A Survey of Alerting Websites: Risks and Solutions

Amrit Kumar and Cédric Lauradoux

INRIA, Grenoble, France
{amrit.kumar,cedric.lauradoux}@inria.fr

Abstract. In the recent years an incredible amount of data has been leaked from major websites such as Adobe, Snapchat and LinkedIn. There are hundreds of millions of usernames, email addresses, passwords, telephone numbers and credit card details in the wild. The aftermath of these breaches is the rise of *alerting websites* such as `haveibeenpwned.com`, which let users verify if their accounts have been compromised. Unfortunately, these seemingly innocuous websites can be easily turned into phishing tools. In this work, we provide a comprehensive study of the most popular ones. Our study exposes the associated privacy risks and evaluates existing solutions towards designing *privacy-friendly alerting websites*. In particular, we study three solutions: private set intersection, private set intersection cardinality and private information retrieval adapted to membership testing. Finally, we investigate the practicality of these solutions with respect to real world database leakages.

Keywords: Data leakages, Phishing, Private Set Intersection, Private Information Retrieval, Bloom filter

1 Introduction

In the recent years, we have witnessed an increasing number of data leaks from major Internet sites including Adobe, Snapchat, LinkedIn, eBay, Apple and Yahoo ! (see `bit.ly/19xscQO` for more instances). While in most of the cases passwords' files have been targeted; database of identifiers, phone numbers and credit card details have also been successfully exfiltrated and published. These leakages dealt a substantial blow to the trust of people in computer security.

The aftermath of these leakages has led to three pivotal developments. First, the bad security policies of major websites have been exposed, and better policies have been proposed to survive leakages (see [27], [20], [18]). In [27], Parno et al. design an architecture to prevent database leakage. At CCS 2013, Kontaxis et al. propose SAuth [20], an authentication scheme which can survive password leakage. At the same conference, Juels and Rivest present the Honeywords [18] to detect if a passwords' file has been compromised.

Second, security community has obtained datasets to study the password habits of users. In [7], Das et al. consider several leaked databases to analyze password reuse. De Carnavalet et al. [8] use these datasets to test the effectiveness of password meters.

Third, a new kind of websites has appeared: *alerting website*. Users can check through these sites whether their accounts have been compromised or not. In order to check whether a user is a victim of data leakage, alerting websites ask for an identifying data such as username or email address and sometimes even password. These

websites are maintained by security experts such as `haveibeenpwned.com` by Troy Hunt, security companies e.g., LastPass, and even government institutions like the German Federal Office for Information Security (Bsi): `sicherheitstest.bsi.de`.

On one hand, these websites are very useful in alerting users, while on the other hand, they are real "*booby traps*". The problem is the following: when a user submits a username or an email address or a password, the site searches whether it exists or not in the leaked database. If it exists, the user is warned and the website has accomplished its purpose. However, if it is not present in the database, the site owner learns for free a username/email address/password.

Most of these sites advert to the users that they do not indulge in phishing activities but this is the only guarantee available to the user. **The goal of alerting websites is to reduce the effect of data leakage but not amplify it!** Considering the risks of using alerting websites, we naturally raise the following question: How to design alerting websites which cannot be turned into a phishing trap? The user must have a guarantee that it is not possible for the database owner to collect his information during a search query.

With the increasing frequency of data leakages, these websites are fast becoming a *sine qua non* for the victims of data leakages. Consequently, an analysis of these websites and their service is necessary. Our work presents a comprehensive study of alerting websites from two angles: the associated privacy risks and possible solutions to improve the service.

Contribution. The contribution of the paper is threefold:

1. We examine 17 popular alerting websites (Section 2) and analyze their working mechanism, and their approach to deal with privacy. Our findings reveal that several of these websites have huge phishing potential and hence users should be careful while visiting any of these websites.
2. We evaluate existing solutions for designing *privacy-friendly alerting websites*. Two different scenarios have been considered depending on whether or not the database is public. In case of private database (Section 4), *private set intersection protocol* and its variant *private set intersection cardinality protocol* yield an immediate solution. The scenario of public database (Section 5) requires us to adapt *private information retrieval protocol* for membership testing. This is achieved by combining it with *Bloom filters* (Section 5.1). These protocols subsumed under the name of *Private Membership Query* protocols ensure user's privacy in the *honest-but-curious* model.
3. Finally, we experimentally analyze the merits of these solutions with respect to real world data leakages (Section 6).

2 Alerting Websites: Risks

Users can be alerted on the fact that their account or personal information has been leaked. We discuss the characteristics of these websites which evidently offer opportunities for sophisticated phishing attacks.

Websites alerting users about their account or data leakage can be divided into three types according to their sources of information. In the sequel, we categorically discuss our findings.

Single-source (S). Some websites are associated with a single data leakage. This is the case for `adobe.cynic.al`, `bit.ly/1by3hd9`, `lucb1e.com/credgrep` and `adobe.breach.il.ly`. These websites are related to the Adobe data leakage of 153 million accounts which occurred in October 2013. The last three websites were successfully tested on 22/12/2014 but cannot be accessed anymore. Other websites for instance `snapcheck.org`,[1] `findmysnap.com`, and `lookup.gibsonsec.org` are similarly associated with the SnapChat leakage (4.6 million usernames, phone numbers and city exposed in January 2014).

Aggregator (A). We observe that 5 of these websites search through several databases to inform users if their data has been exposed. For instance, `shouldichangemypassword.com` (`bit.ly/1aJubEh` for short) allegates to use 3346 leaked databases while only 194 are officially known. The remaining four are maintained by private companies: `lastpass.com`, `bit.ly/1fj0SqV`, `bit.ly/1aJubEh` and `dazzlepod.com/disclosure`. The last remaining site `haveibeenpwned.com` is designed and maintained by security expert Troy Hunt.

Harvester (H). Three sites claim to have created their own databases from harvested data. Two of these are maintained by famous security companies `hacknotifier.com` and `pwnedlist.com/query`. The last site is maintained by the German Federal Office for Information Security (Bsi).

The `rue89.nouvelobs.com` site is slightly different from the others. In September 2014, this French news website bought on the Darknet 20 million French email addresses for 0.0419 bitcoins (see article `bit.ly/1lKAxsB`). The article offers the opportunity to check if the reader's addresses are included in the leak.

Table 1. Analysis of 17 alerting websites (* result as on 22/12/2014).

Websites	Type	Database(s)	https	Statement	Answer	Descrip.
`rue89.nouvelobs.com`	S	Unknown	✗	✓	✓	✗
`adobe.cynic.al`	S	Adobe	✗	✗	✓	✓
`bit.ly/1by3hd9`*	S		✓	✓	✓	✗
`lucb1e.com/credgrep`*	S		✗	✗	✗	✗
`adobe.breach.il.ly`*	S		✗	✗	✗	✗
`snapcheck.org`	S	SnapChat	✗	✗	✓	✗
`findmysnap.com`	S		✗	✗	✓	✗
`lookup.gibsonsec.org`	S		✗	✗	✓	✗
`didigetgawkered.com`	S	Gawker	✗	✗	✗	✗
`lastpass.com`	A	6	✓	✓	✓	✗
`haveibeenpwned.com`	A	9	✓	✓	✓	✓
`bit.ly/1fj0SqV`	A	12	✓	✗	✗	✗
`dazzlepod.com/disclosure`	A	28	✗	✓	✓	✗
`bit.ly/1aJubEh`	A	3346/194	✓	✗	✓	✗
`hacknotifier.com`	H	Unknown	✗	✓	✓	✗
`pwnedlist.com/query`	H	Unknown	✓	✗/✓	✓	✓
`sicherheitstest.bsi.de`	H	Botnets	✓	✓	✓	✗

[1] The database cannot be accessed anymore (10/12/2014).

We have reviewed 17 alerting sites and our findings are summarized in Table 1. To measure if a user can trust the service offered by these sites, we have considered four criteria:

- The usage of a secure connection through HTTPS.
- The existence or not of a security/privacy statement.
- The fact that the site responds or not with an answer.
- A technical description of all the operations performed on the data.

From Table 1, we observe that ten of these sites do not use HTTPS which means that the traffic towards them can be easily eavesdropped. Single-source alerting sites are the least trustworthy of all because most of them do not have a privacy statement. The website `bit.ly/1by3hd9` is a notable exception. Aggregator sites in general perform better. Most of them use HTTPS and have a statement concerning privacy or phishing. The website `haveibeenpwned.com` even has a description of how it works.

The harvesters are more controversial: `hacknotifier.com` claims that "*we use a 256-bit secured and encrypted SSL connection*", but does not use HTTPS or any encryption.[2] The website `pwnedlist.com/query` claims that "*this is not a phishing site*", but they also state (`pwnedlist.com/faq`) that "*Over the past years we've built an advanced data harvesting infrastructure that crawls the web 24/7 and gathers any potentially sensitive data ...*".

Four sites do not give any answer: either they are not working anymore (like `lucb1e.com/credgrep`) or they are real phishing traps.

Almost all the sites receive account information in clear. However, there are two notable exceptions `lastpass.com` and `dazzlepod.com/disclosure`. The former uses cryptographic hash functions and truncation to obfuscate the query and seems to be the most transparent and trustworthy of all. Table 2 presents a summary of our observations on `lastpass.com`. The latter source, `dazzlepod.com/disclosure` only recommends to truncate the email address. With `pwnedlist.com/query`, it is also possible to submit the SHA-512 digest of the email address instead of the address itself.

Table 2. Detailed analysis of `lastpass.com`.

Victim	**Query**	**Policy**	**Privacy method**
Adobe	Email	non-storage	None
LinkedIn	password	non-storage and non-logging	SHA-1
Snapchat	user name	non-storage and non-logging	SHA-1
Apple	UDID		Truncation
Last.fm	password	non-storage and non-logging	MD5
eHarmony	password	non-storage and non-logging	MD5

Cryptographic hash functions, e.g. MD5, SHA-1 or SHA-3 are however not enough to ensure the privacy of passwords, identifiers or email addresses: these data do not have full entropy. Email addresses were recovered from Gravatar digests [2] as well as passwords (see [25] for instance). Apple's **U**nique **D**evice **ID**s *aka* UDIDs are no exceptions. They are computed by applying SHA-1 on a serial number, IMEI or ECID,

[2] Actually, subscribing for the `hacknotifier.com` watchdog is also not secure.

the MAC address of WiFi and the MAC address of Bluetooth. The values used to produce a UDID can be guessed and LastPass asks only for the first 5 characters of UDID. It reduces the amount of information submitted to the site but the user is not warned if he provides more than 5 characters.

As a general conclusion, the measures taken by these websites are clearly not adequate to ensure the privacy of users' queries. In the remainder of the paper, we evaluate how existing cryptographic and privacy preserving primitives can solve the problems associated to alerting websites. These privacy-friendly solutions should guarantee that the websites cannot harvest any new data from a user's query.

3 Privacy-Friendly Solutions: Private vs. Public Database

As previously discussed, the existing alerting websites in general do not respect the privacy of a user and entail huge phishing potential. The need of the hour is to design *privacy-friendly alerting websites*. These websites would rely on what we refer as *Private Membership Query* protocols – allowing a user to privately test for membership in a given set/database. Such a protocol would guarantee that no new data can be harvested from a user's query.

To this end, two different privacy objectives can be defined depending on the privacy policy of the database owner. One that we henceforth refer as *Private Membership Query to Public Database*, and the other as *Private Membership Query to Private Database*. This classification arises due to the fact that most of these leaked databases are available on the Internet (as hackers have acquired the database dump) and hence can be considered as public in nature. However, even though they are public in terms of availability, an ethical hacker might want to ensure that the leaked information is not used for malicious purposes and hence the database cannot be accessed in a public manner to consult private information corresponding to other users. Rendering the database private could also be of interest for government agencies such as BSI `sicherheitstest.bsi.de`.

We highlight that a private membership query protocol provides a direct solution to the problem of designing privacy-friendly alerting websites. A user wishing to know whether his data has been leaked would be required to invoke the private membership protocol with the database owner and learns whether he is a victim of the breach. Thanks to the user's privacy provided by the protocol, no new data can then be harvested by the website. Consequently, in the rest of this work, we concentrate on evaluating solutions for private membership query problem. In the sequel, we formalize the privacy policies and examine viable solutions in the two database scenarios.

4 Solutions for Private Databases

The scenario of private membership query to private database involves a private database $\mathcal{DB}$ and a user $\mathcal{U}$. The database $\mathcal{DB} = \{y_1, \ldots, y_n\}$, where $y_i \in \{0,1\}^\ell$ consists of n bit-strings each of length ℓ. User $\mathcal{U}$ owns an arbitrary string $y \in \{0,1\}^\ell$. Private membership query to $\mathcal{DB}$ consists in knowing whether or not user's data y is present in the database while keeping y private to the user and $\mathcal{DB}$ private to the database.

Adversary model: The client and the database-owner are supposed to be *honest-but-curious* i.e. each follows the protocol but tries to learn information on the data held by the other player.

The above problem is very closely related to the problem of *Private Set Intersection*, hence we examine its applicability to designing privacy-friendly alerting websites.

Private Set Intersection (PSI). PSI protocol introduced by Freedman et al. [13] considers the problem of computing the intersection of private datasets of two parties. The scenario consists of two sets $\mathcal{U} = \{u_1, \ldots, u_m\}$, where $u_i \in \{0,1\}^\ell$ and $\mathcal{DB} = \{v_1, \ldots, v_n\}$, where $v_i \in \{0,1\}^\ell$ held by a user and the database-owner respectively. The goal of the user is to privately retrieve the set $\mathcal{U} \cap \mathcal{DB}$. The privacy requirement of the scheme consists in keeping $\mathcal{U}$ and $\mathcal{DB}$ private to their respective owner. Clearly, the private membership query to private database problem reduces to PSI for $m = 1$.

There is an abounding literature on novel and computationally efficient PSI protocols. The most efficient protocols are the ones by De Cristofaro et al. [10], Huang et al. [17] and Dong et al. [12]. The general conclusion being that for security of 80 bits, protocol by De Cristofaro et al. performs better than the one by Huang et al., while for higher security level, the latter protocol supersedes the former. The most efficient of all is the protocol by Dong et al. as it primarily uses symmetric key operations. We however note that the communication and the computational complexity of these protocols is linear in the size of the sets.

Private Set Intersection Cardinality (PSI-CA). PSI-CA is a variant of PSI where the goal of the client is to privately retrieve the cardinality of the intersection rather than the contents. While generic PSI protocols immediately provide a solution to PSI-CA, they however yield too much information. While several PSI-CA protocols have been proposed [13], [19], [16], [29], we concentrate on PSI-CA protocol of De Cristofaro et al. [9], as it is the most efficient of all. We also note that PSI-CA clearly provides a solution to the membership problem: if the size of the intersection is 0, then the user data is not present in the database.

5 Solutions for Public Databases

This scenario is modeled using a public database $\mathcal{DB}$ and a user $\mathcal{U}$. The database as in the previous scenario is $\mathcal{DB} = \{y_1, \ldots, y_n\}$, where $y_i \in \{0,1\}^\ell$. User $\mathcal{U}$ owns an arbitrary string $y \in \{0,1\}^\ell$ not necessarily in $\mathcal{DB}$. Private membership query consists in knowing whether or not user's data y is present in the database while keeping y private to the user.

The difference to the previous problem (Section 4) is that the database in this context is public. This leads to a trivial solution ensuring absolute privacy consisting in sending the database to the user, who using the available resources performs a search on the database. With huge databases of order GB, the trivial solution is not the most desirable one for low memory devices. In this scenario, a user would wish to securely outsource the search to the database-owner. In the following we present tools which provide a solution in the public database case.

5.1 Tools

In the first place we present a protocol called *Private Information Retrieval* [6], which is the closest to our needs. In the sequel we present Bloom filter and finally show that combining these tools allows us to obtain a protocol for private membership query to public database.

Private Information Retrieval (PIR). PIR first introduced in the seminal work by Chor et al. [6] is a mechanism allowing a user to query a public database while keeping his intentions private. In the classical setting of PIR [6], a user wants to retrieve the bit at index $1 \leq j \leq n$ in a database $\mathcal{DB} = \{y_1, \ldots, y_n\}$, where $y_i \in \{0,1\}$, but does not want the database to learn j.

Adversary model: The database owner is supposed to be honest-but-curious.

Since the work by Chor et al., several variants of PIR have been studied which include Private Block Retrieval (PBR) scheme – where the goal is to retrieve a block instead of a bit and **P**rivat**E** **R**etrieval by **K**e**Y**words (PERKY) [5] – where the user only holds a keyword kw instead of an index j. While PIR may either be built on single or replicated database copies, most of the latter works only consider the more realistic single database scenario. These works improve on the communication complexity [3,4], [14], [21,22]. The current best bound of $\mathcal{O}(\log^2 n)$ is independently achieved in [22], [14]. In this work, we only consider single database protocols. The principle reason being that in our context a user interacts with only one website.

Bloom Filter. Bloom filter [1] is a space and time efficient probabilistic data structure that provides an algorithmic solution to the *set membership query problem,* which consists in determining whether an item belongs to a predefined set.

Classical Bloom filter as presented in [1] essentially consists of k independent hash functions $\{h_1, \ldots, h_k\}$, where $\{h_i : \{0,1\}^* \rightarrow [0, m-1]\}_k$ and a bit vector $\boldsymbol{z} = (z_0, \ldots, z_{m-1})$ of size m initialized to $\mathbf{0}$. Each hash function uniformly returns an index in the vector $\boldsymbol{z}$. The filter $\boldsymbol{z}$ is incrementally built by inserting items of a predefined set $\mathcal{S}$. Each item $x \in \mathcal{S}$ is inserted into a Bloom filter by first feeding it to the hash functions to retrieve k indices of $\boldsymbol{z}$. Finally, insertion of x in the filter is achieved by setting the bits of $\boldsymbol{z}$ at these positions to 1.

In order to query if an item $y \in \{0,1\}^*$ belongs to $\mathcal{S}$, we check if y has been inserted into the Bloom filter $\boldsymbol{z}$. Achieving this requires y to be processed (as in insertion) by the same hash functions to obtain k indexes of the filter. If any of the bits at these indexes is 0, the item is not in the filter, otherwise the item is present (with a small *false positive probability*).

The space and time efficiency of Bloom filter comes at the cost of false positives. If $|\mathcal{S}| = n$, i.e., n items are to be inserted into the filter and the space available to store the filter is m bits, then the optimal number of hash functions to use and the ensuing optimal false positive probability f satisfy:

$$k = \frac{m}{n} \ln 2 \qquad \text{and} \qquad \ln f = -\frac{m}{n} (\ln 2)^2 . \tag{1}$$

Membership Query to Bloom Filter: 2-party setting. Let us assume that *Alice* wants to check if her value y is included in the Bloom filter $\boldsymbol{z}$ held by *Bob*. The easiest way to do so consists for Alice to send y to Bob. Bob queries the filter on input y. He then sends 0 or 1 to Alice as the query output. If the canal between Alice and Bob has limited capacity, another strategy is possible and is described in Fig. 1.

Alice cannot send y due to some channel constraints but she can send $a_i = h_i(y)$, for $1 \leq i \leq k$. We suppose that Alice and Bob first agree on the hash functions to be used. Then Alice sends a_i to Bob. In reply, Bob returns the bit at index a_i of $\boldsymbol{z}$ to

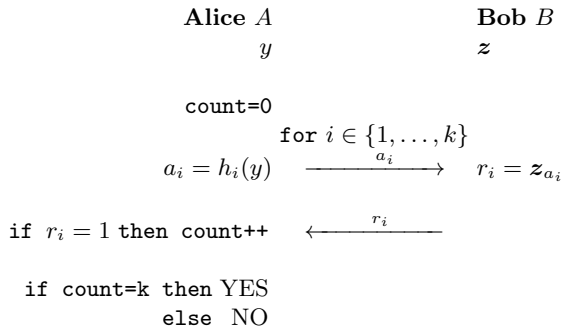

Fig. 1. Verification on a constraint channel.

her. If she only receives 1, y is included in $\boldsymbol{z}$ (with a small false positive probability f) otherwise it is not.

Remark 1. A possible private membership query protocol in the case of private database can be built by combining PSI/PSI-CA and Bloom filter. The idea would be to build a Bloom filter corresponding to the database entries and generate the set $\mathcal{DB} = \text{supp}(\boldsymbol{z})$, where $\text{supp}(\boldsymbol{z})$ represents the set of non-zero coordinate indices of $\boldsymbol{z}$. The client on the other hand generates $\mathcal{U} = \{h_1(y), \ldots, h_k(y)\}$ for a data y. Finally, the client and the database owner invoke a PSI/PSI-CA protocol to retrieve the intersection set/cardinality of the intersection respectively. However, this solution is less efficient than a PSI/PSI-CA protocol on the initial database itself. The reason being the fact that, with optimal parameters the expected size of $\text{supp}(\boldsymbol{z}) = m/2 = 2.88kn$ (see [24] for details) . Hence, the number of entries of the database in PSI/PSI-CA when used with Bloom filter is greater than the one of the original database.

We note that despite the similarity of the two problems: PIR and private membership to public database, PIR stand-alone does not provide a solution to our problem. Nevertheless, we show that when combined with a Bloom filter, PIR behaves as a private membership query protocol. Details are given in the following section.

5.2 Membership Query Using PIR

To start with, we note that classical PIR *per se* cannot be applied to our context since the user holding a data (email address, password, etc.) present in a database does not know its physical address in the database. Furthermore, PIR does not support non-membership queries as the database is constructed in a predefined manner and has only finite entries, while the set of all possible queries is infinite. PERKY resolves the problem of physical address as it only needs kw, and not the index. However, stand-alone it still suffers from the non-membership issue for the same reason as that in case of PIR.

Despite these issues, we can still design a private membership query protocol using PIR as a subroutine and by changing the database representation to Bloom filters which support non-membership queries as well. The idea then is to invoke PIR on each query to the filter.

The protocol explained below requires that the database owner builds a bloom filter $\boldsymbol{z}$ using the entries and a user queries the filter and not the database.

– Database owner builds the Bloom filter $\boldsymbol{z}$ using k hash functions $\{h_1, \ldots, h_k\}$.
– User for a data y generates $\{h_1(y), \ldots, h_k(y)\}$.
– For each $1 \leq i \leq k$, the user invokes a single-server PIR on index $h_i(y)$ and retrieves $z_{h_i(y)}$.
– If $z_{h_i(y)} = 0$ for any i, then y is not in the database, else if all the returned bits are 1, then the data is present (with a false positive probability f).

The only difference with the classical use of Bloom filter (Fig. 1) in the protocol is that the bit is retrieved using PIR.

Remark 2. As in the case of PIR, the database owner in our scenario is honest-but-curious. This attack model for instance does not allow the database owner to return a wrong bit to the user. Under this adversary model, the above protocol modification is private (i.e., keeps user's data private), if the underlying PIR scheme is private. PIR hides any single query of the user from the database owner. Therefore, any k different queries of the user are also hidden by PIR.

5.3 Extension with PBR Protocol

The adapted protocol in its current form requires a bit retrieval PIR scheme. Nevertheless, it can be easily modified to work even with a block retrieval *aka* PBR protocol. The essential advantage of using a PBR protocol instead of a classical PIR protocol would be to increase the throughput i.e. decrease the number of bits communicated to retrieve 1 bit of information. In fact, the most efficient PIR schemes [14], [22] are block retrieval schemes. The modification required to incorporate PBR would consist in using a *Garbled Bloom filter* (see [12]) instead of a Bloom filter. We briefly explain below the garbled Bloom filter construction, and later we present the modification required.

Garbled Bloom Filter. At a high level Garbled Bloom Filter $(k, m, \mathcal{H}_k, \lambda)$ GBF [12] is essentially the same as a Bloom filter. The parameter k denotes the number of hash functions used, while $\mathcal{H}_k$ is a family of k independent hash functions as in a Bloom filter. The size of the filter is denoted by m, and λ is the size of the items to be included in the filter. The difference with respect to a Bloom filter is that at each index in GBF, a bit string of length λ is stored instead of just storing the bit 1. In order to include an item $y \in \{0,1\}^\lambda$, one randomly generates k shares $\{r_1^y, \ldots, r_k^y\}$, where $r_i^y \in \{0,1\}^\lambda$ such that $y = \oplus_i r_i^y$. As in a Bloom filter, one then generates the k indices $i_1^y, \ldots, i_k^y$ by computing the hashes as $i_j^y = h_j(y)$ and truncating them by taking modulo m. Finally, at index i_j^y of the filter, the bit string r_j^y is stored. Collisions on two values y and y' for a certain hash function h_j are handled by choosing the same r_j for both the values.

To check if a given item is in GBF, one computes the truncated hashes and retrieves the shares stored at these indices in GBF. If the XOR of these shares is the same as the given item, then the item is in the filter, or else not. More details on the probability of collisions and the probability of false positives can be found in [12].

Private Membership Query using PBR. This protocol essentially follows the same principle as the one which combines PIR and a Bloom filter. The database owner now builds a GBF $(k, m, \mathcal{H}_k, \lambda)$ using the entries and a user queries the GBF instead of the database. Again k PBR invocations are required to retrieve the k random shares. This adapted protocol is private if the underlying PBR scheme is private, i.e., does not reveal the user's queries.

Remark 3. At this juncture, we have two solutions for private membership query to public database: 1) k invocations of single server PIR/PBR to Bloom filter/GBF, 2) Send the complete filter for a local query. On one hand, any PIR based solution only provides computational privacy, has a communication cost, the best being $\mathcal{O}(\log^2 m)$ and involves cryptographic computations and hence entails a significant time complexity. While on the other hand sending the filter ensures absolute privacy, but has a larger communication complexity m bits (still much better than the trivial PIR i.e., sending the initial database) but has a very low time complexity (has to invoke the protocol in Fig. 1 locally). Since the size of the database gets drastically reduced with Bloom filter, this solution provides a competitive alternative to trivial PIR even for low memory devices.

6 Practicality of the Solutions

We reiterate that a private membership query protocol provides an immediate solution for designing privacy-friendly alerting websites. For the sake of practicality, any realistic privacy-friendly alerting websites should provide response to a user's query in real time. It is hence highly important to evaluate the practicality of the underlying protocol.

We first discuss the practicality of the solutions based on PIR/PBR and Bloom filter in case of public database and in the sequel we discuss the practicality of PSI/PSI-CA protocol in case of private database.

Since Bloom filter is highly efficient in space and time, the practicality of PIR/PBR based protocol depends on the practicality of the underlying PIR/PBR scheme. Hence we first discuss its practicality as perceived in the literature and later by experimentally evaluating PIR/PBR protocols.

For experimental evaluation, the tests were performed on a 64-bit processor desktop computer powered by an Intel Xeon E5410 3520M processor at 2.33 GHz with 6 MB cache, 8 GB RAM and running 3.2.0-58-generic-pae Linux. We have used GCC 4.6.3 with `-O3` optimization flag.

6.1 Applicability of PIR

Sion and Carbunar [28] evaluate the performance of single database PIR scheme. The authors show that the deployment of non-trivial single server PIR protocols on real hardware of the recent past would have been orders of magnitude less time-efficient than trivially transferring the entire database. The study primarily considers the computational PIR protocol of [21]. The authors argue that a PIR is practical if and only if per-bit server side complexity is faster than a bit transfer. With a normal desktop machine, trivial transfer (at 10MBps) of the database is 35 times faster than PIR. This ultimately restricts the use of PIR protocols for low bandwidths (tens of KBps).

Olumofin and Goldberg [26] refute the general interpretation [28] that no PIR scheme can be more efficient that the trivial one. Authors evaluate two multi-server information-theoretic PIR schemes by Chor et al. [6] and by Goldberg [15] as well as a single-server lattice-based scheme by Aguilar-Melchor and Gaborit [23]. The later scheme is found to be an order of magnitude more efficient over the trivial scheme for situations that are most representative of today's average consumer Internet bandwidth. Specifically, for a database of size 16 GB, the trivial scheme outperforms the lattice based scheme only at speeds above 100 Mbps.

6.2 Experimental Analysis

We have implemented two PIR/PBR protocols: 1) Cachin et al. [3], which is the most efficient (in terms of communication) bit retrieval scheme 2) Aguilar-Melchor and Gaborit [23] (implemented in parig-gp[3]) which is the most computationally efficient PBR protocol. We have also implemented RSA-OPRF PSI protocol of De Cristofaro et al. [10] and PSI-CA protocol of De Cristofaro et al. [9]. The existing implementation[4] of protocol by Dong et al. [12] seems not to execute correctly. Even after correcting the initial compilation errors, the code seems not to be executing the protocol till the end. We hence do not consider it for our evaluation.

Table 3. Results for the leaked databases using SHA-1. Databases contain single data for a user, for instance Snapchat contains only username and ignores other auxiliary leaked information.

Database	**Size**	n	$-\log_2 f$	m (MB)	**Build time (mins)**	**Compress. ratio**
SNAPCHAT	49 MB	4609621	128	102	6	0.48
			64	52	2	0.94
			32	26	1	1.88
LINKEDIN	259 MB	6458019	128	142	10	1.82
			64	72	3	3.60
			32	36	1.5	7.19
ADOBE	3.3 GB	153004872	128	412	198	8.20
			64	206	72	16.4
			32	102	30	33.13

Public Database. The cost of using PIR-based schemes reduces to the cost of building the filter combined with the cost of k PIR invocations on the filter. We present the time required to build a Bloom filter for the leaked databases corresponding to SNAPCHAT, LINKEDIN and ADOBE in Table 3. The filter is constructed using SHA-1 which generates 20 bytes' digest.

From Table 3, we can observe that the filter size grows slowly and that the computational time of the filter is reasonable. Initially, all the computations are performed in a sequential manner. We have then distributed the computation on 4 computers (with similar characteristics). Parallelizing the creation of the Bloom filter is straightforward and we nearly achieved a $4\times$ speedup (50 mins). With a few computers, it is possible to reduce the computational time for creating the filter to a desired threshold. We further note that building a Bloom filter involves only a one-time cost.

Despite the space and time efficiency of Bloom filter, the huge cost of PIR invocation (using the existing primitives) makes such protocols impractical. The protocol [3] takes over 6 hours in case of Snapchat database for one invocation. If the probability of false positive is 2^{-32} i.e. $k \approx 32$, the estimated time for 32 PIR invocations is over 32×6 hours i.e. over 8 days. The PBR protocol [23], takes around 2 hours for 1 PBR invocation on Snapchat garbled Bloom filter. The security level considered here is of

[3] pari.math.u-bordeaux.fr/
[4] Available at bit.ly/1k75nu6

100 bits. However, considering the household network bandwidth of 10 Mbps, the time to download the filter would take 20 seconds. The time efficiency of the trivial PIR with Bloom filter seems unmatchable.

Private Database. Table 4 presents results obtained for the PSI protocol by De Cristofaro et al. [10] for 80 bits of security.

Table 4. Cost for PSI protocol [10] with 80 bits of security using SHA-1.

Database	Cost (mins)
SNAPCHAT	48
LINKEDIN	68
ADOBE	1600

Table 5. Cost for PSI-CA protocol [9] with 80 bits of security using SHA-1.

Database	Cost (mins)
SNAPCHAT	9
LINKEDIN	12
ADOBE	301

As the user's set has only one data, his computational cost is negligible. To be precise, a user's computational cost consists in computing a signature and n comparisons. The authors in [11] claim that the result of the server's computation over its own set can be re-used in multiple instances. Hence, the server's cost can be seen as a one-time cost, which further makes it highly practical.

Table 5 presents results obtained using PSI-CA protocol by De Cristofaro et al. [9]. Recommended parameters of $|p| = 1024$ and $|q| = 160$ bits have been chosen.

Clearly, PSI-CA outperforms PSI by a factor of 5. The reason behind this performance leap is that the exponents in modular exponentiations are only 160 bits long in PSI-CA as opposed to 1024 bits in PSI.

Table 6. Summary of the results on Snapchat with $f = 2^{-32}$.

		Cost	
Protocol	**Type**	**Commun.**	**Comput.**
Trivial PIR with Bloom filter	PIR	26 MB	1 min
Cachin et al. [3]	PIR	7.8 KB	$>$ 8 days
Melchor et al. [23]	PBR	12.6 TB	$>$ 2.5 days
De Cristofaro et al. [10]	PSI	562 MB	48 mins
De Cristofaro et al. [9]	PSI-CA	87.92 MB	9 mins

Table 6 summarizes the results obtained on Snapchat database for $f = 2^{-32}$. Clearly, in the public database case, sending the Bloom filter is the most computationally efficient solution. While, in the private database scenario, PSI-CA provides a promising solution. Comparing the two cases, we observe that the private database slows down the query time by a factor of 9.

We highlight that PSI/PSI-CA protocols perform much better than PIR/PBR protocols. This is counter-intuitive, as in case of PIR the database is public while in PSI the database is private. A protocol on private data should cost more than the one on public data. With a theoretical stand-point, there are two reasons why private set

intersection protocols perform better than PIR protocols: 1) the computational cost in PSI/PSI-CA protocols is reduced at the cost of communication overhead, 2) the size of the security parameter is independent of the size of the database. More precisely, the communication cost of the most efficient PSI/PSI-CA protocols [9, 10], [17], [12] is linear while the goal of PIR protocols is to achieve sub-linear or poly-logarithmic complexity. This indeed comes at a cost, for instance the size of RSA modulus in PSI [10] for 80 bits of security is 1024 bits and hence independent of the size of the sets involved. While in case of PIR [3], the size of the modulus used is $\log^{3-o(1)}(n)$ bits. Hence for a million bit database, the modulus to be considered is of around 8000 bits, which leads to a very high computational cost.

7 Conclusion

In this work, we examined websites alerting users about data leakage. With the current rate of leakage, these websites will be needed for a while. Unfortunately, it is currently difficult to determine whether or not these websites are phishing sites since they do not provide any privacy guarantee to users. Our work exposes the privacy risks associated to the most popular alerting websites. We further evaluate how state-of-the-art cryptographic primitives can be applied to make private query to an alerting site possible. Two different scenarios have been considered depending on whether the database is public. While PSI/PSI-CA protocols provide a straightforward solution in the private database scenario, a tweak using Bloom filter transforms PIR/PBR into private membership protocols for public database.

Our experimental evaluation shows that PSI/PSI-CA protocols perform much better than PIR/PBR based protocol. This is an encouraging result for the ethical hacking community or security companies. Yet the cost incurred by these ad hoc solutions is considerable and hence there remains the open problem of designing dedicated and more efficient solutions.

Acknowledgements

This research was conducted with the partial support of the Labex PERSYVAL-LAB(ANR–11-LABX-0025) and the project-team SCCyPhy.

References

1. Burton H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13, July 1970.
2. Dominique Bongard. De-anonymizing Users of French Political Forums. In *Passwords 2013*, 2013.
3. Christian Cachin, Silvio Micali, and Markus Stadler. Computationally Private Information Retrieval with Polylogarithmic Communication. In *Advances in Cryptology– EUROCRYPT '99*. Springer Berlin Heidelberg, 1999.
4. Yan-Cheng Chang. Single Database Private Information Retrieval with Logarithmic Communication. In *Information Security and Privacy*, volume 3108. Springer Berlin Heidelberg, 2004.
5. B. Chor, N. Gilboa, and M. Naor. Private information retrieval by keywords, 1998.

6. Benny Chor, Oded Goldreich, Eyal Kushilevitz, and Madhu Sudan. Private Information Retrieval. In *Annual Symposium on Foundations of Computer Science, FOCS 1995*, 1995.
7. Anupam Das, Joseph Bonneau, Matthew Caesar, Nikita Borisov, and Xiaofeng Wang. The Tangled Web of Password Reuse. In *Network and Distributed System Security Symposium, NDSS 2014*, 2014.
8. Xavier de Carné de Carnavalet and Mohammad Mannan. From Very Weak to Very Strong: Analyzing Password-Strength Meters. In *Network and Distributed System Security Symposium, NDSS 2014*, 2014.
9. Emiliano De Cristofaro, Paolo Gasti, and Gene Tsudik. Fast and private computation of cardinality of set intersection and union. In *Cryptology and Network Security*, Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2012.
10. Emiliano De Cristofaro and Gene Tsudik. Practical Private Set Intersection Protocols with Linear Complexity. In *Proceedings of the 14th International Conference on Financial Cryptography and Data Security*, 2010.
11. Emiliano De Cristofaro and Gene Tsudik. Experimenting with Fast Private Set Intersection. In *Trust and Trustworthy Computing*. Springer Berlin Heidelberg, 2012.
12. Changyu Dong, Liqun Chen, and Zikai Wen. When private set intersection meets big data: an efficient and scalable protocol. In *ACM Conference on Computer and Communications Security*, 2013.
13. MichaelJ. Freedman, Kobbi Nissim, and Benny Pinkas. Efficient Private Matching and Set Intersection. In *Advances in Cryptology - EUROCRYPT 2004*. Springer Berlin Heidelberg, 2004.
14. Craig Gentry and Zulfikar Ramzan. Single-Database Private Information Retrieval with Constant Communication Rate. In *ICALP*, 2005.
15. I. Goldberg. Improving the Robustness of Private Information Retrieval. In *Security and Privacy, 2007. S&P '07. IEEE Symposium on*, 2007.
16. Susan Hohenberger and StephenA. Weis. Honest-Verifier Private Disjointness Testing Without Random Oracles. In *Privacy Enhancing Technologies*, Lecture Notes in Computer Science, pages 277–294. Springer Berlin Heidelberg, 2006.
17. Yan Huang, David Evans, and Jonathan Katz. Private Set Intersection: Are Garbled Circuits Better than Custom Protocols? In *NDSS*, 2012.
18. Ari Juels and Ronald L. Rivest. Honeywords: making password-cracking detectable. In *ACM SIGSAC Conference on Computer and Communications Security, CCS'13*, 2013.
19. Lea Kissner and Dawn Song. Privacy-Preserving Set Operations. In *Advances in Cryptology – CRYPTO 2005*. Springer Berlin Heidelberg, 2005.
20. Georgios Kontaxis, Elias Athanasopoulos, Georgios Portokalidis, and Angelos D. Keromytis. SAuth: protecting user accounts from password database leaks. In *ACM SIGSAC Conference on Computer and Communications Security, CCS'13*, 2013.
21. E. Kushilevitz and R. Ostrovsky. Replication is Not Needed: Single Database, Computationally-private Information Retrieval. In *Proceedings of the 38th Annual Symposium on Foundations of Computer Science*, 1997.
22. Helger Lipmaa. An Oblivious Transfer Protocol with Log-Squared Communication. In *ISC*, 2005.
23. C.A. Melchor and P. Gaborit. A fast private information retrieval protocol. In *Information Theory, 2008. ISIT 2008. IEEE International Symposium on*, 2008.
24. Michael Mitzenmacher. Compressed bloom filters. In *ACM Symposium on Principles of Distributed Computing - PODC 2001*, 2001.
25. Arvind Narayanan and Vitaly Shmatikov. Fast dictionary attacks on passwords using time-space tradeoff. In *ACM Conference on Computer and Communications Security, CCS 2005*, 2005.
26. Femi Olumofin and Ian Goldberg. Revisiting the Computational Practicality of Private Information Retrieval. In *Financial Cryptography and Data Security*. Springer Berlin Heidelberg, 2012.

27. Bryan Parno, Jonathan M. McCune, Dan Wendlandt, David G. Andersen, and Adrian Perrig. CLAMP: Practical Prevention of Large-Scale Data Leaks. In *IEEE Symposium on Security and Privacy - S&P 2009*, 2009.
28. Radu Sion and Bogdan Carbunar. On the Practicality of Private Information Retrieval. In *NDSS*, 2007.
29. Jaideep Vaidya and Chris Clifton. Secure set intersection cardinality with application to association rule mining. *J. Comput. Secur.*, 13(4):593–622, July 2005.