that the smoothing factor α does not have a strong influence on the results. Besides, for small and high similarities, the value of α has almost no influence on the distribution (the curves are placed on top of each other).

Furthermore, as depicted by Figure 6, 14.8% of queries have a similarity with their requester user profile higher than 0.99. One can wonder how it is possible for a query to have a similarity metric almost equal to 1. If we look in detail at these queries, it appears that most of the time, they are one-word queries with a term frequently used by the user (e.g., "google" or "amazon.com"). To better understand how the value of the similarity metric can almost equal 1, we imagine a fictive user profile with 400 queries and consider a one-word protected query. Among the 400 queries, we assume that the protected query appears 20 times. Consequently, if we compute the Dice similarity between the protected query and all the queries in the user profile, we obtain a distribution with 20 times 1 and 380 times 0. Then, by applying the exponential smoothing we end up with a similarity metric equal to 0.99920207733 and 0.999999999965 for a smoothing factor α respectively

Figure 6

The smoothing factor α does not have a strong influence on the similarity metric computed by SimAttack.



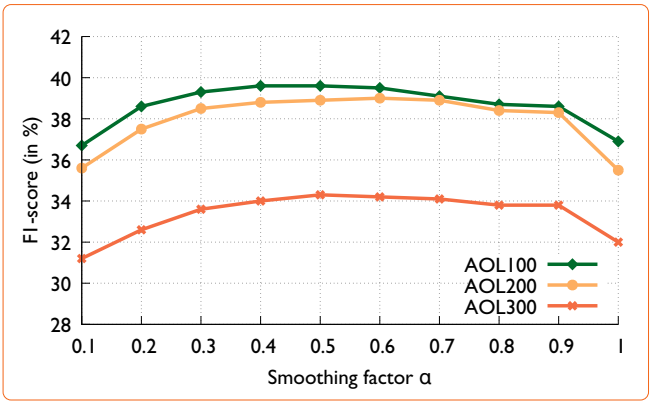


Figure 7

The smoothing factor α does not have a significant impact on the F1-score of SimAttack.

equal to 0.3 and 0.7. As a consequence, even though the protected query does not appear so frequently in the user profile, the similarity between the protected query and the user profile can be close to 1.

3.3.6.3 Influence of the Smoothing Factor on the Results of SimAttack

Finally, we discuss the choice of the smoothing parameter α for our evaluations. As detailed in the previous section, α does not strongly influence the value of the similarity metric. We are interested to see if the value of the parameter α has an impact on the attack. We report in Figure 7 the results of SimAttack (F1-score) obtained for different values of α on three datasets: `AOL100`, `AOL200`, `AOL300`. We show that α has a limited impact on the performance of SimAttack. For instance, for the dataset `AOL100`, the F1-score only varies from 36.9% to 39.6% over the entire range of variation of α. The same result was obtained with the five other datasets (i.e., `AOL500` to `AOL15000`). Furthermore, as the best F1-score is observed on average for α equals 0.5, we set the parameter α to 0.5 in the remaining evaluations.

3.4 Evaluation

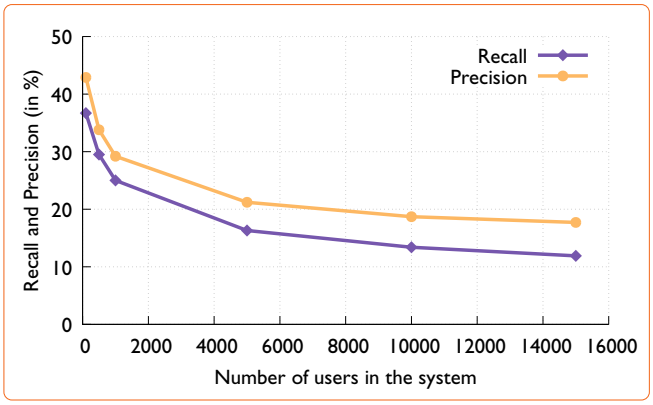
In this section, we provide an evaluation of the robustness of the main state-of-the-art solutions presented in Section 3.3.2: unlinkability solutions, TrackMeNot, and GooPIR. To quantify the results of the attacks we use the recall, the precision, and the F1-score as presented in Section 3.3.4. We first present the results regarding the privacy protection, and then compare the execution time of SimAttack with its concurrent machine learning attack published in the literature.

3.4.1 Privacy Protection

In this section, we evaluate the privacy protection obtained with SimAttack for unlinkability solutions, TrackMeNot, and GooPIR. We compare these results to the ones obtained with its corresponding machine learning attack. In addition, as the three aforementioned solutions fail to properly protect user queries (results presented in the first three sections), we carry out two further experiments which combine unlinkability and indistinguishability. We first study TrackMeNot over an unlinkability solution and then GooPIR over an unlinkability solution.

Figure 8

Increasing the number of users in the system decreases both the recall and the precision of SimAttack.



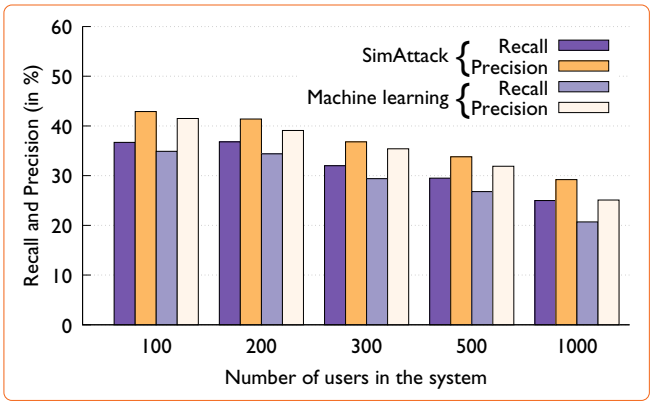
3.4.1.1 Unlinkability Solutions

We evaluate the capacity of SimAttack to de-anonymize queries (i.e., Algorithm 2 presented in Section 3.2.2). Figure 8 depicts the recall and the precision of SimAttack for a different number of users. For instance, the recall of SimAttack is 36.7% (considering 100 users); this means that SimAttack de-anonymizes 36.7% of user queries. Besides, the precision of SimAttack is 42.9%; this means that among the queries de-anonymized by SimAttack, 42.9% of queries are correctly de-anonymized. Furthermore, the results show that the recall and the precision decrease when the number of users in the system increases. For instance, SimAttack de-anonymizes 36.7% of queries with a precision of 42.9% if we consider 100 users, while for 15,000 users, it only de-anonymizes 11.9% of queries with a precision of 17.7%. Nevertheless, this decrease is not linear and tends to stabilize for a number of users equal to 15,000.

We now compare the performance of both SimAttack and ML Attack against unlinkability solutions. Figure 9 measures for the two attacks the precision and the recall for different numbers of users in the system. The results show that both attacks have a comparable recall and precision but overall SimAttack is slightly better. On average, the recall and the precision of SimAttack are respectively 2.8% and 2.2% higher than the ones obtained with the machine learning attack. Nevertheless, if we look in detail, the difference between the two attacks increases according to the number of users. For instance, if we consider 100 users in the system, the recall of SimAttack is 1.8% higher, while for 1,000 users the difference between the recall of the two attacks is 4.3%. For the precision, the differences are respectively 1.4% and

Figure 9

SimAttack and ML Attack have a similar recall and precision.



4.1%. Consequently, according to our experiments, SimAttack is better than ML Attack especially for a high number of users.

Appendix B contains further experiments on SimAttack for unlinkability solutions. We study different parameters that also influence its results: the size of the user profiles, the number of users retrieved for a given anonymized query (top-p instead of top-1), and the number of user profiles owned by the adversary.

3.4.1.2 TrackMeNot

In this section, we evaluate the capacity of SimAttack to distinguish fake queries generated by TrackMeNot from real user queries (i.e., applying Algorithm 3 presented in Section 3.2.3 on TMN). We conduct our experiment with the dataset `TMN100`. As presented in Section 3.3.1.4, `TMN100` contains the 100 most active users protected by TMN that generates 60 queries per hours (the default TMN setting). The number of users do not influence the results of SimAttack, as knowing the identity of the requester, the adversary performs the attack on each user separately. As described in Section 3.2.3, the adversary might have a prior knowledge about the RSS feeds used by TMN to generate the fake queries. Therefore, we design two versions of SimAttack: one that have no information about the RSS feeds and one that exploit this knowledge. The former considers queries close to their requester user profile as real queries (otherwise, they are considered as fake). The notion of proximity with the user profile is defined by the threshold δ: Queries with a similarity metric greater than the threshold δ are considered as real queries (otherwise, they are considered as fake ones). Figure 10 presents the recall, the precision, and the F1-score of SimAttack for several values of δ (i.e., considering that the adversary does not have any information about the RSS feeds used by TMN). The results show that the value of δ significantly impacts the performance of SimAttack. The best results in terms of F1-score are obtained for a δ equal to 0.5. For such a threshold, an adversary is able to identify 36.8% of initial queries with a precision of 62.4%. Nevertheless, while the value of δ can be adjusted to achieve a higher precision, it will be, however, at the cost of decreasing the recall. For instance, for a threshold δ equals to 0.2, the precision is 60.9% but the recall is only 10.6%.

We now analyze the robustness of TMN considering an adversary that knows the RSS feeds used by TMN. Under such a configuration, SimAttack does not need a threshold δ. It takes advantage of the RSS feeds to distinguish fake queries from the user queries. As explained in Section 3.2.3, δ is equal to the similarity metric between the query and a user profile composed of fake

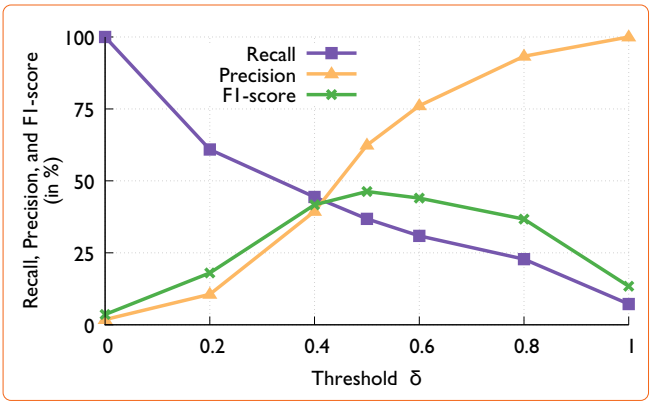


Figure 10

The value of δ significantly impacts the performance of SimAttack.

Table 5

SimAttack performs better considering an adversary with prior knowledge about RSS feeds.

Attack	Recall	Precision	F1-score
With RSS feeds	45.3%	87.1%	61.0%
Without RSS feeds ($\delta = 0.5$)	36.8%	62.4%	46.3%

queries. Table 5 lists the performances of SimAttack in terms of precision, recall, and F1-score in that case. The results show that SimAttack is able to identify 45.3% of initial results with a precision of 87.1%. Compared to the results previously obtained when no RSS feeds were considered, having such a knowledge on the RSS feeds significantly increases the recall and the precision (45.3% versus 36.8% for the recall, and 87.1% versus 62.4% for the precision). Consequently, knowing the RSS feeds used by TMN to generate fake queries is real advantage. As a consequence, users should not use the default RSS list provided by TMN, as the list is publicly known and thus can be exploited by an adversary.

In addition, we compare the efficiency of SimAttack with ML Attack against TMN. As mentioned in Section 3.3.3.2, two machine learning algorithms can be used: Support Vector Machine and Logistic Regression. Table 6 lists the precision, the recall, and the F1-score for the two classifiers. The results illustrate that compared to the logistic regression classifier, the SVM classifier provides better performance in terms of precision while achieving a slightly lower recall. Overall, the F1-score obtained by the SVM classifier (57.8%) is much higher than the one obtained with the logistic regression (37.4%). Interestingly, the machine learning attack published in the literature [48] did not consider the SVM classifier, as its authors "*neglect other better classifiers like SVM, so as to estimate the lowest accuracies that can be achieved*" [150]. Looking for the lowest accuracies is surprising, and our study effectively shows that SVM performs better than logistic regression. Moreover, we are interesting to compare these results with SimAttack. As ML Attack requires prior knowledge on the RSS feeds, we consider the version of SimAttack that uses the same knowledge. Table 5 gives the results for SimAttack. We note that SimAttack provides a lower recall compared to ML Attack with both classifiers (45.3% versus 54.2% and 46.0% for logistic regression and for SVM, respectively). But, in terms of precision, SimAttack outperforms ML Attack (87.1% versus 29.8% and 77.8% for logistic regression and SVM, respectively). Besides, analyzing the recall and the precision together, SimAttack is much better than the two ML Attacks (its F1-score equals 61.0% versus 37.4% and 57.8% for the logistic regression and the SVM, respectively).

Finally, TMN allows users to set up the number of fake queries (as discussed in Section 3.3.2.2). By default, it sends "60 fake queries per hour". To analyze the impact of the number of fake queries, we assess the robustness of TMN with varying numbers of fake queries sent per hour (i.e., 1, 10, 30, and 60 fake queries per hour). These different settings add 248; 2,480; 7,440 and 14,880 fake queries per user, respectively. Figure 11 depicts the precision and the recall provided by SimAttack both with and without using the RSS feeds. The results show that the recall remains unchanged regardless of the number of fake queries (i.e., 36.8% and 45.3% with and without using the RSS

Table 6

Performance of the machine learning classifiers on queries protected by TrackMeNot.

Classifier	Recall	Precision	F1-score
Logistic Regression	54.2%	29.8%	37.4%
Support Vector Machine	46.0%	77.8%	57.8%

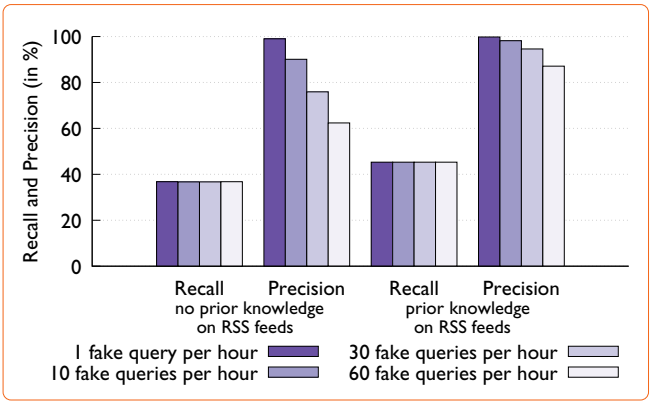


Figure 11

The number of fake queries sent by TrackMeNot only impacts the precision of SimAttack.

feeds, respectively). Indeed, changing the number of fake queries sent per hour does not affect the user profiles, and thus the same proportion of initial queries is retrieved. However, the results show that the precision decreases as the number of fake queries increases (for 1 fake query per hour, the precision is respectively 99.0% and 99.8% with and without exploiting the RSS feeds, while for 60 fake queries per hour, the precision is 62.4% and 87.1%). Indeed, increasing the number of fake queries increases accordingly the number of misclassified fake queries and therefore reduces the precision. Consequently, sending too few fake queries allows the adversary to retrieve initial queries with high precision (almost 100% of precision for one fake query per hour). Therefore, to enforce the protection, users should send a high number of fake queries. But, as shown by Figure 11, even though they selection 60 fake queries per hour, the adversary is able to obtain a relatively high recall (62.4% and 87.1% with and without exploiting the RSS feeds).

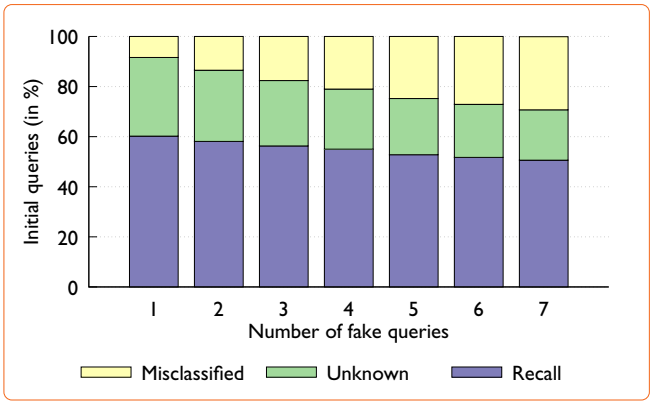
Appendix B contains a complementary experiment on SimAttack for TMN. We study the influence that the number of queries in the user profiles has on the efficiency of SimAttack. The results show that larger user profiles produce better results.

3.4.1.3 GooPIR

GooPIR obfuscates a query by adding to the original query k fake queries; all queries are separated by the logical OR operator. SimAttack tries to identify which query among the $(k+1)$ sub-queries is the initial one. In this section, we assess the capacity of SimAttack to retrieve the initial query. Our experiments use the dataset `AOL100` for which GooPIR generated up to seven fake queries. The number of users do not influence the results, as the attack is performed on each user separately. Figure 12 presents the performance of SimAttack for varying numbers of fake queries (from one to seven). The recall represents the proportion of obfuscated queries for which SimAttack correctly retrieves the initial query. We do not display the precision, as an attack against GooPIR is a unary classification problem and thus, for that specific case, the precision equals the recall. Besides, to better understand the outcomes of the attack, we also categorize queries as Misclassified and Unknown. *Misclassified* represents the proportion of obfuscated queries for which SimAttack retrieves a fake query as initial query, and *Unknown* represents the proportion of obfuscated queries for which SimAttack is not able to classify any query as initial query (i.e., the similarity of all $(k+1)$ queries with the user profile equals zero. The results show that the number of fake queries generated by GooPIR has a limited impact on the privacy protection of the

Figure 12

The number of fake queries generated by GooPIR has a limited impact on the privacy protection of the user.



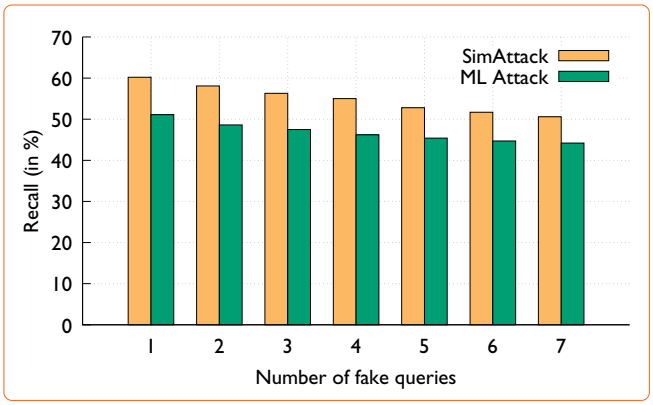
user: SimAttack retrieves 60.2% of initial queries when only one fake query is generated while 50.6% of initial queries are retrieved for seven fake queries generated. Besides, regardless of the number of fake queries, the percentage of initial queries retrieved by SimAttack remains relatively high (more than half of initial queries are identified).

Furthermore, we study the reason why some queries are not identified by SimAttack. These queries are classified as *Misclassified* and *Unknown* (see Figure 12). The results show that the proportion of queries in these two categories changes according to the number of fake queries. For instance, for one fake query, unknown queries represent 78.9% of non-identified queries while misclassified queries represent 21.1%. If we consider seven fake queries, these percentages change to 40.8% and 59.2%, respectively. From the high proportion of unknown queries (20.1% for seven fake queries), we deduce that a lot of initial queries have a similarity equal to 0 with their requester user profile. This means that 20.1% of user queries contained keywords that had never been used by the user. The non-negligible proportion of unknown queries (20.1%) also indicates that the quality of the fake queries generated by GooPIR is not satisfactory as they often have a similarity with the user profile equal to zero. Consequently, increasing the number of fake queries maximizes the probability to have at least one fake query that is slightly related to the user profile.

Finally, we compare the performance of SimAttack with ML Attack against GooPIR. It is an extension of the previous machine learning attacks (see Section 3.3.3.3) that we create to establish a baseline and compare the results obtained with SimAttack. Figure 13 measures the recall of the two attacks for a number of fake queries varying between one and seven. The results

Figure 13

The number of fake queries generated by GooPIR has a limited impact on the privacy protection of the user.



show that on average SimAttack retrieves 8.1% more queries that ML Attack. Nevertheless, this difference is not homogenous: for one fake query, SimAttack retrieves 9.1% more queries while for seven fake queries, the difference between the two attacks concerns only 6.4% of queries. Overall, SimAttack retrieves a higher proportion of initial queries.

We study in Appendix B the influence that the number of queries in the user profiles has on the efficiency of SimAttack. We show that larger user profiles make SimAttack retrieves a higher number of queries.

3.4.1.4 TMN Over an Unlinkability Solution

In this section, we evaluate the performance of SimAttack against TrackMeNot over an unlinkability solution. In this hybrid solution, users send their real queries and the TMN fake queries anonymously. An adversary has to distinguish between user queries and fake ones, and then she has to identify the user that requested the real queries. We analyze with SimAttack the robustness of TMN over an unlinkability solution considering that the adversary has no information about the RSS feeds used by the users. Figure 14 presents the recall, the precision, and the F1-score for several values of δ and considering the `TMN100` dataset. The results show that δ significantly impacts the performance of SimAttack: the F1-score varies from 6.4% to 27.3% and gives its best value for δ equals 0.8. For this value, an adversary is able to identify 20.5% of real queries with a precision of 40.7%. Compared to the results obtained with TrackMeNot alone (see Table 5), adding the unlinkability solution decreases the recall by 16.3% (20.5% versus 36.8%) and the precision by 21.7% (40.7% versus 62.4%). Indeed, as the adversary does not know the identity of the user, a lot of queries are now misclassified. In addition, compared to an unlinkability solution alone (see Figure 8), combining TMN with an unlinkability solution decreases the percentage of real user queries successfully de-anonymized by SimAttack from 36.7% to 20.5%. Consequently, using TMN with an unlinkability solution protects 16.2% more queries than an unlinkability used alone.

We next analyze the performance of SimAttack when the adversary leverages prior knowledge on the RSS feeds used by the users. This knowledge helps the adversary from distinguishing fake queries from real queries. Table 7 lists the recall, the precision, and F1-score obtained by the attack. The results show that SimAttack succeeds in identifying 35.4% of the user's queries with a precision of 14.7%. Compared to the results of TrackMeNot alone (see Table 5), adding the unlinkability solution improves the user protection, as 9.9% more real user queries are not identified by SimAttack. However, the

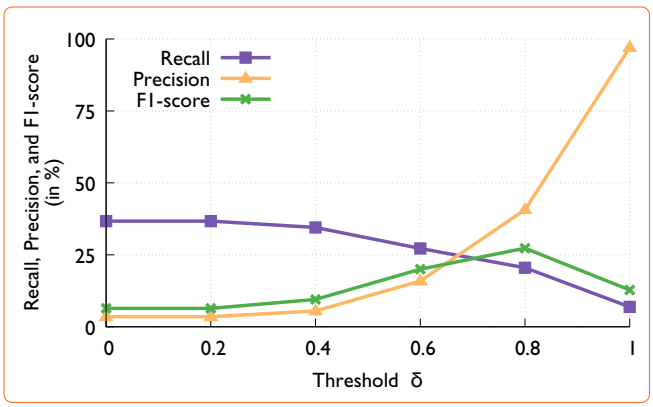


Figure 14
The value of δ significantly impacts the performance of SimAttack against TMN over an unlinkability solution.

Table 7

Performance of the privacy attack considering TMN over an unlinkability solution and the exploitation of the RSS feeds.

Attack	Recall	Precision	F1-score
SimAttack	35.4%	14.7%	20.7%
ML Attack	33.0%	11.4%	16.9%

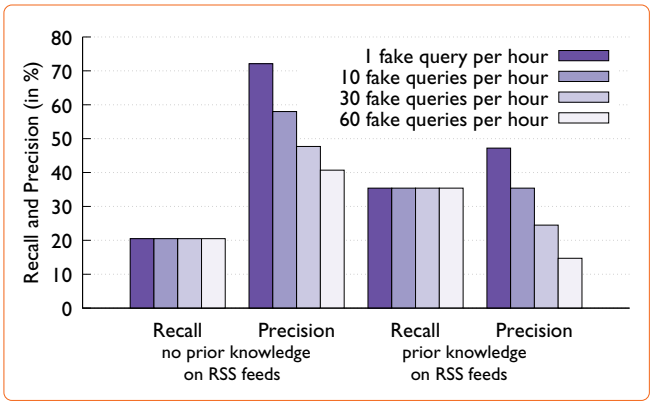
precision has significantly dropped: from 87.1% to 14.7% without and with the unlinkability solution, respectively. The decrease of the precision is a direct result of the high ratio of misclassified fake queries. Indeed, due to the unlinkability solution, a lot of fake queries is not identified as such but is considered as being issued by one of the users in the system. To better understand this ratio, we compare the precision obtained by SimAttack against an unlinkability solution alone (see Figure 8). Introducing the TMN fake queries decreases the precision from 42.9% to 14.7%. This result illustrates that the TMN fake queries strongly decrease the precision.

Furthermore, we compare the performance obtained with SimAttack depending if the adversary knows the RSS feeds employed by TMN. We note from Figure 14 and Table 7 that using the RSS feeds increases the recall by 14.9% (35.4% versus 20.5%) but decreases the precision by 26.0% (14.7% versus 40.7%). Overall, exploiting the RSS feeds gives lower results: the F1-scores are respectively 20.7% and 27.3% with and without exploiting the RSS feeds. Interestingly, this result differs from TMN alone as for the latter, exploiting the RSS feeds dramatically increases the performance of SimAttack (as depicted on Figure 5, the F1-scores obtained for TMN alone are 61.0% and 46.3% with and without the RSS feeds, respectively). It is surprising that for TMN over an unlinkability solution, the exploitation of more data produces lower results. If we look in detail, we note the there is a huge different in terms of precision (14.7% when the RSS feeds are exploited, otherwise 40.7%). Therefore, we deduce that exploiting the RSS feeds results in misclassifying more fake queries than the non-exploitation of the RSS feeds. Indeed, in the non-exploitation of the RSS feeds, SimAttack identifies fake queries with the threshold δ equals to 0.8. This high value prevent the adversary from misclassifying a lot of fake queries (i.e., associating a fake query to a real user).

In addition, we compare the efficiency of SimAttack against ML Attack. Table 7 exhibits the recall, the precision and the F1-score for the two attacks. The results illustrate that SimAttack better identifies real queries: the recall and the precision of SimAttack are respectively 2.4% and 3.3% higher than ML Attack. This is confirmed by the F1-score which is 3.8% higher for SimAttack than the one obtained for ML Attack (20.7% versus 16.9%).

Figure 15

The number of fake queries sent by TrackMeNot over an unlinkability solution does not change the recall but strongly impacts the precision of SimAttack.



Finally, we evaluate the protection offered by TMN over an unlinkability solution according to the number of fake queries periodically sent by TMN. Figure 15 presents the recall and the precision for SimAttack with and without prior knowledge on RSS feeds. The results show that the number of fake queries does not change the recall (i.e., 35.4% and 20.5% with and without using prior knowledge on RSS feeds, respectively). However, the precision strongly depends on the quantity of fake queries sent per hour. Indeed, decreasing the number of fake queries from 60 to 1 fake query per hour, increases the precision by 31.4% and 32.5% (for SimAttack without and with exploiting the knowledge on the RSS feeds, respectively). Consequently, if the user wants a proper protection, TMN has to send a high number of fake queries per hour.

In Appendix B, we present an additional experiment on the size of the user profiles. The results show that the number of queries contained in the user profiles impact the results of SimAttack: more queries in the user profiles result in identifying a higher proportion of queries.

3.4.1.5 GooPIR Over an Unlinkability Solution

We assess the efficiency of SimAttack against a solution that combines GooPIR and an unlinkability solution. Such a protection mechanism obfuscates the user query with k fake queries and sends the protected query anonymously. Therefore, SimAttack aims at retrieving (i) the user query among the $(k+1)$ queries, and (ii) the identity of the user who issued the protected query (among all the users in the system). Figure 16 shows the percentage of initial queries retrieved by SimAttack for a varying number of fake queries (from one to seven). The results show that the number of fake queries has a limited impact on the user protection. Changing from one to seven fake queries protects 5.1% more queries (the recall drops from 32.7% to 27.6%, respectively). In addition, compared to the results of GooPIR alone (see Figure 12), the current percentage of initial queries identified by SimAttack is 23% lower for seven fake queries. Initially GooPIR protects 50.6% of queries, while on top of an unlinkability solution, it protects 27.6% of queries (for seven fake queries). Therefore, adding the unlinkability solution on top of GooPIR helps to dramatically improve the user protection. But, the recall achieves for seven fake queries (27.6%) remains relatively high, especially because queries are protected by two independent private Web search solutions.

Furthermore, we compare the efficiency of SimAttack with ML Attack (Algorithm 6 presented in Section 3.3.3.4). Figure 16 presents the results of the two attacks for a different number of fake queries (from zero to seven).

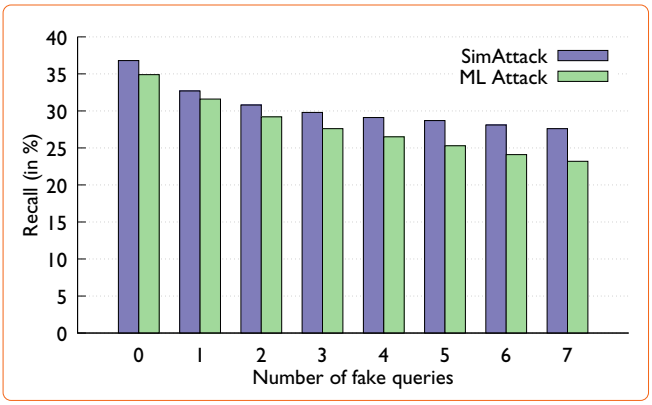


Figure 16
The number of fake queries has a limited impact on the protection of queries.

On average, SimAttack retrieves 2.7% more queries than ML Attack. Nevertheless, the difference between the recall of the two attacks evolves with the number of fake queries: For one fake query, SimAttack retrieves 1.1% more queries than the machine learning attack while for seven fake queries the difference increases to 4.4%. Consequently, besides a better identification of fake queries, SimAttack is more robust to an increase of the number of fake queries.

Appendix B introduces two further experiments for SimAttack against GooPIR over an unlinkability solution. The results show first that the efficiency of SimAttack depends on the number of queries contained in the user profile (a higher number increases its performance). Then, we also study the influence of the number of pair $(query, user)$ returned by SimAttack. In this section, we consider one pair (the most probable one), but to maximize the results of the attack an adversary might be interested in increasing this number. We show empirically that the gain obtained by increasing the number of pairs retrieved by SimAttack is relatively low.

3.4.2 Execution Time

In this section, we compare the execution time of SimAttack with its corresponding machine learning attack. The comparison involves unlinkability solutions, TrackMeNot, and GooPIR.

3.4.2.1 Unlinkability Solutions

We compare the execution time of SimAttack against the concurrent machine learning approach. More precisely, we evaluate their execution time for a varying number of users in the system. Figure 17 depict the execution times from 100 users to 1,000 users using a logarithmic scale. The results show that the execution time for SimAttack increases linearly with respect to the number of users while ML Attack evolves exponentially. In addition, SimAttack is much faster than ML Attack, especially for a high number of users. For 1,000 users, the machine learning attack takes 434,322 seconds (including the time to train models) while SimAttack takes 1,598 seconds, 271 times faster. Without considering the time required to train the classifier (as the adversary might exploit the same classifier for multiple attacks), SimAttack still remains 158 times faster than ML Attack. This huge difference is partially explained, as SimAttack exploits the eight cores available on the tested architecture (contrary to ML Attack). Besides, ML Attack is implemented through Weka that is not optimized for a large number of features and data.

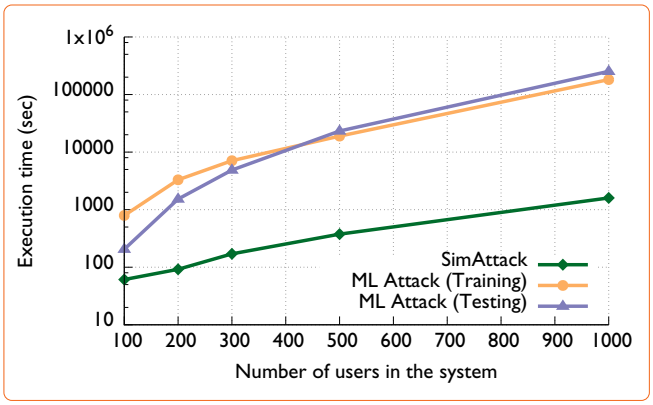


Figure 17
SimAttack is faster than the machine learning attack, especially for a high number of users.

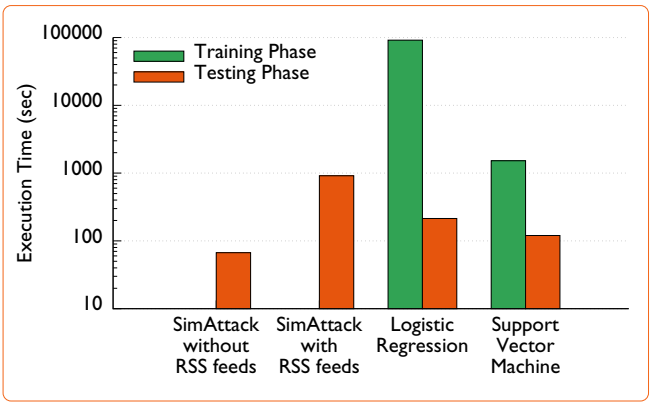


Figure 18

Considering TMN, SimAttack identifies user queries faster than the machine learning approach for both classifiers.

3.4.2.2 TrackMeNot

We compare the execution time of SimAttack on TMN with its concurrent machine learning approach on the `TMN100` dataset. As ML Attack requires the RSS feeds to perform, it makes more sense to compare it with the version of SimAttack that exploits the RSS feeds. But we also include in our comparison the version of SimAttack that does not exploit the RSS feeds. The results are reported in Figure 18 using a logarithmic scale to depict the execution time. We show that SimAttack is faster than ML Attack, especially when ML Attack is based on the logistic regression classifier. For instance, SimAttack (with RSS feeds) takes 914 seconds to execute while logistic regression and SVM take respectively 91,727 seconds and 1,644 seconds (considering the training and testing phases). But Figure 18 also indicates that the training phase is responsible for these large execution times. In consequence, if we only consider the testing phase (considering that the adversary can reuse multiple times the same classifier), ML Attack is faster than SimAttack: SimAttack (with RSS feeds) requires 914 seconds to execute while ML Attack takes 214 seconds and 120 seconds considering respectively the testing phase obtained with the logistic regression and SVM classifiers.

Nevertheless, we note that the execution time of SimAttack without considering the RSS feeds is 67 seconds. Consequently, not considering the RSS feeds makes SimAttack 13.6 times faster (67 seconds versus 914 seconds). Indeed, by considering the RSS feeds, SimAttack engenders extra computations, as it determines the similarity metric between each query and the fake queries generated by the adversary. Therefore, the overhead due to the fake query exploitation depends on the number of fake queries generated by the adversary. Furthermore, we note that, by not exploiting the RSS feeds, SimAttack is much faster than ML Attack: its execution time takes 67 seconds while for ML Attack it takes 214 seconds and 120 seconds for respectively the testing phases of the logistic regression and SVM classifiers.

3.4.2.3 GooPIR

We compare the execution time of SimAttack against queries protected by GooPIR with its concurrent machine learning attack. We measure the execution time of the two attacks considering that GooPIR generates seven fake queries. The results displayed in Figure 19 show that SimAttack is 45.7 times faster than the SVM classifier: it takes 37 seconds for SimAttack to identify the initial queries while, for the SVM classifier, it requires 1,691 seconds (considering the training and the testing phases). More precisely, if we focus on the testing phase (considering that the adversary reuses the same trained clas-

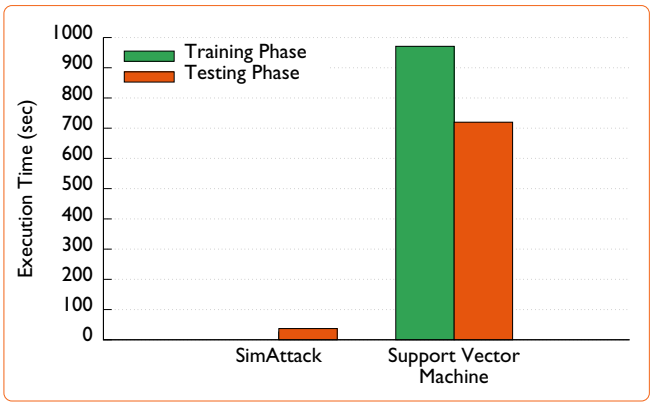


Figure 19
Considering GooPIR, SimAttack performs much faster than the machine learning attack.

sifier for different attacks), SimAttack is still faster as it performs 19.5 faster than the testing phase of the SVM classifier (37 seconds versus 720 seconds).

3.5 Conclusion

We introduced in this chapter SimAttack, a generic attack against different types of private Web search solutions (e.g., Tor, GooPIR, TMN). SimAttack is able to break such protections by computing a similarity metric between queries and user profiles. In the design of SimAttack, we had two initial objectives: (i) retrieve a higher proportion of queries than the concurrent machine learning attack, and (ii) require less computation time than the previous attacks in the literature. We empirically showed using real search queries that SimAttack fulfills these two objectives. Considering unlinkability solutions with 1,000 users, the recall of SimAttack is 4.3% higher than the one obtained with the machine learning attack, while at the same time, SimAttack divides the execution time by 158.

Furthermore, we used SimAttack to analyse the robustness of three representative state-of-the-art solutions (i.e., unlinkability solutions, TrackMeNot, and GooPIR). We established that none of these solutions are satisfactory to protect the user privacy. For instance, considering GooPIR, an adversary retrieved with SimAttack 50.6% of queries (case of seven fake queries).

Nevertheless, SimAttack (as the machine learning attacks) requires an adversary with an external knowledge about users. This assumption is realistic, as search engines collect user queries. Therefore, the external knowledge owned by the adversary is collected before users switch to a private Web search solution (i.e., when users were not using any protection mechanism). In our evaluations, we used the queries issued right after the switch as testing set. As the training set and the testing set contain queries issued in a relatively close period, it could explain the good results we obtained. It is known that user interests change over time [151] (also known as concept drift) and thus, the attack might not be as efficient if the previously-collected queries were issued a long time before the protected user queries. Due to a lack of real data, we were not able to analyze such a situation.

Queries employed in SimAttack are modeled with binary word vectors. This model is a basic representation of queries. For instance, it considers two synonyms as two different concepts. A more advanced representation could lead to a better identification of user queries. Indeed, queries could be enriched by WordNet [82] to take the keyword semantic into consideration. Another method could use Word2Vec [152] to identify keywords that share a common context and represent them under the same feature.

Furthermore, SimAttack distinguishes fake queries from real ones using previously-collected queries. Other criteria could have been investigated to distinguish them. Indeed, automatically generated data is a difficult task that is often not perfect. For instance, GooPIR generates fake queries by aggregating random keywords together. But, as these keywords might never occur together in a reference corpus, such a query might be easily identified as fake. In addition, all keywords employed in fake queries are contained in a dictionary, but not necessarily the ones contained in real queries (e.g., slang words, misspelled keywords). A more advanced attack could exploit these criteria to identify fake queries.

Finally, SimAttack considers queries in a vector space model where each dimension corresponds to a keyword. Consequently, a large number of keywords implies a large dimensional space. This can be an issue, as a large dimensional space increases the computational time of the similarity metric. This can be overcome by a dimensional space reduction (for instance, using Singular Value Decomposition [153]). The same technique could be employed with the machine learning attacks to decrease the number of features.

CHAPTER 4

PEAS, a New Robust and Efficient private Web Search Solution

4.1 Objective

As we have seen in the previous chapter, the existing private Web search solutions are not so satisfactory to protect the user privacy, as an adversary is able to bypass the protection for many protected queries. Even though combining an indistinguishability solution with an unlinkability solution gives better results than the state-of-the-art solutions, an adversary is still able to break a significant proportion of protected queries. Consequently, we aim at designing a new solution that is robust against an adversary that owns prior knowledge about users. But, at the same time, our objective is to have a solution that does not introduce a large overhead in terms of computations and latency. We also want to keep the best possible quality of results. These objectives are important, otherwise the solution will strongly degrade the user experience and users will not see the benefit of using it.

4.2 PEAS in a Nutshell

As already mentioned, unlinkability and indistinguishability solutions are complementary. The former hides the identity of the requester (without modifying the initial query) while the latter only masks the initial query (without faking the user identity). The results of the previous chapter show that to efficiently protect users during their Web search, unlinkability and indistinguishability have to be combined. For that reason, as shown in Figure 20,

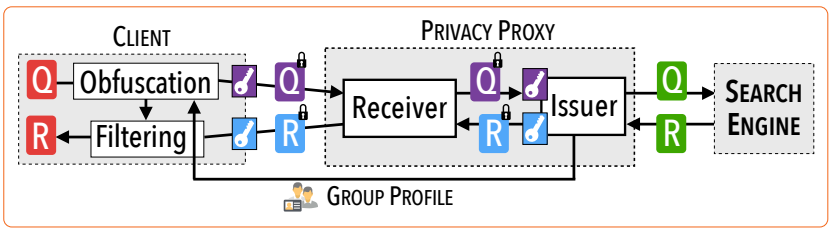


Figure 20

PEAS combines an unlinkability solution provided by the privacy proxy and an indistinguishability solution performed by the client that obfuscates the user queries.

we design PEAS by combining a new efficient unlinkability protocol (i.e., the privacy proxy) with a new accurate indistinguishability mechanism (i.e., obfuscation and filtering).

The privacy proxy is composed of two non-colluding nodes: the *receiver* and the *issuer*. The non-collusion assumption means that, except for a normal execution of the protocol, the two nodes are not allowed to exchange extra information with each other. We also consider these nodes as honest-but-curious, meaning that they strictly follow the protocol but can monitor and exploit information they receive. The privacy proxy works as follows: The receiver gets ciphered queries directly from the clients and forwards them to the issuer which deciphers the queries and forwards them to the search engine. Upon receiving a response from the search engine, the issuer ciphers the results and forwards them to the receiver which forwards these messages to the client. Finally, the client deciphers the message and retrieves the results. In the protocol, the receiver knows the identity of the requesting users (i.e., their IP address) but it is not able to analyze the content of their queries (as they are ciphered by the clients). The issuer, in turn, knows the queries sent by the users (as it is able to decipher the message) but it is not able to identify the requesting users (as it receives the queries from the receiver).

The obfuscation mechanism of PEAS associates user queries with multiple fake queries using the logical OR operator. For instance, PEAS obfuscates the user query q by sending instead the following query:

$$q \text{ OR } \underbrace{fq_1 \text{ OR } \ldots \text{ OR } fq_k}_{\text{k fake queries}},$$

where fq_i is the i$^{\text{th}}$ fake query generated by PEAS for $i \in [\![1, k]\!]$. By doing so, PEAS aims to mislead the adversary about the initial query. Nevertheless, this approach can be inefficient if fake queries are not generated appropriately. To produce realistic fake queries, we identify two key requirements: Firstly, fake queries shall have the same number of keywords, the same language as the initial one, and a realistic usage frequency. Secondly, fake queries generated for one specific user shall be built from real past queries from other users of the system. Indeed, this makes a fake query more realistic and ensures that a de-obfuscation attack will tend to identify other existing users for each fake query and then mislead the attacker. For that reason, we create in our system a *group profile*. This structure contains an aggregation of past user queries. The issuer of the privacy proxy is responsible for aggregating queries in this structure and periodically releasing a new version (as shown in Figure 20). The obfuscation mechanism uses this group profile to generate fake queries.

Finally, obfuscating user queries alters the quality of the results received by the users as they may contain answers related to the fake queries. To overcome this issue, PEAS filters out the results to remove irrelevant answers (i.e., answers corresponding to fake queries).

4.3 PEAS Detailed Description

PEAS combines an unlinkability and an indistinguishability mechanisms to preserve the user privacy. To achieve that, PEAS relies on a piece of software running on the user machine (called the PEAS client in the following) and on a privacy proxy composed of two nodes the receiver and the issuer. In this section, we give a detailed description of PEAS. More precisely, we describe the privacy proxy (Section 4.3.1), the obfuscation mechanism using the group profile (Section 4.3.2), and the filtering mechanism of PEAS (Section 4.3.3).

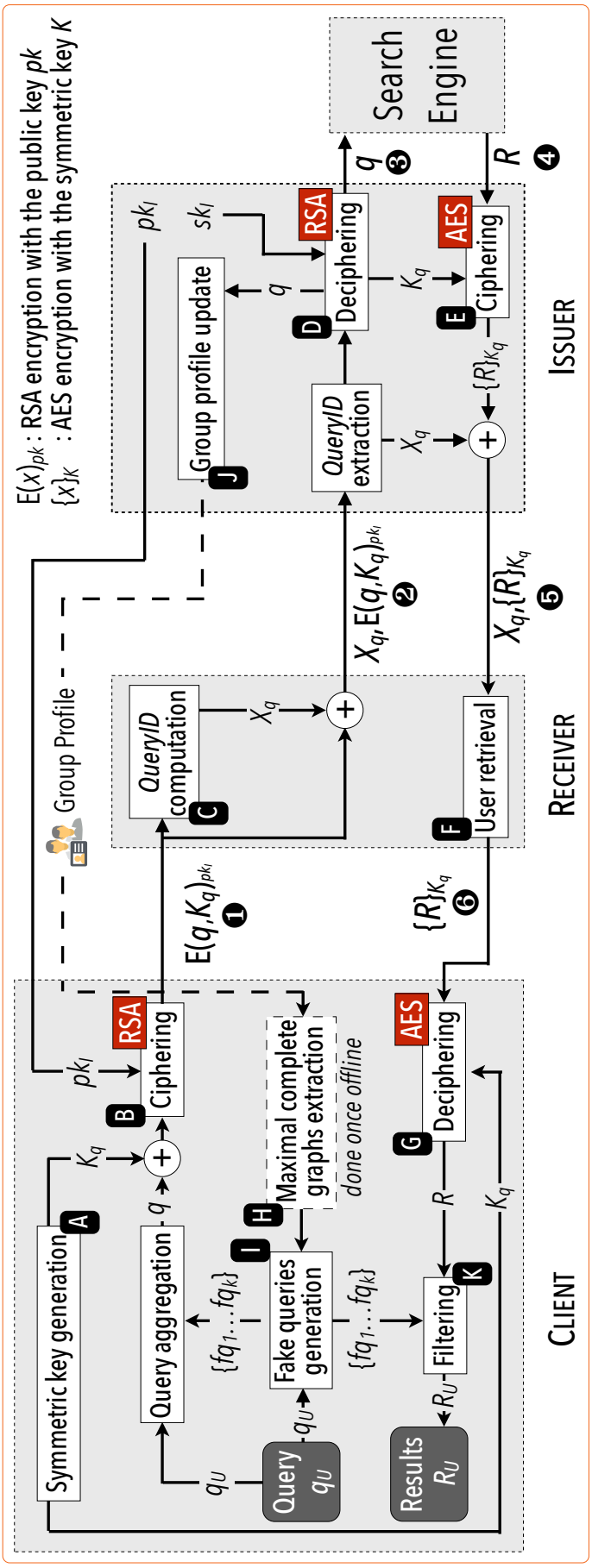


Figure 21
Sequence diagram of PEAS.

4.3.1 Privacy Proxy

The privacy proxy aims to ensure unlinkability between the user and her queries. To efficiently achieve this goal, the privacy proxy is split into two separated nodes: the *receiver* which only knows the identity of the users (e.g., their IP address) and the *issuer* which only knows the content of their queries. Figure 21 details the operations performed by the privacy proxy.

Firstly, when the user issues a query q, the client creates a new symmetric cryptographic key K_q (block A in the figure). Then, using the RSA encryption algorithm and the issuer's public key pk_I[1], the client ciphers the aggregation of her query q and the key K_q (block B). The cipher-text $E(q, K_q)_{pk_I}$ is then sent to the receiver (message ❶). As the receiver does not know the issuer's private key, it is not able to access the content of the initial query. When the receiver gets the message, it maintains a routing table by assigning a unique identifier X_q to the query with the identity of its requester (block C). The receiver then forwards the message along with the identifier X_q to the issuer (message ❷). As the issuer receives the cipher-text from the receiver, it has no information about the user identity. Afterwards, the issuer uses its private key to decipher the message (block D) and to retrieve the concatenation of the user's query q and the user's symmetric key K_q. Lastly, the issuer submits the initial query q to the search engine (message ❸).

Upon receiving the results R from the search engine (message ❹), the issuer uses the user's symmetric key K_q to encrypt the search engine's answer R with the AES encryption algorithm (block E). The issuer then returns the new cipher-text $\{R\}_{K_q}$ along with the query's identifier X_q to the receiver (message ❺). The receiver then retrieves the requester identity from its routing table using the associated identifier X_q (block F) and sends the encrypted results $\{R\}_{K_q}$ to the initial client (message ❻). The client finally uses the cryptographic key K_q to retrieve the results of the user's query (block G).

For privacy reasons, the client needs to generate a new cryptographic key K_q every time the user issues a new query. If not, this key might be used by the issuer as a quasi-identifier to link a set of consecutive queries issued by the same user and ultimately recover her identity (as shown in [17]). For a better efficiency, our protocol does not contain a key establishment step that sets up a symmetric key (like in Tor [39]) but it sends the symmetric key with each new query. Moreover, a key assumption to ensure the preservation of the user privacy, is that the receiver and the issuer are honest-but-curious. This model means that they strictly follow the protocol, and cannot collude or exchange extra information with each other. Nevertheless, they can locally monitor information and exploit this knowledge together with all additional public knowledge (typically, information available on the Internet).

4.3.2 Obfuscation Mechanism

The goal of the obfuscation mechanism in PEAS is to hide user queries among multiple realistic fake queries in such a way that an adversary cannot distinguish the initial queries from the fake ones. To do that, the proposed obfuscation mechanism aggregates the initial query by k fake queries separated with logical OR operators. We ensure that the k generated fake queries have the same structural properties as the initial query. More precisely, these fake queries must have the same number of keywords and the same language as the initial query. Their keyword usage frequency must also be realistic. In

[1] Similarly to Tor [39], public key information are publicly available on a directory system (e.g., published on the institution website that manages the issuer).

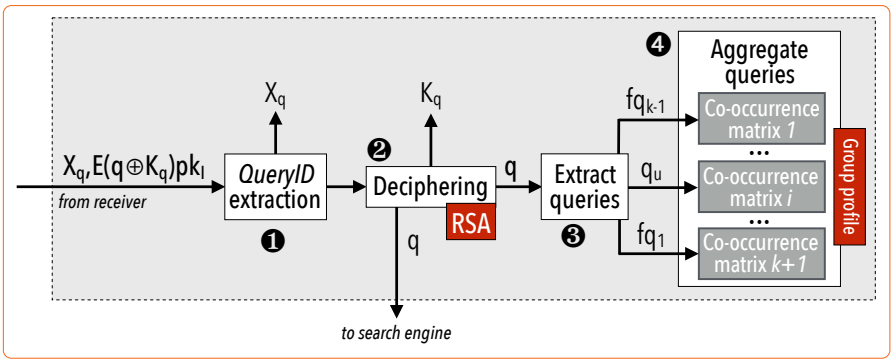


Figure 22
A partial view of the operations performed by the issuer.

addition, to mislead the adversary about the requester identity, these fake queries must not contain keywords that have already been used by the requester. Conversely, they should contain keywords already used by other users in the system. In other words, the generated fake queries should not be related to previous queries issued by the requester but be similar to queries issued by other users. To achieve that, each client locally stores a query history of its associated user. However, as a client has no direct access to the past queries of other users, the proposed obfuscation mechanism relies on the concept of *group profile*.

4.3.2.1 Group Profile

Fake queries are generated from the group profile as detailed in the next section. A group profile contains an aggregation of users' past queries. These queries are aggregated in $k+1$ co-occurrence matrices of disjoint users (meaning that, for a given user, her initial queries are always aggregated in the same co-occurrence matrix). The construction of the $k+1$ co-occurrence matrices is performed by the issuer (as shown by Figure 22). To do so, the issuer extracts from the protected query the $k+1$ sub-queries: the initial user query and k fake ones (step ❸). Then, using their order in the protected query, it aggregates them in their corresponding co-occurrence matrix: The first subquery in the protected query belongs to the first matrix, etc. (step ❹). Therefore, to generate the protected query in the correct order, the PEAS client needs to know which co-occurrence matrix its initial queries belong in. This is done by sending a one-off special query to the receiver. When the receiver gets this special query, it randomly assigns the client to one of the $k+1$ co-occurrence matrices and returns this number to the client. Structuring the group profile through $k+1$ co-occurrence matrices (and not one single co-occurrence matrix) enables the creation of robuster fake queries. This is explained in detail in Section 4.5.5.

Each co-occurrence matrix M is represented using the Vector Space Model [154]: Each dimension of the vector space represents the vocabulary (i.e., one dimension is a keyword used in a past query). Each element $M(a,b)$ of the matrix represents the co-occurrence frequency of the keywords a and b. As the co-occurrence matrices is an aggregation of bi-occurrence frequencies, only an approximation of past queries can be retrieved from them. Figure 23 gives an example of such a co-occurrence matrix. The value corresponding to the pair ("HIV", "treatment") is 1 and means that one query containing the keywords "HIV" and "treatment" has been issued in the past.

The group profile (containing the $k+1$ co-occurrence matrices) is periodically published by the issuer. Clients retrieve this group profile by contacting

Figure 23

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

	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
symptoms	11	4	13	0	0

the issuer through the receiver. For better performance, clients keep the last version of the group profile in cache and download it only when a new version is published.

At the bootstrap of the system, the group profile is empty. A solution to deal with this issue creates a group profile with dummy queries extracted from external sources (e.g., AOL search logs [16], Google Trends [155]). By doing so, the privacy risks for users would be similar to those of our competitors (e.g., GooPIR), which rely on similar data to generate their fake queries.

4.3.2.2 Fake Queries Generation

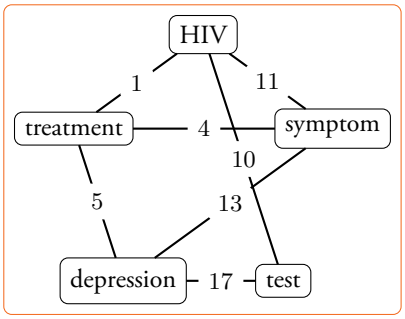
To generate fake queries, the PEAS client transposes each co-occurrence matrix in the group profile to a co-occurrence graph, and computes all maximal complete graphs, also called *maximal cliques* (block H in Figure 21). Formally, a maximal complete graph in an undirected graph $G = (V, E)$ is the vertex set $C \subseteq V$, such that (i) every two vertices in C are connected by an edge and (ii) there is no vertex in $V \setminus C$ for which it exists edges connecting all vertices in C. The associated completeness property is a necessary condition to retrieve already formulated queries (i.e., already formulated queries are necessarily complete graphs of keywords). Furthermore, the associated maximality property, in turn, aims to maximize the probability of selecting past queries and not a subset of past queries (see the evaluation in Section 4.7.1.3). For instance, from the previous example, we obtain the graph displayed in Figure 24. From this graph, it is possible to extract four maximal cliques:

1. "HIV", "treatment", "symptom"
2. "treatment", "symptom", "depression"
3. "test", "depression"
4. "test", "HIV"

These maximal cliques might correspond to previous queries. Nevertheless, it might be the case that, instead of a single query composed by the keywords "HIV", "treatment" and "symptom", two queries were issued: one containing

Figure 24

Example of a co-occurrence graph extracted from a co-occurrence matrix.



"HIV", "treatment" and another one containing "HIV", "symptom" (assuming that the second clique "treatment", "symptom", "depression" was also issued as query).

PEAS computes the maximal cliques of an undirected graph by using the Bron-Kerbosch algorithm [156]. This algorithm is a recursive backtracking algorithm that investigates for all the given vertices at each recursive step if they form a maximal complete graph with their direct neighbors. As investigating all given vertices at each recursion step is inefficient (the same subgraph is checked multiple times), Bron and Kerbosch introduced a *pivot vertex*. At each recursion step, a pivot vertex u is randomly chosen such that the subgraph that u forms with its direct neighbors is checked only once. To minimize the computation cost associated with the Bron-Kerbosch algorithm, the extraction of the maximal cliques is performed only once for each new version of the group profile. In addition, PEAS extracts the maximal cliques in a background task (during that period, PEAS uses the maximal cliques extracted from the previous group profile to generate the fake queries). The new version of the group profile is used as soon as all the extraction of the maximal cliques is finished.

Algorithm 8: Generation of the fake queries.

input : q_u : initial query sent by u,
k : number of fake queries to create,
M : $k+1$ co-occurrence matrices contained in the group profile,
C_u : $k+1$ sets of maximal cliques extracted from the co-occurrence matrices M; each set of cliques is filtered to remove maximal cliques that contain keywords already used by the user u,
id_u : the index of the co-occurrence matrix where the initial queries sent by the user u are aggregated.

$q_p \leftarrow \emptyset$; `// The protected query`
$isMaximal \leftarrow$ `isMaximal`$(C_u[id_u], q_u)$; `// Test if the query` q_u `is a maximal clique`
for $i \in [\![0, k]\!]$ **do**
  if $i \neq id_u$ **then**
    `// Sample a maximal clique`
    $q \leftarrow$ `sample`$(C_u[i], |q_u|, isMaximal)$;
    if $isMaximal$ AND $(|q| = |q_u|$ OR $|q| = |q_u| + 1)$ **then**
      $q \leftarrow$ `removeKeyword`(q);
    while $|q| \neq |q_u|$ **do**
      $q \leftarrow$ `addKeyword`$(q, M[i])$;
  else
    $q \leftarrow$ `shuffle`(q_u) ; `// Shuffle the initial query`
  `// Aggregate the query` q `in the protected query` q_p
  if $|q_p| = 0$ **then** $q_p \leftarrow q$; **else** $q_p \leftarrow q_p +$ "OR" $+ q$;
return q_p;

The PEAS fake query generation (block ❶ in Figure 21) is presented in Algorithm 8. Input parameters include the initial query q_u sent by the user u, the number of fake queries k PEAS needs to create, the $k+1$ co-occurrence matrices M contained in the group profile, the $k+1$ sets of maximal cliques C_u extracted from the co-occurrence matrices M (each set of maximal cliques

is filtered to remove keywords already used by the user), and finally the index of the co-occurrence matrix id_u where the real queries issued by the user u are aggregated.

The co-occurrence matrices M and their corresponding sets of maximal cliques C_u are identified by an integer i bounded between 0 and k. For each co-occurrence matrix (except the one which contains aggregated information about initial queries of user u), Algorithm 8 generates a fake query q by randomly choosing a maximal clique among the filtered set of maximal cliques $C_u[i]$ (line 5). This step aims to select keywords already used in a previous query. In addition, as fake queries are generated from a co-occurrence matrix that differs from the one used by the user u, the generated fake queries are likely to be close to the profile of another user. To be more realistic, we randomly select maximal cliques according to their usage frequency. We define the usage frequency of a maximal clique c by:

$$\text{usageFreq} = \min_{(a,b)\in c^2, a\neq b} M(a,b)$$

where a and b are two keywords contained in the maximal clique c and $M(a,b)$ is the number of times the two keywords a and b co-occur. This metric gives an approximation about how many time the clique c was previously used. Nevertheless, a mechanism that only creates fake queries from maximal cliques is not secured. By looking if each sub-query is a maximal clique, an adversary might be able to retrieve the initial one (the initial query is not always a maximal clique). For that reason, we distinguish two cases: (i) if the initial query q_u is a maximal clique (line 2), the method `sample()` returns a maximal clique of the same size as the initial query (line 5); (ii) if the initial query q_u is not a maximal clique, the same method `sample()` returns a maximal clique of size $|q_u|-1$, $|q_u|$, or $|q_u|+1$. In this second case, to obtain a fake query that is not a maximal clique, the method `removeKeyword()` randomly removes one keyword of the fake query q if its size equals $|q_u|$ or $|q_u|+1$ (line 7). Then, the method `addKeyword()` adds new keywords to obtain a fake query with the same size as the initial one (line 9). This method initializes a bag of words b with the keywords that co-occur with those in q, and their co-occurrent keywords. The keywords added by the method `addKeyword()` are chosen randomly among b. Moreover, it might be the case that the method `sample()` does not return a maximal clique. For instance, if $C_u[i]$ is empty. Therefore, in that case, the method `addKeyword()` randomly selects one keyword and applied the previous algorithm to generate a fake query q. Another part of the algorithm shuffles the keyword order of the initial query (line 11). Indeed, as the group profile does not contain information about the word order, it might exist inconsistencies in the word order of the fake queries and thus reveal that they are fake. By mixing the keyword order in the initial query, we make sure that an adversary cannot exploit the word order to distinguish fake queries (see discussion in Section 4.5.5). Finally, the k fake queries and the initial query are aggregated to form the protected query q_p. The complexity of the generation of the fake queries is $O(k+|q_u|)$.

4.3.3 Filtering Irrelevant Results

The filtering step aims at removing the irrelevant results returned by the search engine due to the fake queries (block K in Figure 21). Indeed, the results returned by the search engine contain a mix of answers corresponding to the $(k+1)$ queries (i.e., k fake queries and the initial one). This filtering step is performed by the client which is the only one that knows the initial

query. In the literature, GooPIR proposed an algorithm to remove irrelevant results. It identifies the results related to the initial query by checking if their title, description and URL contain all keywords of the initial query. In practice, as most results only contain a subset of keywords of the initial query, GooPIR discards a lot of relevant results. In addition, as fake queries and the initial one might have keywords in common, the filtering step has to take the fake queries into consideration. Due to these reasons, we propose Algorithm 9. It checks for each results if the initial query contains the highest number of keywords in its title, description and URL. More precisely, for each result r contained in the result set, the algorithm assigns a score to the $k+1$ queries sent by the user. The scores are equal to the number of occurrences the keywords of each query have in the title, description and URL of the results r (lines 4 to 6). Algorithm 9 computes the score using the method `nbCommonWords`(q,t). This method determines the number of words a query q and a text t have in common. It is defined as $|q \cap t| + |q \cap HW_t|$, where HW_t is the set of highlighted words in t (i.e., some search engines mark some words in bold in the description of the result). A result r is considered as related to the initial query if and only if the initial query has the largest score (lines 7 and 8). The complexity of such an algorithm is $O(k \cdot |R| \cdot |q_u|)$.

Algorithm 9: Results filtering.

input : q_u : initial query,
$FQ = \{fq_1, \ldots, fq_k\}$: set of fake queries,
R : set of results for $q_u \text{OR} fq_1 \text{OR} \ldots \text{OR} fq_k$.

1 $\hat{R} \leftarrow \emptyset$; `// the filtered results`
2 **for** $r \in R$ **do**
3 **for** $q_i \in \{q_u\} \cup FQ$ **do**
4 $score[q_i] \leftarrow$ `nbCommonWords`$(q_i,$ `title`$(r))$
5 + `nbCommonWords`$(q_i,$ `desc`$(r))$
6 + `nbCommonWords`$(q_i,$ `url`$(r))$;
7 **if** $score[q_u] = \max_{q_i \in q^+}(score[q_i])$ **then**
8 $\hat{R} \leftarrow \hat{R} \cup \{r\}$;
9 **return** $\hat{R}$;

4.4 PEAS Distributed Deployment

To support a larger number of users, PEAS can adopt a distributed deployment so as to distribute the load across multiple receivers and multiple issuers. Moreover, this distributed setting makes the non-collision assumption between receivers and issuers more acceptable, as all nodes should collude to break the user privacy.

Distributing PEAS across multiple receivers and issuers only introduces two modifications in the protocol presented in Section 4.3. Firstly, a new preliminary step is performed by the client to select one receiver and one issuer among all available ones. Secondly, as shown in Figure 25, the group profile can be either (i) centralized and shared between all issuers (right side of the figure), or (ii) split across multiple group profiles where each issuer has its own group profile (left side of the figure). Depending on the group profile management selected by the system designer, the client either retrieves the centralized group profile or the group profile associated with its selected issuer.

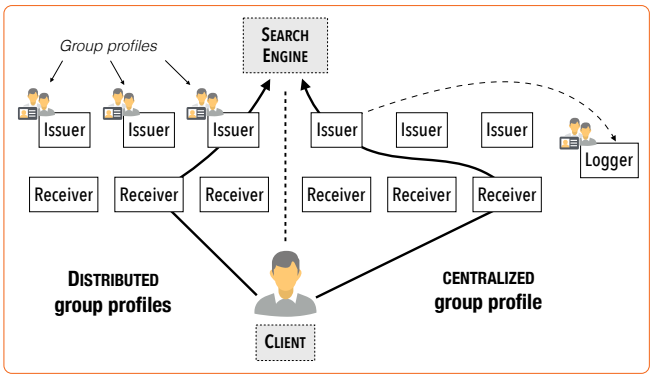


Figure 25
The two possible PEAS distributed architectures: a centralized group profile and a decentralized group profile.

In the rest of this section, we detail both the receiver and issuer selection and the group profile management.

4.4.1 Receiver and Issuer Selection

The distributed deployment of PEAS is composed of multiple receivers and issuers. Every time a user issues a query, the associated PEAS client first randomly selects one receiver and one issuer among all available ones[2]. As for the non-distributed PEAS protocol, clients then need firstly to collect the group profile maintained by every issuer (if they do not have the last version stored locally), and secondly to compute the maximal cliques.

4.4.2 Group Profile Management

To maintain one centralized group profile in a distributed environment (right side of Figure 25), PEAS relies on a new entity called the *logger*. The goal of the logger is to receive the group profiles sent from all issuers and aggregate them into a single group profile. The aggregation of two group profiles GP_1 and GP_2 composed of $k+1$ co-occurrence matrices results in adding their corresponding co-occurrence matrices together (adding the first matrix of GP_1 with the first matrix of GP_2, etc.). In addition, the logger is also in charge of publishing the group profile. To reduce the cost due to the aggregation process, it is performed only once before each update of the group profile. Clients contact the logger through the receiver to retrieve the last version of the group profile. Similarly to the non-distributed PEAS protocol, clients cache the last version of the group profile.

Regarding the decentralized group profile architecture (left side of Figure 25), each issuer aggregates queries locally in its own group profile. The PEAS client randomly selects an issuer and a receiver every time the user issues a query. Then, similarly to the original PEAS protocol, the client first contacts the selected issuer (through the receiver) to retrieve the last version of its group profile. As clients interact with multiple issuers, they stored locally multiple group profiles (one for each issuer). Furthermore, we note that the fake queries generated by the client depend on the selected issuer. Two different issuers have two different group profiles and thus produce different fake queries.

These two solutions present advantages and drawbacks. On the one hand, the centralized group profile solution needs to introduce a new entity, a logger, which is responsible for aggregating all group profiles in a single group

[2]For better performance and security, this choice may take the load of the nodes and their reputation into account. These two aspects remain outside the scope of this thesis.

profile. Therefore, the aggregation of the group profiles introduces periodically extra costs (i.e., sending group profiles to the logger and merging all the group profiles). On the other hand, the decentralization of the group profile increases the traffic as clients have to collect a group profile each time they select a new issuer. It also requires PEAS clients to extract maximal cliques for these multiple group profiles. But, as the distributed group profiles contain fewer queries than the centralized group profile (in the decentralized group profile architecture, queries are distributed among multiple group profiles) and knowing that the maximal clique extraction step follows an exponential running time (see Section 4.7.3.1), there is no real difference in terms of computational costs.

4.5 Security Analysis

This section discusses the security guarantees offered by PEAS. More precisely, we review the choice of the cryptographic algorithms used in the privacy proxy protocol and validate its unlinkability guarantee. We also discuss the practicability of the non-collusion assumption between the receiver and the issuer. Then, we focus on the possible issues due to the publication of the group profile and the distinguishability between fake queries and real ones. Lastly, we evoke the choice of the number k of fake queries generated by PEAS.

4.5.1 Cryptographic Algorithms Used by PEAS

As mentioned in Section 4.3.1, PEAS employs the RSA encryption algorithm as asymmetric encryption (to cipher user queries) and the AES encryption algorithm as symmetric encryption (to cipher the results provided by search engines). These two ciphering techniques were chosen as they are popular and widely used to encrypt data [157]. In addition, these algorithms have been validated and advised by ANSSI (the French Network and Information Security Agency) [158] and NIST (the National Institute of Standards and Technology) [159], two expert institutions.

Moreover, regarding the size of the cryptographic keys used in PEAS, we follow the suggestions provided by ANSSI and NIST. Both organisations have stated that a robust RSA encryption is achieved by a cryptographic key length higher than 2,048 bits. Due to this reason, issuers generate 2,048 bits RSA cryptographic keys. Regarding the AES encryption, the two aforementioned institutions suggest symmetric keys with a length higher than 128 bits. Therefore, PEAS's clients generate a 128 bits symmetric key. To ensure a robuster protection, higher lengths could have been chosen. But, as we want PEAS to be efficient, we limit the key lengths to their minimum acceptable value and thus avoid extra cryptographic costs.

4.5.2 Unlinkability Guarantee of the Privacy Proxy

To formally prove the unlinkability property of the privacy proxy, we used ProVerif [107]. ProVerif is a well-known tool to perform security analysis of cryptographic protocols. Based on a formal model (e.g., π-calculus), it can verify several security properties. We model our protocol according to the description given in Section 4.3. The four entities *user*, *receiver*, *issuer* and *search engine* are defined through subprocesses. We send all messages through a public channel c to make the adversary able to eavesdrop on messages. Furthermore, we set up an adversary that tries to link a query to its requester.

Internally, ProVerif attempts to prove that a state in which the adversary links the query with the user is unreachable. However, ProVerif does not natively implement the notion of linkability. It can state that an adversary knows the query q and the user identity u, but it cannot determine if they are related with each other. In other words, if the query q was issued by the user u. To overcome this limitation, we created a function `LINK(`x`,`y`)` symbolizing that x and y are linked, and we made this function transitive with another function `INFER`, meaning that if an adversary knows `LINK(`x`,`y`)` and `LINK(`y`,`z`)`, she can deduce `LINK(`x`,`z`)`. Consequently, under this model, the adversary tries to obtain `LINK(`q`,`u`)`. Our ProVerif model formalizing the privacy proxy protocol is available in Appendix C. As indicated by the outcome of ProVerif, no attack has been found to link the query q with its requester u.

If we pay more attention to the privacy proxy protocol, we note that an adversary who monitors the network can obtain the same knowledge as a curious receiver. Therefore, an adversary cannot learn new information from the receiver. We then deduce that the critical entity in our protocol is the issuer. If this entity colludes with an adversary, the unlinkability property cannot be guaranteed.

4.5.3 Non-Collusion Assumption between the Receiver and the Issuer

The privacy guarantees of PEAS rely on the assumption that the issuer and the receiver do not collude. This assumption is also adopted by other well-known anonymous or privacy-preserving protocols such as Tor or KOI [106]. In order to mitigate the risk of collusion, Tor enables users to use a higher number of relays in the path forwarding chain (by default, three). In our case, our design involves only two nodes: the receiver and the issuer. Our choice was motivated by the performance, as we want to ensure with the privacy proxy a low overall latency. In the literature, KOI's authors face the same issue in their protocol (i.e., KOI also uses two nodes: the matcher and the combiner). To reduce the risk of collusion, they advise a deployment by trusted and privacy-aware institutions (e.g., Mozilla, academic institutions). In our context, we can imagine that the receiver and the issuer are hosted by two different privacy-aware institutions. Furthermore, using different receivers and issuers in the distributed deployment can reduce the risk of data leakage in case of collusion.

4.5.4 Publication of the Group Profile

To reduce the risk of data leakage induced by the publication of the group profile, the issuer publishes the group profile in the form of co-occurrence matrices. Indeed, this structure aggregates queries about all the users and thus gives no information about individual users. Besides, as the vectorial space is a set of unigrams (and not n-grams), the co-occurrence matrix only contains a sub-part of the initial information. For instance, if the adversary knows that a co-occurs with b, b co-occurs with c and c co-occurs with a, it cannot know whether the initial query was "$a\ b\ c$" or whether it was a set of three independent queries "$a\ b$", "$b\ c$" and "$c\ a$".

Nevertheless, by extracting all maximal cliques, an adversary could retrieve potential information about past user queries. However, as we consider the search engine as a potential adversary (Section 1.2), the adversary already has this knowledge. Indeed, by receiving queries from the issuer, the search engine knows queries sent by the user. Consequently, an adversary cannot ex-

tend her knowledge by performing such an extraction. In addition, choosing a low frequency for refreshing the group profile (e.g., every day) improves the probability to mix the activity of a large number of users in one single co-occurrence matrix.

4.5.5 Indistinguishability of the Initial Queries

Fake queries generated by PEAS have equivalent properties to the initial query (e.g., the same number of keywords, the same language). Their keywords have also a realistic usage frequency, and they are built using co-occurrence information taken from aggregated user queries. Consequently, without using background information on the user, an adversary cannot distinguish fake queries from real ones. In addition, PEAS shuffles the keywords of the initial queries. This operation prevents an adversary from identifying fake queries due to inconsistencies in their keyword order. For instance, a fake query "Floyd Pink" is not realistic as real users would have written "Pink Floyd". By shuffling the word order of the initial query, all queries have inconsistencies in their word order and thus no distinction between fake queries and real ones is possible. Nevertheless, modern search engines exploit the word order to retrieve the results. Therefore, shuffling the keyword order of the initial query increases the security of the protocol but impacts the accuracy of the results. We assess its impact in Section 4.7.2.2.

Furthermore, we construct the k fake queries from k different co-occurrence matrices to increase their robustness.Indeed, fake queries generated for one specific user do not contain keywords already used by this user. Consequently, in the obfuscated query, the initial query is the only one to integrate keywords already used by the requester. As a consequence, if two queries (in the obfuscated query) are close to the same user profile, they are necessarily fake. To avoid such an attack, the obfuscation mechanism of PEAS ensures by design that each fake query refers to a different group of users. This is done by structuring the group profile with multiple co-occurrence matrices. Each co-occurrence matrix contains, in addition to the fake queries, the initial queries issued by different groups of users. More precisely, the initial query of a given user is always aggregated in the same co-occurrence matrix.

4.5.6 Number of Fake Queries k

As explained in the description of PEAS, the number of fake queries k generated by the clients has to be equal to the number of co-occurrence matrices contained in the issuer's group profile. Therefore, the number of fake queries generated by the PEAS client is decided by the issuer and not by the user. To allow more flexibility in the choice of the number of fake queries, we propose two options: (i) the protocol could tolerate a lower number of fake queries. For instance, a user could generate l fake queries ($l < k$) by replacing the $k - l$ missing fake queries by `null`. The issuer could identify the `null` fake queries and correctly aggregate the $l + 1$ sub-queries in their corresponding co-occurrence matrices. (ii) another option is to deploy multiple issuers with a different number of co-occurrence matrices in their group profile. Therefore, a user chooses the issuer according to her desired number of fake queries. Nevertheless, the latter requires several users to use the same number of fake queries. Otherwise, the obfuscation mechanism cannot work, as the group profile contains too few queries mostly issued by only one user.

4.6 Experimental set-up

We exhaustively evaluated PEAS using a real dataset of search queries and a real deployment on Grid5000 [160]. In this section, we provide the experimental set-up of our evaluation. We present the dataset of search queries we used to assess PEAS together with the applied methodology and metrics.

4.6.1 Dataset

We evaluated PEAS using queries issued by the most active users in the AOL query logs presented in Section 3.3.1. The most active users of these datasets represent the users for which the adversary has collected the most of preliminary data (i.e., users with the largest user profile). As explained in the previous chapters, privacy attacks leverage the pre-built user profiles to break privacy. As a consequence, the largest profiles provide an upper-bound of the capacity of the attack to achieve its goal. To focus our evaluation on this upper-bound, we want to consider the 100 most active users contained in the AOL dataset. Nevertheless, to avoid potential variabilities of the dataset, we do not consider the dataset `AOL100` (i.e., the dataset of 100 users used in the previous chapter). Instead, we construct three datasets of 100 users. We randomly split queries of the 300 most active users into three datasets of 100 users. The 300 most active users issued during the collecting period 343,548 queries (i.e., on average 1,145 queries per user). Consequently, our experiments were performed three times on three datasets of 100 users.

4.6.2 Methodology and Metrics

In this section, we present the methodology and the metrics used to evaluate PEAS. More precisely, we focus on three evaluations: (i) measuring the privacy protection by running SimAttack and the machine learning attack on the protected queries, (ii) measuring the influence of PEAS on the quality of the results returned to users, and (iii) assessing the efficiency of PEAS in terms of throughput and latency.

4.6.2.1 Privacy

The privacy evaluation measures the level of protection offered by PEAS on queries contained in the testing set. The level of protection is measured using two privacy attacks: (i) the machine learning attack (presented in Section 3.3.3) and, (ii) SimAttack. To be executed, these two attacks required external knowledge on users. As mentioned in Section 1.2, an adversary can exploit previously-collected queries about users. She can also use all public knowledge available on the Internet. Nevertheless, as we do not know useful information from the Internet that can be used to bypass the PEAS protection, we limit our evaluation to the previously-collected queries. The measure of the outcomes of the attacks are made with the recall and the precision as introduced in Section 3.3.4.

To generate fake queries, PEAS requires a non-empty group profile. As mentioned in Section 4.3.2.1, several solutions exist to initialize the group profile. For our experiments, we used previous user queries to set up the group profile. These queries are contained in the training set (i.e., the same queries used as external knowledge by the adversary). Our experiments vary the number of fake queries from zero to seven. To avoid any influence due to the assignment of the users to the $k+1$ co-occurence matrices (k is the number of fake queries generated by the PEAS client) and to have a fair comparison

between experiments conducted with different numbers of fake queries (i.e., variations could come from a lower number of queries in the co-occurence matrices, and not from an increase of the number of fake queries), we only consider one co-occurence matrix per group profile (and not $k+1$ as proposed in the protocol). For the recall, we introduced the $k+1$ co-occurence matrices to make fake queries and real ones indistinguishable; not because of the privacy attacks (see discussion in Section 4.5.5). Besides, we cover the influence that the number of co-occurence matrices has on the privacy protection with the experiments on the distributed deployment of PEAS (Section 4.7.1.4).

4.6.2.2 Accuracy

The obfuscation mechanism of PEAS (i.e., adding k fake queries to the initial query) modifies the results returned by a search engine. We evaluated the capacity of PEAS to filter and remove all answers related to the fake queries from the results returned by the search engine. Our experiment compares the results obtained by sending a query directly to the search engine and by sending it through PEAS (i.e., considering the obfuscation and the filtering steps). Each set of results contains 30 results.

Furthermore, to obtain the results of the initial query and the protected query, we used Google [3]. However, as the OR operator implemented by Google only works with single-word queries, we simulated the execution of an obfuscated query $q^+ = q_0$ OR ... OR q_k by submitting each sub-query q_i independently and by randomly merging the $(k+1)$ result sets. In reality, most search engines do not necessarily return a balanced number of results for sub-queries. Therefore, with our simulation, we measure the accuracy in the optimal scenario. The worst scenario occurs if, due to an unbalanced number of results, no answer is returned for the initial user query and thus only irrelevant results are displayed to the user. We present the merging algorithm in Algorithm 10. The algorithm takes as input the $(k+1)$ sets of results $S_0, ..., S_k$. Then, it randomly selects a result from one of the sets of results and adds it in the result list R. The algorithm stops when all results have been merged. The result list is then filtered by PEAS and only the first 30 results are considered. Due to the query limitations of Google (100 queries per day), we ran the evaluation on a subset of the testing set composed by 1,000 queries.

Moreover, the obfuscation mechanism of PEAS also modifies the keyword order of the initial query (to prevent an adversary from identifying fake queries due to illogical word orders). Consequently, we also evaluated the influence of the query word order on the accuracy of the results. We compared the results returned by the same 1,000 queries (from which we removed all one-word queries, as they are not impacted by the word order) and one of their shuffled version.

To measure the accuracy, we use the same metrics as presented in [45]. We consider as first metric QM_1 which expresses the percentage of results obtained for the initial query (i.e., without any protection) that are also retrieved by PEAS after filtering the irrelevant results:

$$QM_1 = \frac{|R \cap \hat{R}|}{|R|},$$

where R and $\hat{R}$ are the finite lists of results returned by Google and PEAS, respectively. QM_1 equals 1 when R and $\hat{R}$ contain the same elements. If R and $\hat{R}$ do not share any common results, QM_1 equals 0.

Algorithm 10: Simulation of the execution of the OR operator.

input : $S_0, ..., S_k$: results obtained for each $k+1$ query.

$I \leftarrow [\![0, k]\!]$;
$R \leftarrow \emptyset$; // Results returned for $q^+ = q_0$ OR...OR q_k
while $|I| \neq 0$ **do**
    // Select randomly one set of results
    $i \leftarrow \texttt{random}(I)$;
    // Retrieve the first result in the set S_i
    $r \leftarrow \texttt{remove}(S_i)$;
    // Add the result r in the set R
    $R \leftarrow R \cup \{r\}$;
    // If all results in S_i has been merged in R, remove its index i
    if $|S_i| = 0$ **then**
        $I \leftarrow I \setminus \{i\}$;
return R ;

The second metric we used is QM_2. It computes the average of the absolute ranking differences over the results that are in both R and $\hat{R}$. QM_2 is defined as:

$$QM_2 = \frac{\sum_{r \in R \cap \hat{R}} |rank_R(r) - rank_{\hat{R}}(r)|}{|R \cap \hat{R}|},$$

where $rank_L(e)$ denotes the rank of an element e in a list L. This metric compares the rankings of the expected list of results R with another list of results $\hat{R}$. QM_2 equals 0 if the elements of $(R \cap \hat{R})$ appear in the same position in R and $\hat{R}$. The maximum value for QM_2 varies with the number of elements in $(R \cap \hat{R})$:

$$max(QM_2) = \begin{cases} n^2/2 & \text{if } n \text{ is an even number} \\ (n^2-1)/2 & \text{if } n \text{ is an odd number} \end{cases},$$

where n is the number of results in $(R \cap \hat{R})$.

Furthermore, QM_2 is dependent of QM_1. For instance, if the two lists of results R and $\hat{R}$ are equal (i.e., QM_1 = 100%), QM_2 is necessarily equal to 0. This is due to the merging algorithm used to simulate the OR operator. The algorithm preserves the order for each query. Meaning that, if we consider the two results r_1 and r_2 retrieved for one of the sub-queries (r_1 is retrieved before r_2), after the merging r_1 is still ranked before r_2 in the results list.

4.6.2.3 Performance

We assessed the performance of PEAS with a theoretical and a practical evaluation. The theoretical evaluation studies the number of costly operations (e.g., ciphering, deciphering, messages exchanged) performed to send a query while the practical evaluation analyzes the maximum load that PEAS can deal with, the computation cost required for the client, and the effect of the fake queries on the latency and on the size of messages sent over the network.

We implemented PEAS in Java. We optimized our implementation by using non-blocking sockets (Java NIO) and an efficient cryptographic library (i.e., Bouncycastle [161]) to perform the encryption and decryption operations with native code. Our implementation also takes advantage of hardware

multi-threading (e.g., it is able to perform multiple encryption and decryption operations simultaneously). Besides, in order to test our implementation in a real environment, we deployed PEAS on Grid5000 [160]. This infrastructure provides an access to a large amount of resources. For our experiments, we used up to 63 nodes to act as clients, receivers or issuers. For simplicity all the considered nodes were part of a homogeneous cluster. Besides, running receivers and issuers on the same architecture enables a straight-forward comparison between them. They were equipped with two quad-core Intel Xeon E5520 clocked at 2.27 GHz and 24 GB of RAM. Finally, we assume that a user waits for the result of her query before sending a new one. We also consider that clients, receivers and issuers know all the necessary cryptographic keys in advance[3].

To perform the theoretical analysis of PEAS, we distinguish three costly operations: (i) the number of symmetric encryption and decryption operations, (ii) the number of asymmetric encryption and decryption operations, and (iii) the number of generated messages. This theoretical evaluation counts the number of times the considered operations are performed in the whole system for one query.

For the practical evaluation, we perform three experiments: analyze the time required by the client to extract the maximal cliques, measure the impact of the number of fake queries on the latency (i.e., the period of time during which the client waits for the results of her query), and study the throughput and the latency according to the number of users in the system. We define the throughput as the number of queries the privacy proxy is able to answer per second. This throughput can be measured either on the receiver or on the issuer. Indeed, as a client waits for the results of her query before sending a new one, the throughput remains identical in the whole privacy proxy. To avoid perturbations in the measurement of the scalability of PEAS due to network delay or search engine load, we do not query the search engine for this last experiment but we consider that users send a message of 136 bytes and we model the search engine's answer by a message of 166 kilobytes. Consequently, latencies reported in the scalability evaluation do not include the latency between the issuer and the search engine, and the delay needed by the search engine to prepare the results.

Finally, we compare the performance of the privacy proxy with unlinkability solutions published in the state of the art. According to [58], Tor [39] is the most widely used and effective unlinkability solution. Therefore, we compare the performance of the privacy proxy with Tor. To have a fair comparison, we consider only two onion routers (by default, Tor uses at least three onion routers) and a regeneration of the symmetric keys for each query (by default, Tor established a new symmetric key with each onion router every ten minutes).

4.7 PEAS Experimental Evaluation

This section presents the experimental evaluations of PEAS in terms of privacy-protection, accuracy and performance. Firstly, PEAS outperforms GooPIR over an unlinkability solution by decreasing the number of queries linked by an attacker to its originating user by up to 18.4%. Secondly, the filtering of PEAS successfully removes a large proportion of irrelevant results. For 95.5%

[3]Receivers and Issuers have to share their public keys only once with each client. Therefore, considering that clients send multiple queries through the same receiver and issuer, the exchange of the public keys is an insignificant step and thus is not addressed by our evaluations.

of the protected queries, 80% of the expected results are correctly retrieved by PEAS. This result is twelve times higher than one obtained with GooPIR. Finally, the performance of the privacy proxy clearly outperforms the onion routing baseline. It reduces the latency by 7.5 and the throughput by 3.1.

4.7.1 Privacy

In this section we focus on a privacy evaluation of PEAS. The evaluation analyzes the robustness of PEAS regarding privacy attacks and compares PEAS against its closest competitor GooPIR over an unlinkability solution. We also assess the influence of the distributed deployment of PEAS on the user privacy.

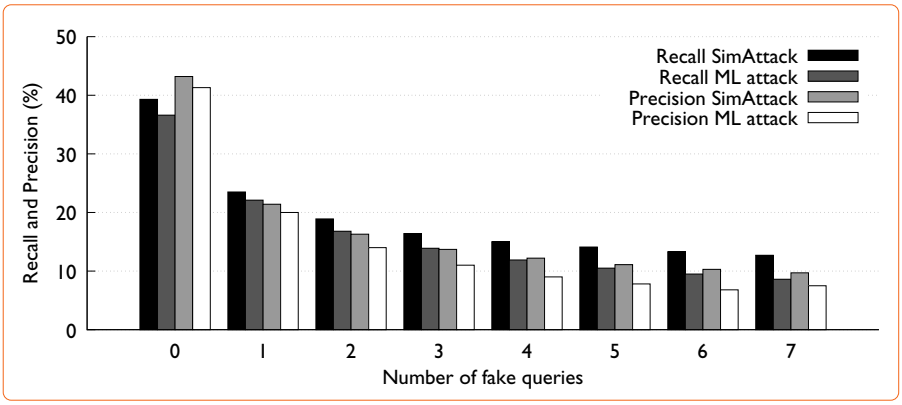
4.7.1.1 Robustness Against Privacy Attacks

We first analyze the robustness of PEAS against the ML Attack and SimAttack. Figure 26 depicts the recall and the precision achieved by both privacy attacks on PEAS for an increasing number of fake queries. When no fake query is used, the results represent the outcome of the attacks against an unlinkability mechanism (no obfuscation). Adding one fake query divides the number of queries retrieved by the attack (the recall) by 1.66 on average, from 39.3% to 23.5% for SimAttack and from 36.6% to 22.1% for ML Attack. Increasing the number of fake queries up to seven decreases the outcome of both attacks by more than 3 times, from 39.3% to 12.7% for SimAttack and from 36.6% to 8.6% for ML Attack. In addition, the precision also decreases according to the number of fake queries. Without obfuscation, the attacks perform with a precision of 43.2% and 41.3% respectively for SimAttack and ML Attack. The precision drops to 9.7% and 7.5% for seven fake queries. Consequently, increasing the number of fake queries dramatically improves the user protection. However, we note that PEAS is not able to protect all users queries. The results show that an attacker is still able to retrieve around 10% of queries (considering seven fake queries). It is difficult to protect these queries, as most of them have been already issued by the user. Indeed, most of them are contained in the previous-collected queries owned by the adversary.

We also compare the robustness of PEAS and GooPIR over an unlinkability solution against the ML Attack and SimAttack. Figure 27 depicts the recall achieved by both attacks against PEAS and GooPIR over an unlinkability solution for a varying number of fake queries. We show that PEAS clearly outperforms GooPIR over an unlinkability solution regardless of the number

Figure 26

The robustness of PEAS increases according to the number of fake queries.



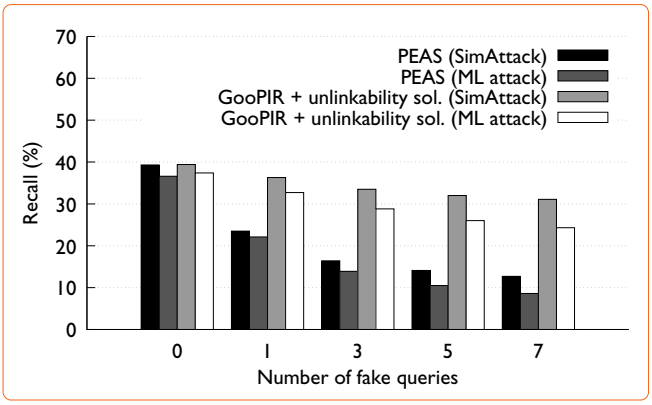


Figure 27

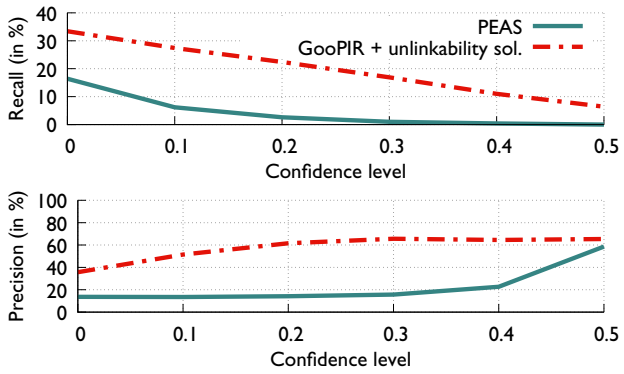
PEAS outperforms GooPIR regardless of the number of fake queries.

of fake queries. Indeed, the PEAS recall is at least 11.7% and 17.0% lower than the recall obtained by GooPIR over an unlinkability solution with one and seven fake queries, respectively. On average, the recall obtained by PEAS is 15.0% lower than the recall obtained by GooPIR over an unlinkability solution. As we obtain a similar conclusion for the precision (the precision is 19.9% lower for PEAS than GooPIR over an unlinkability solution on average), we do not consider it in Figure 27.

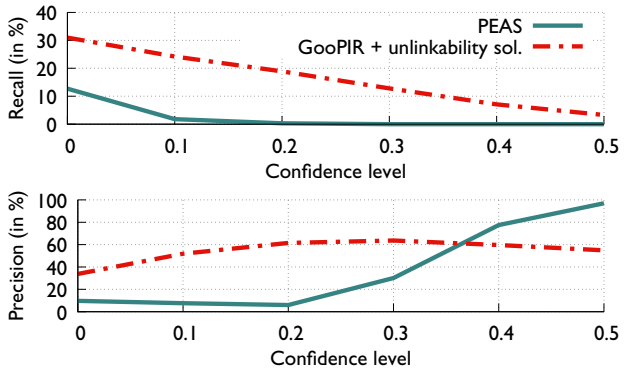
We now mitigate these results by introducing the confidence level that the attacker has on these queries. The current attack considers the pair query and user that gives the highest similarity value. However, if another pair has a slightly lower similarity, the user and the query returned by the attack might not be relevant. The two pairs would have approximately the same probability of being the good one. Consequently, we define the confidence level as the difference between the two highest similarities. As shown in Algorithm 4 (Section 3.2.4), the confidence level corresponds to the threshold cf. This confidence value is ranged from zero to one. We consider that the de-anonymization is successful if the correct user is identified and if the confidence level is higher than the considered value. Figure 28 shows the recall and the precision for different confidence levels considering an obfuscation mechanism that generates three and seven fake queries. We omit to report results for the ML Attack as this second attack gives similar results to SimAttack. The results show that for a confidence level of 0.2, the percentage of de-anonymized queries for PEAS drops from 16.4% and 12.7%, to 2.6% and 0.3% with three and seven fake queries, respectively. For the same level of confidence, the percentage of de-anonymized queries for GooPIR over an unlinkability solution only decreases from 33.5% and 31.1% to 22.4% and 18.9% with three and seven fake queries, respectively. As a consequence, if the adversary slightly increases the confidence level (by default, the value is defined at 0.01), she is able to retrieve only few queries protected by PEAS (0.3%) but for GooPIR she is still able to retrieve a significant quantity (18.9%). One could object that if the confidence level is defined with a value larger than 0.5, the two private Web search solutions could perform similarly, as in both cases the recall would be close to 0. Nevertheless, it is not interesting for the adversary to define such a high threshold, as a high confidence level does not allow the adversary to de-anonymize any query. Consequently, a confidence level is only interesting for low values. More precisely, the confidence level is interesting for the adversary if it increases the precision. According to Figure 28, the precision of PEAS is constant for low confidence levels (for three fake queries, the precision equals approximately 13.5% for $cf < 0.3$, while for seven queries, the precision is around 7.6% for $cf < 0.2$), and then increases

Figure 28

Taking the confidence level into consideration decreases the recall of the privacy attacks but increase the precision .



(a) *In the case of three fake queries.*



(b) *In the case of seven fake queries.*

rapidly (to 58.6% and 97.0% for three and seven fake queries, respectively). The constant part is explained by the fact that the confidence level discards the same quantity of queries correctly and incorrectly retrieved by the attack. But, above a certain threshold, as the confidence level discards more queries incorrectly retrieved by the attack, the precision of PEAS increases. Therefore, in the case of PEAS, the confidence level has no real interest. The adversary does not have any gain in increasing its value. However, if we consider GooPIR over an unlinkability mechanism, the confidence level increases the precision up to 60%. Therefore, for this solution, the adversary can trade off recall for precision. Depending on the goal of the adversary, she might be interesting to retrieve a lower number of queries with a higher precision.

4.7.1.2 Relevance of Fake Queries

Fake queries generated for a given user u are designed to be (i) far from previous queries sent by the user u but (ii) close to queries sent by the user $v \neq u$. The first aspect is achieved by design, as fake queries generated by u do not contain keywords already used by u. This ensures that no relation exists between previous queries sent by u and fake queries generated by u. To assess the relevance of the second point (i.e., being close to queries sent by $v \neq u$), we compute the similarity between fake queries generated by a user u and the previously-collected queries sent by all users $v \neq u$. More precisely, we compute the cosine similarity between a fake query and each previous query sent by these users. Then, we extract, from this collection of cosine similarities, the highest cosine similarity. A similarity value close to one indicates that the fake query is similar to a previous real query while a value close to zero means

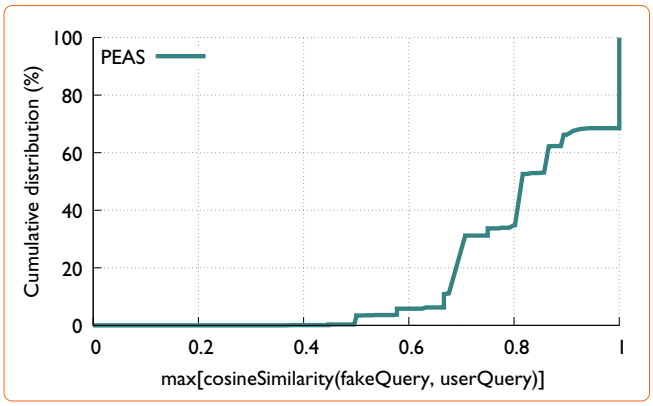


Figure 29
PEAS fake queries are close to previous queries sent by other users in the system.

that the query is dissimilar to any previous real queries. Figure 29 gives the cumulative distribution of this similarity for all generated fake queries. We can see that PEAS fake queries are similar to previous real queries, as all fake queries have a cosine similarity higher than 0.5. Moreover, almost a third of the fake queries (31.5%) were already issued by a user $v \neq u$. It corresponds to the fake queries with a cosine similarity equal to one. These results show that PEAS fake queries are effectively close to queries sent by another user in the system.

Nevertheless, this is not enough to satisfactory protect users. Privacy attacks try to retrieve from an obfuscated query, the most probable query. To ensure that the adversary cannot perform such attacks, we have to verify that fake queries are closer to a user profile $v \neq u$ than the initial query with its user profile u. For that purpose, we compute the mean of the previous distribution, m_{fq}. It gives the average similarity between fake queries and the other user profiles. We compute a similar metric m_q between real queries and their requester user profile. Therefore, to correctly protect users, the mean m_q (corresponding to initial queries) should be lower than the mean m_{fq} (corresponding to fake queries). We computed these two values and we found that $m_{fq} = 0.84$ and $m_q = 0.49$. Consequently, as m_{fq} is higher than m_q, a privacy attack is more likely to retrieve a fake query with a wrong requester than the correct query with the correct requester.

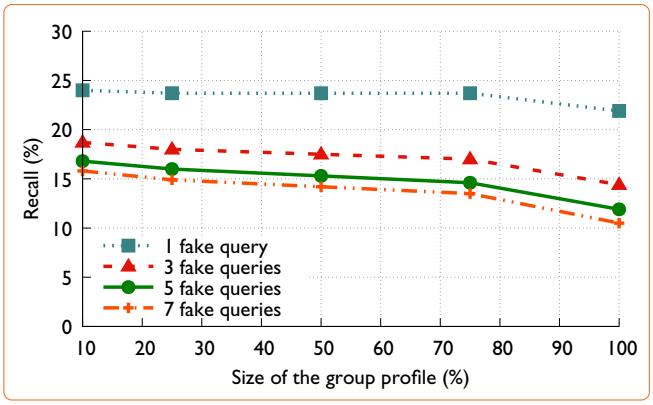
4.7.1.3 Impact of the Size of the Group Profile

PEAS leverages a group profile to generate fake queries. As described in Section 4.3.2.1, this group profile contains an aggregation of user queries. In this section, we evaluate the impact of the size of this group profile on the quality of the obfuscation. To achieve that, Figure 30 reports the recall provided by SimAttack according to the size of the group profile. The results show that the size of the group profile influences the outcome of the attack: Increasing the considered size better protects users. SimAttack provides a recall of 15.8% when only 10% of the group profile is considered, while this recall drops to 10.5% when the whole group profile is considered (both results are obtained when PEAS generates seven fake queries).

This result is counter-intuitive, as increasing the size of the group profile should decrease the quality of the generated fake queries. Fake queries are generated by extracting cliques from the group profile. A group profile contains co-occurrence matrices in which previous queries are aggregated. The aggregation makes the retrieval of the original data difficult due to a loss of information. As the vectorial space is a set of unigram (and not n-grams),

Figure 30

Increasing the size of the user profile improves the quality of fake queries.



some relations between keywords are lost. For instance, let us consider q the query "a b c", where a, b, and c are three different keywords. From the co-occurrence matrix which contained only q, it is possible to deduce that a was used with b, b was used with c, and c was used with a. But there is no information that a, b, and c were used in the same query. Nevertheless, as a, b, and c form a maximal clique, one can deduce that they might have been used together in a single query. But, considering two more queries "d b c" and "a d" (where d is a keyword different from a, b, and c) in the co-occurrence matrix, the maximal clique is now "a b c d", even though such a query was never sent. From this example, we deduce that it is easier to retrieve the original data from a small group profile than a larger one, as a small group profile might contain more queries with disjoint keywords than larger one and thus, the maximal cliques extracted from a small group profile might correspond to real queries and not to the merge of multiple real queries. Due to this reason, fake queries generated from a small group profile should be closer to real queries than fake queries generated from a large one. Consequently, increasing the size of the group profile should give a lower protection to users (as increasing the size of the group profile would produce fake queries that are farther from real queries). The reason why generating fake queries from a small group profile gives a lower protection is that to generate fake queries PEAS manipulates the maximal cliques extracted from the group profile (e.g., removing and adding keywords). For a small group profile, these operations transform the extracted maximal cliques into fake queries that differ from real ones. But, for a larger group profile, the manipulations produce fake queries that are closer to real ones.

4.7.1.4 Distributed Deployment of PEAS

As presented in Section 4.4, two group profile managements are available to deploy PEAS through a distributed setting. The group profile can be centralized (i.e., shared by all issuers), or decentralized (i.e., each issuer manages its own group profile). In terms of privacy, a centralized group profile provides the same guarantee as the original PEAS protocol (i.e., without a distributed deployment), as the centralized group profile is not impacted by the number of issuers. Therefore, the outcomes of the attacks against PEAS deployed with a centralized group profile are equivalent to those presented in Section 4.7.1.1.

However, the decentralization of the group profile between all issuers impacts the privacy. Indeed, the number of issuers changes the composition of their group profile and therefore, the generated fake queries. Figure 31

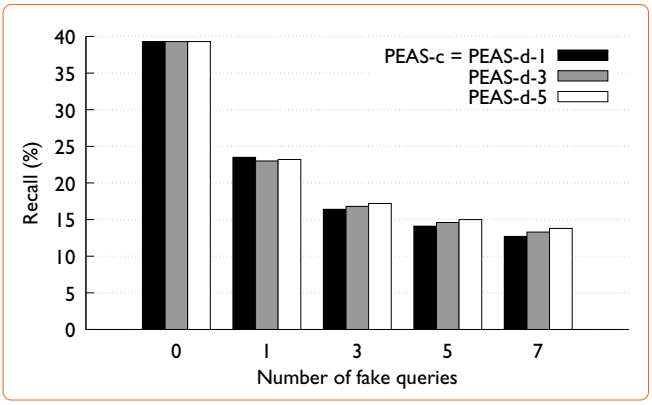


Figure 31
The distributed deployment does not really impact the privacy protection.

presents the recall achieved by SimAttack[4] for the two group profile managements according to the number of fake queries. The label "PEAS-c" corresponds to a centralized group profile while 'PEAS-d-x" denotes a decentralized group profile between x issuers. The results show that increasing the number of issuers and thus the number of group profiles, slightly affects the quality of the obfuscation. By increasing the number of issuers from 1 to 5, the recall increase by 1.1% on average. These results confirm the observations made in Section 4.7.1.3: Smaller group profiles tend to slightly reduce the privacy protection offered by PEAS. Indeed, increasing the number of issuers, decrease the size of the group profiles (as queries are split between multiple group profiles). Due to this reason, increasing the number of issuers slightly decreases the privacy protection.

4.7.2 Accuracy

In this section we evaluate the accuracy of PEAS. Our evaluation is performed in two steps: (i) we evaluate the filtering algorithm (i.e., more precisely, the capacity of PEAS to remove irrelevant results introduced by the fake queries), and (ii) we assess the influence of shuffling the keywords of the initial user query on the quality of the results. For the first step, we consider that the PEAS obfuscation mechanism does not shuffle the keywords of the initial user query, while for the second step, we consider that the PEAS obfuscation mechanism only shuffles keywords and does not introduce any fake query.

4.7.2.1 Impact of the Filtering Algorithm

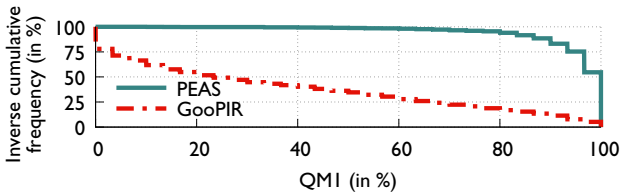
In this section, we measure the quality of the filtering algorithm by studying the proportion of results obtained without any protection mechanisms that are also retrieved by PEAS. Such a proportion corresponds to the metric QM_1 introduced in Section 4.6.2.2. For this experiment, we consider that PEAS does not shuffle the keywords of the initial user query. Figure 32(a) depicts the inverse cumulative frequency distribution of the QM_1 metric. The results show that for more than 50% of the initial queries, PEAS returns all the expected results (i.e., $QM_1 = 100\%$ for 54.6% of initial queries). Furthermore, for 95.5% of the initial queries, PEAS returns more than 80% of the expected results. This means that for 95.5% of the initial queries, more than 24 results out of 30 are correct.

Secondly, we measure the difference in the rankings between the results obtained for the initial query without obfuscation and the ones provided

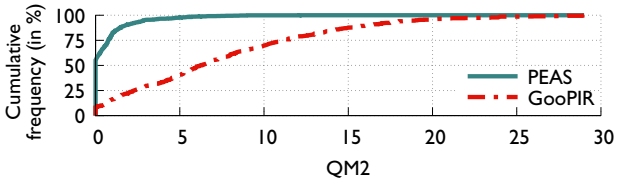
[4]We omit to report the results for ML Attack which provides a similar recall.

Figure 32

Comparison between the filtering performed by PEAS and the filtering performed by GooPIR.



(a) *PEAS filtering outperforms GooPIR filtering regardless of the number of expected results retrieved.*



(b) *PEAS better preserves the rankings of the expected results.*

by PEAS (i.e., through the QM_2 metric as defined in Section 4.6.2.2). Figure 32(b) presents the cumulative frequency distribution of the QM_2 metric. We show that the rankings of the results of PEAS are similar to the expected results for more than 50% of the initial queries. Moreover, for 90% of the initial queries, the ranking of the results after the filtering process only differs from the expected results by two units (i.e., QM_2 is less than or equal to two).

Lastly, we observe that the filtering step of PEAS retrieves significantly more results than GooPIR (on average 95.3% for PEAS against 36.3% for GooPIR). In addition, the results returned by PEAS are more accurate than GooPIR as the ranking of the results only differ on average by 0.6 units for PEAS compared to 5.8 units for GooPIR. The main difference between the two filtering algorithms is that GooPIR consider a result relevant if it contains all keywords used in the initial query, while PEAS identifies a result as relevant if the initial query contains the highest number of keywords in the results (compared to the fake queries).

4.7.2.2 Impact of the Word Order

As mentioned in Section 4.3.2.2, PEAS shuffles the keywords of all initial user queries to prevent an adversary from distinguishing fake queries from real ones. Fake queries might have an irrelevant keyword order (e.g., "Floyd Pink" instead of "Pink Floyd"). Therefore, by shuffling the keywords of the initial user queries, both real and fake queries might have inconsistencies. Such an operation reinforces the indistinguishability property between fake queries and real ones but might degrade the accuracy of the results, as search engines might exploit the keyword order to establish the result list. Consequently, we evaluate the influence of shuffling the keywords of the initial queries on the quality of the results. To do so, we compare the results returned by Google if the keyword order of a query is shuffled or not. We measure the difference between these two result list using QM_1. In this experiment, we consider that the PEAS obfuscation mechanism does not introduce any fake query. Figure 33 depicts the QM_1 metric for a varying number of results returned by Google (from 5 results to 50). The results confirm that Google takes into consideration the word order. For instance, for five results, the QM_1 metric is on average equal to 78.8%. Meaning that, on average 78.8% of the results returned for the non-shuffled query are also returned for the shuffled query. Therefore, due to the keyword shuffling, 21.2% of the results are not retrieved

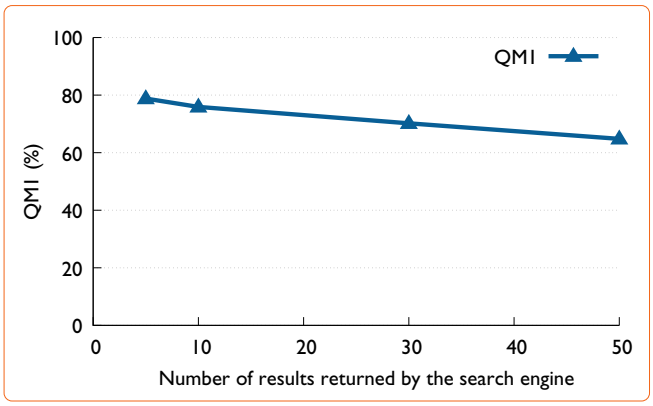


Figure 33
The word order of the query has a non-negligible impact on the results returned by the search engine.

for the shuffled query. Considering fifty results, QM_1 drops on average to 64.8% and thus, 35.2% of the results are not retrieved for the shuffled query.

Moreover, as users generally do not consult results beyond the top-10, we can consider that shuffling keywords in PEAS degrades the accuracy by approximately 20%. This percentage is relatively high but, it can be partially mitigated by reconstructing the keyword order. For instance, by using Markov chains [162] built on a large corpus, the issuer could reorder keywords to obtain more logical queries (e.g., reorder "Floyd Pink" in "Pink Floyd").

4.7.3 Performance

In this section, we evaluate the performance of PEAS on several aspects: execution time required on the client side, influence of the number of fake queries, comparison with the onion routing baseline (from both a theoretical and a practical perspective).

4.7.3.1 PEAS Client Evaluation

The PEAS client is in charge of many operations in the generation of fake queries. We focus the evaluation of the PEAS client on the most costly one, the extraction of the maximal cliques. We leverage the Bron-Kerbosch algorithm [156] to do this operation, the worst-case running time of this algorithm is $O(3^{n/3})$ [163] where n is the number of words in the co-occurrence matrices. However, as these co-occurrence matrices are generally sparse, it can be reduced to $O(dn3^{d/3})$ [164], where d is the maximum number of times a keyword co-occurs with other keywords. Figure 34 depicts the computation time required by the Bron-Kerbosch algorithm to extract maximal cliques

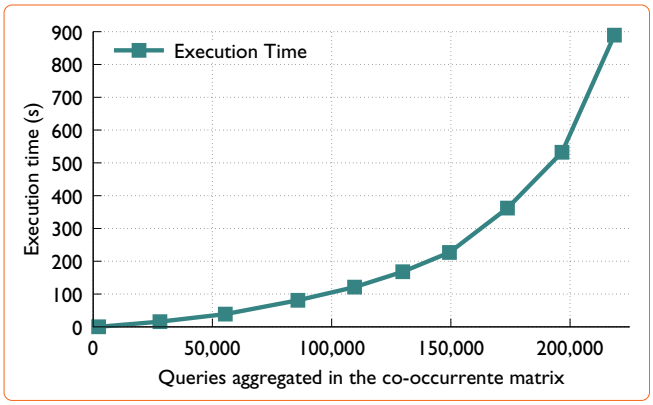


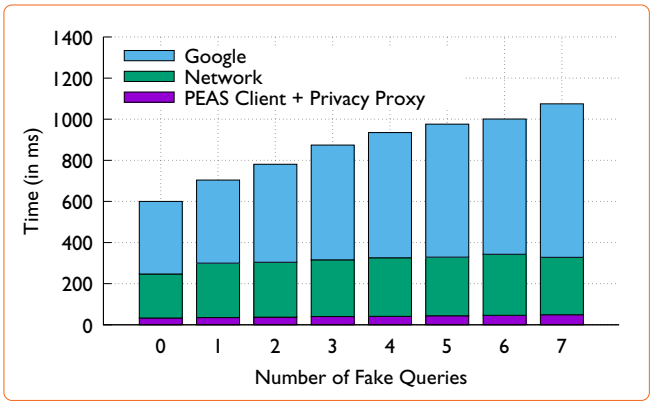
Figure 34
The maximal clique extraction performed by the client follows an exponential running time.

for an increasing number of queries aggregated in the co-occurrence matrix. Queries were randomly selected in the training set. The results show that the extraction follows an exponential running time. This could be an issue in practice but, to minimize the cost of this operation on the client, the extraction is executed only once upon the reception of a new version of the group profile. In addition, this extraction is performed in a background execution so that the performance of PEAS are not impacted by the clique extraction. The new version of the group profile is used when the extraction is completed.

4.7.3.2 Impact of the Number of Fake Queries

We want to measure the influence of the number of fake queries on the performance of PEAS. We measure the latency for a different number of fake queries. In the computation of the latency, we omit the maximal clique extraction step, as it is done only once for each new version of the group profile. The experiment was conducted on Google with 1,000 queries selected randomly from the test dataset. To minimize the influence of the size of the queries, we select only three-word queries. As previously mentioned, Google does not support multi-keyword queries sent with the OR operator. For that reason, to make the protected queries compliant with Google, we use the "Exact expression" feature. For instance, for a user query q sent with two fake queries fq_1 and fq_2, we issue to Google "fq_2" OR "q" OR "fq_1", where the quotes indicate that the exact expression. Besides, we consider that the search engine returns an equal number of results for each sub-query. Therefore, to return ten results related to the initial query, the total number of results evolves linearly according to the number of fake queries. We explicitly ask the search engine to return $10 \times (1 + nb_{fq})$ results, where nb_{fq} is the number of fake queries contained in the protected query. Figure 35 depicts how the latency evolves according to the number of fake queries. We note that the latency increases with the number of fake queries. For instance, sending the original query without fake queries takes on average 602.0 ms while for seven fake queries, it takes 1076.0 ms. More precisely, the operations performed by the client and the privacy proxy are negligible compared to the network delay and the time required by the search engine to retrieve the results. Furthermore, the main reason behind the increase of the latency is that the search engine requires more time to process larger queries. For zero fake query, Google takes 356.9 ms to process the query while for seven fake queries it takes 747.8 ms. Therefore, adding seven fake queries increases the time taken by Google to compute its results by 52.3%. It is difficult to explain this result, as Google's algorithms are not public. But two elements might explain this: (i) increas-

Figure 35
The latency is impacted by the number of fake queries.



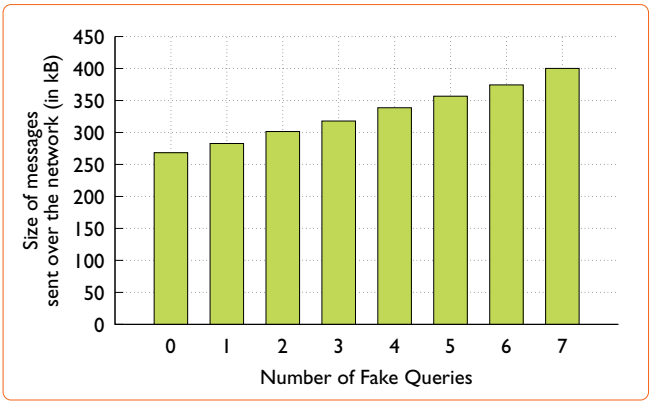


Figure 36
The size of the messages sent over the network increases according to the number of fake queries.

ing the number of fake queries makes the search engine looking for more documents, and (ii) in our evaluation, for a higher number of fake queries, Google is asked to return more results.

Furthermore, we also study the effect of the fake queries on the size of the messages sent over the network. We measure both the size of the query sent by the client and the response it receives from the search engine. Figure 36 relates the evolution of the size of these messages according to the number of fake queries. The results show that the size increases with the number of fake queries. If we consider a query sent without fake queries, the messages represent 268.4 kB while, for seven fake queries sent together with the initial query, the size increases to 400.2 kB. As a consequence, adding seven fake queries increases by 1.5 the size of the messages sent over the network. This increase can be explained by the experimental set up, as for a higher number of fake queries, we ask Google for a higher number of results (i.e., $10 \times [1 + nb_{fq}]$, where nb_{fq} is the number of fake queries). Consequently, if the answer returned by Google contains more results, its size naturally increases.

4.7.3.3 Scalability of PEAS

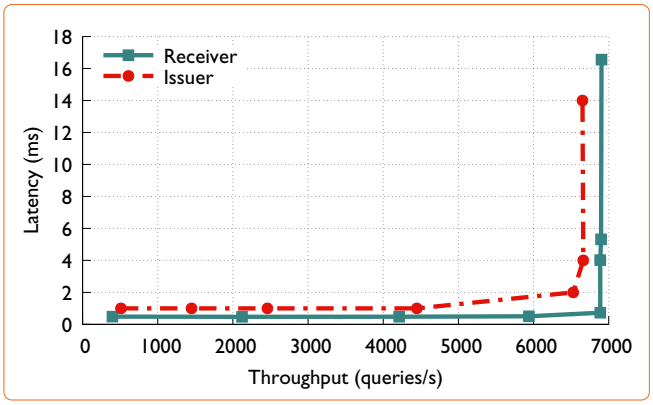
To practically evaluate the performance of PEAS, we measure the scalability of the components contained in the privacy proxy, namely the receiver and the issuer. Figure 37 reports the latency of the receiver and the issuer to separately process a query according to the load of the system (i.e., an increasing throughput). The results show that both the receiver and the issuer are able to deal with the increasing rate of queries up to a certain load where they become saturated and are not able to answer clients straightaway (i.e., incoming messages are queued). As a consequence, the saturation dramatically increases the latency.

In addition, the results show that the receiver supports a larger load than the issuer. Their maximum throughputs are 6,892.2 queries per second and 6,669.3 queries per second, respectively. Besides, for a given throughput, the latency of the receiver is 3.3 times lower than the latency observed on the issuer (on average the latency on the receiver is 472.0 μs against 1,542.7 μs on the issuer). Consequently, the receiver succeeds in processing more queries per second than the issuer. The better performance of the receiver is explained as, compared to the issuer, the receiver does not perform any cryptographic operation.

Furthermore, as the scalability of the receiver is only slightly better than the issuer, we deduce that the additional cryptographic operations performed by the issuer, do not have a strong impact on its performance. In addition, the

Figure 37

The scalability of the receiver and the issuer are similar, both serve queries up to a certain load where they become saturated and are no longer able to answer clients straightaway.



memory of the two nodes is not saturated, as 24 GB of RAM is large enough to execute the protocol. Therefore, we deduce that the limiting factor of PEAS comes from the network (and not from the cost of the protocol itself). The nodes used as receiver and issuer cannot deal with a higher number of messages.

4.7.3.4 Comparison With Onion Routing

Firstly, we theoretically evaluate the number of costly operations performed to send a query without protection mechanisms and protected by PEAS or the onion routing baseline. Table 8 compares the number of cryptographic operations and the number of messages for PEAS, for the onion routing baseline, and for a direct access to the search engine without privacy protection. We show that PEAS requires less cryptographic operations than onion routing baseline (half as much RSA encryption and decryption, and six times less AES encryption and decryption operations). In terms of traffic, PEAS sends 38% fewer messages than the onion routing baseline. Indeed, to query a search engine using PEAS requires 15 messages (9 for the TCP handshake and 6 for the protocol messages), while using Onion Routing requires 39 messages (9 for the TCP handshake, 18 for the TLS handshake and 12 for the protocol messages).

We also empirically compare the performance of PEAS against the onion routing baseline. Figure 38 reports the latency of both solutions for an increasing load. The results show that the latency of PEAS is 7.5 times lower than the latency achieved by the onion routing baseline (2 ms for PEAS against 15 ms for the onion routing baseline). As shown in Table 8, this difference is mainly due to the additional number of cryptographic operations performed by the onion routing baseline and to the associated renegotiation of the symmetric keys for each query.

In terms of scalability, PEAS manages to process more queries per second than the onion routing baseline. On average, the PEAS throughput is 3.1 times higher than the one obtained with the onion routing baseline. As for the latency, the difference is explained by the number of cryptographic

Table 8

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

Protocol	RSA operations	AES operations	Messages
Direct Access	0	0	5
PEAS	2	2	15
Onion Routing	4	12	39

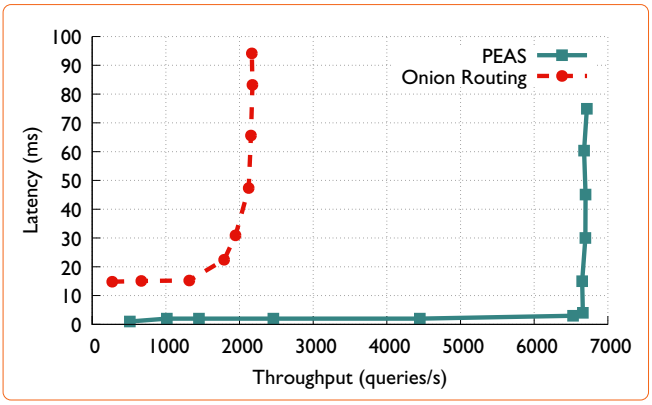


Figure 38
Comparison between the scalability of PEAS and Onion Routing.

operations performed and messages sent (see Table 8).

4.7.3.5 Scalability of PEAS Distributed Deployment

Finally, we evaluate the scalability improvement provided by the distributed deployment of PEAS. As shown previously, both the receiver and the issuer have approximately the same throughput. As a consequence, we consider the same number of receivers and issuers in this distributed deployment. We denote by *PEAS x-x* the deployment of the PEAS with x receivers and x issuers. Figure 39 depicts the latency for different deployments of PEAS (from one to ten receivers and issuers) and an increasing throughput. The results show that the latency slightly increases according to the throughput when the receivers and the issuers are not saturated. For instance, with a deployment of ten receivers and issuers, a throughput of 520.7 queries per second provides a latency of 2 ms, while a throughput of 24,345.1 queries per second only rises the latency to 4.7 ms. The increase of the latency is relatively important (more than two time) but, due to the order of magnitude, it is imperceptible to the users. Consequently, we can consider that the latency is stable as long as the receivers and the issuers are not saturated. Indeed, from a certain load, the system becomes saturated and the latency drastically increases (i.e., the system is no longer able to serve clients properly). Obviously, the more receivers and issuers in the system, the more the system can serve clients. Regarding the throughput, a distributed deployment of PEAS with one, two, five, and ten receivers and issuers provides a maximum throughput of 6,669, 13,284, 19,562, and 31,823 queries per second, respectively. Consequently, distributing the privacy proxy of PEAS with two, five and ten receivers and issuers increases

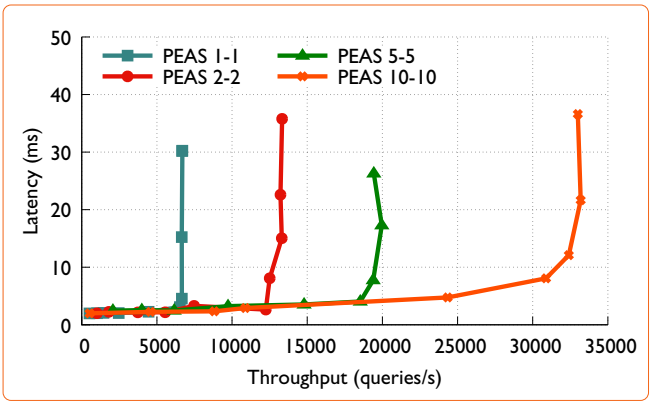


Figure 39
*Comparison between the scalability of PEAS for a different number of servers (*PEAS x-x *means that the PEAS protocol was deployed with x receivers and x issuers).*

the throughput of the system by 2.0, 2.9, and 4.8 respectively (compared to a non-distributed deployment). We note that the scalability is not linear. It is due to the socket management between receivers and issuers that requires synchronizations and therefore reduces the throughput of each issuer.

4.8 Conclusion

This chapter presented PEAS, a new privacy-preserving solution to query search engines. PEAS combines a new efficient unlinkability protocol with a new accurate indistinguishability solution. While the former hides the identity of a requester, the latter obfuscates user queries with k fake queries to ensure that an adversary will not be able to identify user queries.

We exhaustively evaluated PEAS using real search queries and showed that PEAS meets the expectations: it improves the privacy protection by reducing the number of de-anonymized queries compared to (i) an onion routing baseline and (ii) GooPIR over an unlinkability solution. The de-anonymized queries decrease respectively by 26.6% and 18.4%. Furthermore, PEAS reduces the overhead in terms of latency and throughput compared to the onion routing baseline (the latency of PEAS is 7.5 lower and its throughput 3.1 higher). Regarding the quality of the results, PEAS is better than GooPIR, as PEAS retrieves on average 95.3% of the initial results while GooPIR retrieves only 36.3%. Finally, the distributed deployment of PEAS provides a better scalability of the system (the throughput increases by 4.8 for ten issuers) without significantly reducing the privacy protection (the recall of SimAttack increases by 1.1%).

The obfuscation of PEAS strongly relies on the group profile. We did not investigate in our work how the system designer should configure the system to obtain the optimal obfuscation. Two factors should be studied for a deployment: how often the group profile is updated and how many queries the group profile should contain. Moreover, PEAS obfuscates user queries by adding k fake queries. Nevertheless, as mentioned in Section 4.6.2.2, the current search engines (i.e., Google, Yahoo!, and Bing) do not process correctly boolean queries. Therefore, PEAS should find another mechanism to protect queries with fake queries. A possible solution is to send queries and fake queries successively (as it is done in TMN). Besides, such a solution improves the accuracy (the results of each query are returned independently) but decreases the performance (more traffic generated).

CHAPTER 5

SAM and LAM, Identifying Sensitive Queries to Improve the Efficiency of Private Web Search Solutions

5.1 Objective

Private Web search solutions allow users to protect their privacy against a curious search engine. But these mechanisms require (i) more computation than sending non-protected queries to search engines, and (ii) more data sent over the network. They also degrade the accuracy of the results (e.g., introducing irrelevant results due to the fake queries). These drawbacks can be reduced by protecting queries that only require a protection. For instance, if banal queries do not reveal personal information, they do not need to be protected. Therefore, many resources (e.g., CPU, network traffic) could be saved by adapting the protection to each query. This chapter studies the trade-off between protecting the lowest number of queries (i.e., the sensitive queries) and preserving the user privacy. Ultimately, we want to adapt current privacy-preserving Web search solutions to only protect sensitive queries, but without decreasing the user protection.

In this work, we denote a sensitive query as a query related to an embarrassing topic (e.g., sexual behavior or health condition) and a linkable query as a query that can be linked back to its originating user. As illustrated in Section 2.2.1, defining embarrassing topics is not easy, as there is no consensus on such a definition. Consequently, we consider in our work a user-centric model. Users are invited to personalize their own list of sensitive topics. We further detail how it is done in practice. Furthermore, linkable queries are defined by one of the privacy attacks presented in this thesis (e.g., SimAttack or the machine learning attack). More precisely, a linkable query is an anonymous query for which the adversary successfully retrieves the identity of its requester.

In the first section, we analyze the overhead produced in protecting all queries similarly. Then, we introduce two modules to identify sensitive queries and discuss their deployment on existing private Web search solutions. Finally, we evaluate the accuracy of these modules and their influence on the

query protection.

5.2 Overhead Made by Private Web Search Solutions

In this section, we assess the proportion of sensitive queries sent by users and deduce the proportion of queries that should not be protected by the current privacy preserving solutions. We perform an empirical evaluation using the dataset of 18,164 users presented in Section 3.3.1 (i.e., a subset of the AOL dataset containing the most active users). The evaluation requires queries to be annotated according to their sensitivity. Nevertheless, as the dataset contains too many queries to manually annotate their sensitivity, it is not suitable for our evaluation. For that reason, we select 198 users among the most active users who sent at least one semantically sensitive query. We identified these users manually from the most active to the least active and stopped when we could extract a test set of 10,000 queries. Moreover, we consider an adversary with prior knowledge on users. Therefore, similarly to the previous chapters, we created a training set and a test set by splitting queries sent by the 198 users in ⅔, ⅓. The training set contains the first ⅔ of queries sent by the 198 users, while the test set contains 10,000 queries extracted from the remaining ⅓. These 10,000 queries represent for each user her first 44.6% of queries in the remaining ⅓, considering queries ordered by time. As a result, our training dataset and test dataset are composed of respectively 96,548 queries (487.6 queries per user on average) and 10,000 queries (50.5 queries per user on average).

In the following sections, we show that private Web search solutions produce an unnecessary overhead by protecting semantically non-sensitive queries (i.e., queries not related to an embarrassing topic) and non-linkable queries (i.e., queries that cannot be linked back to their requester user profile). We focus our analysis on Tor (unlinkability solutions), GooPIR, and PEAS.

5.2.1 Overhead Due to the Protection of Semantically Non-sensitive Queries

We first analyze the proportion of semantically sensitive queries sent by a user. Then, we illustrate the overhead produced by the three aforementioned private Web search solutions.

5.2.1.1 Proportion of Semantically Sensitive Queries

To analyze the proportion of semantically sensitive queries sent by a user, we conducted a crowd-sourcing study in which we asked workers to annotate a set of queries. The study consisted in annotating the 10,000 queries contained in the test dataset according to their sensitivity. In the instructions we gave to the workers (see Appendix D), we defined a semantically sensitive query as a "*query that they do not want to make public (e.g., relating to an embarrassing topic, revealing personal behavior/information)*". After reading the instructions, the workers did the questionnaire that contained between six and eight queries for a reward of €0.05. Each query was annotated by five different annotators to obtain multiple opinions. The crowd-sourcing campaign was conducted using Crowdflower[1]. We obtained a total of 1,456 workers who participated in our study. Some of them participated more than once, meaning that they answered multiple times the questionnaire with different queries.

[1]Accessible at: `http://www.crowdflower.com`

Nevertheless, we limited to two the number of questionnaires a single user could complete. Thus, a user annotated a maximum of between 12 and 16 queries. We also configured *golden questions* to avoid free-riders. They correspond to queries we annotated ourselves so that Crowdflower can compare users' results and detect misbehaving users. These queries were randomly introduced among the 10,000 queries annotated by workers. Furthermore, to validate the quality of the annotations made by the workers, we use Fleiss' kappa [165]. This metric assesses the reliability of agreement between annotators. We obtained a score of 0.44, which according to Landis and Koch's scale [166], can be considered as a moderate agreement. We found two reasons to explain this level of agreement: (i) annotators do not always agree on what is semantically sensitive, e.g. for the query *treatment marijuana addiction*, three annotators said that it is sensitive whereas two other annotators said it is non-sensitive, and (ii) some queries can have two different meanings: one semantically sensitive and another one semantically non-sensitive, e.g. the query *young Russian girl* can either refer to teenagers in Russia or be interpreted as a pornographic query. To create the final dataset of annotated queries, we needed to aggregate the five annotations into a single one. Several aggregation techniques have been published in the literature: Majority Vote [167], EM [168], GLAD [169]. For our study, to limit the impact of the outliers, we assigned to each query the annotation chosen by a majority of workers. Of the 10,000 queries annotated, we found that 1,574 queries are semantically sensitive while the remaining 8,426 queries are not semantically sensitive. If we analyze the proportion of semantically sensitive queries per user, we note that on average users send 18.9% of semantically sensitive queries (with a standard deviation of 18.1%). These results were obtained with different users than the ones that originally issued the queries. For that reason, our results would have been different with the original users. Therefore, we can conclude from this study that *a priori* semantically non-sensitive queries represent a majority of users' queries.

5.2.1.2 Impact on Protecting Semantically Non-sensitive Queries

If we consider the results of the previous section, there are 1,574 semantically sensitive queries in the annotated dataset. Consequently, instead of protecting the 10,000 queries, protection mechanisms could only protect the 1,574 sensitive queries. Therefore, the different parties (i.e., users, relays, the search engine, and the network) could save resources. Indeed, if we consider Tor, GooPIR, and PEAS, the cost on the user-side due to the fake queries or the cryptographic operations could decrease by 84.3%. More precisely, the number of fake queries generated by GooPIR or PEAS decreases from 70,000 to 11,018 and the number of cryptographic operations performed by Tor would decrease from 150,000 to 23,610. In addition, in the specific case of Tor, the number of messages processed by users would drop by 73.7% (from 80,000 messages to 21,018 messages). Furthermore, for the relays (involved in Tor or PEAS), adapting the protection mechanism could reduce cryptographic operations and exchanged messages by 84.3%. In practice, such a decrease could allow relays to process queries faster and thus reduce the latency. Regarding the search engine, the benefit due to a lower number of fake queries would save its resources by 73.7%. This gain is very profitable in particular for widely used search engines like Google, as they receive approximately 3.5 billion search queries per day. Finally, the adaptation profits to the network as a lower number of smaller messages would be sent.

5.2.2 Overhead Due to the Protection of Non-Linkable Queries

Linkable queries are anonymous queries that can be linked back to their original users. In practice, we consider that linkable queries are queries correctly de-anonymized by at least one of the privacy attacks: SimAttack (see Chapter 3) or ML Attack (see Section 3.3.3). These attacks consider an adversary who previously collected queries of the users in the system. The results presented in this section give a lower bound on the proportion of linkable queries, as an adversary might find a better attack to de-anonymize queries.

In the next section, we analyze the proportion of linkable queries in the annotated dataset. Then, based on this value, we quantify the cost of protecting non-linkable queries for PEAS. Indeed, only PEAS is concerned by adapting its protection mechanism to linkable queries, as the notion of *linkability* requires the use of an unlinkability solution and the notion of *adaptation* requires the use of an extra protection mechanism. As a result, adapting protection mechanisms to linkable queries concerns indistinguishability solutions that runs on top of an unlinkability protocol. To the best of our knowledge, only PEAS corresponds to such a solution in the literature.

5.2.2.1 Proportion of Linkable Queries

We assess the proportion of linkable queries contained in the annotated dataset. Our experiment considers linkable queries as anonymous queries correctly retrieved by at least SimAttack or ML Attack. These privacy attacks require an adversary with prior knowledge on the users. Therefore, we consider queries in the training dataset as the previous queries collected by the adversary. Our experiment anonymizes the 10,000 queries contained in the test dataset (such that the adversary has no information about the identity of their requesters) and executes the two privacy attacks to de-anonymize them. In Figure 40, we present the proportion of linkable queries for each of the 198 users. These users are ranked by their proportion of linkable queries. We note that the proportion of linkable queries significantly varies from user to user (from 3.0% to 87.6%). On average, 35.3% of user queries are linkable. Furthermore, we study the overlap between the results of the two privacy attacks. We found that 65.1% of linkable queries were retrieved by the two privacy attacks. It shows that a large proportion of linkable queries can be easily de-anonymized. Nevertheless, these results are established on a small dataset where the 198 users do not represent the billions of users using search engines. Therefore, in a real context, the proportion of linkable queries might be lower.

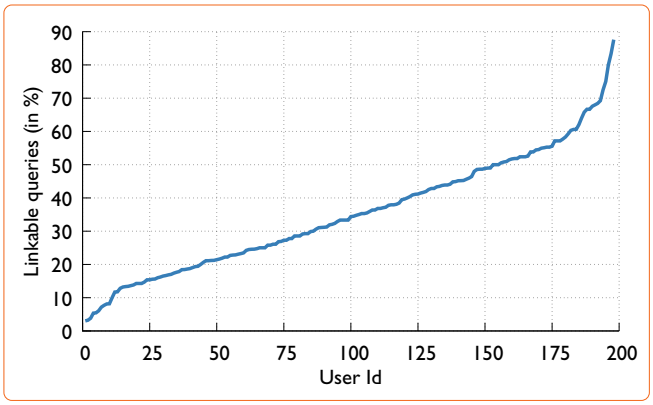


Figure 40
Percentage of linkable queries per user.

5.2.2.2 Impact on Protecting Non-Linkable Queries

Non-linkable queries are well protected by an unlinkability protocol (e.g., an anonymous network protocol), as they cannot be de-anonymized by the existing linkability attacks. As we have seen in the previous section, linkable queries represent 35.3% of users' queries in the annotated dataset. Hence, non-linkable queries represent a large majority of users' queries (i.e., on average, 64.7% of their queries are non-linkable). This large proportion suggests that PEAS wastes many resources by obfuscating these queries. By protecting non-linkable queries with fake queries, PEAS engenders extra computations for the client and for the search engine (i.e., more queries need to be processed). In addition, the network is also impacted: more fake queries increases the size of the exchanged packets. Adapting PEAS to linkable queries consists in obfuscating linkable queries only. Such enhancement of PEAS decreases up to 64.7% the cost due to the obfuscation of queries.

5.3 Sensitivity Assessment

To reduce the cost of protecting queries, we design two assessment modules to identify sensitive queries: (i) the Semantic Assessment Module (SAM) detects semantically sensitive queries, and (ii) the Linkability Assessment Module (LAM) identifies linkable queries. We first give an overview of the two assessment modules and formalize the notion of sensitivity (considering a user-centric model). Then, we present the two assessment modules in detail: first the Semantic Assessment Module and then the Linkability Assessment Module.

5.3.1 Overview

Our work aims to identify if a given query is sensitive. By sensitive, we target both semantically sensitive queries and linkable queries. For that reason, we design two independent modules: the Semantic Assessment Module (SAM) and the Linkability Assessment Module (LAM). These modules need to respect two requirements: (i) they cannot rely on external services, as we do not want to trust any distant server, and (ii) they cannot rely on costly algorithms or heavy databases, as they have to run on a personal computer or a smartphone. For that reason, SAM identifies semantically sensitive queries by detecting queries related to an embarrassing topic. The identification is performed using a local copy of WordNet [82] (a lexical database) and a mapping in a pre-defined set of categories. Concerning LAM, it detects linkable queries by computing a similarity metric with all queries already sent by the user.

5.3.2 User Centric Sensitivity Modeling

The sensitivity of a query depends on two aspects: if the query is related to an embarrassing topic or if the query is linkable to its requester's profile. As we have seen in Section 5.2, these two concepts have a different meaning from user to user. Therefore, we need to personalize the definition of linkable queries for each user. For embarrassing topics, we consider that users can select in a pre-established list of categories, the ones that they find embarrassing. To establish the list of categories, we consider the ones provided by eXtended WordNet Domains (XWND) [170], as our implementation of SAM use this library. Using the same categories to model users avoids the

problem of mapping a pre-defined set of categories to another one. XWND provides 170 categories (all available categories are listed in Appendix E). The relatively high number of categories gives users the possibility to carefully personalize their embarrassing topics.

Regarding the notion of linkability, we personalize its definition for each user by considering her previous queries. Indeed, to de-anonymize a query, the adversary uses a set of previously-collected queries she has about the user (i.e., when the user did not protect her queries). Therefore, the linkability of the query is assessed by measuring its similarity with the set of queries previously sent by the user. Depending on the value of the metric, the query is considered either linkable or non-linkable. Nevertheless, the notion of linkability depends on the popularity of a query. If a query has been sent by many users, it might be difficult for privacy attacks to retrieve her original requester precisely. But, as a given user has in general no information about the other users in the system, LAM cannot assess the popularity of a query. Consequently, LAM might over-detect linkable queries.

5.3.3 Semantic Assessment Module (SAM)

As mentioned in Section 5.3.1, assessment modules cannot rely on external services, costly algorithms or heavy databases. For that reason, existing query categorizers [135, 129, 87] cannot be used to detect sensitive queries, as they exploit DBpedia or require to set up a local search engine. Consequently, we propose a new module to identify queries related to an embarrassing topic. We base our module on two libraries: (i) WordNet [82], a lexical database, and (ii) eXtended WordNet Domains (XWND) [170], an extension of WordNet to add categories. WordNet groups words into sets of synonyms called *synsets* and provides the relations between those synsets (e.g., hypernym, hyponym) while XWND enriches WordNet by mapping those synsets to a set of categories. The storage required by the two libraries is compatible with our requirements: the former takes 37.4 MB while the latter requires 5.8 MB. For instance, they can easily be stored on smartphones, as the current minimum storage capacity is approximately 8 GB. Regarding the computation constraints, exploiting these libraries does not suppose heavy computations. WordNet can be seen as an index between words and synsets (i.e., we only exploit hypernym/hyponym relationships to disambiguate synsets) and XWND is an index between synsets and categories.

SAM as presented in Algorithm 11 takes as input a query q, a set of sensitive categories SC (categories classified as embarrassing by the user), a dictionary D (that pairs categories with a set of specific terms), the maximum number of synsets per word s and the maximum number of categories per synset c. Algorithm 11 calls the function `getMostProbSynsets(`q, s`)` (line 2) to identify for each keyword w in the query q all possible synsets in WordNet (i.e., all possible meanings of the keyword in WordNet). This function returns for each keyword w the s most probable synsets (using a graph-based disambiguation method [171] with the Wu and Palmer metric [172]). However, it appears that keywords from specific domains cannot be found in WordNet (e.g., slang words contained in porn-related queries or specific diseases contained in health queries). Therefore, we introduce a dictionary D that pairs categories k with a set of specific terms v. If the keyword w is contained in one of the sets of terms v, the category k (corresponding to the set of terms v) is aggregated in the set C (lines 3 to 5). Otherwise, in the general case, the algorithm maps each synset retrieved by WordNet to multiple categories by calling `getCategories()` (line 7). This function uses XWND to return the c

Algorithm 11: Semantic Assessment Module.

```
input: q: query,
       SC: set of sensitive categories,
       D: dictionary that pairs categories with a set of terms,
       s: maximum number of synsets per word,
       c: maximum number of categories per synset.
1  C ← ∅ ; // Categories related to the query q
2  for (w, S) ∈ getMostProbableSynsets(q, s) do
3      if |S| = 0 then
           // Identify terms that are not in WordNet
4          for (k, v) ∈ D do
5              if w ∈ v then C ← C ∪ {k} ;
       // Map synset to categories
6      for synset ∈ S do
7          C ← C ∪ getCategories(synset, c) ;
   // Look if C contains sensitive categories
8  if |C ∩ SC| > 0 then
9      return sensitive ;
10 return nonSensitive ;
```

most probable categories associated to a synset. The categories retrieved for the s synsets associated to the keyword w are aggregated in the set C. Finally, SAM identifies sensitive queries by looking if the set C contains a sensitive category. If it is the case, SAM considers the query q as sensitive (line 9). Otherwise, the query q is considered as non-sensitive (line 10). The complexity of this algorithm is $O(|q|^3)$. In Section 5.6.1, we discuss the choice of the number of synsets s, the number of categories c, and the dictionary D.

5.3.4 Linkability Assessment Module (LAM)

The goal of the Linkability Assessment Module (LAM) is to determine if knowing a set of previous queries sent by the user, an adversary is able to de-anonymize a protected query. We identify two possibilities to obtain a successful attack: (i) the user has already issued the query (i.e., the query is already in the user profile) or (ii) the query is close to a previous query sent by the user (e.g., a subset of its keywords was already used in a past query which is stored in the user profile). Therefore, we design LAM to detect if a query is in one of these two cases. Nevertheless, the decision regarding the linkability needs to exploit the number of common keywords the query has with the previous ones and the frequency of these previous queries. For instance, if the query was already issued 20 times, it is more likely to suffer from a re-identification attack than if it was issued only once. Similarly, if a query has five words in common with a previous query, it is more likely to be retrieved than if it has only one.

To implement LAM, we use the similarity metric designed for SimAttack. This metric takes the two previous aspects into consideration (i.e., the frequency and the number of common keywords). The algorithm to compute the similarity metric was presented in Section 3.2.1 with Algorithm 1. It returns a similarity value between a query and a user profile. We consider as user profile the collection of queries previously sent by the user. Then, we define a threshold λ above which the query is considered as linkable. This threshold

varies between 0 and 1. Therefore, we consider that a query with a similarity value above λ is similar enough to its requester's profile and thus can potentially be de-anonymized by a privacy attack. We present in Section 5.6.2 how we empirically chose the smoothing factor α (used in Algorithm 1) and the linkability threshold λ.

5.4 Deployment

In this section, we detail the deployment of the two assessment modules for three existing private Web search solutions: Tor, GooPIR, and PEAS. We summarize this deployment in Figure 41 . The adaptation of Tor sends queries through the Tor network only if they are assessed as sensitive by SAM. Otherwise, they are sent directly to the search engine. We denote this protection mechanism by Tor+SAM. The adaptation of GooPIR is similar to Tor: if the query is sensitive according to SAM, it is protected with fake queries. Otherwise, it is sent without any fake query. This adapted protection is called in the following sections GooPIR+SAM. The adaptation of PEAS uses both SAM and LAM. It works as follows: (i) if the query is identified as non-sensitive by SAM, the query is not protected; (ii) if the query is determined as sensitive (by SAM) and non-linkable (by LAM), the query is protected by the unlinkability protocol of PEAS (i.e., queries are only sent through the privacy proxy); and (iii), if the query is identified as sensitive (by SAM) and linkable (by LAM), the query is protected by the full PEAS solution (the unlinkability protocol and the fake queries). One can wonder why in (ii) queries are protected with the unlinkability protocol and not by generating fake queries. As these queries are non-linkable, an adversary is not able to de-anonymize them (if they are protected by the unlinkability protocol) whereas, if they were protected with the obfuscation mechanism, there is no guarantee that an adversary could not de-obfuscate them. We refer to this solution later on the manuscript by PEAS+SAM+LAM.

5.5 Experimental Set-up

In this section, we provide the experimental set-up for our evaluation. We present the datasets of real search queries we used to evaluate the two assessment modules separately and their deployment on existing private Web search solutions. We formalize the definition of sensitive queries we followed

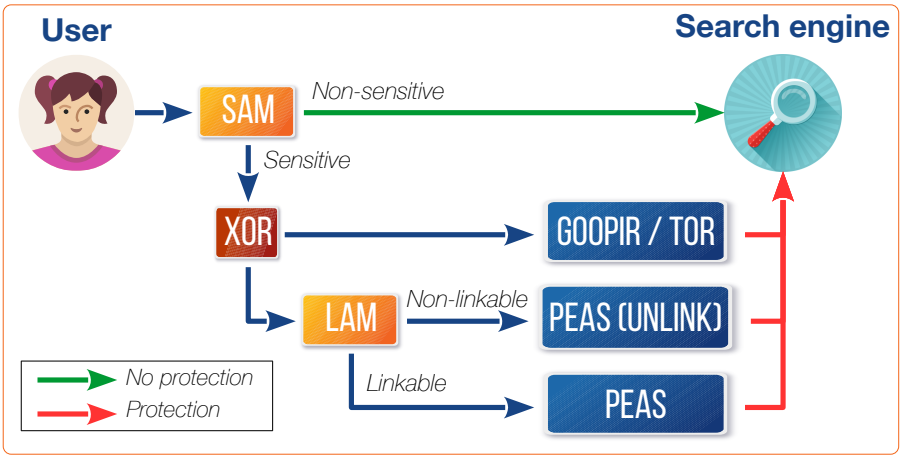


Figure 41
Deployment of SAM and LAM for the three existing private Web search solutions: Tor, GooPIR, and PEAS.

in the evaluation section. Finally, we present the metrics we used to evaluate the assessment modules and discuss the selection of the four parameters they used.

5.5.1 Datasets

To properly evaluate our assessment modules, queries from the AOL dataset need two types of annotations: if they are linkable and if they are semantically sensitive. The annotation of sensitive queries requires a manual annotation. Consequently, we constructed two different datasets, a small one due to the manual annotation, and a bigger one to only evaluate LAM. Regarding the linkability annotation, we use the datasets introduced in Section 3.3.1.4. It consists of five datasets with the most active users in the AOL search log dataset [45]. They contain respectively the 100, 200, 300, 500, and 1,000 most active users.

Regarding the creation of the sensitivity annotated dataset, we conducted a crowd-sourcing campaign to annotate queries according to their sensitivity (semantically). However, the annotations made in the crowd-sourcing campaign presented in Section 5.2 cannot be used in our evaluation section, as we do not know the categories – among the 170 available categories – the annotators considered as embarrassing. Consequently, we cannot configure correctly SAM and thus validate its efficiency. To overcome this problem, we selected first the categories we consider as embarrassing and then, with a new crowd-sourcing campaign, we asked annotators to identify queries related to such categories. To define which categories we found sensitive, we used the definition provided by Google. It is written in Google's privacy policy [3] that sensitive personal information concerns "*confidential medical facts, racial or ethnic origins, political or religious beliefs or sexuality*". Consequently, we consider only queries which refer to "Health/Medical facts", "Politics", "Porn/Sexuality", and "Religion" as semantically sensitive. In the new crowd-sourcing campaign we conducted, we asked annotators to classify queries according to these pre-defined sensitive categories. For this campaign, we selected the same queries as we selected for the first campaign presented in Section 5.2 (i.e., 10,000 queries issued by 198 users). To validate the quality of the annotations, we used Fleiss kappa [165] and obtained a score of 0.73. This score corresponds to a substantial agreement (according to Landis and Kochs scale [166]). In this dataset, queries are distributed as follows: 16.4% for "Health/Medical facts", 2.2% for "Politics", 12.3% for "Porn/Sexuality", 3.9% for "Religion" and 65.3% for "Other". Consequently, we obtained a dataset of 3,473 sensitive queries and 6,527 non-sensitive queries (semantically). Compared to the first annotated dataset obtained in Section 5.2, this new dataset contained two times more sensitive queries (3,473 sensitive queries versus 1,574). This means that *a priori* the users involved in the crowd-sourcing campaign do not consider that all queries related to health, politics, porn, and religion are sensitive. Furthermore, to evaluate the deployment of the assessment modules on real solutions, we also annotated these queries in terms of linkability. As a result, we obtained a dataset of 10,000 queries annotated according to their sensitivity (semantically) and their linkability.

5.5.2 Embarrassing Categories

To set up SAM, users have to indicate a list of categories they consider as sensitive. For our evaluation, we define semantically sensitive queries as queries that refer to "Health/Medical facts", "Politics", "Porn/Sexuality", "Religion".

Table 9
Categories considered as embarrassing categories.

anatomy	medicine	psychiatry	radiology	sexuality
dentistry	pharmacy	psychology	religion	surgery
health	politics	psychoanalysis	roman catholic	theology

In order to have a compliant definition with the Semantic Assessment Module, we need to map these four categories to the 170 categories available in SAM (see Appendix E). Therefore, we manually identified 15 categories corresponding to our definition of sensitive queries (see Table 9).

5.5.3 Evaluation Metrics

Our evaluation section contains two types of experiments. We first evaluate the assessment modules, and then we analyze their deployment on real solutions. To evaluate the assessment modules, we define two metrics: the recall and the fall-out. The recall is computed as the percentage of sensitive queries correctly identified as sensitive while the fall-out is the percentage of non-sensitive queries identified as sensitive. More formally, the recall and the fall-out are defined as follows:

$$recall = \frac{TP}{TP + FN},$$
$$fall\text{-}out = \frac{FP}{FP + TN},$$

where TP is the number of true positives (i.e., semantically sensitive queries identified as semantically sensitive), FN is the number of false negatives (i.e., semantically sensitive queries identified as semantically non-sensitive), FP is the number of false positive (i.e., semantically non-sensitive queries identified as semantically sensitive) and TN is the number of true negative (i.e., semantically non-sensitive queries identified as non-sensitive). The recall provides an insight about the privacy protection (i.e., the percentage of sensitive query that will be protected), while the fall-out gives an indication of the performance (i.e., the percentage of non-sensitive queries that will be protected). A perfect module has a recall of 100% and a fall-out of 0%.

The recall and the fall-out can be combined in a single metric with the harmonic mean H between $recall$ and $(1 - fall\text{-}out)$:

$$H = 2 \times \frac{(1 - fall\text{-}out) \times recall}{1 - fall\text{-}out + recall}.$$

The best value for the fall-out is 0. Therefore, to have a value compatible with the recall (which have 1 as best value), we operate the fall-out as $(1 - fall\text{-}out)$. This metric put the same weight on the recall and the fall-out. Nevertheless, as our objective is to improve the performance without impacting the user privacy, it might be interesting to define a metric that puts more weight on the recall than the fall-out. Similarly to the F_β score, we define the J_β score:

$$J_\beta = (1 + \beta^2) \times \frac{(1 - fall\text{-}out) \times recall}{\beta^2 \times (1 - fall\text{-}out) + recall}.$$

This measure gives β times more importance to the recall than the fall-out.

5.5.4 Creating the dictionary D used by SAM

Even though WordNet contains a reference to a large number of words (i.e., 147,278 unique words for the version 3.0), it does not contain all possible

words. Therefore, if a query q contains some keywords that are not in WordNet, its correct categories cannot be found. To solve this issue, SAM uses a dictionary D that contains external sets of terms. These terms are grouped by categories such that if a keyword in the query q is included in a set of terms associated with the category c, SAM deduces that the query q is related to the category c. For the experimentation, we use two different sets of terms: a set of medical terms[2] and a set of pornographic terms[2]. These two sets of terms allow SAM to identify slang words contained in porn-related queries or specific diseases contained in health queries.

5.6 Evaluation

We first evaluate how SAM and LAM identifies semantically sensitive queries and linkable queries. Then, we analyze the impact of deploying these two modules on existing private Web search solutions: Tor, GooPIR, and PEAS. The evaluation employs the machine learning attack to (i) define the notion of linkable queries and (ii) test the robustness of private Web search solutions. We note that SimAttack could have been used but, as LAM and SimAttack identify linkable queries with the same similarity metric, using SimAttack could have led to a bias. In addition, Chapter 3 showed that SimAttack and ML Attack have similar results (even though SimAttack is slightly better). Therefore, it makes sense to limit our evaluation to ML Attack.

5.6.1 Evaluation of the Semantic Assessment Module

The evaluation of SAM is conducted with the dataset of sensitive queries annotated with the crowd-sourcing campaign. For each query in the dataset, SAM identifies if it belongs to an embarrassing category. Then, by comparing the results of SAM with the ones obtained through the crowd-sourcing campaign, we compute the recall and the fall-out (metrics presented in Section 5.5.3). Nevertheless, to identify semantically sensitive queries, SAM requires to set up two parameters: the maximum number of synsets per keyword s and the maximum number of categories per synsets c. Figure 42 presents the recall and the fall-out for different pairs (s,c), where s and c varies from 1 to 4. We note the large influence that the two parameters s and c have on the performance of SAM: the recall evolves between 68.9% and 91.0% while the fall-out evolves between 18.6% and 59.0%. As we can

[2] Accessible at: https://framabin.org/?8ba726c0801b4c84#RprFwEJ9jHmcS05rGbpGnAGVyG7roaX1nZ3nwGFzwGI=

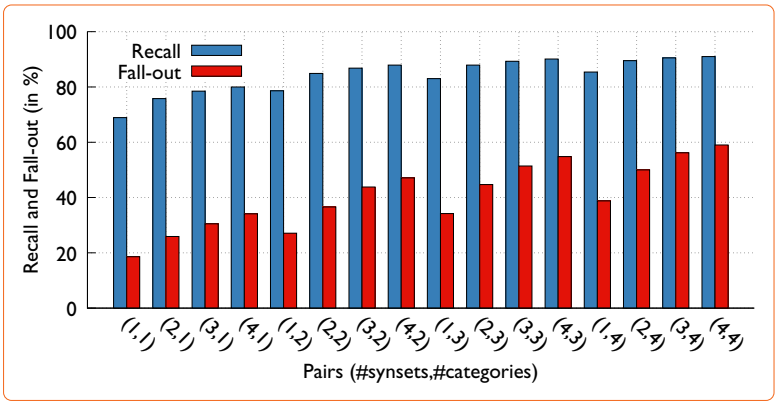


Figure 42

The recall and the fall-out obtained with SAM according to the maximum number of synsets per keyword s and the maximum number of categories per synsets c.

Table 10

Impact of the number of synsets and categories on the J_3 score (in %) obtained with SAM.

		Number of categories c			
		1	2	3	4
Number of synsets s	1	70.0	78.0	80.9	82.2
	2	75.7	82.1	83.0	82.9
	3	77.5	82.3	82.4	81.8
	4	78.3	82.5	82.0	81.1

notice, it is hard to decide which pair of parameters s and c gives the best trade-off between recall and fall-out. Our objective is to find the pair that maximizes the recall (which provides an insight about the user protection) and minimizes the fall-out (which gives an insight about the performance). To simplify the decision, we can combine the recall and the fall-out in a single metric through the J_β score. As our objective is to improve the performance of private Web search solutions without impacting the user privacy, we chose the J_3 score[3]. We depict in Table 10 the J_3 score for different values of s and c. We chose s and c such as they maximize the J_3 score. We note that the best value (83.0%) is obtained for s equals two and c equals three. Therefore, we chose these values in our experiments. Nevertheless, as these two parameters might be dataset dependent, they might not maximize the J_3 score on a different dataset. To solve this issue in practice, users can personalize these two parameters for their own queries. For a short period of time, they can annotate the sensitivity of their own queries, and then SAM can set-up s and c that maximize the J_3 score on their own queries.

With s equals two and c equals three, we obtain a recall of 87.9% and a fall-out of 44.7%. Consequently, SAM identifies a large number of semantically sensitive queries (87.9%) and misclassifies a reasonable number of semantically non-sensitive queries (44.7%). In practice, this means that SAM divides by two the number of semantically non-sensitive queries protected without significantly degrading the level of protection (only 12.1% of semantically sensitive queries are not identified by SAM). Nevertheless, depending on users, 12.1% of semantically sensitive queries not identified by SAM can be too much. A better protection can be achieved by giving more weight on the recall than the fall-out (using for instance the J_4 score or the J_5 score). But a better protection is obtained at the cost of the performance. Therefore, choosing s and c that maximize a J_β score with a higher β will increase the recall but also the fall-out. In addition, Figure 42 indicates that for s and c varying between 1 and 4, the best recall achieved by SAM is 91.0% (for $s = 4$ and $c = 4$). As a result, the SAM cannot identify all semantically sensitive queries.

5.6.2 Evaluation of the Linkability Assessment Module

LAM relies on two parameters: the smoothing factor α (used in the computation of the similarity metric between the query and the user profile of the user) and the threshold λ (to decide whether a query is linkable). Figure 43 depicts the recall and the fall-out for different number of pairs (α,λ) considering 100 users in the system. We used as ground truth the machine learning attack (as SimAttack and LAM identify linkable queries with the same similarity metric that could lead to a bias). We note that the recall and the fall-out

[3]Depending on the trade-off between privacy and performance desired by users, the importance of the recall on the fall-out can be adjusted. The J_3 score gives three times more importance to the recall than the fall-out.

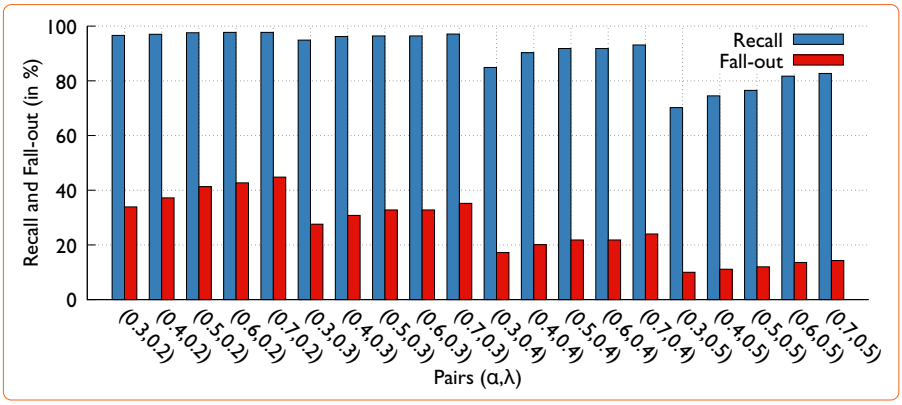


Figure 43

The recall and the fall-out obtained with LAM according to the smoothing factor α and the threshold λ (considering 100 users in the system).

vary respectively between 70.2% and 97.7%, and between 10.0% and 44.8%. Similarly to SAM, it is not easy to determine which pair (α,λ) gives the best trade-off between recall and fall-out. Therefore, we study the J_3 score to determine the best parameters α and λ. Table 11 presents the J_3 score for several values of α and λ. We can see that multiple pairs imply a J_3 score close to its maximum value (92.6%). For instance, LAM obtains a J_3 score of 92.6%, 92.4%, 92.4%, and 92.5%, for λ equal 0.3 and α respectively equal 0.4, 0.5, 0.6, and 0.7. Consequently, for a value of α varying between 0.4 and 0.7 and a λ equal 0.3, the J_3 score is relatively stable. We note that the J_3 score is maximal for $\alpha = 0.4$ and $\lambda = 0.3$. It corresponds to a recall of 96.2% and a fall-out of 30.8%. These two metrics indicate that (i) linkable queries are correctly identified by LAM and (ii) the fraction of non-linkable queries misclassified by LAM is rather small. We compute the J_3 score for datasets that contain different numbers of users (i.e., 200, 300, 500 and 1,000 users). We find that $\alpha = 0.4$ and $\lambda = 0.3$ is the pair of values that also maximizes the J_3 score on these datasets. For that reason, we use these two parameters in our experiments.

We purse the evaluation of LAM by studying the recall and the fall-out for different numbers of users (from 100 to 1,000 users), as the linkability of a query depends on the number of users in the system. If we consider more user profiles, it is harder for an adversary to retrieve the right user profile. Figure 44 shows the performance of LAM for different numbers of users. The results show that the recall and the fall-out increase with the number of users in the system. This can be explained by the fact that increasing the number of users in the system decreases the number of linkable queries. For 100 users, LAM correctly identifies 94.9% of linkable queries while for 1,000 users, the recall

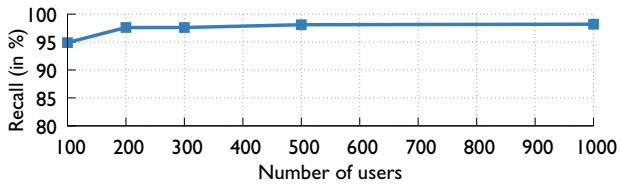
Table 11

Impact of the thresholds λ and α on the J_3 score (in %) obtained with LAM (considering 100 users in the system).

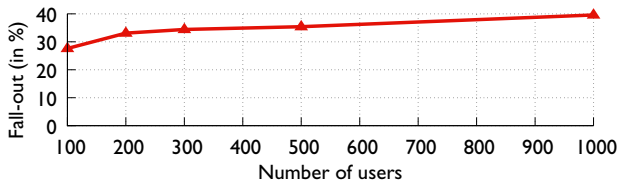
		Threshold λ				
		0.2	0.3	0.4	0.5	0.6
Smoothing Factor α	0.2	88.5	84.8	74.6	63.7	54.5
	0.3	92.3	92.0	84.7	71.8	61.8
	0.4	92.0	92.6	89.1	75.7	67.6
	0.5	91.5	92.4	90.2	77.5	71.4
	0.6	91.2	92.4	90.2	82.1	73.3
	0.7	90.7	92.5	91.0	83.0	77.8
	0.8	90.6	92.1	91.3	83.9	78.6

Figure 44

Impact of the number of users on the recall and the fall-out obtained using the Linkability Assessment Module.



(a) *Recall.*



(b) *Fall-out.*

increases up to 98.2%. Consequently, LAM identifies very well the linkable queries. Nevertheless, the recall needs to be appreciated with the fall-out. We note that the fall-out is relatively high. For 100 users, LAM achieves a fall-out of 30.8% that increases to 39.6% considering 1,000 users. In practice, this means that LAM misclassifies a significant proportion of non-linkable queries (more than one third). Therefore, the gain in terms of performance is not optimal, but at least the user protection is well preserved.

5.6.3 Impact of Using SAM and LAM on Existing Private Web Search Solutions

In this section, we evaluate the deployment of SAM and LAM on three existing solutions: Tor, GooPIR and PEAS. Our evaluation assesses two criteria: the privacy protection and the efficiency of the adaptation.

5.6.3.1 Tor

The adaptation of Tor with SAM detects semantically sensitive queries to only protecting these queries. Therefore, semantically non-sensitive queries are no longer protected. We depict in Table 12 the impact on the privacy protection of adapting Tor with SAM. The results show that the adaptation slightly decreases the user protection, as the adversary is able to de-anonymize 8.7% more queries. But, if we compute the relative loss, it corresponds to a loss of 28.5%. This increase corresponds to semantically sensitive queries that are no longer protected by Tor due to SAM (i.e., sensitive queries identify as semantically non-sensitive by SAM). Furthermore, if ML Attack correctly de-anonymizes a larger proportion of queries, it necessarily misclassifies less queries. Therefore, adapting PEAS with SAM increases the precision. As shown by Table 12, the precision increases by 10% (or 30.3%, if we focus on the relative loss). Moreover, adapting Tor improves its performance: Instead

Table 12

Comparison between the privacy protection offered by Tor and Tor+SAM.

	Recall	Precision
Tor	30.5%	33.0%
Tor+SAM	39.2%	43.0%
Absolute loss	8.7%	10.0%
Relative loss	28.5%	30.3%

	Number of fake queries k				
	1	3	5	7	Average
GooPIR	72.1%	62.5%	59.0%	56.1%	62.4%
GooPIR+SAM	77.9%	69.2%	65.9%	63.1%	69.0%
Absolute loss	5.7%	6.7%	7.0%	7.1%	6.6%
Relative loss	8.0%	10.7%	11.7%	12.5%	10.6%

Table 13
Comparison between the protection offered by GooPIR and GooPIR+SAM.

of protecting 10,000 queries, TOR+SAM only protects 5,971 queries. Its performance is thus improved by 40.3%. Such a gain has a positive impact on users and relays, as the latter performs less cryptographic operations, and exchanges less messages.

5.6.3.2 GooPIR

The adaptation of GooPIR is similar to Tor. If SAM identifies a query as semantically sensitive, GooPIR obfuscates this query with fake queries; otherwise, the query is no longer protected. We measure the impact of adapting GooPIR with SAM. As shown in Table 13, the adapted version of GooPIR decreases the user protection by 6.6% on average. This means that the adversary succeeds in retrieving more semantically sensitive queries with the adapted version: the recall of the attack increases from 62.4% for GooPIR to 69.0% for GooPIR+SAM. Nevertheless, this loss is relatively low compared to the recall obtained with the attack. On average, the adversary retrieves 62.4% of user queries and thus an increase of 6.6% does not change the order of magnitude for the recall (there is a relative loss of 10.6% on average). Besides, we note that the loss increases with the number of fake queries. For one fake query, there is a loss of 5.7% that increases to 7.1% for seven fake queries. Indeed, semantically sensitive queries identified as non-sensitive (and thus non-protected by GooPIR+SAM) do not necessarily modify the recall of the attack, as they would have been retrieved by the attack anyway. But, increasing the number of fake queries, gives a better protection to queries, and therefore less semantically sensitive queries are retrieved by the attack.

Regarding the impact in terms of performance, the use of SAM significantly increases the performance of GooPIR. GooPIR+SAM protects 5,971 queries (instead of 10,000 queries originally). Therefore, the number of fake queries generated by GooPIR+SAM decreases by 40.3%. As a result, the adaptation of GooPIR produces less computation on the user-side. It also decreases the number of queries processed by the search engine.

5.6.3.3 PEAS

The adaptation of PEAS uses SAM and LAM to decrease the number of protected queries. We conduct experiments to evaluate the impact that the adaptation has on PEAS. Table 14 compares the privacy protection obtained with PEAS and its adaptation with SAM and LAM. We observe that, the adaptation of PEAS makes the adversary able to retrieve more semantically sensitive queries (on average, there is an absolute loss of 11.4%). In comparison with the percentage of queries originally retrieved by the adversary (on average 11.9%), obtaining 11.4% more queries doubles the number of queries retrieved by the adversary. In addition, we note that the privacy loss increases with the number of fake queries: for one fake query, there is an absolute loss of 10.7% while, for seven fake queries, the absolute loss increases to 11.8%. As already

Table 14
Comparison between the protection offered by PEAS and PEAS+SAM+LAM.

		Number of fake queries k				
		1	3	5	7	Average
Recall	PEAS[4]	18.9%	12.1%	9.2%	7.6%	11.9%
	PEAS+SAM+LAM	29.6%	23.5%	20.9%	19.4%	23.5%
	Absolute loss	10.7%	11.4%	11.7%	11.8%	11.6%
	Relative loss	56.6%	94.2%	127,2%	155.3%	97.5%
Precision	PEAS[4]	17.4%	11.1%	7.8%	8.0%	11.1%
	PEAS+SAM+LAM	29.1%	23.6%	21.2%	20.3%	23.5%
	Absolute loss	11.7%	12.5%	13.4%	12.3%	12.4%
	Relative loss	67.2%	112.6%	171.8%	153.8%	111.7%

[4]We note that these results for PEAS are lower than the ones obtained in Figure 26. The difference is due to different datasets. On Chapter 4, we consider the mean over three datasets; each of them contains 100 users. These 300 users represent the 300 most active users in the AOL dataset. On this chapter, we consider a dataset of 198 manually selected users that issued at least one sensitive query. Therefore, as explained in Chapter 3, considering more users and considering less active users decrease the efficiency of the attack.

explained in the previous section, the proportion of semantically sensitive queries retrieved by the attack decreases with the number of fake queries and therefore, this lead to a rise of the loss. Furthermore, we observe that the adaptation of PEAS increases the precision of the attack. On average, the precision is 12.4% higher for PEAS+SAM+LAM than PEAS (or 111.7% higher considering the relative loss). Overall, we deduce that the adaptation of PEAS with SAM and LAM noticeably impacts the user protection.

Nevertheless, in terms of performance, the adaptation has a positive impact. Instead of sending 10,000 queries through the privacy proxy, the adaptation of PEAS protections only 5,971 queries (gain of 40.3%). In addition, instead of obfuscating 10,000 queries, PEAS+SAM+LAM generates fake queries for only 3,348 queries (gain of 66.5%). As a result, the adaptation of PEAS dramatically decreases the cost of protecting queries: the users and the relays execute less cryptographic operations, the users generate less fake queries and therefore, the search engine processes less queries.

5.7 Discussion

This section discusses several aspects regarding the use of SAM and LAM with private Web search solutions. Firstly, we analyze a potential issue that could lead to a re-identification of users. Then, we discuss the advantages of an automatic detection of sensitive queries (compared to a manual identification). Lastly, we propose an improvement of LAM for indistinguishability solutions that generate fake queries.

5.7.1 On the Identification of Users' Identity

One may advocate that to preserve the user anonymity, unlinkability mechanisms have to be used in all user communications. Indeed, it might be argued that switching an unlinkability mechanism on and off (due to the adaptation of the protection) could lead to an identification of the user. For instance, the adversary could track users with fingerprinting methods (e.g., cookies, plug-ins installed). Nevertheless, as stated in Section 1.3, we consider that our users

remove all quasi-identifiers[5]. If there is no identifier that allows an adversary to link anonymous queries to non-anonymous queries, the previous attack is not feasible. Therefore, under our considerations, switching on and off the unlinkability mechanism does not put users in danger.

Furthermore, removing all quasi-identifiers prevents an adversary from grouping queries issued by an identical user. As shown by the AOL scandal, being able to group queries issued by a single user tends to compromise her privacy. Indeed, exploiting personal information spreads in her queries might give enough details to retrieve her identify. As a result, the adaptation of the unlinkability solution does not compromise the unlinkability property.

5.7.2 On the Advantages of the Assessment Modules

As we have seen in the evaluation section, assessment modules do not correctly classify sensitive queries. Indeed, such a classification is intuitive for humans (at least for their own queries) but extremely complex for a computer. For that reason, one might argue that users should classify queries themselves. For instance, we can imagine that users indicate their sensitive queries through a dedicated button. But a manual identification presents huge drawbacks: (i) users will sometimes choose the wrong classification by accident (sensitive instead of non-sensitive, and vice versa), and (ii) identifying sensitive queries is a restrictive task, so much that users might get tired of doing it, and therefore always indicate that their queries are sensitive (as such, users keep a high protection with low interaction). For these two reasons, a manual identification of sensitive queries might not necessarily lead to a better judgment than automatic modules.

5.7.3 On the Adaptation of the Number of Fake Queries

As presented in Section 5.3.4, LAM computes the similarity between the query and its requester user profile and then decide if the query is linkable according to a threshold λ. Nevertheless, this binary decision is not the optimal decision to adapt private Web search solutions, especially the ones that generate fake queries. The current adaptation protects queries by generating either zero or k fake queries. This specific type of solution could profit from a more advanced adaptation; the number of fake queries could evolve according to the similarity metric. Such an adaptation would give a better trade-off between efficiency and privacy protection.

[5] This thesis does not address countermeasures against fingerprinting methods. Readers who may be interested in such a question are referred to the literature (e.g., [36]).

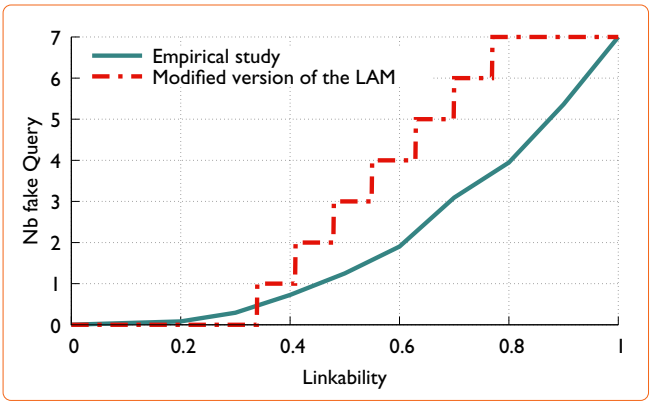


Figure 45
Mapping of the similarity metric between the query and its requester user profile to a number of fake queries.

To have a better understanding of the adaptation, we assess, for all queries in our dataset, the number of fake queries PEAS needs to generate to resist against ML Attack, and their corresponding similarity value. In Figure 45, we present the average number of fake queries generated by PEAS according to the similarity metric between the query and its requester user profile. We bounded the number of fake queries to seven. The results confirm that the binary option is not optimal. For instance, queries with a similarity metric over 0.3 are protected with seven fake queries while, Figure 45 shows that the number of fake queries can evolve linearly to seven fake queries. Consequently, instead of a binary decision, LAM could return the number of fake queries that PEAS should generate. To do so, we introduce in LAM a mapping between the similarity metric and the number of fake queries. We base the mapping on the previous empirical results obtained with PEAS. It follows the function f:

$$f(S) = \begin{cases} k_{min} & \text{if } S < 0.3 \\ k_{max} & \text{if } S > 0.8 \\ \text{Round}\left(\frac{k_{min}(S-S_{max})+k_{max}(S_{min}-S)}{S_{min}-S_{max}}\right) & \text{otherwise} \end{cases}$$

We display in Figure 45 the mapping function f. Between 0 and 0.3, we design LAM to generate k_{min} fake queries. Then, between 0.3 and 0.8, LAM computes the number of fake queries with a linear function (evolving from k_{min} to k_{max}) rounded to the nearest integer. Finally, between 0.8 and 1, LAM returns k_{max} fake queries. This last range of values does not follow the empirical results but we assume that above a similarity of 0.8, queries are likely to be retrieved by an attack and thus, we protect them with k_{max} fake queries. Furthermore, we set up by default k_{min} and k_{max} to zero and seven respectively, as these numbers give a good trade-off between privacy and performance. Nevertheless, users are able to modify them according to their requirements (i.e., focusing more on performance or privacy).

We evaluate the adaptation of PEAS with the new version of LAM. Table 15 shows the results in terms of privacy protection. We note that the adaptation of PEAS increases the recall by 15.1%. Compared to the previous version of LAM (see Table 14), the adversary is able to retrieve 3.3% more queries (22.7% instead of 19.4%). Nevertheless, the performance of PEAS increases, as the number of fake queries generated by PEAS is divided by 2.1 (i.e., 23,436 fake queries are generated with the first LAM version, while with the new version, this number drops to 11,075 fake queries). Therefore, the first version of LAM increases the performance of PEAS by 66.5%, but the second version increases its performance by 84.2%. As a result, for a small loss in privacy protection (3.3%), the new LAM version increases dramatically the performance (17.7%).

Table 15

Impact on the robustness of PEAS considering the new version of LAM that adapts the number of fake queries.

	Recall	Precision
PEAS	7.6%	8.0%
PEAS+SAM+LAM	22.7%	23.0%
Absolute loss	15.1%	15.0%
Relative loss	198.7%	187.5%

5.8 Conclusion

We have seen in this chapter that existing private Web search solutions over-protect a majority of queries, as most of them are either semantically non-sensitive or non-linkable. For that reason, we proposed two modules to identify these two types of sensitivity: the Semantic Assessment Module and the Linkability Assessment Module. We showed that these modules correctly identify sensitive queries even though the identification of semantically sensitive queries is less accurate than the identification of linkable queries (SAM obtains a recall of 87.9%, while LAM obtains a recall of 96.2%, for 100 users). We aimed with these modules to adapt the protection of existing private Web search solutions. We presented the adaptation of three representative private Web search solutions (i.e., Tor, GooPIR, and PEAS) and evaluated their impact in terms of privacy protection and performance. We showed that adapting private Web search solutions practically halved the cost of protecting queries while the user protection is, in the worst case, divided by two.

These assessments modules dramatically improve the performance of the current private Web search solutions. Nevertheless, they also degrade the user privacy protection. A first reason is that our experiments were performed on 198 users; a higher number of users (like in practice) would have less impact on the privacy protection, as privacy attacks retrieve less queries for a higher number of users. In addition, the decrease in the privacy protection is also due to SAM. This module does not identify all semantically sensitive queries. Improving SAM is not easy because of the constraints on the running environment (i.e., no distant services, and limited resources). But, as the current implementation is relatively simple, there is room for improvement. For instance, the set of categories on which SAM bases its decision contains a lot of noise (i.e., many retrieved categories are not relevant for the query). Consequently, a future work should focus on improving the relevance of these categories. In addition, misspelled words in queries prevent SAM from correctly identifying categories. One could introduce another pre-processing step to solve those misspelled words. Finally, queries often contain slang words or specific words that are not covered by WordNet. We use a dictionary of medical terms and pornographic terms to identities such words. Nevertheless, this approach does not cover all possible words and a more advanced techniques should be considered.

CHAPTER 6

Conclusion and Perspectives

6.1 Conclusion

Controlling her personal data is becoming increasingly important in today's world. The technological progress has made the processing of data easily accessible for all online providers. These providers collect and analyze data from billions of users. They mainly use it for targeted advertising, but one can imagine worst scenarios like selling sensitive data to insurance companies. This issue with Web providers concerns in particular search engines that receive billions of search queries every day. The processing of this data reveals many personal information about their users. Several solutions have been published in the literature allowing users to control the usage of their personal data. In this thesis, we studied such a type of solutions. More precisely, we focused on solutions that do not require any modifications on existing search engines. Therefore, we excluded from the scope of this thesis private information retrieval (PIR) protocols, as they require search engines from implementing specific recommendation algorithms based on homomorphic encryption. Furthermore, our study considered an adversary interested in collecting user queries. We assumed that this adversary was able to previously collect non-protected user queries, i.e., before the users employ a privacy-protection mechanism.

6.1.1 Robustness of Existing Private Web Search Solutions

In the first part of the thesis, we aimed to assess the robustness of the existing private Web search solutions. For this purpose, we developed SimAttack, an attack that breaks the protection offered by all types of private Web search solutions: unlinkability solutions, indistinguishability solutions, and indistinguishability solutions run on top of unlinkability solutions. SimAttack relies on a similarity metric between a query and a user profile (i.e., set of queries previously collected by the adversary). To find the identity of the requester that issued an anonymous query (case of an unlinkability solution), it identifies among all available user profiles the most likely one, while for queries protected with fake queries (case of an indistinguishability solution), it retrieves the initial query by finding the query that is the most related to the user profile. We showed that SimAttack outperforms similar attacks pub-

lished in the literature. It succeeds in breaking a larger proportion of queries in a much lower amount of time. For instance, considering unlinkability solutions with 1,000 users, the recall of SimAttack is 4.3% higher than the one obtained with the machine learning attack (competitor from the literature), while SimAttack divides the execution time by 158. In addition, SimAttack targets protection mechanisms that had never been attacked so far (e.g., GooPIR).

Furthermore, we conducted an analysis of the robustness of state-of-the-art solutions using SimAttack. We established that none of the existing private Web search solutions are satisfactory to protect users. We empirically proved that for each tested solution (i.e., unlinkability solutions, TrackMeNot, and GooPIR) an adversary that has previously-collected queries is able to break these protections and retrieve a large proportion of user queries. For instance, considering GooPIR, an adversary retrieved with SimAttack 50.6% of queries (case of seven fake queries). Moreover, we investigated the combination of two types of private Web search solutions and thus, analyzed the protection offered by TrackMeNot over an unlinkability solution and GooPIR over an unlinkability solution. We confirmed an intuitive thought that combining two independent solutions gives a better protection to users. For instance, by using GooPIR over an unlinkability solution, the recall dropped from 50.6% to 27.6%. As we can notice, SimAttack was still able to retrieve a non-negligible number of protected queries. We can therefore conclude that none of these solutions protects users in a satisfactory manner.

6.1.2 A New Efficient Private Web Search Solution

As no solution successfully protects users, we designed PEAS, a new private Web search solution to overcome this problem. The novelty of our approach consists in combining an indistinguishability solution on top of an unlinkability protocol. This enables a better protection by integrating the two types of solution together. The indistinguishability part relies on the generation of fake queries, while the unlinkability part is composed of two nodes: a receiver and an issuer. To mislead the adversary about the identity of the requester, the indistinguishability part aims at generating, in a privacy-preserving way, fake queries already issued by other users in the system. For the unlinkability solutions, our new protocol aims at being efficient as well as robust.

We empirically assess PEAS using a dataset of real queries released by AOL in 2006. The privacy evaluation of PEAS was conducted using SimAttack (i.e., the first contribution of this thesis) and a machine learning attack. We showed that PEAS outperforms all existing solutions in terms of privacy; for instance, it reduces the number of de-anonymized queries by 26.6% (compared to an onion routing baseline) and by 18.4% (compared to GooPIR over an unlinkability solution). Moreover, we established that the indistinguishability part of PEAS does not strongly impact the quality of the results. Due to an efficient filtering mechanism, PEAS removes most of the irrelevant results (i.e., results related to the fake queries). On average, only 4.7% of the results displayed by PEAS are irrelevant. Finally, regarding the performance of PEAS, we determined that PEAS is more efficient than other protections mechanisms: Compared to onion routing, it has a maximum throughput 3.1 times higher. Overall, PEAS achieves an excellent user protection (i.e., only few queries are still sensitive to the privacy attacks) without significantly degrading the user experience (i.e., the quality of the results and the latency are not strongly impacted).

6.1.3 Adapting the Protection to the Query Sensitivity

Existing private Web search solutions do not adapt their protection mechanism according to the query. All queries are protected in a similar way: Queries related to a sensitive topic receive the same protection than non-sensitive queries. Therefore many resources are wasted to protect queries that should not be protected. Due to this reason, we investigated the possibility to adapt all private Web search solutions according to the query sensitivity. By sensitive queries, we identify two types of sensitivity: queries related to an embarrassing topic (e.g., sex, diseases) and linkable queries (i.e., queries that suffer from privacy attacks). Consequently, we design two modules to identify these two types of sensitivity: SAM and LAM. SAM uses WordNet to categorize queries and determines if they belong to a sensitive category, while LAM exploits previous queries sent by the user to decide if the query is linkable (i.e., close to the previous queries). As we consider the privacy protection as more important than the performance, we configured these modules to better identify sensitive queries than non-sensitive queries (i.e., some queries could be overprotected). We showed that these modules correctly identify sensitive queries even though the identification of semantically sensitive queries is less accurate (SAM obtains a recall of 87.9%) than the identification of linkable queries (LAM obtains a recall of 96.2%, for 100 users).

We deployed the two modules on existing private Web search solutions. Their deployment does not require a great effort. For most private Web search solutions, it introduces a preliminary step to assess the sensitivity of the query and, according to the results, either protect the query or send it directly to the search engine. The empirical evaluation of their deployment exhibits the huge gain in terms of performance. In particular PEAS, which combines an indistinguishability mechanism on top of an unlinkability protocol, dramatically improve its performance (up to 66.5%). Nevertheless, the adaptation mechanism decreases the user protection, as some sensitive queries are not correctly detected by SAM and LAM. Therefore, as such queries are no longer protected, the user protection decreases. On average, the recall obtained with ML Attack against PEAS increases from 11.9% to 23.5%. Consequently, SAM and LAM are aimed mainly at people that agree to trade part of their privacy for better performance. Privacy-protection mechanisms increase the time waited by users to obtain their results, therefore with these modules this time is closer to the one obtained when no protection is used, i.e. almost instantaneously.

6.2 Perspectives

We present several perspectives for future work. First, we discuss the settings of this thesis (our evaluation and our adversary model). Then, we give some perspectives regarding the three contributions presented in this thesis.

6.2.1 Dataset Employed and Adversary Model

The evaluations conducted in this thesis used the AOL dataset. This dataset was created in 2006. Nevertheless, the Web is constantly changing: new technologies (e.g., HTML5, CSS3, jQuery), new services (e.g., social networks), and new content. These evolutions impact users' behavior as they have to continuously adapt themselves to these changes. Consequently, there will definitely be a difference between a dataset of queries from 2006 and one from 2017. Therefore, we might wonder about the influence of the time on

the user protection. Moreover, the dataset published by AOL does not reflect the large quantity of data collected by current search engines. First, it only contains three months of search queries while search engines have a larger query history. Next, the AOL dataset contains few active users (users that issued a significant number of queries). Lastly, millions of users are querying search engines every day while we could only extract from the AOL dataset a maximum of 15,000 active users. Due to the lack of other available data, we used the AOL dataset for our evaluation. Similar solutions in the literature have done the same (e.g., in [47], 60 AOL users were used in the evaluation, while [49] uses 100 AOL users). Search engines have larger datasets (used for internal purposes) but they do not publish them. Further experiments should be conducted with the help of the search engines to benefit from these larger datasets.

We consider in this work an adversary that previously-collected queries about all users employing a privacy-preserving mechanism. Therefore, we assume in our attacks that the adversary has prior queries for each user attacked. Nevertheless, for the unlinkability solutions, the adversary has no information about the current requesters and thus, it might be hard for an adversary to use the correct user profiles in the attack. This is even harder if we consider a more realistic scenario where search engines receive queries from millions of users every day. Such a hypothesis supposes that the adversary is able to identify among this large number of users, the few that are using the unlinkability solution. A future work should focus on the feasibility of the hypothesis. For instance, it might be possible to retrieve these users by identifying the ones that stop querying the search engines. Besides, it is likely that these users might have issued queries related to privacy protection.

Furthermore, we consider an adversary who focuses her effort on the exploitation of previously collected queries. Nevertheless, such attacks do not consider specific flaws in the protection mechanisms. For instance, regarding indistinguishability solutions, there is a question of distinguishing fake queries from real ones. SimAttack solves this problem from a semantical point of view: comparing keywords from fake queries and real ones. Nevertheless, we can imagine an attack that exploits other criteria: keyword order, co-occurrence between keywords. Identifying precisely these criteria and exploiting them might enhance SimAttack.

6.2.2 Privacy Attacks

We have seen in this thesis that privacy attacks suffer from a high execution time. This is mainly due to the number of features exploited by the attacks. Queries are represented in a vector space model and thus a large number of keywords implies a large number of features. This can be overcome by a dimensional space reduction (e.g., using Singular Value Decomposition [153]). Consequently, in terms of execution time, introducing a preliminary step to decrease the number of features will speed up the attacks. But the dimensional reduction will also affect the results of the attack. From the dimensional reduction, we can expect a better identification of users, as the singular values often removes noise and linearly dependent elements [173, 174].

Similarly, one could investigate the use of word embeddings to represent keywords [175]. Most word embedding techniques rely on a neural network architecture (e.g., Word2Vec [152]). Word embeddings map in a continuous vector space semantically similar words to nearby points. Therefore, such representation contains more information than a basic dictionary of words used in the current representation of queries in SimAttack. Due to this reason, rep-

resenting queries with word embeddings would likely improve the efficiency of SimAttack.

Another limitation with privacy attacks is that they only exploit keywords. They do not consider external information (e.g., the semantic relation between keywords) or exploit timing information (e.g., workers do not use their home computer during working hours). A more advanced feature selection could lead to better distinction between users. In addition, queries are often issued in search sessions. They correspond to a short period of time where users refine their queries to obtain a more relevant answer to their problems. Existing works (e.g., [176, 177]) have shown that it is possible to identify search sessions as these queries share common keywords. Nevertheless, current attacks do not exploit such a technique and consider all queries independently. By identifying search sessions, current attacks could increase their performance: (i) a necessary condition for these queries is that they have been issued by the same user, and (ii) identifying such search sessions will reveal more information about the initial requester than processing each query independently.

Last but not least, current attacks only exploit a query history of users. As we have seen in Chapter 3, SimAttack was not able to identify some fake queries or some requesters because the keywords of the targeted queries were never used in the query history. Viejo et al. [75, 76] designed a privacy protection based on user profiles extracted from social networks. Therefore, we can imagine that SimAttack could similarly take advantage of social networks. The data public available on social networks could enrich the query history collected on users. For instance, the terms used in tweets or Facebook status could be taken into consideration in the computation of the similarity metric. A more advanced metric could improve the efficiency of SimAttack.

6.2.3 Private Web Search Solutions

The unlinkability protocol suggested in PEAS requires the nodes to be honest. This means that they cannot deviate from the protocol. It has been shown, through the Tor project, that in practice it is not the case [178, 179]. For instance, some exit nodes modify the search engine answer returned to the users. To overcome this issue, the unlinkability protocol should be enhanced with accountability techniques (e.g., [180, 181]) such that nodes could record all transactions they send and receive to periodically audit each other and verify that the transactions made by the nodes were correct. In particular, during the audition, they can check the integrity of the search engine answer previously processed by the audited node. To do so, they re-issue the user query and compare the two versions. If there is a difference, they have to check if it is due to the search engine (its answer might not be deterministic) or to the audited node (if it introduces or modifies intentionally the answer). Therefore, misbehaving nodes could be detected and removed from the system.

Furthermore, in this thesis, we established that the combination of an unlinkability solution with an indistinguishability solution improves the user protection. Nevertheless, the weakness of such a protection is the fake queries. If they are not realistic, users are not well protected. But, generating realistic fake data is a difficult task. In this thesis, we suggested that fake queries should be generated from previous user queries. But, as we have seen in the evaluation section of PEAS, the clique extraction step is computationally expensive. Therefore, works should be conducted to improve the generation of fake queries at a lower cost. As explained in Chapter 4, we want to generate fake queries that (i) are plausible and that (ii) refer to real users. Social

networks (e.g., Twitter [182]) can be an alternative source to generate fake queries. They contain real data produced by real users. Therefore, as the data is publicly available, it is easily extractable. Such a source would decrease the cost, as it avoids the extraction of the maximal cliques. Nevertheless, as messages on social networks differ from user queries, further experiments should investigate if such a data can replace the PEAS group profile.

6.2.4 Adaptation of Existing Solutions

The identification of semantically sensitive queries is a difficult task as many factors influence the meaning of a query, in particular its context. Nevertheless, the results obtained with SAM are relatively good, especially if we compare them to the recall achieved by query categorizers (e.g., the winner of the KDD competition [135] has a recall of 47.9%). But, in terms of security, SAM divides by two the user protection in the worst case. A future work could improve the detection of sensitive queries to reinforce the privacy protection. Instead of relying on WordNet [82] and XWND [170], one could use machine learning techniques to automatically learn what the users consider as sensitive. Therefore, such algorithms do not require users to specify categories they found sensitive, but instead they have to annotate queries. Then, by training a binary classifier on these annotated queries, we obtain a classifier that distinguishes sensitive queries from non-sensitive ones. As annotating queries is less subjective than choosing categories, machine learning could improve the detection of sensitive queries.

Regarding the notion of sensitivity, we consider in this thesis semantically sensitive queries and linkable queries. Nevertheless, it is also possible to consider a third type of queries: A query itself might not contain sensitive information, but linked with other queries, it might disclose new information about the user. For instance, it has been shown in [22] that the age of a user or her gender can be inferred from her queries even though they do not contain such information explicitly. Consequently, a future work should focus on inferring straight-forward information (e.g., age, gender, location, wealth, health, politics) to validate the feasibility of such an attack. Besides, detecting such a piece of information could help in the user protection. For instance, fake queries should be generated to fake the values retrieved by the inference attack (e.g., modify the age). A naive solution is to design fake queries with contrary information, e.g., a fake query about a very expensive product for a young man and another fake query about an unexpressive product for an old lady. The inference attack could be used to monitor the noise introduced with the fake queries and adapt it according to the user requirements.

APPENDIX A

Stopwords

We display below the stopwords we removed from queries used in our evaluation.

's	below	ho	others	underneath
I	beneath	how	otherwise	unless
a	beside	however	our	unlike
aboard	besides	i	ours	until
about	between	if	ourselves	unto
above	bewteen	in	out	up
across	beyond	inside	outside	upon
after	bi	insofar	outta	uppon
afterwards	both	instead	over	us
against	but	into	per	via
agin	by	it	rather	vis-a-vis
ago	ca.	its	regardless	vis-‡-vis
agreed-upon	can	itself	round	we
ah	de	la	se	well
alas	des	le	she	what
albeit	despite	les	should	whatever
all	do	lest	since	whatsoever
all-over	down	lieu	so	when
almost	due	like	some	whenever
along	durin	me	someone	where
alongside	during	minus	something	whereas
altho	each	moreover	than	wherefore
although	eh	my	that	whereupon
amid	either	myself	the	whether
amidst	en	near	their	which
among	every	near-by	them	whichever
amongst	ever	nearer	themselves	while
an	everyone	nearest	then	who
and	everything	neither	there	whoever
another	except	nevertheless	therefore	whom
any	far	next	these	whose

anyone	fer	no	they	why
anything	for	nor	thine	with
around	from	not	this	withal
as	go	nothing	those	within
aside	goddamn	notwithstanding	thou	without
astride	goody	o	though	ye
at	gosh	o'er	through	yea
atop	half	of	throughout	yeah
avec	have	off	thru	yes
away	he	on	till	yet
back	hell	once	to	yonder
be	her	one	together	you
because	herself	oneself	toward	your
before	hey	only	towardes	yours
beforehand	him	onto	towards	yourself
behind	himself	or	uh	yourselves
behynde	his	other	under	

APPENDIX B

Complementary Experiments on SimAttack

In this appendix, we present further experiments we performed with SimAttack. These experiments target unlinkability solutions, TMN, GooPIR, TMN over an unlinkability solution and GooPIR over an unlinkability solution. We first present an analysis about how the number of previously-collected queries impacts the results of SimAttack. Then, we depict specific experiments we run for unlinkability solutions and GooPIR over an unlinkability solution.

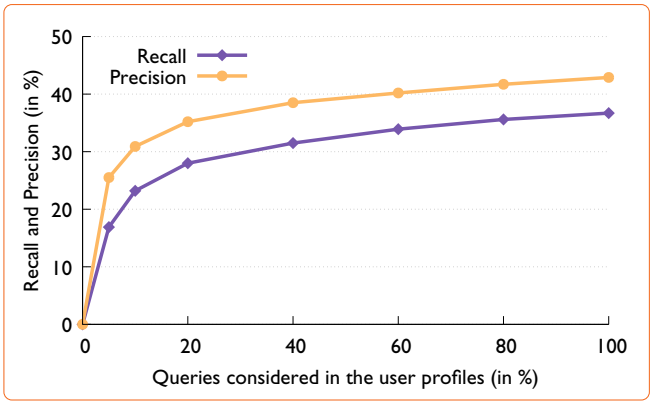
B.1 Impact of the Size of the User Profiles

As current search engines collect all user queries, we assume in this thesis an adversary that has been collecting user queries, in particular to de-anonymize queries or retrieve (among fake queries) the initial one. In this section, we evaluate how the number of previously-collected queries influences the results of SimAttack. We could use different datasets that contain users who issued different quantities of queries. Nevertheless, comparing datasets with different users is not easy as many statistical properties (e.g., term frequency, topics) might influence the results. For that reason, we consider only one dataset `AOL100`, and vary the size of training set by considering different proportions of queries. We select successively the first 0%, 5%, 10%, 20%, 40%, 60%, 80%, 100% of queries contained in the training set.

B.1.1 Unlinkability Solutions

We measure the influence of the number of previously-collected queries on the results of SimAttack. Figure 46 depicts the precision and the recall of SimAttack considering the dataset `AOL100`. The results show that the efficiency of the attack decreases according to the proportion of queries considered in user profiles. Considering the full training set provides a recall of 36.7% while considering only 5% of queries drops this value to 16.9%. Similar results are obtained for the precision: a drop from 42.9% to 25.5%. These results illustrate that exploiting less accurate user profiles (i.e., preliminary

Figure 46
The efficiency of the attack decreases according to the proportion of queries considered in user profiles.

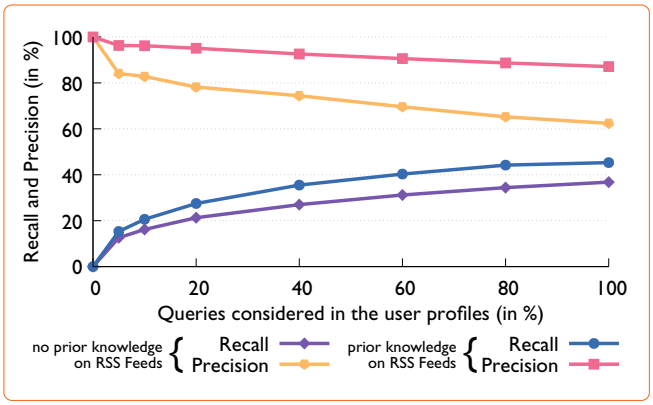


information about users) makes it harder for the adversary to de-anonymize queries. Nevertheless, we note that, when the proportion of queries in user profiles drops from 100% to 20%, the number of de-anonymized queries decreases by 8.7%. Therefore, removing 80% of queries does not significantly impact the results. Furthermore, we note that no query in the training set makes SimAttack unable to de-anonymize any queries. Indeed, SimAttack bases its re-identification on previously-collected queries. Therefore, without past queries, SimAttack cannot de-anonymize any queries.

B.1.2 TrackMeNot

We mesure the impact that the number of queries owned by the adversary has on the TMN protection. We depict in Figure 47 the precision and the recall of SimAttack for the dataset `TMN100` considering its training set varying from 0% to 100%. The results show that exploiting smaller user profiles makes it harder for SimAttack to identify user queries. For instance, if we consider the full user profiles, SimAttack identifies 36.8% or 45.3% of queries (with and without exploiting the RSS feeds, respectively) while this number drops to 12.6% or 15.3% if we consider only 5% of the query history in the user profiles. However, considering smaller user profile makes the precision increases: decreasing the training set from 100% to 5% increases the precision from 21.7% to 92%. Indeed, with less accurate user profiles, SimAttack does not have enough information to correctly identify user queries. Consequently, increasing the size of user profiles increases the recall of SimAttack, but also decreases the precision as more queries get misclassified.

Figure 47
Exploiting smaller user profiles makes it harder for SimAttack to identify user queries.



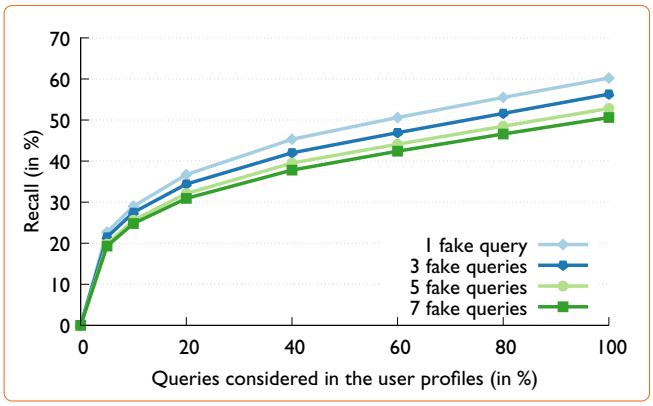


Figure 48

GooPIR ensures a better protection if the adversary has only a smaller and less accurate user profile.

B.1.3 GooPIR

To measure the influence of the number of previously-collected queries on the efficiency of SimAttack against GooPIR, we assess the performance of SimAttack for a varying number of queries in the training set (from 0% to 100%). Figure 48 depicts for AOL100 dataset, the percentage of identified queries according to the ratio of the query history considered in the user profile. The results show that GooPIR ensures a better protection if the adversary has only a smaller and less accurate user profile. For instance, when 7 fake queries are generated by GooPIR and only 5% of the query history is considered in the user profiles, SimAttack retrieves 19.3% of the initial query while this percentage increases to 50.6% if 100% of the query history is considered in the user profiles. Consequently, if this profile is less accurate, fewer user queries can be identified.

Furthermore, changing the number of fake queries generated by GooPIR impacts the percentage of identified queries, only if the adversary has collected a significant number of queries to create accurate user profiles. For instance, adding six fake queries decreases by 9.6% the percentage of query identified when the full query history is taken into account. This percentage drops to 3.4% when only 10% of the query history is considered.

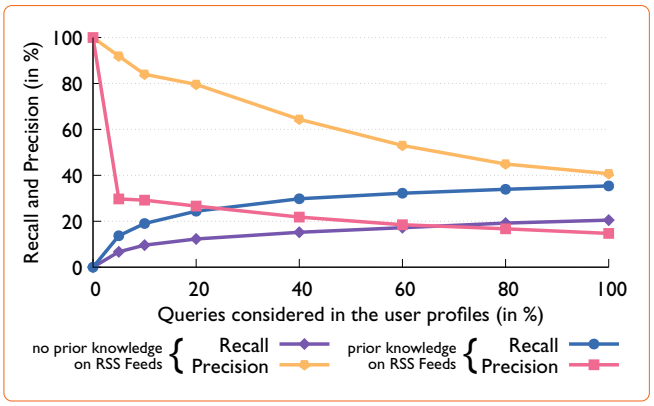
Lastly, as SimAttack bases its identification exclusively on the user profile, if there is no query in the user profiles, the recall obtains by SimAttack is 0%.

B.1.4 TMN Over an Unlinkability Solution

We evaluate the influence that the size of the user profiles owned by the adversary has on SimAttack when TrackMeNot is combined with an unlinkability solution. Figure 49 depicts the recall and the precision of SimAttack with and without using the RSS feeds. From the results, we deduce that smaller user profiles decrease the performance. For instance, decreasing the size of the user profiles from 100% to 5% makes SimAttack able to identify 13.8% and 21.7% fewer queries without and with prior knowledge on RSS feeds, respectively. In addition, the results show that the precision decreases according to the size of the user profile: considering from 100% to 5% of the user profiles, the precision loses 51.2% and 15.0%, respectively, for SimAttack with and without prior knowledge on RSS feeds of users. Furthermore, compared to the results obtained with TrackMeNot alone, the recall is slightly lower due to the unlinkability solution (e.g., for 5% of the user profiles, the recalls are respectively 6.7% and 12.6% considering no RSS feeds). A similar result is obtained for the precision when the RSS feeds are used (e.g., for 5% of the user profiles, the precisions are respectively 91.9% and 84.1%). But, surprisingly,

Figure 49

Exploiting smaller user profiles makes it harder for SimAttack to identify user queries.



the precision obtained when the RSS feeds are exploited is very different than TMN alone: for 5% of the user profiles, the precision drops receptively from 96.3% to 14.7%. As explained Section 3.4.1.4, this is due to a high number of fake queries misclassified.

B.1.5 GooPIR Over an Unlinkability Solution

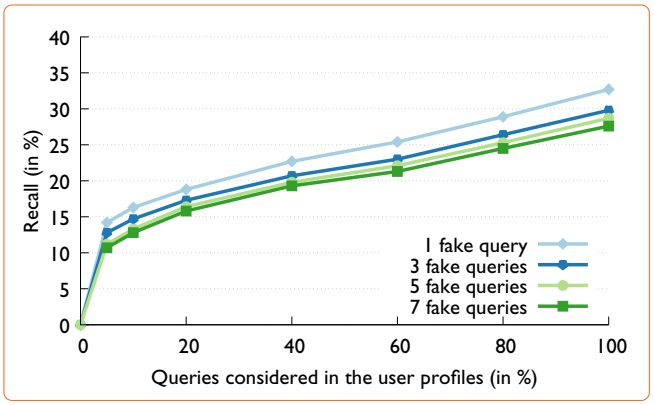
We finally evaluate the influence of the size of the user profiles on the performance of SimAttack when GooPIR is combined with an unlinkability solution. Figure 50 presents the recall of SimAttack for different proportions of queries in the user profile (from 0% to 100%). The results show that GooPIR over an unlinkability solution protects users' queries more strongly if the adversary has smaller user profiles: for one fake query generated by GooPIR, SimAttack identifies 16.1% of queries when 5% of the user profile is considered, while 32.7% of the queries are identified when 100% of the profile is considered. Therefore, the more information is owned by the adversary, the more queries she is able to retrieve.

B.2 Unlinkability Solutions

We introduce two further experiments on SimAttack considering an unlinkability solution. First, we study the influence of considering more user profiles than the users protected by the unlinkability solution. Then, we analyse the efficiency of SimAttack if, instead of considering the most probable user, SimAttack considers a query correctly de-anonymized if its correct requester is retrieved among the p most probable users.

Figure 50

Exploiting smaller user profiles make it harder for SimAttack to identify user queries.



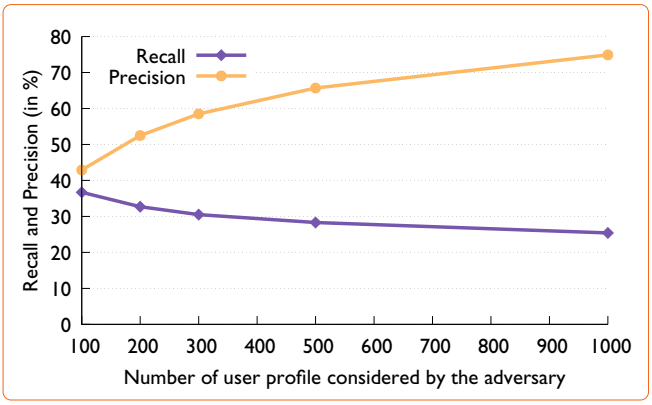


Figure 51

Increasing the number of user profiles, with respect to the number of users in the system, decreases the precision while slightly reducing the recall.

B.2.1 Impact of the Number of User Profiles

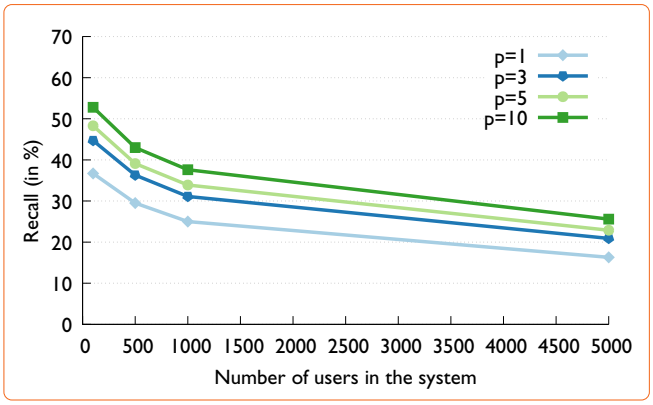
In previous evaluations, we considered that the adversary has pre-built as many user profiles as the number of users in the system. However, in practice, the adversary might consider more user profiles than the number of users in the system. Consequently, we present in Figure 51 the precision and the recall of SimAttack when the system gathers 100 users while the adversary considers from 100 to 1,000 user profiles. The results show that increasing the number of user profiles owned by the adversary decreases the recall (i.e., adding 900 extra user profiles decreases the recall by 11.3%). A higher number of user profiles increase for SimAttack the number of potential requesters. Therefore, a given query correctly de-anonymized for 100 users might not be for 1,000 user profiles, as SimAttack retrieves a more probable user. On the contrary, the precision significantly increases according to the number of user profiles considered by the adversary. For instance, introducing 900 extra user profiles increases the precision by 32.0%. Indeed, taking extra user profiles into account splits the misclassified queries between a larger number of user profiles. Consequently, for a given user profile, the number of queries incorrectly mapped by SimAttack decreases (i.e., queries issued by a different user than the one retrieved by SimAttack) and thus the precision computed for this profile increases. So, if all precisions related to user profiles increase, the precision of SimAttack also increases.

B.2.2 Impact of Targeting p Users With the Highest Similarity Instead of the Highest One

Depending on the intentions of the adversary, she might consider that the attack succeeds if the initial user is retrieved among the 3, 5 or 10 most probable users. Her goal might be to associate a query with her potential requesters. Consequently, we adapt SimAttack to link a query to the p most probable users. Figure 52 illustrates the results of this experiment for different numbers of users in the system. Obviously, the number of de-anonymized queries increases according to the number of users considered by the adversary. However, this increase is rather small: The adversary de-anonymizes on average 12.9% more queries if the 10 most probable users are targeted compared to targeting the most probable one (i.e., $p = 10$ versus $p = 1$). In addition, if the adversary considers the 10 most probable users while the systems gather 100 users, the recall only reaches 52.8%. This result is counter-intuitive, as the 10 most probable users represent 10% of the dataset, the recall should be close to 100% (as there is a high probability that an initial user is among the 10 most probable users). The explanation is that a large proportion of non-

Figure 52

The number of de-anonymized queries increases according to the number of probable users returned by SimAttack.



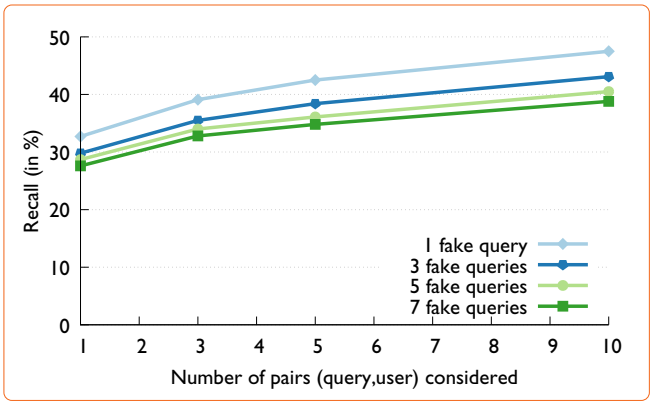
retrieved queries (74.8% if we consider 100 users) was not retrieved because they do not contain keywords already used by the user in her previous queries. Therefore, these queries cannot be retrieved by SimAttack as their similarity with the requester user profile equals zero.

B.3 GooPIR Over an Unlinkability Solution

SimAttack for GooPIR over an unlinkability solution presented in Section 3.2.4 retrieves for a protected query (i.e., an obfuscated query anonymized) the most probable pair $(query, user)$. For the same reason as explained for unlinkability solutions, an adversary might be interested in the p most probable pairs $(query, user)$. In Figure 53, we report the recall of SimAttack considering a successful attack if the correct pair $(query, user)$ is retrieved among the p most probable pairs. We vary p from one to ten. The results show that the recall increases according to the number of pairs considered by the adversary. For instance, considering ten pairs instead of one makes the recall increase on average by 13.4%. Nevertheless, this recall improvement remains relatively low. Considering more pairs $(query, user)$ makes the adversary retrieve irrelevant information (e.g., associated a fake query with a non-related user). For ten pairs, the adversary retrieves nine out of ten pairs that might not be relevant.

Figure 53

The recall slightly increases according to the number of pairs $(query, user)$ returned by SimAttack.



APPENDIX C

Proverif code of PEAS

We present the ProVerif code we wrote to model the privacy proxy used in PEAS. This code has been used with ProVerif to prove that an adversary cannot link a query sent through the privacy proxy with its requester. We consider an adversary that listens to the network and replays all messages. We do not consider in this model the receiver or the issuer as a potential adversary.

```
(*
Message 1: U -> R : U, { q, kU }pkI
Message 2: R -> I : X, { q, kU }pkI
Message 3: I -> S : (q)
Message 4: S -> I : (a)
Message 5: I -> R : X, { a }kU
Message 6: R -> U : U, { a }kU
*)

(* A public channel *)
free c: channel.

type IP.
type id.
type request.
type answer.
type key.
type pkey.
type skey.
type link.

(* The protagonists *)
free receiver: IP.
free issuer: IP.
free server: IP.

free q: request [private].
free user: IP [private].

fun LINK_a(IP,bitstring): link [private].
fun LINK_b(IP,id): link [private].
fun LINK_c(id,bitstring): link [private].
fun LINK_d(id,request): link [private].
fun LINK_e(IP,request): link [private].

reduc forall x: IP, y: bitstring, z: id;
   INFER(LINK_a(x,y),LINK_c(z,y)) = LINK_b(x,z).
reduc forall x: IP, y: id, z: request;
```

```
39    INFER1(LINK_b(x,y),LINK_d(y,z)) = LINK_e(x,z).
40
41  (* Symmetric key encryption *)
42  fun sencrypt(answer, key): bitstring.
43  fun sdecrypt(bitstring, key): answer.
44  equation forall m: answer, k: key; sdecrypt(sencrypt(m,k),k) = m.
45  equation forall m: bitstring, k: key; sencrypt(sdecrypt(m,k),k) = m.
46
47  (* Asymetric key encryption *)
48  fun pk(skey): pkey.
49  fun aencrypt(bitstring, pkey): bitstring.
50  reduc forall m: bitstring, k: skey; adecrypt(aencrypt(m,pk(k)),k) = m.
51
52  (* Functions *)
53  fun ans(request): answer [private].
54  table receiverTable(IP, id).
55  table issuerTable(request, key).
56
57  query attacker(LINK_e(user, q)).
58
59
60  (* User *)
61  let userProcess(user: IP, q: request, pkI: pkey) =
62     new kU: key;
63     let payload = aencrypt((q, kU), pkI) in
64     (*       Message 1        *)
65     out(c, (user, receiver, payload));
66     out(c, LINK_a(user,payload));
67     (*       Message 6        *)
68     in(c, (=receiver, =user, m: bitstring));
69     let a = sdecrypt(m, kU) in
70     0.
71
72
73  (* Receiver *)
74  let receiverProcess =
75     (*       Message 1        *)
76     in(c, (user_X: IP, =receiver, m: bitstring));
77     let (m3: bitstring) = adecrypt(m, skP) in
78     new rid: id;
79     insert receiverTable(user_X,rid);
80     (*       Message 2        *)
81     out(c, (receiver, issuer, rid, m3));
82     out(c, LINK_c(rid,m3));
83     (*       Message 5        *)
84     in(c, (=issuer, =receiver, rid_X: id, mI: bitstring));
85     get proxyTable(user_Y,=rid_X) in
86     (*       Message 6        *)
87     out(c, (receiver, user_Y,mI));
88     out(c, LINK_a(user_Y,mI)).
89
90
91  (* Issuer *)
92  let issuerProcess(skI: skey) =
93     (*       Message 2        *)
94     in(c, (=receiver, =issuer, rid: id, m: bitstring));
95     let (qI: request, kU: key) = adecrypt(m, skI) in
96     insert issuerTable(qI,kU);
97     (*       Message 3        *)
98     out(c, (issuer, server, qI));
99     (*       Message 4        *)
100    in(c, (=server, =issuer, q2: request, a: answer));
101    get issuerTable(=q2,kUI) in
102    let payload = sencrypt(a, kUI) in
103    (*       Message 5        *)
104    out(c, (issuer, receiver, rid, payload));
105    out(c, LINK_c(rid, payload)).
106
107
108 (* Search Engine *)
```

```
109 let serverProcess =
110     (*      Message 3       *)
111     in(c, (src: IP, =server, q2: request));
112     (*      Message 4       *)
113     out(c, (server, src, q2, ans(q2))).
114
115
116 (* Start process *)
117 process new skI: skey;
118     let pkI = pk(skI) in
119     out(c, pkI);
120     (
121       (userProcess(user,q,pkI)) |
122       (!receiverProcess) |
123       (!issuerProcess(skI)) |
124       (!serverProcess)
125     )
```

APPENDIX D

Instructions Given in the Crowd-sourcing Task Conducted with Crowdflower

We present below the instructions we gave to annotators hired through the crows-sourcing platform. We provided a short description of the work and gave examples to illustrate the task.

Help us categorize Web queries

Overview

Given a query issued by real users, choose the best category for this query from a drop-down list and indicate if the query is sensitive or non-sensitive. We define sensitive queries as queries that you do not want to make public (e.g., relating to an embarrassing topic, revealing personal behavior/information). Otherwise, the query is non-sensitive.

Process

1. Read the query carefully
2. Choose one of the 4 categories we provide:
   - Health / Medical facts
   - Politic
   - Porn / Sexuality
   - Religion
3. Choose "Other" if and only if none of the categories above fit.
4. Answer the second question to indicate if the query is "Sensitive" or "Non Sensitive"

Examples

For the question "Choose the best category for this query"

- *used dirty gay underwear* -> "Porn / Sexuality" is the right category
- *iran buying modren weapons* -> "Politic" is the right category
- *short term dangers of ecstasy* -> "Health / Medical fact" is the right category
- *how does god reveal truth to you* -> "Religion" is the right category

For the question "Indicate if this query is sensitive or non-sensitive"

- *different pictures of fruit and vegetables* -> is "Non Sensitiv"
- *music composition lessons* -> is"Non Sensitive"
- *gays erotic nude wrestling* -> is "Sensitive"
- *surving an affair* -> is "Sensitive"

Summary

This task is about putting queries into appropriate categories and annotating sensitive queries. You need to choose the best available category for each query, or select "Other". You should consider the default categories carefully before choosing other: only select "Other" for queries that do not match any of available categories at all. Regarding the sensitivity of a query, you need to think as if you were sending this query and decide if it is a problem for you to reveal publicly this query (e.g., to your family, to your friends, to your insurance company). Sensitive queries are for instance queries that reveal personal information or indicate your interest in embarrassing topics.

Thank You!

Your answers will help us to adapt private Web search solutions by taking into account the category of a query and the sensitivity of a query. We appreciate your work on this task!

APPENDIX E

List of Categories Available in the Semantic Assessment Module (SAM)

acoustics
administration
agriculture
anatomy
animal husbandry
animals
anthropology
applied science
archaeology
archery
architecture
art
artisanship
astrology
astronautics
astronomy
athletics
atomic physic
aviation
badminton
banking
baseball
basketball
betting
biochemistry
biology
body care
book keeping
bowling
boxing
buildings
factotum
fashion
fencing
finance
fishing
folklore
food
football
free time
furniture
gas
gastronomy
genetics
geography
geology
geometry
golf
grammar
graphic arts
health
heraldry
history
hockey
home
humanities
hunting
hydraulics
industry
insurance
jewellery
law
philosophy
photography
physics
physiology
plants
plastic arts
play
politics
post
psychiatry
psychoanalysis
psychological features
psychology
publishing
pure science
quality
racing
radio
radiology
railway
religion
roman catholic
rowing
rugby
school
sculpture
sexuality
skating
skiing
soccer
social

card
chemistry
chess
cinema
color
commerce
computer science
cricket
cycling
dance
dentistry
diplomacy
diving
drawing
earth
economy
electricity
electronics
electrotechnology
engineering
enterprise
entomology
environment
ethnology
exchange
factotum
linguistics
literature
mathematics
mechanics
medicine
meteorology
metrology
military
money
mountaineering
music
mythology
nautical
number
numismatics
occultism
oceanography
optics
painting
paleontology
paranormal
pedagogy
person
pharmacy
philately
philosophy
social science
sociology
sport
statistics
sub
surgery
swimming
table tennis
tax
telecommunication
telegraphy
telephony
tennis
theatre
theology
time period
topography
tourism
town planning
transport
tv
university
vehicles
veterinary
volleyball

Bibliography

[1] Ralph Gross and Alessandro Acquisti. Information revelation and privacy in online social networks. In *Proceedings of the 2005 ACM Workshop on Privacy in the Electronic Society*, WPES '05, pages 71–80, 2005.

[2] Susan B Barnes. A privacy paradox: Social networking in the united states. *First Monday*, 11(9), 2006.

[3] Google Inc. Privacy & terms. `http://www.google.com/policies/privacy/key-terms/#toc-terms-sensitive-info`. Key Terms.

[4] Glenn Greenwald, Ewen MacAskill, and Laura Poitras. Edward snowden: the whistleblower behind the nsa surveillance revelations. *The Guardian*, 9(6), 2013.

[5] Yabing Liu, Krishna P Gummadi, Balachander Krishnamurthy, and Alan Mislove. Analyzing facebook privacy settings: user expectations vs. reality. In *Proceedings of the 2011 ACM SIGCOMM conference on Internet measurement conference*, IMC'11, pages 61–70, 2011.

[6] The European Parliament and The Council of the European Union. Data Protection Rules - Directive 95/46/EC (Article 12-b). `http://eur-lex.europa.eu/LexUriServ/LexUriServ.do?uri=CELEX:31995L0046:en:HTML`, 1995.

[7] Court of Justice of the European Union (CJEU). Judgment of the Court – Case C-131/12. `http://curia.europa.eu/juris/document/document.jsf?docid=152065&doclang=EN`, 2014.

[8] Google Inc. European privacy requests for search removals. `https://www.google.com/transparencyreport/removals/europeprivacy/`.

[9] The European Parliament and The Council of the European Union. Data Protection Rules - Regulation (EU) 2016/679 (Article 17). `http://eur-lex.europa.eu/legal-content/EN/TXT/HTML/?uri=CELEX:32016R0679`, 2016.

[10] Samuel Gibbs. Google to extend 'right to be forgotten' to all its domains accessed in EU. *The Guardian*, 2016.

[11] Microsoft Inc. Bing To Use Location for RTBF. `http://blogs.bing.com/search/august-2016/bing-to-use-location-for-rtbf`, 2016.

[12] The WWW Virtual Library. `http://vlib.org/`.

[13] Looksmart. `http://www.looksmart.com/`.

[14] Magdalini Eirinaki and Michalis Vazirgiannis. Web mining for web personalization. *ACM Transactions on Internet Technology (TOIT)*, 3(1):1–27, 2003.

[15] Daniel E Rose and Danny Levinson. Understanding user goals in web search. In *Proceedings of the 13th International Conference on World Wide Web*, WWW '04, pages 13–19. ACM, 2004.

[16] Greg Pass, Abdur Chowdhury, and Cayley Torgeson. A picture of search. In *InfoScale*, volume 152, page 1, 2006.

[17] Michael Barbaro, Tom Zeller, and Saul Hansell. A face is exposed for aol searcher no. 4417749. *New York Times*, 9(2008):8, 2006.

[18] Eytan Adar, Daniel S Weld, Brian N Bershad, and Steven S Gribble. Why we search: visualizing and predicting user behavior. In *Proceedings of the 16th international conference on World Wide Web*, WWW'07, pages 161–170. ACM, 2007.

[19] David J Brenes and Daniel Gayo-Avello. Stratified analysis of aol query log. *Information Sciences*, 179(12):1844–1858, 2009.

[20] Jeff Huang and Efthimis N Efthimiadis. Analyzing and evaluating query reformulation strategies in web search logs. In *Proceedings of the 18th ACM conference on Information and knowledge management*, CIKM'09, pages 77–86, 2009.

[21] Paul Heymann, Georgia Koutrika, and Hector Garcia-Molina. Can social bookmarking improve web search? In *Proceedings of the 2008 International Conference on Web Search and Data Mining*, WSDM '08, pages 195–206, 2008.

[22] Rosie Jones, Ravi Kumar, Bo Pang, and Andrew Tomkins. "i know what you did last summer": Query logs and user privacy. In *Proceedings of the 16th ACM conference on Conference on information and knowledge management*, CIKM'07, pages 909–914. ACM, 2007.

[23] Bin Bi, Milad Shokouhi, Michal Kosinski, and Thore Graepel. Inferring the demographics of search users: social data meets search queries. In *Proceedings of the 22nd international conference on World Wide Web*, WWW'13, pages 131–140, 2013.

[24] Daniel J Solove. *Nothing to hide: The false tradeoff between privacy and security*. Yale University Press, 2011.

[25] Gary T Marx. Privacy and technology. `http://web.mit.edu/gtmarx/www/privantt.html`. Revision of material that appeared in *The World and I*, Sept. 1990 and *Telektronik*, Jan. 1996.

[26] Bruce Schneier. The eternal value of privacy. `https://www.schneier.com/essays/archives/2006/05/the_eternal_value_of.html`, 2006.

[27] DuckDuckGo. `https://duckduckgo.com`.

[28] StartPage. `https://startpage.com`.

[29] Qwant. `https://www.qwant.com`.

[30] Gabriel Weinberg. DuckDuckGo Privacy Policy. `https://duckduckgo.com/privacy#s4`, 2010.

[31] HweeHwa Pang, Xuhua Ding, and Xiaokui Xiao. Embellishing text search queries to protect user privacy. *Proceedings of the VLDB Endowment*, 3(1-2):598–607, 2010.

[32] Hweehwa Pang, Jialie Shen, and Ramayya Krishnan. Privacy-preserving similarity-based text retrieval. *ACM Transactions on Internet Technology (TOIT)*, 10(1):4, 2010.

[33] Michael Naehrig, Kristin Lauter, and Vinod Vaikuntanathan. Can homomorphic encryption be practical? In *Proceedings of the 3rd ACM Workshop on Cloud Computing Security Workshop*, CCSW '11, pages 113–124, New York, NY, USA, 2011. ACM.

[34] Felipe Saint-Jean, Aaron Johnson, Dan Boneh, and Joan Feigenbaum. Private web search. In *Proceedings of the 2007 ACM Workshop on Privacy in Electronic Society*, WPES '07, pages 84–90. ACM, 2007.

[35] Sasha Romanosky and Cynthia Kuo. Foxtor: Anonymous web browsing. *Tor GUI Competition*, 2006.

[36] Collin Jackson, Andrew Bortz, Dan Boneh, and John C Mitchell. Protecting browser state from web privacy attacks. In *Proceedings of the 15th international conference on World Wide Web*, WWW'06, pages 737–744, 2006.

[37] Ghostery. `https://www.ghostery.com`.

[38] Paul F Syverson, David M Goldschlag, and Michael G Reed. Anonymous connections and onion routing. In *Proceedings of the 1997 IEEE Symposium on Security and Privacy*, SP'97, pages 44–54, 1997.

[39] Roger Dingledine, Nick Mathewson, and Paul Syverson. Tor: The second-generation onion router. In *Proceedings of the 13th Conference on USENIX Security Symposium*, USENIX Security'04, pages 21–21, 2004.

[40] Henry Corrigan-Gibbs and Bryan Ford. Dissent: Accountable anonymous group messaging. In *Proceedings of the 17th ACM Conference on Computer and Communications Security*, CCS'10, pages 340–350, 2010.

[41] David Isaac Wolinsky, Henry Corrigan-Gibbs, and Bryan Ford. Dissent in numbers: Making strong anonymity scale. In *Proceedings of the 10th USENIX Conference on Operating Systems Design and Implementation*, OSDI'12, 2012.

[42] Sonia Ben Mokhtar, Gautier Berthou, Amadou Diarra, Vivien Quéma, and Ali Shoker. Rac: A freerider-resilient, scalable, anonymous communication protocol. In *Proceedings of the 2013 IEEE 33rd International Conference on Distributed Computing Systems*, ICDCS'13.

[43] Josep Domingo-Ferrer, Maria Bras-Amorós, Qianhong Wu, and Jesús Manjón. User-private information retrieval based on a peer-to-peer community. *Data & Knowledge Engineering*, 68(11):1237–1252, 2009.

[44] Michael K Reiter and Aviel D Rubin. Crowds: Anonymity for web transactions. *ACM Transactions on Information and System Security (TISSEC)*, 1(1):66–92, 1998.

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

[46] Vincent Toubiana, Lakshminarayanan Subramanian, and Helen Nissenbaum. Trackmenot: Enhancing the privacy of web search. *arXiv preprint arXiv:1109.4677*, 2011.

[47] Sai Teja Peddinti and Nitesh Saxena. On the effectiveness of anonymizing networks for web search privacy. In *Proceedings of the 6th ACM Symposium on Information, Computer and Communications Security*, ASIACCS '11, pages 483–489, 2011.

[48] Sai Teja Peddinti and Nitesh Saxena. On the privacy of web search based on query obfuscation: a case study of trackmenot. In *Proceedings of the 10th International Conference on Privacy Enhancing Technologies*, PETS'10, pages 19–37. Springer, 2010.

[49] Arthur Gervais, Reza Shokri, Adish Singla, Srdjan Capkun, and Vincent Lenders. Quantifying web-search privacy. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, CCS '14, pages 966–977. ACM, 2014.

[50] Peter Eckersley. How unique is your web browser? In *Proceedings of the 10th International Conference on Privacy Enhancing Technologies*, PETS'10, pages 1–18, 2010.

[51] Nick Nikiforakis, Alexandros Kapravelos, Wouter Joosen, Christopher Kruegel, Frank Piessens, and Giovanni Vigna. Cookieless monster: Exploring the ecosystem of web-based device fingerprinting. In *Proceedings of the 2013 IEEE Symposium on Security and Privacy*, SP'13, pages 541–555. IEEE, 2013.

[52] Eran Gabber, Phillip B Gibbons, David M Kristol, Yossi Matias, and Alain Mayer. Consistent, yet anonymous, web access with lpwa. *Communications of the ACM*, 42(2):42–47, 1999.

[53] Marc Shapiro. Structure and Encapsulation in Distributed Systems: the Proxy Principle. In *Proceedings of the 1986 IEEE 6th International Conference on Distributed Computing Systems*, ICDCS'86.

[54] H.A. Seid and A.L. Lespagnol. Virtual private network, June 16 1998. US Patent 5,768,271.

[55] David L Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM*, 24(2):84–90, 1981.

[56] Oliver Berthold, Hannes Federrath, and Stefan Köpsell. Web mixes: A system for anonymous and unobservable internet access. In *Designing Privacy Enhancing Technologies*, pages 115–129. Springer, 2001.

[57] Benedikt Westermann, Rolf Wendolsky, Lexi Pimenidis, and Dogan Kesdogan. Cryptographic protocol analysis of an. on. In *International Conference on Financial Cryptography and Data Security*, pages 114–128. Springer, 2010.

[58] The Tor Project Inc. Estimated number of clients in the Tor network. `https://metrics.torproject.org/clients-data.html`.

[59] Alexandre Viejo and Jordi Castellà-Roca. Using social networks to distort users' profiles generated by web search engines. *Computer Networks*, 54(9):1343–1357, 2010.

[60] Arnau Erola, Jordi Castellà-Roca, Alexandre Viejo, and Josep M Mateo-Sanz. Exploiting social networks to provide privacy in personalized web search. *Journal of Systems and Software*, 84(10):1734–1745, 2011.

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

[62] Cristina Romero-Tris, Alexandre Viejo, and Jordi Castella-Roca. Improving query delay in private web search. In *Proceedings of the 2011 International Conference on P2P, Parallel, Grid, Cloud and Internet Computing*, 3PGCIC'11, pages 200–206. IEEE, 2011.

[63] Yehuda Lindell and Erez Waisbard. Private web search with malicious adversaries. In *Proceedings of the 10th International Conference on Privacy Enhancing Technologies*, PETS'10, pages 220–235. Springer, 2010.

[64] Cristina Romero-Tris, Jordi Castella-Roca, and Alexandre Viejo. Multiparty private web search with untrusted partners. In *Proceedings of the 2011 7th International Conference on Security and Privacy in Communication Networks*, SecureComm'11.

[65] Wai Han Soo, Azman Samsudin, and Alwyn Goh. Efficient mental card shuffling via optimised arbitrary-sized benes permutation network. In *Proceedings of the 5th International Conference on Information Security*, pages 446–458. Springer, 2002.

[66] Taher ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In *Proceedings of Advances in Cryptology*, CRYPTO'84.

[67] Markus Jakobsson and Ari Juels. Millimix: Mixing in small batches. Technical report, DIMACS Technical report 99-33, 1999.

[68] Zhengjun Cao, Lihua Liu, and Zhenzhen Yan. An improved lindell-waisbard private web search scheme. *International Journal of Network Security*, 18(3):538–543, 2016.

[69] David Rebollo-Monedero, Jordi Forne, and Josep Domingo-Ferrer. Query profile obfuscation by means of optimal query exchange between users. *IEEE Transactions on Dependable and Secure Computing*, 9(5):641–654, 2012.

[70] Claude Elwood Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27(3):379–423, 1948.

[71] Yuval Elovici, Bracha Shapira, and Adlai Maschiach. A new privacy model for hiding group interests while accessing the web. In *Proceedings of the 2002 ACM Workshop on Privacy in the Electronic Society*, WPES'02, pages 63–70, 2002.

[72] Yuval Elovici, Chanan Glezer, and Bracha Shapira. Enhancing customer privacy while searching for products and services on the world wide web. *Internet Research*, 15(4):378–399, 2005.

[73] Bracha Shapira, Yuval Elovici, Adlay Meshiach, and Tsvi Kuflik. Praw - a privacy model for the web. *Journal of the American Society for Information Science and Technology*, 56(2):159–172, 2005.

[74] Mummoorthy Murugesan and Chris Clifton. Providing privacy through plausibly deniable search. In *SDM*, pages 768–779. SIAM, 2009.

[75] Alexandre Viejo and Dominick Sanchez. Providing useful and private web search by means of social network profiling. In *Proceedings of the 11th Annual International Conference on Privacy, Security and Trust*, PST'13, pages 358–361. IEEE, 2013.

[76] Alexandre Viejo and David Sánchez. Profiling social networks to provide useful and privacy-preserving web search. *Journal of the Association for Information Science and Technology*, 65(12):2444–2458, 2014.

[77] Wikimedia Foundation Inc. Wikinews Print Edition. `https://en.wikinews.org/wiki/Wikinews:Print_edition/October_2007`, 2007.

[78] Daniel C Howe and Helen Nissenbaum. Trackmenot: Resisting surveillance in web search. *Lessons from the Identity Trail: Anonymity, Privacy, and Identity in a Networked Society*, 23:417–436, 2009.

[79] Alexandre Viejo, Jordi Castella-Roca, Oriol Bernadó, and Josep M Mateo-Sanz. Single-party private web search. In *Proceedings of the 10th Annual International Conference on Privacy, Security and Trust*, PST'12, pages 1–8. IEEE, 2012.

[80] David Sánchez, Jordi Castellà-Roca, and Alexandre Viejo. Knowledge-based scheme to create privacy-preserving but semantically-related queries for web search engines. *Information Sciences*, 218:17–30, 2013.

[81] AOL Inc. DMOZ, An Open Directory Project (ODP). `http://www.dmoz.org`, 1998.

[82] George A Miller. Wordnet: a lexical database for english. *Communications of the ACM*, 38(11):39–41, 1995.

[83] David Rebollo-Monedero and Jordi Forné. Optimized query forgery for private information retrieval. *IEEE Transactions on Information Theory*, 56(9):4631–4642, 2010.

[84] Shaozhi Ye, Felix Wu, Raju Pandey, and Hao Chen. Noise injection for search privacy protection. In *Proceedings of the 2009 International Conference on Computational Science and Engineering*, volume 3 of *CSE'09*, pages 1–8. IEEE, 2009.

[85] Avi Arampatzis, Pavlos S Efraimidis, and George Drosatos. A query scrambler for search privacy on the internet. *Information retrieval*, 16(6):657–679, 2013.

[86] Avi Arampatzis, George Drosatos, and Pavlos S Efraimidis. Versatile query scrambling for private web search. *Information Retrieval Journal*, 18(4):331–358, 2015.

[87] Marc Juárez and Vicenç Torra. Toward a privacy agent for information retrieval. *International Journal of Intelligent Systems*, 28(6):606–622, 2013.

[88] Marc Juárez and Vicenç Torra. A self-adaptive classification for the dissociating privacy agent. In *Proceedings of the 11th IEEE International Conference on Privacy, Security and Trust*, PST'13, pages 44–50. IEEE, 2013.

[89] Marc Juarez and Vicenç Torra. Dispa: An intelligent agent for private web search. In *Advanced Research in Data Privacy*, pages 389–405. Springer, 2015.

[90] Yabo Xu, Ke Wang, Benyu Zhang, and Zheng Chen. Privacy-enhancing personalized web search. In *Proceedings of the 16th international conference on World Wide Web*, WWW'07.

[91] Gang Chen, He Bai, Lidan Shou, Ke Chen, and Yunjun Gao. Ups: efficient privacy protection in personalized web search. In *Proceedings of the 34th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR'11, pages 615–624, 2011.

[92] Claudia Díaz, Stefaan Seys, Joris Claessens, and Bart Preneel. Towards measuring anonymity. In *Proceedings of the 2nd International Conference on Privacy Enhancing Technologies*, PETS'02, pages 54–68. Springer-Verlag, 2003.

[93] Edward W Felten and Michael A Schneider. Timing attacks on web privacy. In *Proceedings of the 7th ACM conference on Computer and communications security*, CCS '00, pages 25–32, 2000.

[94] Riccardo Focardi, Roberto Gorrieri, Ruggero Lanotte, Andrea Maggiolo-Schettini, Fabio Martinelli, Simone Tini, and Enrico Tronci. Formal models of timing attacks on web privacy. *Electronic Notes in Theoretical Computer Science*, 62:229–243, 2002.

[95] Nathan S Evans, Roger Dingledine, and Christian Grothoff. A practical congestion attack on tor using long paths. In *Proceedings of the 18th Conference on USENIX Security Symposium*, USENIX Security'09, pages 33–50, 2009.

[96] Rami Al-Rfou, William Jannen, and Nikhil Patwardhan. Trackmenot-so-good-after-all. *arXiv preprint arXiv:1211.0320*, 2012.

[97] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine learning*, 20(3):273–297, 1995.

[98] David R Cox. The regression analysis of binary sequences. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 215–242, 1958.

[99] James MacQueen et al. Some methods for classification and analysis of multivariate observations. In *Proceedings of the 5th Berkeley symposium on mathematical statistics and probability*, BSMSP'67.

[100] Chih-Wei Hsu and Chih-Jen Lin. A comparison of methods for multiclass support vector machines. *IEEE Transactions on Neural Networks*, 13(2):415–425, 2002.

[101] Jason Weston, Chris Watkins, et al. Support vector machines for multi-class pattern recognition. In *ESANN*, volume 99, pages 219–224, 1999.

[102] John C Platt. 12 fast training of support vector machines using sequential minimal optimization. *Advances in kernel methods*, pages 185–208, 1999.

[103] John H Aldrich and Forrest D Nelson. *Linear probability, logit, and probit models*, volume 45. Sage, 1984.

[104] Edward W Forgy. Cluster analysis of multivariate data: efficiency versus interpretability of classifications. *Biometrics*, 21:768–769, 1965.

[105] Yiming Yang. An evaluation of statistical approaches to text categorization. *Information retrieval*, 1(1-2):69–90, 1999.

[106] Saikat Guha, Mudit Jain, and Venkata N Padmanabhan. Koi: A location-privacy platform for smartphone apps. In *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*, pages 14–14. USENIX Association, 2012.

[107] Bruno Blanchet. An efficient cryptographic protocol verifier based on prolog rules. In *Proceedings of the 14th IEEE Workshop on Computer Security Foundations*, CSFW '01, page 82. IEEE Computer Society, 2001.

[108] Alessandro Armando, David Basin, Yohan Boichut, Yannick Chevalier, Luca Compagna, Jorge Cuéllar, P Hankes Drielsma, Pierre-Cyrille Héam, Olga Kouchnarenko, Jacopo Mantovani, et al. The avispa tool for the automated validation of internet security protocols and applications. In *Computer Aided Verification*, pages 281–285. Springer, 2005.

[109] Mark B Abbott and Larry L Peterson. Increasing network throughput by integrating protocol layers. *IEEE/ACM Transactions on Networking (TON)*, 1(5):600–610, 1993.

[110] Yong Ki Lee, Kazuo Sakiyama, Lejla Batina, and Ingrid Verbauwhede. Elliptic-curve-based security processor for rfid. *IEEE Transactions on Computers*, 57(11):1514–1527, 2008.

[111] Nachiketh R Potlapally, Srivaths Ravi, Anand Raghunathan, and Niraj K Jha. A study of the energy consumption characteristics of cryptographic algorithms and security protocols. *IEEE Transactions on mobile computing*, 5(2):128–143, 2006.

[112] Charles Spearman. 'footrule'for measuring correlation. *British Journal of Psychology, 1904-1920*, 2(1):89–108, 1906.

[113] Maurice G Kendall. A new measure of rank correlation. *Biometrika*, 30(1/2):81–93, 1938.

[114] Kalervo Järvelin and Jaana Kekäläinen. Cumulated gain-based evaluation of ir techniques. *ACM Transactions on Information Systems (TOIS)*, 20(4):422–446, October 2002.

[115] Xuehua Shen, Bin Tan, and ChengXiang Zhai. Privacy protection in personalized search. In *ACM SIGIR Forum*, volume 41, pages 4–17. ACM, 2007.

[116] Xuehua Shen, Bin Tan, and ChengXiang Zhai. Implicit user modeling for personalized search. In *Proceedings of the 14th ACM international conference on Information and knowledge management*, CIKM'05, pages 824–831. ACM, 2005.

[117] Microsoft Corporation. Microsoft creative acceptance policy. http://fp.advertising.microsoft.com/en-uk/WWDocs/User/display/cl/content_standard/2007/global/Microsoft-Advertising-Creative-Acceptance-Policy-Guide.pdf.

[118] Yahoo! Inc. Yahoo advertising policies. https://adspecs.yahoo.com/pages/yahooadpolicies/.

[119] Digital Advertising Alliance (DAA). Self Regulatory Principles for Online Behavioral Advertising Implementation Guide (FAQ). http://www.aboutads.info/resource/download/OBA%20Self-Reg%20Implementation%20Guide%20-%20Frequently%20Asked%20Questions.pdf, 2010.

[120] French Republic. Loi n° 78-17 du 6 janvier 1978 relative à l'informatique, aux fichiers et aux libertés. https://www.legifrance.gouv.fr/jo_pdf.do?id=JORFTEXT000000886460, 1978.

[121] The European Parliament and The Council of the European Union. Data Protection Rules - Directive 95/46/EC (Article 8). http://eur-lex.europa.eu/LexUriServ/LexUriServ.do?uri=CELEX:31995L0046:en:HTML, 1995.

[122] United Kingdom. Data Protection Act 1998. http://www.legislation.gov.uk/ukpga/1998/29/pdfs/ukpga_19980029_en.pdf, 1998.

[123] Article 29 Working Party. Advice paper on special categories of data ("sensitive data"), 2011.

[124] The U.S. Congress. Fair Debt Collection Practices act (FDCPA). https://www.ncjrs.gov/pdffiles1/Digitization/55324NCJRS.pdf, 1977.

[125] The U.S. Congress. Health Insurance Portability and Accountability Act (HIPAA). http://www.gpo.gov/fdsys/pkg/PLAW-104publ191/html/PLAW-104publ191.htm, 1996.

[126] The U.S. Congress. Electronic Communications Privacy Act (ECPA). http://www.gpo.gov/fdsys/pkg/STATUTE-100/pdf/STATUTE-100-Pg1848.pdf, 1986.

[127] State of California (USA). California Civil Code. http://www.leginfo.ca.gov/cgi-bin/displaycode?section=civ&group=01001-02000&file=1798.80-1798.84. Section 1798.83.

[128] Sai Teja Peddinti, Aleksandra Korolova, Elie Bursztein, and Geetanjali Sampemane. Cloak and swagger: Understanding data sensitivity through the lens of user anonymity. In *Proceedings of the 2014 IEEE Symposium on Security and Privacy*, SP'14, pages 493–508, 2014.

[129] Prashant Ullegaddi and Vasudeva Varma. A simple unsupervised query categorizer for web search engines. In *ICON'10*, 2011.

[130] Lin Li, Guandong Xu, Zhenglu Yang, Yanchun Zhang, and Masaru Kitsuregawa. A feature-free flexible approach to topical classification of web queries. In *Proceedings of the 7th IEEE International Conference on Semantics Knowledge and Grid*, SKG'11.

[131] Milad Alemzadeh and Fakhri Karray. An efficient method for tagging a query with category labels using wikipedia towards enhancing search engine results. In *Proceedings of the 2010 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology*, volume 1, pages 192–195. IEEE, 2010.

[132] Steven M Beitzel, Eric C Jensen, Ophir Frieder, David D Lewis, Abdur Chowdhury, and Aleksander Kolcz. Improving automatic query classification via semi-supervised learning. In *Proceedings of the 5th IEEE International Conference on Data Mining*, ICDM'05, pages 8–pp. IEEE, 2005.

[133] Steven M Beitzel, Eric C Jensen, Ophir Frieder, David Grossman, David D Lewis, Abdur Chowdhury, and Aleksandr Kolcz. Automatic web query classification using labeled and unlabeled training data. In *Proceedings of the 28th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '05, pages 581–582. ACM, 2005.

[134] Steven M Beitzel, Eric C Jensen, David D Lewis, Abdur Chowdhury, and Ophir Frieder. Automatic classification of web queries using very large unlabeled query logs. *ACM Transactions on Information Systems (TOIS)*, 25(2):9, 2007.

[135] Dou Shen, Rong Pan, Jian-Tao Sun, Jeffrey Junfeng Pan, Kangheng Wu, Jie Yin, and Qiang Yang. Query enrichment for web-query classification. *ACM Transactions on Information Systems (TOIS)*, 24(3):320–352, 2006.

[136] Zsolt T Kardkovács, Domonkos Tikk, and Zoltán Bánsághi. The ferrety algorithm for the kdd cup 2005 problem. *ACM SIGKDD Explorations Newsletter*, 7(2):111–116, 2005.

[137] David Vogel, Steffen Bickel, Peter Haider, Rolf Schimpfky, Peter Siemen, Steve Bridges, and Tobias Scheffer. Classifying search engine queries using the web as background knowledge. *ACM SIGKDD Explorations Newsletter*, 7(2):117–122, 2005.

[138] Lee R Dice. Measures of the amount of ecologic association between species. *Ecology*, 26(3):297–302, 1945.

[139] Amit Singhal. Modern information retrieval: A brief overview. *IEEE Data Eng. Bull.*, 24(4):35–43, 2001.

[140] Paul Jaccard. The distribution of the flora in the alpine zone. *New phytologist*, 11(2):37–50, 1912.

[141] Christopher D. Manning, Mihai Surdeanu, John Bauer, Jenny Finkel, Steven J. Bethard, and David McClosky. The Stanford CoreNLP natural language processing toolkit. In *ACL'14*, pages 55–60, June 2014.

[142] Martin F Porter. An algorithm for suffix stripping. *Program*, 14(3):130–137, 1980.

[143] Soutik Biswas for BBC News. Digital Indians: Ben Gomes. `http://www.bbc.com/news/technology-23866614`, 2013.

[144] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H Witten. The weka data mining software: an update. *ACM SIGKDD explorations newsletter*, 11(1):10–18, 2009.

[145] John Platt et al. Sequential minimal optimization: A fast algorithm for training support vector machines. 1998.

[146] Chih-Chung Chang and Chih-Jen Lin. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2:27:1–27:27, 2011.

[147] University of Waikato. Javadoc related to the class libsvm implemented in weka. `http://weka.sourceforge.net/doc.stable/weka/classifiers/functions/LibSVM.html`.

[148] S. le Cessie and J.C. van Houwelingen. Ridge estimators in logistic regression. *Applied Statistics*, 41(1):191–201, 1992.

[149] Kenneth Lange. A quasi-newton acceleration of the em algorithm. *Statistica sinica*, pages 1–18, 1995.

[150] Sai Teja Peddinti and Nitesh Saxena. Web search query privacy: Evaluating query obfuscation and anonymizing networks1. *Journal of Computer Security*, 22(1):155–199, 2014.

[151] Alexey Tsymbal. The problem of concept drift: definitions and related work. *Computer Science Department, Trinity College Dublin*, 106, 2004.

[152] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[153] Gene H Golub and Christian Reinsch. Singular value decomposition and least squares solutions. *Numerische mathematik*, 14(5):403–420, 1970.

[154] G. Salton, A. Wong, and C. S. Yang. A vector space model for automatic indexing. *Communications of the ACM*, 18(11):613–620, November 1975.

[155] Google Inc. Google trends. `https://www.google.com/trends/`, 2012.

[156] Coen Bron and Joep Kerbosch. Algorithm 457: Finding all cliques of an undirected graph. *Communications of the ACM*, 16(9):575–577, September 1973.

[157] Caroline Fontaine and Fabien Galand. A survey of homomorphic encryption for nonspecialists. *EURASIP Journal on Information Security*, 2007(1):1–10, 2007.

[158] Agence Nationale de la Sécurité des Systèmes d'Information (ANSSI). Annexe B1 au Référentiel général de sécurité (version 2.03) : Choix et dimensionnement des mécanismes cryptographiques.

[159] Elaine Barker and Allen Roginsky. NIST Special Publication 800-131A Transitions (revision 1): Recommendation for Transitioning the Use of Cryptographic Algorithms and Key Lengths. 2015.

[160] Daniel Balouek, Alexandra Carpen Amarie, Ghislain Charrier, Frédéric Desprez, Emmanuel Jeannot, Emmanuel Jeanvoine, Adrien Lèbre, David Margery, Nicolas Niclausse, Lucas Nussbaum, Olivier Richard, Christian Pérez, Flavien Quesnel, Cyril Rohr, and Luc Sarzyniec. Adding virtualization capabilities to the Grid'5000 testbed. In *Proceedings of the 2nd International Conference on Cloud Computing and Services Science*, volume 367 of *CLOSER'12*, pages 3–20. Springer International Publishing, 2013.

[161] The Legion of the Bouncy Castle Inc. Bouncy castle cryptographic apis. `https://www.bouncycastle.org`, 2006.

[162] John G Kemeny, James Laurie Snell, et al. *Finite markov chains*, volume 356. van Nostrand Princeton, NJ, 1960.

[163] Etsuji Tomita, Akira Tanaka, and Haruhisa Takahashi. The worst-case time complexity for generating all maximal cliques and computational experiments. *Theoretical Computer Science*, 363(1):28–42, 2006.

[164] David Eppstein, Maarten Löffler, and Darren Strash. Listing all maximal cliques in sparse graphs in near-optimal time. In *International Symposium on Algorithms and Computation*, pages 403–414. Springer, 2010.

[165] Joseph L Fleiss. Measuring nominal scale agreement among many raters. *Psychological bulletin*, 76(5):378, 1971.

[166] J Richard Landis and Gary G Koch. The measurement of observer agreement for categorical data. *biometrics*, pages 159–174, 1977.

[167] Ludmila I Kuncheva, Christopher J Whitaker, Catherine A Shipp, and Robert PW Duin. Limits on the majority vote accuracy in classifier fusion. *Pattern Analysis & Applications*, 6(1):22–31, 2003.

[168] Panagiotis G. Ipeirotis, Foster Provost, and Jing Wang. Quality management on amazon mechanical turk. In *Proceedings of the ACM SIGKDD workshop on human computation*, HCOMP '10, pages 64–67, 2010.

[169] Jacob Whitehill, Ting-fan Wu, Jacob Bergsma, Javier R Movellan, and Paul L Ruvolo. Whose vote should count more: Optimal integration of labels from labelers of unknown expertise. In *Advances in neural information processing systems*, pages 2035–2043, 2009.

[170] Aitor González, German Rigau, and Mauro Castillo. A graph-based method to improve wordnet domains. In *Proceedings of the 13th International Conference on Computational Linguistics and Intelligent Text Processing*, CICLing'12, pages 17–28. Springer, 2012.

[171] Michael L Fredman and Dan E Willard. Trans-dichotomous algorithms for minimum spanning trees and shortest paths. *Journal of Computer and System Sciences*, 48(3):533–551, 1994.

[172] Zhibiao Wu and Martha Palmer. Verbs semantics and lexical selection. In *Proceedings of the 32nd annual meeting on Association for Computational Linguistics*, pages 133–138, 1994.

[173] Yiming Yang. Noise reduction in a statistical approach to text categorization. In *Proceedings of the 18th annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR'95, pages 256–263, 1995.

[174] Hyunsoo Kim, Peg Howland, and Haesun Park. Dimension reduction in text classification with support vector machines. *Journal of Machine Learning Research*, 6(Jan):37–53, 2005.

[175] Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Jauvin. A neural probabilistic language model. *Journal of machine learning research*, 3(Feb):1137–1155, 2003.

[176] Daqing He, Ayşe Göker, and David J Harper. Combining evidence for automatic web session identification. *Information Processing & Management*, 38(5):727–742, 2002.

[177] Rosie Jones and Kristina Lisa Klinkner. Beyond the session timeout: automatic hierarchical segmentation of search topics in query logs. In *Proceedings of the 17th ACM Conference on Information and Knowledge Management*, CIKM'08, pages 699–708. ACM, 2008.

[178] Damon McCoy, Kevin Bauer, Dirk Grunwald, Tadayoshi Kohno, and Douglas Sicker. Shining light in dark places: Understanding the tor network. In *Proceedings of the 8th International Symposium on Privacy Enhancing Technologies*, PETS'08, pages 63–76, 2008.

[179] Niels Provos Panayiotis Mavrommatis and Moheeb Abu Rajab Fabian Monrose. All your iframes point to us. In *Proceedings of the 17th Conference on Security Symposium*, USENIX Security'08, pages 1–15, 2008.

[180] Andreas Haeberlen, Petr Kouznetsov, and Peter Druschel. Peerreview: Practical accountability for distributed systems. In *ACM SIGOPS operating systems review*, volume 41, pages 175–188. ACM, 2007.

[181] Andreas Haeberlen, Paarijaat Aditya, Rodrigo Rodrigues, and Peter Druschel. Accountable virtual machines. In *Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation*, OSDI'10, pages 119–134, 2010.

[182] Twitter. `http://www.twitter.com/`.