April 4, 2019

MASTER THESIS

PRIVACY PRESERVING MATCHING USING BLOOM FILTERS: AN ANALYSIS AND AN ENCRYPTED VARIANT

David Stritzl

Faculty of Electrical Engineering, Mathematics and Computer Science (EEMCS)
Services and Cybersecurity Group (SCS)

Exam committee:
Dr. Andreas Peter
Dr. Maarten H. Everts

UNIVERSITY OF TWENTE.

Privacy-Preserving Matching Using Bloom Filters: An Analysis and an Encrypted Variant

David Stritzl
University of Twente, The Netherlands
d.l.stritzl@student.utwente.nl

Abstract. Record lookup schemes are utilised in distributed database systems to allow clients to efficiently identify databases that contain relevant information. In untrusted environments, where a client should not be able to learn about the database entries that are not explicitly queried, probabilistic record lookup schemes can be used to provide a certain level of privacy. In this paper, we provide a framework for evaluating different probabilistic record lookup schemes in terms of privacy, efficiency and utility. For privacy in such schemes, for the first time, we present an analysis of the privacy implications of additions and removals of database records. Using this framework, we furthermore analyse a Bloom filter-based record lookup scheme, for which we show that it can be privacy-preserving depending on the use case and the configuration of the scheme. However, updates of databases records in this scheme, can introduce a significant privacy impact. Finally, we provide an efficient interactive record lookup protocol using homomorphic encryption that reduces the impact on privacy in the case of database updates.

Keywords: record lookup schemes · Bloom filters · homomorphic encryption

Contents

1 Introduction

While network database systems have been around for the longest part of modern computing history, the introduction of the internet has made it possible for parties within a company or organization to access those systems from virtually anywhere. However, when it comes to inter-organisational collaborations, the process of information sharing becomes more involved as many databases contain company-internal or privacy-sensitive data. Examples of this can be found in the public sector in, among others, health-care and law enforcement institutions, where a substantial amount of the handled information is privacy-sensitive. Various solutions involving centralised databases, managed by a (trusted) third-party, have been criticised due to security flaws [1, 2, 3]. Nonetheless, with medical care becoming more advanced and crime investigation cases growing more complex due to the increasing involvement of digital and international aspects, manual inquiries of can take significant work, therefore inter-organisational data exchange is becoming more important for such institutions.

An alternative to building centralised databases is to keep database systems decentralised and each organisation keeps their own (privacy-sensitive) data. In this case, other parties (manually) have to send an inquiry for certain data, which allows an organisation to handle access authorization for that data internally. However, this introduces the problem that party needs to know where to inquiry. Simply publishing a complete list of (unique) data record attributes, for instance social security numbers, is not only inefficient, it also compromises privacy if identifiable information is involved.

A commonly used approach for record lookup in distributed database systems [4] is built upon Bloom filters [5], which is an efficient set data structure for membership testing. As a side effect of the efficient representation, Bloom filters queries can result in false positives, thereby potentially matching elements that were not explicitly encoded into the filter. In the record lookup scheme, the different database providers can encode a selected of attributes of their data set and share this with a client, who can then use it to inquiry if a database contains some selected data record. Here, false positives can introduce some overhead as some database will be wrongly queried. However, depending on the configuration of a Bloom filter, a trade-off can be made between the false positive probability of a query and the time and space efficiency. Furthermore, utilising the false positive probability, a Bloom filter can provide a certain level of privacy, as more false positives will make it more difficult to discern them from real elements.

In this paper, we consider a record lookup scheme similar to that proposed by Little et al. [4], where all filters are stored on the client-side. For this purpose, we define a set of metrics to evaluate different aspects of probabilistic record lookup schemes as, for instance, Bloom filter-based schemes. Using these metrics, we analyse the effect of different Bloom filter configurations on the privacy, utility and efficiency of the scheme. Lastly, in order to reduce potential leakage when querying, we introduce an efficient interactive variant of this scheme using homomorphic encryption.

2 Background

2.1 Bloom Filters

As mentioned above, a Bloom filter is a probabilistic data structure for highly efficient set membership querying [5]. The data structure is highly configurable, allowing for a trade-off between accuracy and efficiency of the querying. Common applications of Bloom filters are found in networking for efficient data caching [6] and packet routing [7]. Furthermore, several extensions of Bloom filters exist, including counting filters [8], where a count is stored instead of a bit for every filter position, thus allowing for the removal of elements from the filter, and scalable filter setups [9] consisting of multiple filters, where additional filters are used if a certain false positive threshold has been reached.

Regular Bloom filters consist of a bit vector of length m and a set of k independent hash functions mapping data elements to bit positions in that vector.

For the insertion of an element of a data set into the Bloom Filter, the element is hashed by all k independent hash functions and the resulting bit positions are set.

Similarly, for the querying of an element, the element is hashed by all independent hash functions and then checked if each resulting bit position is set. If at least one of the bit positions is not set, then the element is guaranteed to be non-existent in the data set. However, in the case that all bit positions for a member check are set, it is not certain that the checked element is part of the set, as one or more bit positions could have been set by other elements in set. Therefore, for a positive query result, there is a certain probability that the match was a false positive, i.e. there is a match although the element is not in the data set. The false positive probability of a filter depends on its size, the number of hash functions used and the number of elements inserted into the filter.

More formally, for a given universe $\mathbb{U}$ and a data set $S \subseteq \mathbb{U}$, a set of k independent hash functions $\mathsf{hs} = \{h_1, h_2, ..., h_k\}$, where $h_i \colon \mathbb{U} \to \mathbb{Z}_m$, we can define a Bloom filter $\mathcal{BF}(m, \mathsf{hs}, S) \colon (\mathbb{Z}^+, \{\mathbb{U} \to \mathbb{Z}_m\}^k, \{\mathbb{U}\}^*) \to \{\mathbb{Z}_m\}^*$ of size m as follows:

$$\mathcal{BF}(m, \mathsf{hs}, S) := \{h(\mathsf{elem}) \mid h \in \mathsf{hs}, \mathsf{elem} \in S\}$$

For querying elements, we can define the indicator function $\mathbb{1}_{\mathcal{BF}} \colon \mathcal{P}(\mathbb{Z}_m) \to \{0,1\}$ as follows:

$$\mathbb{1}_{\mathcal{BF}} := \begin{cases} 1 & \text{if } \forall h_i \in \mathsf{hs} : h_i(\mathsf{elem}) \in \mathcal{BF} \\ 0 & \text{otherwise} \end{cases}$$

False Positive Rate

As previously mentioned, there is a certain probability that member query results a false positive. Given a Bloom filter of length m and k hash functions, the probability that a bit position is still unset after the insertion of n elements can be computed as follows [10]:

$$p_{\text{unset}} = (1 - 1/m)^{kn} \tag{1}$$

Using the probability of a bit position being unset, the formula for computing the false positive rate of the whole filter can be derived [10]:

$$p = (1 - p_{\text{unset}})^k = \left(1 - (1 - 1/m)^{kn}\right)^k \tag{2}$$

However, eq. (2) is not completely correct in certain edge cases [11]. For smaller filters with $m < 1024$ or filters with a higher m/n ratio, eq. (2) can induce a significant relative error. However, for the purpose of record linkage, the filters that are used in this paper are at least 2048 bits long and have a m/n ratio of at most 64, in which case the absolute error is $\ll 1\,\%$. Furthermore, as the correct computation given by Christensen et al. [11] is significantly more complex, we will therefore use the approximation in eq. (2) for this paper.

Next, the formula for the Bloom filter false positive rate further be simplified [12] as follows:

$$p \approx \left(1 - e^{-kn/m}\right)^k \tag{3}$$

This approximation can then be used to find the optimal number of hash functions k, where the false positive rate is minimal. For this purpose, eq. (3) can be derived over k [6], resulting in the following equality for an optimal

$$k = \frac{n}{m} \ln 2. \tag{4}$$

By substituting k in eq. (3) with this equality, a function for the optimal filter size m, where m is minimal, can be deduced [6],

$$m = \frac{n \ln p}{(\ln 2)^2}, \tag{5}$$

where n is the targeted number of elements for the filter, and p is the targeted false positive rate.

3 Related Work

Distributed Record Lookup

There are various reasons for distributing a database across different systems. For one, it allows for more simultaneous connections by distributing load over different systems. This redundancy also has as an effect that downtime or failure of a single database will not make all data unavailable. Also, with the growing use of data aggregation solutions and other big data applications, it can be simply infeasible for single systems to handle such amounts of data. Furthermore, in the case of multiple different parties providing databases, each party can manage access to their own data sets, thereby allowing more controlable privacy. However, distributed database systems introduce the problem that a client has to know where to find certain information, as querying each separate database is not efficient and might therefore not be feasible.

A solution to this problem is the use database query routers [4, 13, 14], where a client sends a query to some centralized router that keeps track of all the items in the various database systems. Such solutions often requires a single trusted party that has access to sufficient information to be able to create a routing table for the different databases. However, in certain applications such as, for instance, hospital systems storing medical records, such a trusted party might not be available. Moreover, for systems with a large amount of different databases as, for instance, in peer-to-peer networks, it might become infeasible for a single router to keep track of all records available. For most table based record lookup schemes it is, however, possible to store the lookup table at the client, thereby removing the need for trusted party and allowing each separate database server to provide a lookup table to specific clients only.

In [4], Little et al. provide an efficient record lookup scheme based on Bloom filters, where each database server provides a Bloom filter of some attribute of all records. In order to perform a query, a client can then check all local filters for a match, and then issue a query at the corresponding database server. While querying a Bloom filter can produce false positives, the probability is tunable to reach some optimum between filter storage size and average lookup time.

Private Record Linkage

A similar notion to record lookup is record linkage where, for the purpose of combining data from different databases, records in these separate databases have to be linked using some kind of identifier. For databases managed by a single entity, uniquely generated identifiers are frequently used. However, for databases managed by different entities this may not be an option.

One common type of identifier for personal records is the name of a person, however sharing personal data

with other parties, such as names is often undesirable due to privacy concerns. In [15], Quantin et al. provide a private record linking scheme for medical records based on hashing the identifiers, therefore limiting the personal data that is leaked during record linkage.

Furthermore, private record linkage schemes can also be constructed using Bloom filters. In [16], Schnell et al. introduces a method for fuzzy private record linking on names using bi-grams of names stored in Bloom filters is described. However, statistical attacks for leaking names in bi-gram and n-gram based linkage schemes using frequency analysis have been devised [17, 18, 19].

A more general attack on Bloom filters is devised by Alaggan et al., where all elements in a application domain are tried, thereby generating a set of all possible matches including false positives [20].

Bloom Filter Privacy Metrics

In order to evaluate the privacy provided by different Bloom filter-based schemes, a metric for privacy has to be defined.

In [21], Bianchi et al. adapt the privacy notion of k-anonimity [22] to Bloom filters. Here, a filter is defined k-anonymous if at least $k-1$ false positive elements exist that map to each set bit position of the filter.

Furthermore, Bianchi et al. describe the notion of deniability: an element in a filter is deniable if the bit positions that are set for that element could be explained by bit positions that could be set by elements not existing in the filter, i.e. false positive elements. A filter is then called γ-deniable if every element from the source data set can be denied with some probability γ.

Also, Bianchi et al. propose the anonymisation of Bloom filters by settings specific bits, such that possible number of false positives explaining the sets bit for some elements is increased, thereby increasing the γ-deniability.

Lastly, in similar approach, Alaggan et al. adapt the concept of ϵ-differential privacy to Bloom filters by introducing a filter pertubation mechanism through flipping bits in the filter [20]. When adding a single element to a Bloom filter, using this notion, it can be guaranteed that the probability of generating a specific perturbated bit vector due to the addition of that is at least e^{ϵ} times lower than the probability of generating the same bit vector when adding some other element. The authors furthermore define a Bloom filter utility metric based on the recall when querying a filter.

Multiple Bloom Filter Setup

As databases in distributed systems change over time, Bloom filters user for record lookup have to be updated. However, adding elements to an existing filter will increase the false positive rate, potentially more than is desirable for certain use cases. On the other hand, creating new versions of Bloom filters may not be desired, either, due to efficiency or privacy concerns, as will be discussed in section 6.

One method dealing with new elements without recreating the complete filter is to create a separate filter with only the new elements. Record lookup can then be done by checking all filters for matches: when there is at least one positive result, the element could be in the data set. However, as more filters are combined in such a setup, the false positive rate will increase significantly and the efficiency will be lowered.

In [9], Almeida et al. introduce another multiple Bloom filter scheme, where the overall maximum false positive rate can bounded by increasing the size and hash count of each new filter. However, this means that each newer filter has an increasingly lower false positive rate, thereby raising potential privacy concerns for later created filters.

There exist other schemes employing multiple Bloom filters, which utilise different filters to increase the overall space efficiency of the setup. For instance, in [23], Tabataba and Hashemi present a set querying scheme using two Bloom filters. In this scheme, all elements are stored in both Bloom filters, but with a independent set of hash functions for each. When a member check is performed, both filters are queried, thereby decreasing the overall false positive rate of querying when compared to a similarly sized single-filter scheme.

Furthermore, in [24], Lim et al. provide a set querying scheme using a main Bloom filter and two cross-checking filters. Each cross-checking filter contains one disjunct part of the elements encoded in the main filter. During a member query, the main filter is queried first, and only if the returned result is positive, the other two cross-checking filters are tried. Here, Lim et al. show that cross-checking filter schemes are significantly more space-efficient than single or dual Bloom filter schemes for large filter sizes.

However, as Bloom filters are most commonly used for the purposes of caching or lookup in high performance applications where privacy might not be the main concern, the privacy implications of systems incorporating multiple Bloom filter as, for instance, with record lookup schemes, have not been studied in depth yet.

4 Contributions

While Bloom filters have been used in various applications such as caching in the last few decades, the privacy implications of different Bloom filter configurations have not been studied widely yet. In this paper, we will consider a client-side Bloom filter-based record lookup scheme similar to the scheme provided by Little et al. [4].

For this purpose, we first provide a set of metrics for

the evaluation of different probabilistic (private) record lookup schemes that allow for false positives. Since we mainly focus on the privacy aspects of such schemes, we will also provide metrics for the efficiency and utility aspects of different schemes. For privacy, we devise a new metric based on the false positive rate of a record lookup scheme, since privacy notions like k-anonymity and ϵ-differential privacy, as presented in [21, 20] for Bloom filters, are not trivially adaptable to the multiple systems scenario of record lookup schemes. For the purpose of evaluating the utility, however, we opted for a similar metric as the one proposed [20]. While efficiency studies have been performed for most record lookup schemes, we provide our own evaluation in order to analyse potential trade-offs of different schemes and configurations.

Furthermore, using these metrics we will investigate the use of the afore-mentioned Bloom filter-based record lookup scheme by Little et al. in privacy sensitive contexts. In contrast to existing work analysing privacy aspects of Bloom filter, we also focus on the privacy implications of systems incorporating multiple Bloom filters in comparison to systems using only a single filter, as well as the impact of two different filter configuration strategies in the multiple filter case.

Finally, we present a novel interactive record lookup protocol using homomorphic encryption based on the scheme by Little et al. with the aim of improving the privacy measures of the scheme. For this purpose, we build upon on the Bloom filter-based private set intersection protocol introduced Davidson and Cid [25], which is further discussed in section 7.1. Here, we also provide a comparison of our protocol to the initial approach using the above-mentioned metrics.

5 System Model

In this section, we provide some notions regarding Bloom filter-based record lookup schemes similar to that presented in [4]. Furthermore, we will describe our approach for analysing such schemes and extensions thereof.

For the Bloom filter record lookup case, we can distinguish two different parties: the *provider* of a data set, who wants to share data set and provides a Bloom filter for lookup to other parties, and the *consumer* of said data set, who uses a Bloom filter created by some other party. Once a provider has created a Bloom filter from a selected data set, and has distributed it to other parties. These other parties can then query the filter to check if the provider has some data related to the query. In case there is a match, the consumer can invoke (manual) information exchange procedures. If a match is a false positive, these procedures will then reveal this fact and no data exchange will take place. Depending on the configuration of the Bloom filter and its intended use cases, more false positive are generated, therefore requiring more (manual) involvement. On the other hand, lowering the false positive rate will lower the level of privacy provided.

In order to compare different setups of the Bloom filter record lookup scheme, we will define metrics for the evaluation of different aspects in the next section.

5.1 Metrics

In order to compare different approaches to privacy-preserving record lookup schemes, we define metrics for the different aspects of such schemes. For this purpose, we have identified our main concerns with different schemes to be the privacy, utility, efficiency and functionality provided by such scenes.

5.1.1 Privacy

Furthermore, we define a metric for the privacy of a given scheme. In order to measure the privacy provided by a statistical scheme such as Bloom filters, we consider how much information can be gained by an adversary in different scenarios.

For this purpose, we consider three attack scenarios based with differing levels of adversary knowledge, in all of which the adversary does not know the actual entries of the data set being stored.

Agnostic Outsider
In this scenario, an adversary has absolutely no knowledge about the algorithms, data structures, the format of the data and the secrets used in the scheme.

This can be the case if, for instance, some data is accidentally leaked by an insider then an outside adversary captures the data without knowing what it is. In this case, the data must not give give anything away about its own structure.

However, as data almost always comes with some context and metadata, e.g. the origin of the data or a file name, an adversary can try to learn more about the scheme using, for instance, public information or reverse engineering, therefore the practicality of this scenario is limited.

Outsider
The attacker knows the algorithms, data structures and the format of the data used in the scheme, but does not have any knowledge about any secrets being used.

This scenario comes closest to Kerckhoffs' principle [26], where Kerckhoffs states that a system must be secure, even if an adversary knowns everything about the system except for the secrets.

It is most likely in case of an external adversary attacking system and networks that are part of the

scheme, for instance by exfiltrating data from an employee laptop. If the adversary has access to such system, it is not unlikely that documentation, applications or at least metadata related to the data structure and algorithms can be found by the adversary as well.

Insider

Considering probabilistic schemes where privacy is not fully compromised if all shared secrets are known to the attacker, we define an additional scenario based on the regular *Outsider* scenario, where the shared secrets are known as well. In this case, the attacker knows the algorithms, data structures, the format of the data and the shared secrets used in the scheme, but not the entries of the data set.

This scenario is most likely in the case of an internal adversary, as for instance a curious or a disgruntled employee, or powerful external adversary attack a wide range of systems and networks.

Brute-Force Attack Precision

A possible attack type is a brute-force attack as has been described in [17, 20], where elements are taken from some data set or generated randomly, and then matched using a given scheme. In the context of one of the above scenarios, we measure the privacy using the precision of a given attack on a scheme, i.e. the ratio of the positive matches versus all matched elements:

$$\text{Privacy} = \frac{|\text{TP}|}{|\text{TP}| + |\text{FP}|}$$

As this metric only considers positive matches, attacks on a given scheme can still be infeasible if the probability of a false positive is infinitesimal. For this purpose, we define the binary events A and B, representing the event that a query return a positive result for some element z and the event that the queried z exists in the original data set S, respectively.

Using the events, we can redefine the privacy metric from above as follows:

$$\text{Privacy} = P(B \mid A) \tag{6}$$

For non-stochastic schemes, it is given that $P(B \mid A) = 1$, as there are no false positives. For stochastic schemes, this probability can be rewritten, using Bayes' theorem, as follows:

$$P(B \mid A) = \frac{P(A \mid B)P(B)}{P(A)} \tag{7}$$

Here, the probabilities $P(A)$ and $P(A \mid B)$ are dependent on the scheme and will therefore be covered in later sections.

Furthermore, we can also define an absolute version of the privacy metric eq. (6) that also takes negative matches into account:

$$\text{Privacy}_{\text{abs}} = P(B \mid A)P(A) \tag{8}$$
$$= P(A, B) \tag{9}$$

The absolute metric describes the probability of randomly correctly brute forcing a single element. This can, for instance, be used for estimating how many queries will have to be run to produce a single, correct positive match. The main difference from the first metric is that we now consider negatives results as well, as $P(A)$ is the probability of positive query result.

However, for schemes that do not allow for false negatives, i.e. $P(A \mid B) = 1$, as in the case of Bloom filters, we can simply disregard any negative that are produced while querying, as we can guarantee that all negatives were not included in the original data set. Therefore a lower $P(A)$ only affects the runtime of an attack, and not directly the privacy. Nonetheless, in case that the probability of producing some positive result $P(A)$ is infinitesimal, a brute-force attack as described above could become computationally infeasible for an adversary.

However, as $P(A)$ is not only highly dependent on kind and size of data set, but also on the knowledge of an adversary, as will be shown in later sections, we will not consider this variant of the metric for the rest of the paper.

Adversary Data Set

In order to reason about the attack precision, we need to consider the probability $P(B)$ depicting the probability that some queried element z is in the source data set S. This probability depends on both the source data set being used and the knowledge of the querying party which is, in the case of a brute-force attack, the adversary. For this purpose, we will assume that an adversary has some data set used for a brute-force attack denoted by the set S_{adv}, or some algorithm generating said data set.

The size of this data set S_{adv} depends on the kind of data in S, and the knowledge and assumptions of an adversary. For instance, in the case of credit card numbers, the total universe of credit cards is finite and relatively small, so it is trivial for an adversary to create a set S_{adv} for a brute-force attack of this data set. While the total universe of names is limited in practice, such data might not be readily available, and as the theoretical universe of possible names is infinite, it might be infeasible for an adversary to create some data set S_{adv}.

In the latter case, an adversary can try to limit the universe of possibilities by using their knowledge and assumptions. For instance, given some scheme using a data set with dates, if the adversary does not know that a given scheme uses dates, the size of S_{adv} is very

possibly infinite. However, in the case that an adversary does have this information, the size of this data set can severely reduced. While the set of all possible dates is still infinite, an adversary can limit the options using context specific assumptions and common sense, as, for instance, many information systems will only consider events from the last few decades.

Indeed, it is possible that knowledge and assumptions lead to elements not being brute-forceable in case $S \not\subseteq S_{\text{adv}}$. In cases where no feasible data sets or generators of such exist, for example for schemes using names of people, an adversary can only use information that is available to them. Furthermore, in cases where a data set S_{adv} is too large for a brute-force attack, thereby generating a lot of false positives, an adversary might choose to limit this data set by making certain assumption. For instance, in the case of birth dates, it could be beneficial for an adversary to limit such a data set to at most 90 years old, as most the number of people older than that is significantly lower than other age categories. When considering knowledge, another option would be the usage of demographical statistics and other available data sources, such as lists of actual people from, for example, social media accounts.

In the case the sets S and S_{adv} are known, we can write the probability that a random queried element z exists in S as follows:

$$P(B) = P(z \in S) = |S \cap S_{\text{adv}}|/|S_{\text{adv}}|$$

However, if S_{adv} is a superset of S, then $S \cap S_{\text{adv}} = S$, so $P(B) = |S|/|S_{\text{adv}}|$. In this case, we only need the sizes of the sets to compute $P(B)$, therefore, for simplicity, we will assume that S is a subset of S_{adv}, i.e. $S \subseteq S_{\text{adv}}$, for the rest of this paper.

As no adversary will have the same knowledge and skill, it is impossible to make general assumptions about the size of S_{adv}. However, when assessing the risk of some setup for a specific use case, one could define multiple scenarios for different adversary types, e.g. insider/outside attacker or low-/medium-/high-skilled. For each scenario, one could enumerate likely skill levels and sources of knowledge, e.g. email-addresses from public data leaks. Using this, the size of a plausible adversary data set $\tilde{S}_{adv.}$ can then be approximated. Based on the risk-appetite of the data provider, a scheme should be chosen in consideration of the assessed scenarios. For the evaluations in this paper, however, we will make no assumptions about the adversary's skill and knowledge. Using the notion of binary information entropy, we can denote the approximate size of a universe. In our analysis, we will then define the size of a set as $|S_{\text{adv}}| = 2^H$, where H is some binary entropy.

5.1.2 Utility

First, we define a utility metric representing the usefulness of a given scheme. While there are many possible utility aspects to be considered, we will only use the correctness of the matches returned by a lookup scheme, since this can be efficiently measured as opposed to, for instance, the experience of the user of the system. Therefore, this metric will mostly be relevant for probabilistic schemes allowing for false positives, since most other schemes should not result in any false positives or false negatives. For record lookup schemes, false positives can result in unnecessary database queries or manual work, therefore we will use the false positive rate of a scheme in our utility metric as opposed to the utility metric in [20], which uses query recall and therefore does not consider false positives. For this purpose, we define the utility metric as follows:

$$\text{Utility} = \frac{|\text{FP}|}{|\text{FP}| + |\text{TN}|}, \tag{10}$$

which can be easily evaluated for a wide range of probabilistic schemes.

5.1.3 Efficiency

Next, we determine metrics for the efficiency of a record lookup scheme. The efficiency of such a scheme depends on the storage, computation and communication costs of the scheme. Also, we differentiate the efficiency for each involved party, i.e. the provider and consumers of the shared data. Lastly, we divide the efficiency of each scheme into two life cycle phases: the efficiency of the initial setup and the efficiency of as single membership query.

For the case of private record lookup, we will consider the computation and communication cost of a single query, as this will most likely be a limiting factor on the usage of a scheme. However, depending on the use case, other efficiency factors should be taken in consideration as well.

In order to determine the computational efficiency, we will measure the time T_{query} that is required for a single query empirically using a benchmarking setup. While the typical computational complexity classes may be more universal, vastly different schemes cannot be compared effectively using this method.

For the communication cost, however, we will consider the network transfer size as the time spent on network transfers will depend on the network latency which can greatly differ per environment. For this purpose, we will analyse the transaction size of the setup phase L_{setup} and the transaction size for a single query L_{query}.

5.1.4 Functionality

Lastly, we define a category for additional functionality that can be provided by a potential scheme, for instance, regarding the management of the provided data. Below we will describe the functionalities we will consider in this paper.

Secure element addition Does the scheme allow for new elements to be added to a shared data set over time without (negatively) affecting the privacy of the scheme?

Guaranteed element deletion Does the scheme allow for the deletion of elements from the shared data and can the deletion of an element be guaranteed? Naturally, this only applies to data elements that have not been explicitly queried by the data consumer yet.

6 Bloom Filters

As mentioned in section 5, Bloom filters are used in the record lookup schemes similar to Little et al. [4].

Since Bloom filters can be setup in various configurations using different data sets, we will analyse the effect of different Bloom filter configurations on the metrics as defined in section 5.1. For each filter configuration, the optimal size and number of hash functions is used such that the false positive rate is minimal.

Finally, the different Bloom filter configurations will be evaluated in the context of the technology. For this purpose, we created a tool in C++ to empirically evaluate the efficiency and privacy metrics for different filter configurations. We plan to release this tool as an open-source project in the near future.

6.1 Privacy Analysis

In section 5.1.1, we defined the privacy metric as the probability of brute-forcing an element of the original data given some Bloom filter. For this purpose, we define three different attack scenarios based on the adversary's situation and knowledge.

6.1.1 Agnostic Outsider

In the first scenario, an adversary has absolutely no knowledge about the data structures, algorithms, any secrets and data used in the filter. In this scenario, while it might be theoretically possible for a knowledgable adversary to deduce with some certainty that a binary blob might be a Bloom filter, especially if given multiple filters, the parameters of that filter are still unknown. Considering the size of the combined parameter space of the hash type, hash size and any secrets, it is practically impossible to gather any information in this situation.

However, as was mentioned, an adversary could try to learn about the data structures and algorithm using metadata, for instance file names, and public information, as the source and context of a leak is often known.

6.1.2 Outsider

Next, in the second scenario, the attacker knows the data structures and algorithms used, but has no information about the shared secrets and data.

However, following the definition of this scenario, the adversary should not know anything about the secrets or data, therefore this scenario requires that this information is not be present or at least not trivially accessible to users of such systems.

In the case of an active adversary intercepting network traffic, the adversary could, even without knowing the contents of filter, perform bit-flips on the filter before forwarding it to the authorised consumer, in order to increase the false positive rate or to prevent records from being matched. However, this kind of attack can simply be mitigated using by signing every filter before distribution, therefore allowing the consumer to detect changes to a filter.

Next, a passive adversary could try to learn from a (leaked) Bloom filter using statistical brute-force attacks, as mentioned in [17, 20]. However, in the case an adversary does not know the format of the records in the filter, no significant leakage is possible outside of estimates about the number of elements in the filter. Therefore, we can define a simple Bloom filter extension adding a secret to the records, thereby limiting the privacy impact of such leaks, as long as the secret is unknown. For this purpose, we define a new hash function $h'(x, s)$, where a fixed length secret s is appended to message m before hashing it:

$$h'(m, s) := h(m \| s)$$

Here, a simple brute force attack on the secret will not be feasible as it is not possible for the attacker to discern the correct secret and data from all incorrect secrets and data.

While this property holds for one time secrets and long time secrets, which are used for multiple filters, all filters using the same secret will be at risk if the secret is leaked or found using, for instance, the known-record attack described in the next section.

Known-Record Attack
In the case that an adversary knows or guesses some elements stored in a filter, based on context, metadata and publicly known information, the adversary can perform a known-record attack on the secret. A known-record attack, in this case, works similar to regular brute force

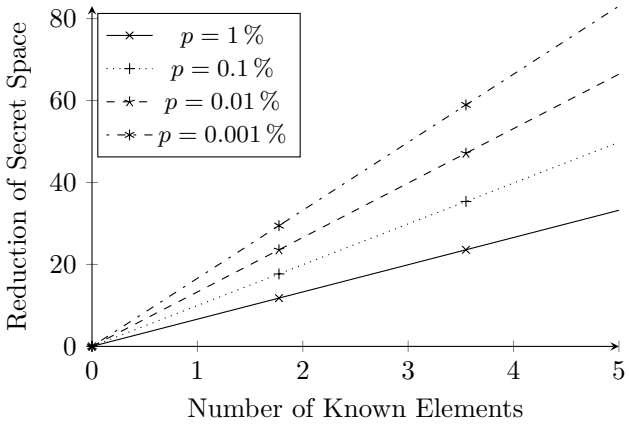


Figure 1: Secret space reduction for a brute force attack on the secret of a Bloom filter for different false positive rates.

attacks, however the attacker can discard all possible secrets that do not result in positive matches for the known elements.

For a known-record attack, we try to find secret for which all known elements must result in positive matches. In case of an incorrect secret, no query can result in true positives in case a secure hash function is used, therefore the matches of all known elements must result in false positives. We can compute the false positive probability p_{known} for q known elements using the false positive rate p as follows:

$$p_{\text{known}} = p^q$$

Assuming that the secrets are distributed uniformly, we can compute the entropy of the secret space $\mathcal{S}$:

$$H(\mathcal{S}) = \log_2(|\mathcal{S}|)$$

Furthermore, we can compute the reduction of the secret space $\Delta H(\mathcal{S})$ using known elements as follows:

$$\Delta H(\mathcal{S}) = \log_2(p^{-q})$$

This will result in an effective secret space $H'(\mathcal{S})$:

$$H'(\mathcal{S}) = H(\mathcal{S}) - \Delta H(\mathcal{S})$$

In fig. 1, the secret space reduction is shown for different false positive rates p. Here, it can be seen that a few known elements can significantly reduce the secret space for positive matches.

However, brute force attacks on larger keys have a significant runtime as the entropy of the secret space $\mathcal{S}$ does not change given some known elements. As a reference, a single-threaded brute-force attack using our tool on a 10 000 element filter using SHA3 hashes with a 0.01 % false positive rate and a 32-bit secret has a worst-case runtime of over 8 hours using our benchmarking setup mentioned in section 6.3. The same attack on a 64-bit secret will have an approximated worst-case runtime of over 4 million years.

Therefore, this kind of attack can be easily mitigated by using a sufficiently large secret space such that it is infeasible to perform a brute force attack considering the average runtime of such an attack.

Other types of cryptographic hash functions, as for instance SHA1 or MD5 could be used as well, since they can be considerably faster and a, in most cases inconsiderably, higher than average collision rate will lead to more false positives, and therefore increased privacy, and lowered utility and efficiency. In this case, attacks like the hash length extension attack on Merkle–Damgård hash functions, such as MD5, SHA1 or SHA2, where an adversary can extend a hash with additional input without knowing the original input including the secret, are not relevant for privacy. It is infeasible to retrieve a complete hash from a Bloom filter given that the secret and input are unknown, and an adversary would, furthermore, not gain anything from extending an element hash.

The use of non-cryptographic hash functions like MurmurHash3 may lead to potential statistical attacks, therefore having adverse effect on the privacy of a Bloom filter. However, as these attacks would depend on specific hash functions, such use cases are out of scope of this paper and will, therefore, not be considered.

6.1.3 Insider

In the last scenario, the adversary knows all the data structures, algorithms and shared secrets, but has no or only limited knowledge about the data being stored.

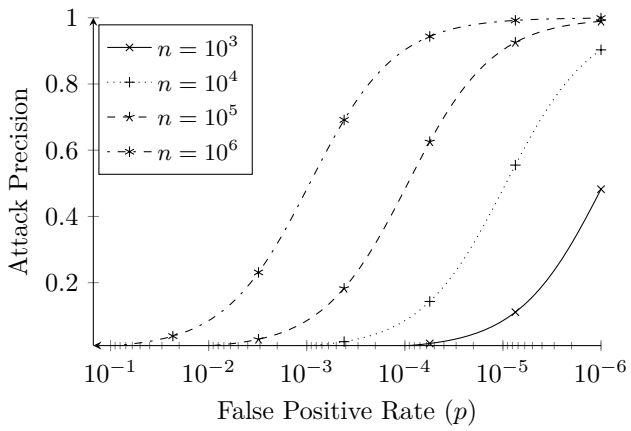
For this case, we can define a brute force attack on Bloom filters, where we query the filter with randomly generated elements. This type of attack will be discussed in the next few sections.

Single Bloom Filter In a single Bloom filter setup, only a single Bloom filter is created from a data set S. This data set S or parts of it cannot be reused in any other Bloom filter, as we would then deal with a dual or multi filter setup, therefore the definitions in this section are only valid for that case.

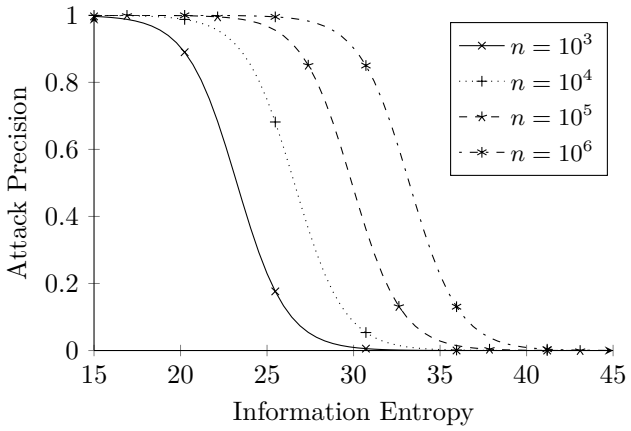
First, we redefine the following binary events from section 5.1.1, given an element z:

$$A := z \in \mathcal{BF}(m, \mathsf{hs}, S)$$
$$B := z \in S$$

In order to measure the possible information leakage for a single filter setup, we can define the precision of a querying strategy, i.e. the probability that a positive match is a actually true positive. Given eq. (6) and



(a) Attack precision for different false positive rates given a total adversary data set entropy of 30 bits.



(b) Attack precision for different data entropies given a number of elements n with a false positive rate of 0.0001.

Figure 2: Single Bloom filter brute force attack precision for different filter and data set parameters.

eq. (7):

$$\begin{aligned}\text{Precision} &= P(B \mid A) \\ &= \frac{P(A \mid B)P(B)}{P(A)}\end{aligned}$$

where

$$P(A \mid B) = 1 \tag{11}$$

as Bloom filter queries cannot produce false negatives. Next, we can rewrite the probability that an element is in the Bloom filter as follows, using Bayes' theorem:

$$P(A) = P(A \mid B)P(B) + P(A \mid \overline{B})P(\overline{B})$$

Here, $P(A \mid \overline{B})$ is the false positive rate p of the Bloom filter as defined in eq. (2). The probability $P(\overline{B})$ that an element occurs not in a data set S is $P(\overline{B}) = 1 - P(B)$.

In fig. 2a, the influence of different false positive rates on the attack precision is shown for filters with a different amount of elements. As more unique elements are stored in a filter, the attack precision will increase since the size of the (approximated) universe is fixed in this case, therefore, the privacy of a filter will be lowered. Furthermore, it is shown that lower false positive rates will lead to less privacy as less false positives will be produced.

Similarly, from fig. 2b, it can be seen that for filters incorporating higher entropy data sets attacks are less effective and, therefore, the level of privacy will be higher.

Dual Bloom Filters

As single Bloom filter setups are limited since they do not allow for filter updates, we must extend this this model to multiple filters. As a first step, we will define a model for two filters.

Similar to the previous section, we can create a two filter setup using data sets S_1 and S_2, where $\mathcal{BF}_i(m_i, \mathsf{hs}_i, S_i)$, for $i \in \{1, 2\}$. This new model is limited to exactly two filters, therefore the data sets S_1, S_2 or parts thereof, cannot be reused in any other filter as this would make it a multiple filter setup.

First, we can redefine the events A and B for two or more filters:

$$\begin{aligned}A_i &:= z \in \mathcal{BF}(m_i, \mathsf{hs}_i, S_i) \\ B_i &:= z \in S_i\end{aligned}$$

Next, we can differentiate between an attack on the intersecting source set of the Bloom filters and an attack on the union set.

Attack on Intersection

First, we can perform an attack the on the intersection of the two data sets. In this case, we only consider elements that occur in both sets. The precision for an attack on the intersection can be defined as as follows:

$$\begin{aligned}\text{Inter. Prec.} &= P(B_1, B_2 \mid A_1, A_2) \\ &= \frac{P(A_1, A_2 \mid B_1, B_2)P(B_1, B_2)}{P(A_1, A_2)}\end{aligned} \tag{12}$$

where $P(A_1, A_2 \mid B_1, B_2) = 1$, similar, to eq. (11). Furthermore,

$$P(B_1, B_2) = P(B_2 \mid B_1)P(B_1) \tag{13}$$

and

$$\begin{aligned}P(A_1, A_2) = {} & P(A_1, A_2 \mid B_1, B_2)P(B_1, B_2) \\ & + P(A_1, A_2 \mid \overline{B_1}, B_2)P(\overline{B_1}, B_2) \\ & + P(A_1, A_2 \mid B_1, \overline{B_2})P(B_1, \overline{B_2}) \\ & + P(A_1, A_2 \mid \overline{B_1}, \overline{B_2})P(\overline{B_1}, \overline{B_2})\end{aligned} \tag{14}$$

Similar to eq. (13), $P(\overline{B_1}, B_2)$ can be computed as follows:

$$P(\overline{B_1}, B_2) = P(B_2 \mid \overline{B_1})P(\overline{B_1})$$

where the probability $P(B_2 \mid B_1)$ depends on the amount of overlap between S_1 and S_2. The probabilities $P(B_1, \overline{B_2})$ and $P(\overline{B_1}, \overline{B_2})$ can be calculated accordingly. Moreover, in case the elements of S_1 and S_2 are chosen independently, $P(B_1, B_2)$ and others can be computed as follows:

$$P(B_1, B_2) = P(B_1)P(B_2)$$

Given that all hash functions in hs_1 and hs_2 are completely independent, i.e. $\forall h_i \in \mathsf{hs}_1, h_j \in \mathsf{hs}_2 : P(h_i(x_i) = y_i, h_j(x_j) = y_j) = |Y_1||Y_2|$, where Y_1 and Y_2 is the output space of the hash function in hs_1 and hs_2 respectively, A_1 and A_2 are independent. Therefore, the partial probabilities from eq. (14) can be defined as follows:

$$P(A_1, A_2 \mid \overline{B_1}, B_2) = P(A_1 \mid \overline{B_1})P(A_2 \mid B_2) \qquad (15)$$

Here, $P(A_i \mid B_i)$ and $P(A_i \mid \overline{B_i})$ can by computed similar to the single filter approach in section 6.1.3. Accordingly, this step can be applied to the probabilities $P(A_1, A_2 \mid B_1, \overline{B_2})$ and $P(A_1, A_2 \mid \overline{B_1}, \overline{B_2})$ as well.

This attack will be evaluated in more detail in section 6.1.3.

Attack on Union

Similarly, we can perform an attack on the union of two Bloom filters. In this case, we again use both filters for the attack, but now we consider all elements that occur at least in one of the two data sets a true positive. Given that the elements of B_1 and B_2 are chosen independently, the precision for a union attack can be defined as as follows:

$$\begin{aligned}\text{Union Prec.} &= P(B_1 \cup B_2 \mid A_1, A_2) \\ &= 1 - P(\overline{B_1}, \overline{B_2} \mid A_1, A_2) \\ &= 1 - \frac{P(A_1, A_2 \mid \overline{B_1}, \overline{B_2})P(\overline{B_1}, \overline{B_2})}{P(A_1, A_2)}\end{aligned}$$

which can be evaluated similar to eq. (12). Here, $P(A_1, A_2 \mid \overline{B_1}, \overline{B_2})$ is the combined false positive rate for both filters, $P(\overline{B_1}, \overline{B_2})$ is the probability of elements occurring in the different data sets, which can be computed using eq. (13), and $P(A_1, A_2)$ is the probability of two elements resolving to true in the filter.

In fig. 3, the joint precision of the same attack against the intersection on a two Bloom filter setup is shown. However, in this case, we evaluate the brute forced elements on the union of all data, i.e. an element considered a true positive if it is exists at least in one of the sets.

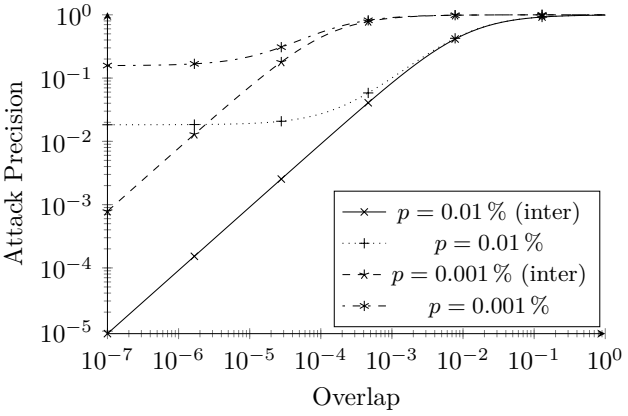


Figure 3: Two Bloom filter brute force attack precision against the intersection of all filter data sets when evaluating against union of elements for different false positive rates given an adversary data set entropy of 30 bits, a filter size of 1000 elements.

Multiple Bloom Filters

From the dual filter setup, this model can be extended to multiple Bloom filters. For q Bloom filters, $\mathcal{BF}_i(m_i, \mathsf{hs}_i, S_i)$, where $1 \geq i \geq q$. For this case, it is required that none of the data of data sets S_i, where $1 \geq i \geq q$, should be reused outside of the given multi filter setup.

First, as we now deal with multiple filters, we need to define some notations for operations on sets of binary events. For some set of binary events $\mathbb{Z}$ and its complement $\mathbb{Z}^{\complement}$, we define the following notation for complementary set of binary events:

$$\overline{\mathbb{X}} = \left\{\overline{X} : X \in \mathbb{X}\right\}$$

Furthermore, we can define notations for binary operations on these sets of events in order to create combined events:

$$\mathbb{X}_{\cap} = \bigcap_{X \in \mathbb{X}} X \qquad \mathbb{X}_{\cap}^{\complement} = \bigcap_{X \in \mathbb{X}^{\complement}} X$$

and

$$\mathbb{X}_{\cup} = \bigcup_{X \in \mathbb{X}} X \qquad \mathbb{X}_{\cap}^{\complement} = \bigcup_{X \in \mathbb{X}^{\complement}} X$$

Attack on Intersection Next, we can now define the precision of an attack on the intersection for q Bloom filters as follows:

$$\begin{aligned}\text{Inter. Prec.} &= P(\mathbb{B}_{\cap} \mid \mathbb{A}_{\cap}) \\ &= \frac{P(\mathbb{A}_{\cap} \mid \mathbb{B}_{\cap})P(\mathbb{B}_{\cap})}{P(\mathbb{A}_{\cap})}\end{aligned}$$

where

$$\mathbb{A} = \{A_1, A_2, ..., A_q\} \quad \mathbb{B} = \{B_1, A_2, ..., B_q\}$$

Similar to probability eq. (11) for a single filter, it is given that:

$$P(\mathbb{A}_\cap \mid \mathbb{B}_\cap) = 1$$

Furthermore, we can recursively define the probability that elements exists in all data sets using Bayes' theorem:

$$P(\mathbb{B}_\cap) = P(B \mid \mathbb{B}'_\cap)P(\mathbb{B}'_\cap) \tag{16}$$

for $B \in \mathbb{B}$, where

$$\mathbb{B}' = \mathbb{B} \setminus \{B\}$$

Or, in case all the elements for the sets are chosen independently, this is equivalent to:

$$P(\mathbb{B}_\cap) = \prod_{B \in \mathbb{B}} P(B)$$

Lastly, similarly to eq. (14), we can define the probability of an element existing in a Bloom filter:

$$P(\mathbb{A}_\cap) = \sum_{\mathbb{B}' \in \mathcal{P}(\mathbb{B})} P(\mathbb{A}_\cap \mid \mathbb{B}'_\cap, \overline{\mathbb{B}'^{\mathsf{C}}}_\cap) P(\mathbb{B}'_\cap, \overline{\mathbb{B}'^{\mathsf{C}}}_\cap) \tag{17}$$

where $\mathcal{P}(\mathbb{B})$ is the power set of $\mathbb{B}$.

In fig. 4a, the precision of a brute force attack against the intersection of elements in a multiple Bloom filter setup can be seen. While attacking more filters can considerably reduce the size of the intersection of the elements in those filters, the attack precision increases significantly the more filters are used.

Attack on Union Similarly, extending from the dual filter approach, we can define the precision of an attack on the union for multiple filters.

$$\begin{aligned}\text{Union Prec.} &= P(\mathbb{B}_\cup \mid \mathbb{A}_\cap) = 1 - P(\overline{\mathbb{B}}_\cap \mid \mathbb{A}_\cap) \\ &= 1 - \frac{P(\mathbb{A}_\cap \mid \overline{\mathbb{B}}_\cap) P(\overline{\mathbb{B}}_\cap)}{P(\mathbb{A}_\cap)}\end{aligned}$$

where

$$P(\mathbb{A}_\cap \mid \overline{\mathbb{B}}_\cap) = P(A_1 \mid \overline{B}_1) ... P(A_q \mid \overline{B}_q)$$

since A_i and A_j are independent, where $i \neq j$, due the pre-image resistance of the hash functions. Furthermore, we can recursively define $P(\overline{\mathbb{B}}_\cap)$ similar to eq. (16):

$$P(\overline{\mathbb{B}}_\cap) = P(\overline{B} \mid \overline{\mathbb{B}'}_\cap) P(\overline{\mathbb{B}'}_\cap)$$

for $\overline{B} \in \overline{\mathbb{B}}$, where

$$\overline{\mathbb{B}'} = \overline{\mathbb{B}} \setminus \{\overline{B}\}$$

Lastly, $P(\mathbb{A}_\cap)$ can be computed as defined in eq. (17).

In fig. 4b, we show the increase of the attack precision when attacking the union of the data sets instead of the intersection. For setups with more Bloom filters, this increase can be significant, especially if the filters have a low overlap. Furthermore, in setups with more elements, the increase in attack precision can be even more substantial. For instance, in a similar setup to the one in fig. 4b with 100 000 elements, the absolute difference almost reaches 20% at points.

6.1.4 Same Parameter Setups

Given that all filters in a setup share the same parameters, namely the size m and the hash functions hs, the privacy impact of dual and multiple filter setups can be partially mitigated. In this case, all elements shared between different filters will set the same bit positions in the separate filters. Therefore, setups with largely overlapping data sets will be more likely to generate the same false positives as well. In the next section, we provide the analysis for the dual filter case.

Dual Filter Setup
First, as the filter now use the same parameters, the events A and B can be redefined as follows:

$$\begin{aligned}A_i &:= z \in \mathcal{BF}(m, \mathsf{hs}, S_i) \\ B_i &:= z \in S_i\end{aligned}$$

The attack precision can then be defined in the same way as for the independent dual filter case in eq. (12), except for the computation of $P(A_1, A_2)$ in eq. (14):

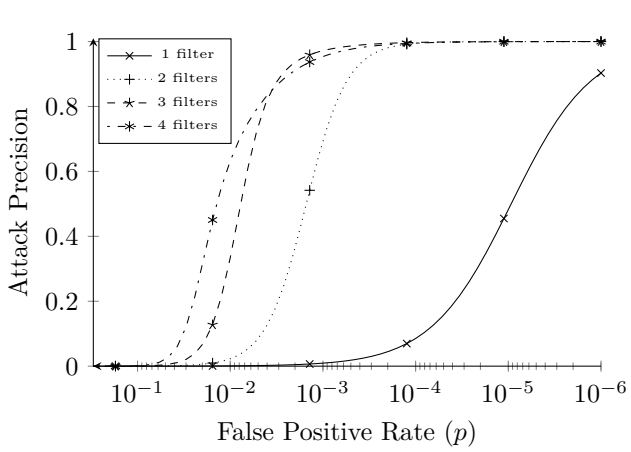
$$\begin{aligned}P(A_1, A_2) = &P(A_1, A_2 \mid B_1, B_2) P(B_1, B_2) \\ &+ P(A_1, A_2 \mid \overline{B_1}, B_2) P(\overline{B_1}, B_2) \\ &+ P(A_1, A_2 \mid B_1, \overline{B_2}) P(B_1, \overline{B_2}) \\ &+ P(A_1, A_2 \mid \overline{B_1}, \overline{B_2}) P(\overline{B_1}, \overline{B_2})\end{aligned}$$

Here, A_1 and A_2 are not independent, since the same hash functions are used for both filters. Therefore, bit positions can be set by overlapping elements in both filters, thereby increasing the chance of the similar false positives in both filters. In this case, we must consider the probability of some bit position being set in both filter. For this purpose, we split all element of S_1 and S_2 into three separate independent sets:

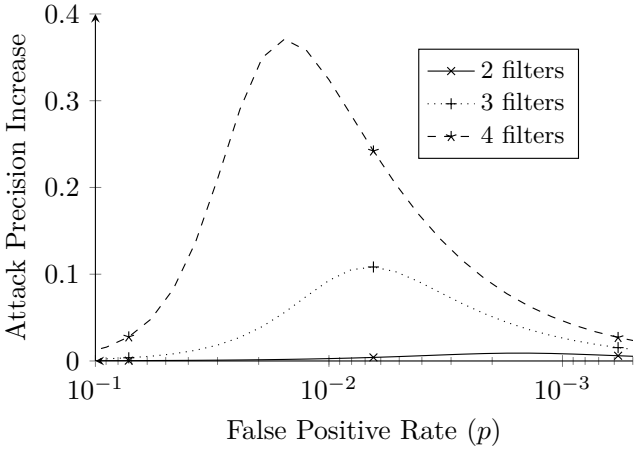
$$\begin{aligned}S_\cap &= S_1 \cap S_2 \\ S'_1 &= S_1 \setminus S_\cap \\ S'_2 &= S_2 \setminus S_\cap\end{aligned}$$

Furthermore, we define a function C_i for the event of a bit b_j being set in the set S_i:

$$C_i := b_j \in \mathcal{BF}(S_i)$$



(a) Attack precision of attack against the intersection of filter data sets.



(b) Absolute increase of attack precision of attack against the union of filter data sets on top of fig. 4a.

Figure 4: Multiple Bloom filter brute force attack precision given a total data entropy of 30 bits, filters with each 10 000 elements and a filter overlap of 25 %.

Here, some bit b_j can only be set in, both, S_1 and S_2, if either the bit is set in $S_\cap$, or it is not set in $S_\cap$, but it is set in, both, S'_1 and S'_2:

$$P(C_1, C_2) = P(C_\cap) + P(C'_1, C'_2, \overline{C_\cap}) \tag{16}$$

The probability of a bit being set due to overlapping elements in the data set $S_\cap$ can be computed similar to eq. (1) for regular Bloom filters:

$$P(C_\cap) = 1 - (1 - 1/m)^{k|S_\cap|}$$

Furthermore, as C'_1, C'_2 and $C_\cap$ are independent, since no elements are shared between the sets, the probability $P(C'_1, C'_2, C_\cap)$ can be simply written as follows:

$$P(C'_1, C'_2, \overline{C_\cap}) = P(C'_1)P(C'_2)P(\overline{C_\cap})$$

Using section 6.1.4, i.e. the probability of a bit being set in both filters, the probability of a double false positive can then be computed similar to eq. (2):

$$P(A_1, A_2 \mid \overline{B_1}, \overline{B_2}) = P(C_1, C_2)^k$$

In fig. 5, an illustrative example of the possible effect on attack precision in the case of two-filter same parameter setups is given. Here, we can see that the attack precision is significantly reduced for same parameter configurations as the overlap of the data set is increased, thereby eventually converging into a single-filter scenario. It should be noted, however, that data sets are likely to diverge more over larger time periods due to database changes, therefore this effect on privacy will also be reduced over time as the overlap will decrease. Nonetheless, for the case of filter updates where a database has little to no changes over time, this can be a valuable brute-force attack mitigation strategy.

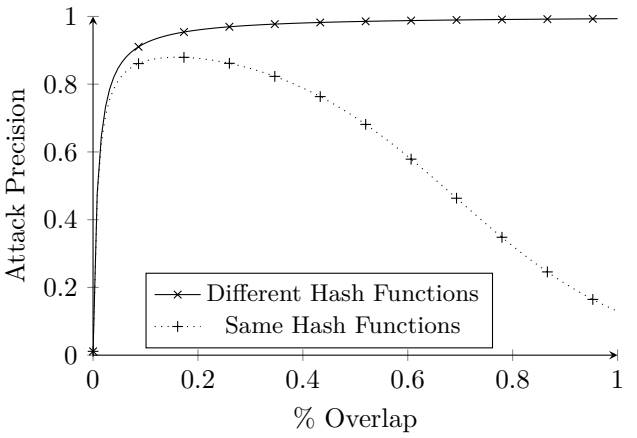
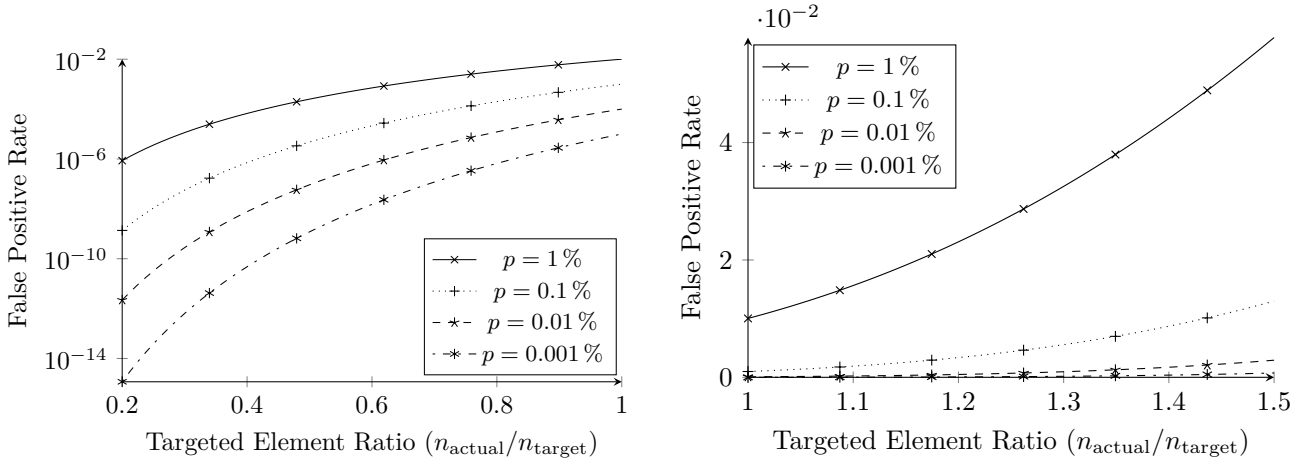


Figure 5: Comparison of the two Bloom filter brute force attack precision against the intersection of the filter data sets for differing parameter setups and same parameter setups. Here, we show filters with 10 000 elements, a false positive rate of 0.001 and 10 hash functions. The adversary data set is a superset of the filter data and has an information entropy of 26 bits.

However, it requires all filters to share the same secret, therefore it is not suitable for cases where the different database providers do not trust each other.

6.2 Utility Analysis

In section 5.1.2, we defined the utility metrics as the true negative rate. In the case of Bloom filters, the true negative rate is the inverse of false positive rate eq. (2). From the approximations eq. (3) and eq. (5) it is shown that the false positive rate mostly depends on



(a) False positive rate for $n_{\text{actual}}/n_{\text{target}} \in [0.2, 1]$.

(b) False positive rate for $n_{\text{actual}}/n_{\text{target}} \in [1, 1.5]$.

Figure 6: Utility for Bloom filters with different targeted false positive rates at for different $n_{\text{actual}}/n_{\text{target}}$ ratios, where n_{actual} is the actual number of elements in the filter and n_{target} is targeted number of elements.

the ratio between the targeted amount of elements and the actual amount of elements $n_{\text{actual}}/n_{\text{target}}$. In fig. 6, the utility is shown for different targeted false positive rates and targeted element ratios.

First, from fig. 6a it is shown that if less then the targeted amount of elements n_{target} are inserted into a filter, the false positive rate can significantly lower and, therefore, the utility score higher.

Similarly, we can see from fig. 6b that the utility significantly decreases if more than the targeted amount of element are inserted into the filter.

6.3 Efficiency Analysis

Next, we evaluate the efficiency as defined in section 5.1.3, where we consider the computational and transactional complexity for the setup and for a single query. As transactions solely take place in the setup phase in the case of Bloom filters where the filters are deployed, we will only consider the computational cost for queries.

6.3.1 Computational Complexity

For the computational complexity, we consider the costs of the setup as well as the costs of a single query.

When creating a filter, each inserted element has to be hashed using k hash functions, therefore the following factors mostly define the filter setup performance:

- The number of elements n inserted into the filter.
- The type of the hash functions being used, e.g. MD5 hashes will be faster than SHA3 hashes.
- The number of hash functions k, as each element has to be hashed k times.

Furthermore, For filter querying, we can define best case and worst-case scenarios. In the best case scenario, the queried element is not a member of the data set and the first bit-position check is negative in which case only one hash has to be computed. In the worst case, the queried element results in either true positive, i.e. the element is part of the filter data set, or a false positive, i.e. the element is not part of the data set. In both cases all hash functions have to be computed as all bit-position checks will be positive. The average-case efficiency is somewhere between, depending on the number of elements currently in the filter, the number of hash functions used and what percentage of the elements being queried is a member of the filter data set. As the average case is dependent on the queries being performed, we will only analyse the best- and worst-case performance for the single query cost.

Similar to the setup phase, we can identify the following factors impacting the querying performance:

- The number of hash functions k has an impact on the average and worst-case performance, since more hashes have to be computed in those cases, depending on the configuration of the filter.
- The effective false positive rate and the actual number of elements in the filter have an influence on the average case, as a higher effective false positive rate will result in more negative bit-position checks.
- The type of the hash functions, similar to the setup phase.

Hash Functions

As the size of the codomain of common hash functions is fixed or at least limited, it will most probably not be equal to size of the filter. Therefore, an alternative

hash function has to defined that maps directly to bit-positions for the membership querying algorithm. A straightforward implementation of such a hash function $h\colon \mathcal{M} \to \mathbb{Z}_m$ could be defined as follows, where $\mathcal{M}$ is the message space:

$$h(z) := h'(z) \mod m, \tag{16}$$

where h' is some cryptographic hash function like SHA3, z is the element being queried and m is the size of the filter. However, such an approach may introduce a bias similar to the modulo bias experienced when mapping randomly generated numbers from one domain to another (non-power-of-two) domain, and will thereby increase the false positive rate of the filter.

In order to prevent this kind of bias, we can define a simple function where we ignore hashes that cannot be directly mapped to a bit-position inside the filter without introducing a bias. For this purpose, we define a recursive hash function $h\colon \mathcal{M} \to \mathbb{Z}_m$ using some keyed-hash function $h_k\colon \mathcal{K} \times \mathcal{M} \to \mathcal{H}$, where $\mathcal{K}$ is the key space and $\mathcal{H} \subset \mathbb{Z}$ the output space of the hash function:

$$h(z) := h'(0, z)$$

$$h'(i, z) := \begin{cases} h_k(i, z) \mod m, & \text{if } h_k(i, z) < h_{\text{limit}} \\ h'(i+1, z), & \text{otherwise} \end{cases}$$

where

$$h_{\text{limit}} = |\mathcal{H}| - (|\mathcal{H}| \mod m),$$

where h_{limit} is the maximum hash value that can be mapped to a bit-position without introducing bias. However, this hash function is now non-deterministic and may require multiple hash computations depending on the size of the filter m, thus increasing the query time in those cases. Given a worst-case Bloom filter setup, where h_{limit} is minimal at $m = \lfloor |\mathcal{H}|/2 \rfloor + 1$, the probability of generating a non-mappable output using a (keyed-)hash function is $p = h_{\text{limit}}/|\mathcal{H}| \approx 0.5$. In this case, we deal with a geometric distribution, therefore the average number of hash computations will then be $1/p \approx 1/0.5 = 2$. However, for most filter configurations h_{limit} will be close to $|\mathcal{H}|$, therefore the average number of hash computations will be significantly lower than that.

An alternative to a keyed-hash function, in this case, is double hashing, where the output of two cryptographic hash functions, h_1 and h_2, is computed and then used to derive different hashes:

$$h'(i, z) := (h_1(z) + i \cdot h_2(z)) \mod |\mathcal{H}|,$$

given $h_2(z) \neq 0$. This function has a constant worst-case and average-case complexities in terms of two hash computations, making it less efficient than the first solution using a keyed-hash function for the average-case. However, this method can be useful for cases where the worst-case amount of computations should be constant.

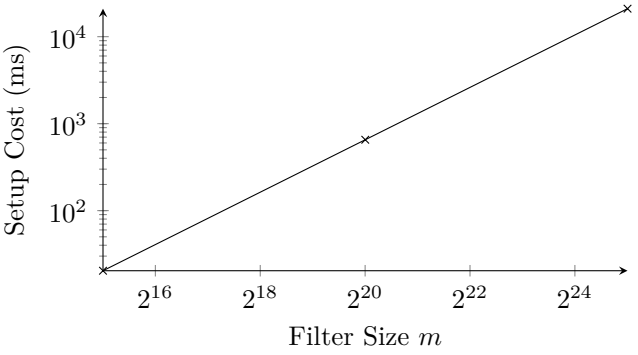


Figure 7: Computational efficiency of Bloom filter setup T_{setup} for different filter sizes m and 10 hash functions.

In order to exclude any non-determinism due to bias, we will only consider Bloom filter configurations where the aforementioned simple hash function section 6.3.1 will introduce no bias, i.e. when $|\mathcal{H}|$ is divisible by m. Nonetheless, for performance critical use cases, potential performance influences due to such biases should be considered.

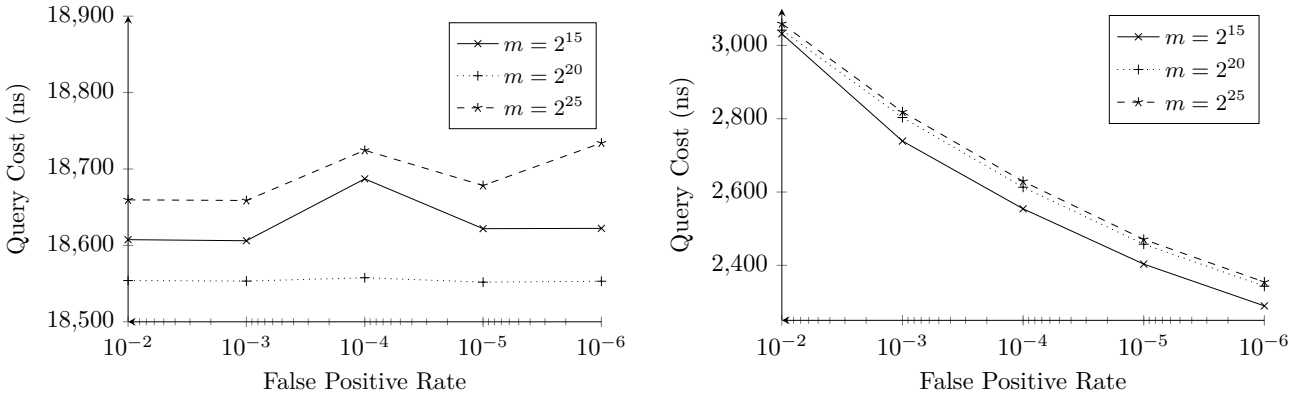
Benchmark

For the purpose of evaluating the runtime performance, we use a benchmarking setup with the following specifications: *Intel i7-4790K* @ 4.5 GHz, 16 GB, 1600 MHz *LPDDR3*, Linux 4.20. Furthermore, we tested Bloom filters of sizes $m \in \{2^{15}, 2^{20}, 2^{25}\}$ using 10 independent SHA3-based hash functions, with the optimal number of elements n according to eq. (4).

In fig. 7, we show the results of the Bloom filter setup benchmarks. Here, it can be seen that the runtime cost of a query is linear in respect to the filter size m and, therefore, also in respect to the number of elements m as can be derived from eq. (4).

The results of the benchmarks for the computational cost of querying member elements $T_{\text{query,mem}}$ and non-member elements $T_{\text{query,nonmem}}$ are shown in fig. 8. The actual average query cost T_{query} of a query depends on the queries of consumer and is bounded by $T_{\text{query,nonmem}}$ and $T_{\text{query,mem}}$.

As can be seen, the cost of member element queries is mostly consistent regardless of the false positive rate. Therefore, we can conclude that this cost is mostly dependent on type and number of hash functions used. On the contrary, the cost of non-member element queries will decrease for lower false positive rates as the likeliness of the same bit-position being set by multiple different elements in the filter is lowered as well. Lastly, the general runtime differences between the different filter sizes are most likely caused by CPU cache misses, which have an increasing impact for larger Bloom filter



(a) Cost of member queries $T_{\text{query,mem}}$.

(b) Cost of non-member queries $T_{\text{query,nonmem}}$.

Figure 8: Computational efficiency of Bloom filter queries at different false positive rates for filters of size m and 10 hash functions.

sizes.

6.3.2 Communication Complexity

For the communication costs, we will only consider the setup phase as querying does not require any transactions.

After a Bloom filter has been created, the only transaction that takes place is the distribution of that filter to the consumer. Here, we can compute the filter transaction cost using eq. (5) as follows:

$$L_{\text{setup}} = m = \frac{n \ln p}{(\ln 2)^2}$$

The communication complexity for querying is, therefore, linear in respect to the number of records n.

6.4 Functionality Analysis

In this section, we will analyse the additional functionalities metric as defined in section 5.1.4 for the case of Bloom filters.

First of all, adding new elements to the data set of the filter provider can be simply achieved by creating and distributing a new version of the filter with the additional elements. However, creating a new Bloom filter with different parameters will result in a multi-filter setting and will most likely leak information about the overlapping data the filters as is described in section 6.1.3. If the filter parameters are kept the same, no additional information regarding the overlapping elements will be leaked. Nonetheless, there will likely be some leakage about the newly added elements as one can now discern bit positions set in the new version of the filter and positions set in previous version, thereby giving clues about the set of added elements. We can, therefore, not securely add an element to a Bloom filter without leaking information at all, but it is possible to so without any additional leakage of the former data set.

Second, while one could remove elements from a data set by distributing a new filter similar to the situation above, another receiving party can just keep the original filter that still includes these elements. As there is no way to ensure that an older filter has been removed by another party, the guaranteed deletion of elements is not possible in this scheme.

6.5 Case Study Ma^{3}tch

In this section, we will evaluate the application of Bloom filters in the context of the Ma^{3}tch technology [27] that is part of FCInet project. FCInet is an initiative by the OECD Forum of Heads of Tax Crime Investigation focusing on improving the collaboration between international financial crime investigation units. The Dutch FIOD (Fiscale inlichtingen- en opsporingsdienst) and the British HMRC (Her Majesty's Revenue and Customs) are leading this initiative. Furthermore, a pilot of FCInet is ongoing as of 2017 with the FIOD, the HMRC and the Belgian BBI (Bijzondere Belastinginspectie). One the of the goals of the FCInet project is the assistance of the investigation process. For this purpose, the project considers the processes for data analysis and information exchange regarding (past) cases, criminals and suspects, as these processes are currently mostly non-automatable due to the lack of tools and legal devices.

Ma^{3}tch is a technology used in FCInet, aiming to make these processes more practical, while also providing a certain level of confidentiality regarding the exchanged data. Currently, Ma^{3}tch is evaluated in a pilot test evaluating the exchange of data about criminals and suspects, where first names, last names and birth dates of individuals are shared in an encoded form.

For this purpose, Ma^{3}tch makes use of Bloom filters,

similar to the scheme by Little et al. [4], where the data to be shared is encoded into a Bloom filter. This filter can be shared with partners, who can then be queried to check if the filter contains an individual. However, as mentioned before in section 2.1, this process can induce false positives, which can be used to provide a certain level of privacy.

6.5.1 Method

In this section, we will solely discuss the privacy aspects of Bloom filters in the Ma^{3}tch context, as the provided utility and efficiency are not impacted by the data that is encoded into the Bloom filter. These aspects of a Bloom filter only depend on the Bloom filter parameters and the amount of elements in the filter, and, therefore, have already been adequately discussed in the previous sections.

For this case study, we worked together with the FIOD as an investigative instance, therefore, for this section, we consider the FIOD as the filter provider. In this scenario, the filters encode first names, last names and birth dates. The filters' privacy will be evaluated for false positive rates of 0.01 %, 0.001 % and 0.0001 %, as such scenarios are assumed to be common in the context of Ma^{3}tch. Since we already analysed the privacy impact of encoding non-optimal amounts of data entries into a filter, we will assume a test set size of 30 000 records.

Furthermore, we will perform, both, a theoretical and an experimental analysis of the privacy in this scenario. In the theoretical analysis, we try to estimate the attack precision based on simulated data for this case. For the experimental analysis, we evaluate the attack on actual data of which we only know that it contains first names, last names and birth dates.

For the experiments, we will assume the *Insider* attack scenario, where a filter including the corresponding secret has somehow been leaked, as the privacy provided by a filter cannot be compromised under the assumptions of the two weaker scenarios. Furthermore, we assume an external adversary that knows that the FIOD is the provider of the filter, and we assume that the adversary has access to publicly available knowledge, except for resources providing full person details such as social media websites or publicly available data leaks of other online services. All conclusions about privacy in this section will, therefore, only hold under these assumptions.

Since the adversary only knows that the filter was provided by the FIOD and cannot use any publicly available records of people, the adversary is bound to use different resources for a brute-force attack, as simply generating a set of all possible strings as names, for instance, is generally not feasible.

Furthermore, as the FIOD is a Dutch authority, we assume that most names that are encoded in a filter will be Dutch. As a reference data set, we use the public Dutch first and last name databases[1][2]. These data sets encompass most names occurring in the Netherlands, which includes a fair amount of non-Dutch names. Moreover, for each name, estimates of the amount of occurrences are provided. Using these data sets, a probabilistic generator can be created that can generate a data set with single first and last name combinations. We do not consider secondary first names, as these would result in a state-explosion making the attack mostly infeasible.

For birth dates, we assume that all people in the data set are born between the 1st of January, 1910 and the 1st of January, 2010. Just as for the names, we could use statistical data for birth date distributions in the Netherlands, as different seasons and years will have different birth rates. However, this data is not as readily available publicly, therefore we assume a uniform distribution for the birth dates.

By combining the data sets for names and birth date, i.e. through the computation of the cartesian product, a brute-force attack can be simulated.

6.5.2 Theoretical Analysis

For a theoretical data set of the filter provider, we use the same assumptions about the data as described in the previous section. Testing with real data would be more appropriate, however this information is generally not available due to privacy concerns. Therefore, we can only create test cases by generating a data set under a set of assumptions.

Considering the adversary, we can approximate the total size of the assumed adversary's data set with all first and last names occurring in the Netherlands using the afore-mentioned name databases: the set of first names has a size of around 2^{17} entries and last names around 2^{19} entries. Both data sets encompass names of almost 16 million people, which almost accounts for the current population in the Netherlands.

As mentioned, for birth dates, we assume that the date occurred somewhere in the last 100 years, which corresponds to a set of around 2^{15} possible dates.

Together, the combination of these sets accumulate to a total approximated adversary set size of 2^{52} possible data entries, which is hard to use in a brute-force attack for a common adversary. However, as not all names are equally common, we can perform a non-exhaustive search for entries using the popularity of different names.

[1] Nederlandse Voornamenbank - `https://www.meertens.knaw.nl/nvb/`

[2] Nederlandse FamilienamenBank - `https://www.meertens.knaw.nl/nfb/`

False Positive Rate	Attack Precision
10^{-4}	0.016
10^{-5}	0.084
10^{-6}	0.256

Figure 9: Result of theoretical analysis of privacy in the Ma^{3}tch scenario.

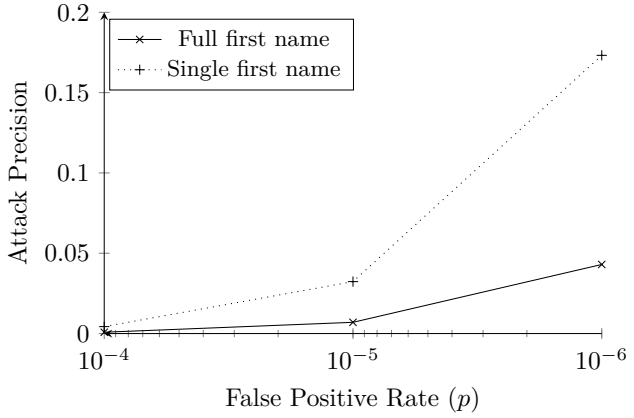


Figure 10: Attack precision of experimental analysis on unknown Ma^{3}tch data with records including first name(s), last name and birth date.

For a non-exhaustive search, we can compute a lower-bound set size for an non-exhaustive search using Shannon's information entropy [28]. For this case, we estimate the lower-bound set size for first and last names of approximately to be 2^7 and 2^{10} data entries, respectively. The set of birth dates, still has around 2^{15} entries, as we assumed a uniform distribution. The combination of all data setts results in a lower-bound data set size of 2^{32} entries. As this set size is a lower-bound and does not include less frequent names, we will assume a total test data set size of 2^{34} entries.

For the evaluation, as was mentioned, we evaluate filters with false positive rates of 0.01 %, 0.001 % and 0.0001 % with 30 000 elements. The results of this theoretical analysis are shown in fig. 9.

6.5.3 Experimental Analysis

For the experimental analysis, we evaluate two filters created from an actual data set, which is unknown to us. The first filter contains full names and birth dates, and the second contains only primary first names, last names and birth dates. We will then use the above-mentioned adversary data set with names and birth dates to perform a non-exhaustive search. The results of this experiment are shown in fig. 10.

6.5.4 Discussion of Single-Filter Scenario

As can be seen from the evaluation, the experimental analysis, a non-exhaustive search using real data is significantly less effective than in the theoretical analysis. There are multiple possible factors that can influence this result.

First, in the first case with full names, it is not possible to find any entries that have secondary first names, as our reference data set only includes single first names. This effect can be seen from the second scenario, where the attack precision is significantly higher as the Bloom filter only entries where all secondary are removed.

Furthermore, the size of the overlap of the actual names in the filter with the Dutch name databases is unknown, therefore it is likely that names unknown to the adversary also reduce the amount of entries that can be found.

Next, as was already concluded from the data-agnostic privacy evaluation of Bloom filters: the assumed adversary data set may be too large for an attack to be feasible, i.e. the combined data set of names and birth dates may result in too many false positives to obtain any significant result.

Lastly, the data set size in the theoretical analysis was based on a best-case scenario (considering the adversary) regarding the distribution of names, as the actual distribution of names in a actual data set is unknown.

Therefore, as we do not know the exact contents of the targeted database, it is not possible to identify the impact of each of these factors on the measured results. Furthermore, repeating this evaluation for another kind of adversary with different knowledge and assumption can yield vastly different results. In order to make more precise theoretical and experimental evaluations, it is, therefore, necessary to include an internal data-analyst, who knows the exact contents of the filter(s), and the expected knowledge and assumptions of an attacker, also considering data that is feasibly attainable by an adversary. Furthermore, for such an evaluation, the privacy implications of database updates due to ongoing investigations or data-retention policies should be anticipated with regard to the multiple Bloom filter setting.

6.6 Discussion

For the purpose of record lookup schemes in distributed systems (and similarly private linkage schemes), Bloom filters can provide a certain tunable level privacy depending on the attack scenario.

In the case of outsider adversaries, leakage is not possible if the systems are sufficiently such that properly generated shared secrets are only used between a single server and client.

For insider adversaries or in the case that a secret is leaked, Bloom filters provide a trade-off between privacy, and utility and efficiency depending on the risk-

appetite of the database providers. However, in case an adversary gets hold of multiple filters with overlapping data, for example due to updates of the data set, the effectiveness of a brute-force attack is further increased depending on the filters they have access to and their data overlap. Available mitigations using configuration strategies are only effective up to a limited number of compromised filters in this case, as the privacy of a filter cannot be increased indefinitely without the utility dropping below usable levels. Furthermore, in the case of filter updates, the deletion of older filter cannot be guaranteed, therefore there is no effective approach for removing records from a database without potential leakage in the future.

When considering the privacy trade-off provided by Bloom filters, a domain expert familiar with the data and the system should be included in this adjustment, as the provided level of privacy is highly dependent on the specific data being stored. Furthermore, different types of adversaries should be considered as brute-force attack will be more effective based on the knowledge of an adversary.

7 Encrypted Bloom Filters

As was shown in the last section, the privacy impact mitigation of brute-force attacks on a Bloom filter in the lookup scheme can significantly reduce the utility of that filter. Furthermore, the current approach does not allow secure deletion of elements, for instance, when releasing updated filters. Ideally, the querying of the Bloom filter should be interactive such that the filter consumer cannot access the filter locally and cannot perform an unlimited amount of queries while not leaking any information about the queries.

Our proposed interactive protocol defines two parties: *Alice* as the providing party with a Bloom filter of a data set S_A and *Bob* as the consuming party with a separate data set S_B. In this case, *Alice* is willing to share with *Bob* which data elements they have in common while allowing for false positives, i.e. the intersection $S_A \widetilde{\cap} S_B$, without learning anything themselves.

7.1 Related Work

With record lookup schemes, some information is shared with other parties that allows them to identify databases that might have the data that they are interested in. While the information that is shared can be different, it becomes similar to the setting of PSI (Private Set Intersection) when operating in the encrypted domain. PSI schemes allow two or more parties compute the intersection of their data sets without either party learning anything more than that intersection. As such, it possible to construct private record lookup schemes based on PSI schemes, where only the querying party learns the result.

One of the first PSI protocols is described in [29] by Meadows. The proposed protocol is based on the Diffie-Hellman key exchange and allows two parties to privately check if they share the same secret, while only requiring a trusted third party for the setup phase of the protocol. Since the protocol provides matching of pairs of secrets, computing the intersection has a worst-case complexity of $\mathcal{O}(nm)$ runs of the protocol given two parties with data sets of sizes n and m.

In [30], Freedman et al. propose PSI protocols for the semi-honest setting based on oblivious polynomial evaluation [31] using homomorphic encryption schemes such as the Paillier cryptosystem [32]. Through reducing the polynomial size by separating the elements into different "buckets", a worst-case computational complexity $\mathcal{O}(n + m \log\log n)$ is achieved. Furthermore, a fuzzy matching scheme is proposed, where each data element consists of a vector of predefined attributes. Here, a match will only have a positive result if at least t attributes are shared between elements.

Hazay and Nissim present an extension to this approach in [33] for the malicious settings while using the ElGamal cryptosystem [34] with homomorphic addition. It offers a similar computational performance as the semi-honest protocol in [30].

In [35], Kerschbaum introduces an Bloom filter public key encryption scheme using the homomorphic properties of the Goldwasser-Micali cryptosystem [36]. Here, the result of a query is provided as a vector of ZKPs (Zero-Knowledge Proofs). However, this leaks the element that is queried to the party creating the ZKP. Kerschbaum describes in [37] interactive PSI protocols based on a similar approach for the semi-honest and malicious scenarios with a linear complexity. Furthermore, a variant of the protocol is proposed where the computation of the intersection can be outsourced to an oblivious service provider.

Perl et al. present a confidential search scheme [38] for bio-medical data that makes use of Bloom filters. For each query, a filter is created and obfuscated to prevent a server from learning the query. Separately, each query is encrypted using the Smart-Vercauteren FHE (Fully-Homomorphic Encryption) scheme [39] using the private key of the client. By evaluating the obfuscated Bloom filters on their database, the server can then limit the possible query results, which are then evaluated homomorphically in the encrypted domain using the query ciphertexts and returned to the client. Here, the client can choose higher false positive rates for the Bloom filters encoding the queries in order to reduce potential leakage. A higher false positive rate will then produce a larger potential result set, in which case the server has to perform more computations for the final encrypted search.

Dong et al. propose a PSI scheme using *garbled Bloom filters* in [40], where all filter position set by a single element are represented by a set of secret shares. A test will then only result in a positive match when all queried filter positions contain corresponding secret shares, thereby making the probability of a false positive negligible. An interactive protocol using this approach with OT (Oblivious Transfer) for the semi-honest and malicious settings is provided as well.

In [41], Pinkas et al. provide an overview of various existing PSI protocols, as well as optimisations to existing protocols such as that described in [40]. Furthermore, the authors also introduce an efficient PSI protocol using OT with quasi-linear complexity with further improvements in [42].

In [43], Egert et al. proposes a multi-party private set-cardinality computation protocol using Bloom filters. The authors present a separate protocol for the two-client case using the additive properties of the ElGamal cryptosystem [34].

Davidson and Cid describe two-party interactive protocols for PSI and PSU (Private Set Union) in [25] using encrypted Bloom filters. The protocols use the Paillier cryptosystem [32] and have linear complexities similar to other Bloom filter-based constructions. Here, the querying party creates a filter of their data set. Before encryption, the bit positions of a Bloom filter are inverted such that queries can be computed using the additive properties of the cryptosystem. The encrypted Bloom filter is then evaluated by the other party under encryption. Finally, the querying party can then decrypt the results of the queries. Both protocols are secure in the semi-honest model and extensions to the malicious model are provided.

7.2 Contributions

Our goal is to move the simple off-line Bloom filter-based scheme described in the first part of this paper to an interactive privacy-preserving protocol. The protocol should allow a client to efficiently perform set membership queries on a Bloom filter of another remote party with a certain false positive rate defined by the latter party. Here, the filter provider should not be able to learn anything about the queries aside from the amount of queries being performed. Furthermore, the protocol should be able mitigate brute-force attacks as described in section 6.1 or allow the filter provider to detect such illegitimate use.

For this purpose, we propose an interactive Bloom filter querying protocol based on the PSI protocol presented by Davidson and Cid in [25]. In our protocol, the party sharing the data creates the filter instead of the querying party, therefore allowing the former to define a false positive rate for the shared filter based on the intended use case. Furthermore, our protocol is implemented using the ElGamal cryptosystem, which can be adapted for additive homomorphism and has a higher computational and communication efficiency than the Paillier cryptosystem. It does not, however, allow for homomorphic multiplications with plaintext messages, although we will show that this is not required for our protocol. Lastly, we provide a prototype implementation of our protocol as well as an efficiency comparison to PSI protocol in [25].

7.3 Primitives

7.3.1 ElGamal on Elliptic Curves

Given a curve $E(\mathbb{F}_p)$ over $\mathbb{F}_p$, with parameters (p, a, b, P, n), *Alice* generates a secret key $\mathsf{sk} = a_A \xleftarrow{R} \mathbb{Z}_n^*$. The public key $\mathsf{pk} = P_A \leftarrow [a_A]P$ is then published together with the parameters of the curve.

For these parameters, the encryption function $\mathsf{Encrypt}(\mathsf{pk}_A, M)$ is defined for some point $M \in \mathbb{F}_p$ as follows:

$$[\![M]\!]_A \leftarrow ([k]G, [k]P_A + M), \quad \text{where } k \xleftarrow{R} \mathbb{Z}_n^* \tag{16}$$

Correspondingly, we define a decryption function $\mathsf{Decrypt}(\mathsf{sk}_A, [\![M]\!]_A)$ for some ciphertext $[\![M]\!]_A = (Q, R)$:

$$M \leftarrow R - [a_A]Q \tag{16}$$

Furthermore, we define a function $\mathsf{Encode}(m)$, which encodes some message $m \in \mathcal{M}$ onto a point in $\mathbb{F}_p$, where $\mathcal{M}$ is the message space:

$$M \leftarrow [m]P \tag{16}$$

Analogously, we define a function $\mathsf{Decode}(M)$ that is able to efficiently decode a point $M = [m]P$ if $m = 0$, i.e. $M = P_\infty$. Due to the discrete logarithm problem, there is no known generally efficient method to compute m given $[m]P$, however one can compare encoded points, therefore we can identify expected values by encoding them beforehand.

For clarity, we will denote homomorphic addition of ciphertexts using the operator $\oplus$, such that:

$$[\![M_1]\!]_A \oplus [\![M_2]\!]_A = [\![M_1 + M_2]\!]_A$$

7.3.2 Additive Bloom Filters

Given a Bloom filter $\mathcal{BF}(m, \mathsf{hs}, S)$ for some data set S and hash functions $\mathsf{hs} = (h_1, ..., h_k)$ that produces the bit-vector $\mathsf{bf} = (b_1, ..., b_m)$, we can test the membership of some element elem by verifying that for all $h_i \in \mathsf{hs}$: $b_{h_i(\mathsf{elem})} \stackrel{?}{=} 1$. Alternatively, this can also be written multiplicatively as $\prod_{h_i \in \mathsf{hs}} b_{h_i(\mathsf{elem})} \stackrel{?}{=} 1$.

However, as we are dealing with an strictly additive homomorphic encryption scheme, multiplication of ciphertexts is not possible, therefore filters must be encoded in a different manner to work additively. For this purpose, we can create an "inverted Bloom filter" ibf by inverting each bit of the filter, i.e. $\mathsf{ibf} \leftarrow (\overline{b_1}, ..., \overline{b_m})$, as proposed in [43, 25]. We can now additively test the membership of some element elem by verifying that $\sum_{h_i \in \mathsf{hs}} \overline{b_{h_i(\mathsf{elem})}} \stackrel{?}{=} 0$, which can now also be implemented using a homomorphic encryption scheme as described above.

In this case, the querying party will learn how many of the k queried bit positions matched, which can give an indication of the number of elements in the filter (given that the false positive rate is known). Therefore, we propose a variation on "inverted Bloom filters" that randomises each unset bit position in $\mathbb{Z}_n$ instead of setting it to 1. Given some Bloom filter with a bit vector $\mathsf{bf} = (b_1, ..., b_m)$, we define a function $\mathsf{ObfuscateFilter(bf)}$ that produces an obfuscated vector obf using a mapping $\{0, 1\} \to \mathbb{Z}_n$:

$$\mathsf{obf} \leftarrow (v_1, ..., v_m), \quad \text{where } v_i = \begin{cases} 0, & \text{if } b_i = 1 \\ r \xleftarrow{R} \mathbb{Z}_n, & \text{otherwise} \end{cases}$$

Given that the querying party conforms to the protocol, i.e. the party queries actual elements using hash functions gs, the party will not be able to distinguish $\sum_{h_i \in \mathsf{hs}} v_{h_i(\mathsf{elem})}$ in case of a negative from some $r \xleftarrow{R} \mathbb{Z}_n$. This is different from the "obfuscated filters" mentioned in [38], as the intention, here, is not the increasing of the false positive rate.

In order to mitigate leakage to non-conforming parties, one could encode the set bits for every encoded element in bf using some secret sharing scheme as, for instance, the one proposed by Dong et al. in [40], such that the k bit positions of a valid element in the filter must be queried in order to get a positive result. However, in this case, false positives cannot occur anymore as the probability that the addition of k random positions will uncover the secret is negligible. Therefore, we will use the above-mentioned obfuscated filter approach and assume that the querying party adheres to the protocol.

7.4 Protocol

Our protocol consists of four phases:

Setup Setup of all cryptographic parameters such as the public curve parameters pp and the public and secret keys, pk_A and sk_A.

Filter Obfuscation Obfuscation, encryption and distribution of some Bloom filter bf.

Querying Computation and blinding of the queries qs.

Result Retrieval Decryption and unblinding of the queries by the relevant parties.

After sharing the initial cryptographic parameters in the **Setup** phase, fig. 11 provides an overview of the subsequent phases of our protocol. Furthermore, the following sections will discuss the phases in more detail.

7.4.1 Setup

For setup, *Alice* generates an ElGamal key pair $(\mathsf{pk}_A, \mathsf{sk}_A)$ on some curve E and shares the parameters of $E(\mathbb{F}_p)$ and the public keys pk_A with *Bob*. Furthermore, *Alice* creates Bloom filter bf from their data set S_A for some filter length m and a set of independent hash functions $\mathsf{hs} = (h_1, ..., h_k)$, and then shares m and a specification of hs with *Bob*.

7.4.2 Filter Encryption

Given some Bloom filter $\mathsf{bf} \leftarrow (b_1, ..., b_m)$ and corresponding hash functions hs, *Alice* creates an obfuscated version of the filter obf:

$$(v_1, ..., v_m) \leftarrow \mathsf{ObfuscateFilter(bf)}$$

Next, *Alice* encodes and encrypts each filter position v_i as follows:

$$V_i \leftarrow \mathsf{Encode}(v_i)$$
$$[\![V_i]\!]_A \leftarrow \mathsf{Encrypt}(\mathsf{sk}_A, V_i)$$

The encrypted filter $[\![\mathsf{obf}]\!]_A = ([\![V_1]\!]_A, ..., [\![V_m]\!]_A)$ is then sent to *Bob*.

7.4.3 Querying

Given a some data set $S_B = \{y_j \in \mathcal{M}\}$ and an encrypted filter $[\![\mathsf{obf}]\!]_A$, *Bob* can create an encrypted query $[\![Q_j]\!]_A$ for each y_j as follows:

$$[\![Q_j]\!]_A \leftarrow \bigoplus_{h_i \in \mathsf{hs}} [\![V_{h_i(y_j))}]\!]_A$$

In case of a match, the encrypted query will be $[\![Q_j]\!]_A = [\![P_\infty]\!]$, i.e. an encryption of zero, otherwise Q_j will be some random point in $E(\mathbb{F}_p)$. Furthermore, *Bob* will add a blinding factor to every encrypted query $[\![Q_j]\!]_A$ such that *Alice* will not learn of the query result:

$$z_j \xleftarrow{R} \mathbb{Z}_n$$
$$[\![Z_j]\!]_A \leftarrow \mathsf{Encrypt}(\mathsf{pk}_A, \mathsf{Encode}(z_j))$$
$$[\![\tilde{Q}_j]\!]_A \leftarrow [\![Q_j]\!]_A \oplus [\![Z_j]\!]_A$$

This vector of encrypted blinded queries $[\![\widetilde{\mathsf{qs}}]\!]_A = ([\![\tilde{Q}_1]\!]_A, ...)$ is then sent back to *Alice*.

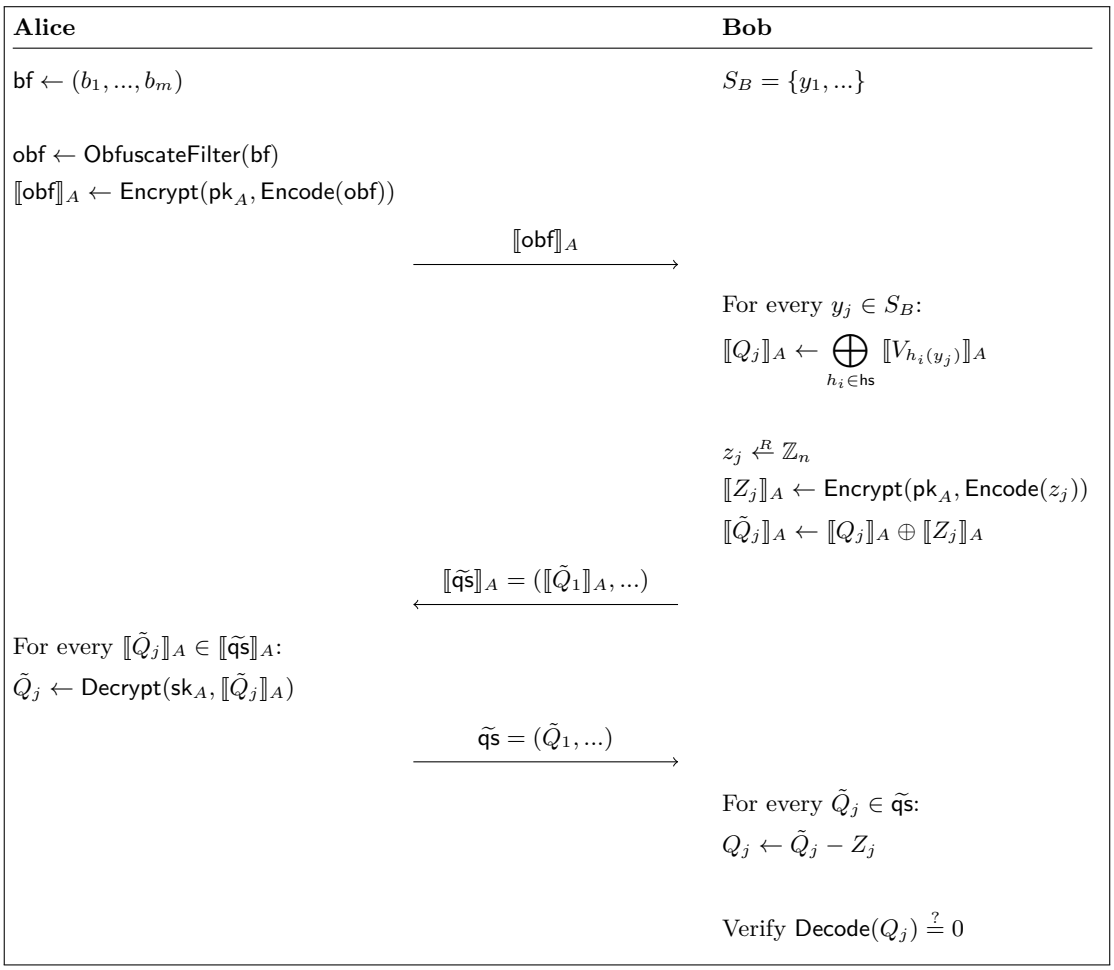


Figure 11: An overview of the transactions and computations in our Bloom filter-based record lookup protocol for some filter provider *Alice* and a filter consumer *Bob*.

7.4.4 Result Retrieval

Alice can now decrypt each blinded query $[\![\tilde{Q}_j]\!]_A$:

$$\tilde{Q}_j \leftarrow \mathsf{Decrypt}(\mathsf{sk}_A, [\![\tilde{Q}_j]\!]_A)$$

The resulting vector $\widetilde{\mathsf{qs}} = (\tilde{Q}_1, ...)$ is then sent back to *Bob*, who can then remove the blinding factor for each $\tilde{Q}_j$:

$$Q_j \leftarrow \tilde{Q}_j - Z_j$$

For each query Q_j, *Bob* can then check for a positive query result as follows:

$$\mathsf{Decode}(Q_j) \stackrel{?}{=} 0$$

In case there is a positive filter match with element $y_j \in S_B$, decoding will result in a zero, otherwise $Q_j = kG$ will be an random unknown value to *Bob*, thus *Bob* will most likely not be able to decode the query and only learns that the query result was negative.

7.5 Privacy Analysis

For the evaluation of the privacy of our protocol, we assume the semi-honest scenario, where an adversary can try to learn about the other party's inputs and (intermediate) outputs, while adhering to the protocol. For this purpose, we use simulation-based security proofs based on the *real-vs.-ideal paradigm* [44]. Here, we assume an *ideal scenario*, where all protocol computations are performed by a trusted third party given only the inputs of the other parties.

For simulation-based security proofs, a protocol Π for the computation of some function f is said to be computed privately, if we can create a simulation S of the protocol in the ideal scenario using only the input and output of either corrupted party, where the (intermediate) results are computationally indistinguishable (denoted by $\stackrel{c}{\equiv}$) from the party's view of the protocol view^{Π} in the real scenario:

$$\{S_A(x, f_A(x,y))_{x,y\in\{0,1\}^*}\} \stackrel{c}{\equiv} \{\mathsf{view}_A^{\Pi}(x,y)_{x,y\in\{0,1\}^*}\}$$
$$\{S_B(y, f_B(x,y))_{x,y\in\{0,1\}^*}\} \stackrel{c}{\equiv} \{\mathsf{view}_B^{\Pi}(x,y)_{x,y\in\{0,1\}^*}\}$$

where x and y are the protocol inputs of parties A and B, and f_a and f_b the outputs of both parties.

In the next sections, we will provide simulations proofs for the cases that either of the parties has been corrupted.

7.5.1 Corrupted Filter Consumer

Assuming that the filter consumer, *Bob*, has been compromised, we can create a simulation of the protocol in the ideal scenario using only *Bob*'s real input and output. For this purpose, we create some Bloom filter bf using the hash functions hs that conforms the bit positions that were set according to the query output that *Bob* receives in the real scenario and the corresponding elements from *Bob*'s input data set S_B. We can then obfuscate and encrypt the filter using *Alice*'s public key pk_A onto $E(\mathbb{F}_p)$, as we do not need to be able to decrypt any data for the simulation. Furthermore, since nothing can be learned from a valid set of hash functions hs on itself, it is considered a public parameter of the system. Now, the obfuscated filter $[\![\mathsf{obf}]\!]_A$ and the specification of the hash functions hs are indistinguishable from the filter transaction (as shown in section 7.4.2) in the real scenario.

Next, in our simulation, we can create a set of queries qs using *Bob*'s data set S_B the obfuscated filter $[\![\mathsf{obf}]\!]_A$ as in the real scenario. These queries can then be blinded using some $z_j \stackrel{R}{\leftarrow} \mathbb{Z}_n$ producing a set of blinded queries $[\![\widetilde{\mathsf{qs}}]\!]$. Here, the queries $[\![\widetilde{\mathsf{qs}}]\!]$ will be indistinguishable from the real query transaction (as shown in section 7.4.3) due to the properties of the ElGamal cryptosystem.

Thereafter, we can decrypt the encrypted blinded queries resulting in a set blinded queries $\widetilde{\mathsf{qs}}$. Due to the blinding factors, this set of queries is then indistinguishable from the result retrieval transaction (as shown in section 7.4.4) in the real scenario.

Finally, in the simulation, we can simply reproduce the protocol output *Bob* receives in the real scenario. The protocol is therefore computed privately in the semi-honest scenario in the case that the filter consumer, *Bob*, is corrupted.

7.5.2 Corrupted Filter Provider

For the case where the filter provider, *Alice*, has been compromised, we will create a simulation for the protocol using only *Alice*'s Bloom filter bf as input of the protocol, as *Alice* receives no output.

First, for the simulation, we can select a vector of m uniformly random elements in $E(\mathbb{F}_p)$ for the initial filter transaction (as shown in section 7.4.2) in the real scenario, making it indistinguishable from the simulation due to the properties of the ElGamal cryptosystem.

Similarly, for the querying transaction (as shown in section 7.4.3), we can do the same to simulate the queries $[\![\widetilde{\mathsf{qs}}]\!]$ as they are blinded, thereby resulting in a uniformly distributed value in $E(\mathbb{F}_p)$ after decryption, therefore being undistinguishable from the real transcript of the protocol.

In this case, the final result retrieval transaction does not matter, as *Alice* does not receive any further transactions based on it. Therefore, the protocol is also secure in the semi-honest scenario when the filter provider, *Alice*, is corrupted.

7.6 Utility Analysis

As our approach is based on Bloom filters in the encrypted domain, the utility will will be the same as for the regular case discussed in section 5.1.2 depending on the parameters for the creation of the filter. However, since our interactive protocol does not require a higher false positive rate as the sole privacy utensil and we can mitigate the effect of brute-force attacks, the false positive rate can be chosen more freely according to the utility required for the use case.

7.7 Efficiency Analysis

In this section, we will discuss the computational and communication complexities of every step for the protocol, as well as an empirical analysis of computational performance.

7.7.1 Computational Complexity

In order to evaluate the computation complexity of our proposed protocol, we will focus on the time complexity, as this will likely be a more limiting factor than the space complexity or the communication complexity, which will be discussed later. For this purpose, we will use the cost of a curve point addition T_{add} and the cost of point doubling T_{double} as the base units, as these are the primary operations in our protocol.

For scalar curve point multiplication, we assume that a *double-and-add* algorithm is used. Furthermore, we will assume that every operation has the worst-case performance. Using these assumptions, we can define the worst-case cost of a scalar multiplication in a finite field $\mathbb{F}_p$ as follows:

$$T_{\text{mul}} = \lfloor \log_2 p \rfloor T_{\text{double}} + \lfloor \log_2 p \rfloor T_{\text{add}}$$

Furthermore, we will define the cost of an Encrypt operation as $T_{\text{encrypt}} := 2T_{\text{mul}} + T_{\text{add}}$, as is shown in section 7.3.1, and the cost of a Decrypt as $T_{\text{decrypt}} := T_{\text{mul}} + T_{\text{add}}$, as is shown in section 7.3.1. Similarly, the Encode operation uses a single multiplication as shown in section 7.3.1, therefore we can define the cost as $T_{\text{encode}} := T_{\text{mul}}$. For the Decode operation, however, only one or more comparisons are required at runtime, as the possible decodable value(s) can be precomputed.

Filter Creation
For the creation of an encrypted filter, *Alice* first needs to create an obfuscated Bloom filter as shown in section 7.3.2. However, since the computational complexity of the CreateFilter operation is trivial compared to the required curve point multiplications for encoding and encryption, we will not consider this cost.

For every filter position, *Alice* needs to encode and encrypt a single plaintext, as is shown in section 7.4.2. As the number of filter positions depends on the size of *Alice*'s data set S_A, where $n = |S_A|$, we define the computational cost for the creation of a filter as follows:

$$\begin{aligned} T_{\text{filter}}(S_A) &:= m(T_{\text{encode}} + T_{\text{encrypt}}) \\ &= |S_A| \frac{\ln p}{(\ln 2)^2} (3T_{\text{mul}} + T_{\text{add}}) \end{aligned}$$

Since the false positive rate p is considered a system parameter, the creation of a filter is linear for both, point addition and point doubling, regarding to the size of *Alice*'s data set S_A, i.e. $T_{\text{filter}} \in \mathcal{O}(|S_A|)$.

Querying & Result Retrieval
In order to perform a single query, *Bob* has to add up k filter positions and a some blinding factor, as is shown in section 7.4.3. The addition of the plaintext blinding factor requires an encoding and an encryption step, therefore, in order to query all elements of *Bob*'s data set S_B:

$$\begin{aligned} T_{\text{batch,B}}(S_B) &:= |S_B|(T_{\text{encode}} + T_{\text{encrypt}}) \\ &= |S_B|(3T_{\text{mul}} + T_{\text{add}}) \end{aligned}$$

Next, *Alice* needs to decrypt every query they received from *Bob*:

$$\begin{aligned} T_{\text{batch,A}}(S_B) &:= |S_B| T_{\text{decrypt}} \\ &= |S_B|(T_{\text{mul}} + T_{\text{add}}) \end{aligned}$$

For the last step in the querying protocol, *Bob* removes the blinding factor and decodes the value of every result. However, as was mentioned, decoding does not require any point additions or doublings at runtime, therefore this step does not induce any significant additional cost.

The computational complexity for both, *Alice* and *Bob*, will therefore be linear in respect to *Bob*'s data set size S_B for point addition and doubling, i.e. $T_{\text{batch,B}}(S_B) \in \mathcal{O}(|S_B|)$ as well as $T_{\text{batch,A}}(S_B) \in \mathcal{O}(|S_B|)$.

The total query time, excluding network overhead, can then be written as follows:

$$\begin{aligned} T_{\text{batch}} &= T_{\text{batch,B}} + T_{\text{batch,A}} \\ &= |S_B|(4T_{\text{mul}} + 2T_{\text{add}}) \end{aligned}$$

Therefore, the total batch query time T_{batch} is also linear in $|S_B|$.

7.7.2 Communication Complexity

In this section, we will discuss the communication complexity of each phase of the protocol. For this purpose, we define the complexity in terms of minimum amount of bits transferred for each phase.

As all transferred messages are curve points, the size of a message depend on the curve $E(G)$ being used and

the corresponding order of the group G. Therefore, we define the size of a (packed) curve point in bits as follows:

$$L_{\text{point}} := \lceil \log_2 (|G|) \rceil + 1$$

As ElGamal ciphertexts on $E(G)$ consist of two curve points, we can define the bit size of a ciphertext correspondingly:

$$L_{\text{ciphertext}} = 2L_{\text{point}}$$

In table 1, an overview of the size of a curve point in bits for different curves is shown.

Curve ($E(G)$)	**Bits Per Point (L_{point})**
NIST224	$224 + 1$
NIST256	$256 + 1$
NIST384	$384 + 1$
NIST521	$521 + 1$

Table 1: Bits required for a point on different curves.

Filter Creation
After the creation of the filter by *Alice*, each encrypted filter position needs to be transferred to *Bob*, therefore the ciphertexts being transmitted during the filter creation phase is equal to the size of the Bloom filter m. As the filter size is dependent on the number of elements n and the false positive rate p, we can define the number of transferred bits using eq. (5):

$$\begin{aligned} L_{\text{filter}}(S_A) &:= mL_{\text{ciphertext}} \\ &= 2L_{\text{point}} \frac{n \ln(p)}{(\ln 2)^2} \\ &= 2L_{\text{point}} \frac{|S_A| \ln(p)}{(\ln 2)^2} \end{aligned}$$

As the false positive rate p and the curve point size L_{point} are considered to be system parameters, the number of ciphertexts transmitted by *Alice* for filter distribution is therefore linear with respect to the amount of elements $|S_A|$, i.e. $L_{\text{filter}}(S_A) \in \mathcal{O}(|S_A|)$. In comparison to the record lookup scheme from section 6, each encrypted filter is $2L_{\text{point}}$ times larger the original filter. For the case of a NIST256 curve, this results in a 514 times transaction size increase, as is shown in table 1.

Querying & Result Retrieval
For each separate queried element, *Bob* needs to send a single ciphertext to *Alice* and then retrieves the result as a single curve point. Given *Bob*'s set of elements S_B, we can define the communication complexity of as single query as follows:

$$L_{\text{query}} = L_{\text{ciphertext}} + L_{\text{point}} = 3L_{\text{point}}$$

Using this, the total number of transmitted bits when querying *Bob*'s entire set of elements S_B is equal to:

$$L_{\text{batch}}(S_B) := L_{\text{query}}|S_B| = 3L_{\text{point}}|S_B|$$

Therefore, the communication complexity for querying is linear in respect to the size of *Bob*'s data set, i.e. $L_{\text{batch}}(S_B) \in |\mathcal{S}_{\mathcal{B}}|$.

7.7.3 Empirical Computational Performance

For the purpose of evaluating the computational performance of our protocol, a Python prototype using the Charm framework [45] was created. It should be noted, however, that this implementation was not particularly optimised for speed and, therefore, only serves for demonstrational purposes. The source code for the prototype will be published as an open-source project in the near future. Furthermore, since we already evaluated the theoretical communication complexity in the previous section, we will exclude any network overhead in this analysis.

In this section, we will evaluate the performance of the filter setup for different filter sizes m, as well as the performance of a single query for different amounts of hash functions k, for which we will use the SHA3 hash function. Furthermore, for this purpose, we will use the same benchmarking setup as mentioned in section 6.3.

The evaluation for cost the filter setup phase T_{setup} is shown in fig. 12. Furthermore, the evaluation of the single query performance T_{query} is shown in fig. 13. Here, the querying performance is lower than that for the record lookup scheme in section 6, however it is sufficiently fast to query larger batches of records. While the setup cost is significantly larger due to all the encryptions, it is most likely not an issue for record lookup if databases are not updating this frequently.

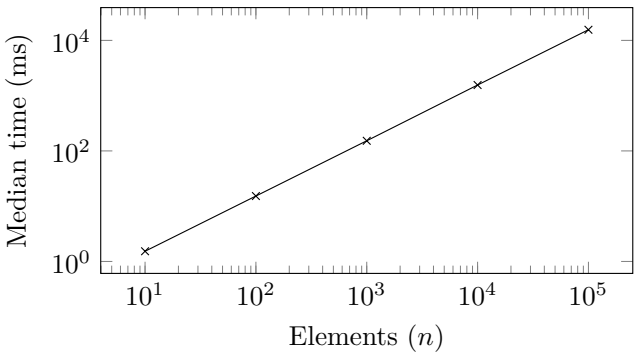


Figure 12: Comparison of the median filter obfuscation performance T_{setup} for different amounts of elements using the NIST256 curve. Furthermore, the number of SHA3 hash functions k is set to 10 and the amount of elements in the filter is chosen correspondingly.

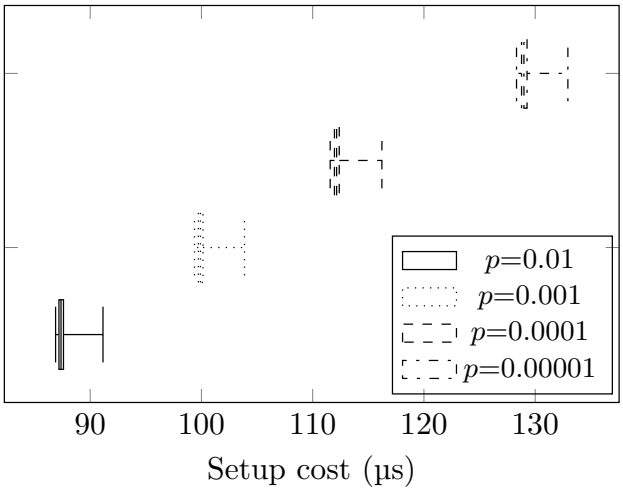


Figure 13: Performance comparison of single (positive) queries T_{query} for different false positive rates using the NIST256 curve. Furthermore, the element count n is 10 and the filter size is set correspondingly. In this figure, the line represents the median, the box denotes the range from the 25th up to the 75th percentile, and the whiskers represents the 2nd and the 98th percentile.

7.8 Functionality Analysis

Lastly, we consider any additional functionality of our protocol, as mentioned in section 5.1.4.

In our proposed protocol, when sharing a Bloom filter with additional elements, a filter provider can simply generate an new key pair for the encryption of the filter and use this key for all future transactions with consumer. The consumer will now be enforced to use the new filter and new public key, as using former versions will product results indistinguishable from uniform randomness. The probability that the result will then be trivially decodable by the consumer is therefore negligible, i.e. in this case the consumer will only retrieve negative querying results.

Indeed, any information shared by queries on the previous filter is still available to the other party, and can therefore leak data about changes to the data sets or changing false positives. The first case, leakage about element changes is, however, inherent to our goal of information sharing. Furthermore, using the same setup parameters for new filter versions if possible can limit leakage about false positives as described in section 6.1.4. Here, the probability that the same false positives are included in a new version of a filter are higher, the smaller the differences between the filters.

Furthermore, the deletion of elements can be done similarly, where the filter provider creates a new key pair for a new Bloom filter and the corresponding transactions. In this case, no data of this previous filter will accessible to the filter consumer, except the information that has already been queried. Furthermore, it is possible for the filter provider to limit the amount of queries that are performed in order to prevent the creation of a local database by the filter consumer, thereby limiting this type of leakage.

8 Conclusions

We have provided a set of metrics for the evaluation of the privacy, efficiency and utility aspects of different record lookup schemes allowing for false positives.

Using these metrics, we have shown that Bloom filter-based record lookup schemes, as the one introduced in [4], can be privacy-preserving by using a shared secret for encoding the data records for scenarios including an outsider adversary. In the case of an insider adversary, such schemes can also provide some level of privacy depending on the configuration of the filter and the data being stored. For the first time, we have furthermore shown that, in the insider scenario, the distribution of new filters, as will happen in the case of database updates, can significantly reduce the privacy as the scheme does not provide measures for guaranteeing the deletion of data records.

Finally, building on this Bloom-filter based record lookup scheme, we have presented a novel interactive record lookup protocol using homomorphic encryption that is practically applicable with regards to lookup performance and network transaction sizes. Our lookup protocol, furthermore, provides a means to limit the effect of brute-force attacks through rate-limiting in an insider setting. Lastly, as it is an interactive protocol, queries using older filters can be prevented, thereby allowing for the addition and the secure deletion of elements without an impact on the privacy.

9 Future Work

In this paper, we have only considered the semi-honest scenario for the privacy analysis of our encrypted Bloom filter-based record lookup protocol, as this resembles the situation of the unencrypted scheme in section 6. A natural extension would, therefore, be an adaption of the protocol to the malicious scenario [44]w here, an adversary can choose to deviate from the protocol in order to learn about any private inputs or outputs.

Moreover, while we have studied Bloom filter-based schemes in this paper, other probabilistic data structures could be considered for record lookup schemes in order to further improve privacy and efficiency. Here, one possibility is the Cuckoo filter [46], which is a data structure based on Cuckoo hashing [47] that has properties similar to Bloom filters, as it allows for false positives, however it is in many cases more storage efficient than a Bloom filter. Furthermore, an efficient

Cuckoo hashing-based PSI protocol has been presented in [48], which could possibly be adapted for use in private record lookup schemes allowing for false positives.

Acknowledgements

This paper was supported by the Dutch *Fiscal Information and Investigation Service* (FIOD). We thank Udo Kroon and Gonnie de Graaff from the Dutch Ministry of Justice and Security for their guidance. Additionally, we thank Tim van de Kamp for his help and feedback.

References

[1] Ian P. Johnson. *Report: Patients data lay unsecured after NaProt app glitch.* (Accessed on 03/11/2019). Mar. 2018. URL: https://p.dw.com/p/2uRYe.

[2] Linda Carrol. *Health data breaches on the rise.* (Accessed on 03/09/2019). Sept. 2018. URL: https://reut.rs/2xBMAlg.

[3] Kate O'Flaherty. *Why Cyber-Criminals Are Attacking Healthcare.* (Accessed on 03/09/2019). Oct. 2018. URL: https://www.forbes.com/sites/kateoflahertyuk/2018/10/05/why-cyber-criminals-are-attacking-healthcare-and-how-to-stop-them/.

[4] M. Little, Santosh K. Shrivastava, and Neil Speirs. "Using Bloom Filters to Speed-up Name Lookup in Distributed Systems." In: *Comput. J.* 45 (Jan. 2002), pp. 645–652. DOI: 10.1093/comjnl/45.6.645.

[5] Burton H. Bloom. "Space/Time Trade-offs in Hash Coding with Allowable Errors". In: *Commun. ACM* 13.7 (July 1970), pp. 422–426. ISSN: 0001-0782. DOI: 10.1145/362686.362692. URL: http://doi.acm.org/10.1145/362686.362692.

[6] Andrei Broder and Michael Mitzenmacher. "Network Applications of Bloom Filters: A Survey". In: *Internet Math.* 1.4 (2003), pp. 485–509. URL: https://projecteuclid.org/euclid.im/1109191032.

[7] Christian Esteve Rothenberg, Carlos Alberto Braz Macapuna, and Alexander Wiesmaier. "In-packet Bloom filters: Design and networking applications". In: *CoRR* abs/0908.3574 (2009). arXiv: 0908.3574. URL: http://arxiv.org/abs/0908.3574.

[8] Li Fan, Pei Cao, J. Almeida, and A. Z. Broder. "Summary cache: a scalable wide-area Web cache sharing protocol". In: *IEEE/ACM Transactions on Networking* 8.3 (June 2000), pp. 281–293. ISSN: 1063-6692. DOI: 10.1109/90.851975.

[9] Paulo Sérgio Almeida, Carlos Baquero, Nuno Preguiça, and David Hutchison. "Scalable Bloom Filters". In: *Information Processing Letters* 101.6 (2007), pp. 255–261. ISSN: 0020-0190. DOI: https://doi.org/10.1016/j.ipl.2006.10.007. URL: http://www.sciencedirect.com/science/article/pii/S0020019006003127.

[10] James K. Mullin. "A Second Look at Bloom Filters". In: *Commun. ACM* 26.8 (Aug. 1983), pp. 570–571. ISSN: 0001-0782. DOI: 10.1145/358161.358167. URL: http://doi.acm.org/10.1145/358161.358167.

[11] Ken Christensen, Allen Roginsky, and Miguel Jimeno. "A New Analysis of the False Positive Rate of a Bloom Filter". In: *Inf. Process. Lett.* 110.21 (Oct. 2010), pp. 944–949. ISSN: 0020-0190. DOI: 10.1016/j.ipl.2010.07.024. URL: http://dx.doi.org/10.1016/j.ipl.2010.07.024.

[12] Li Fan, Pei Cao, Jussara Almeida, and Andrei Z. Broder. "Summary Cache: A Scalable Wide-area Web Cache Sharing Protocol". In: *IEEE/ACM Trans. Netw.* 8.3 (June 2000), pp. 281–293. ISSN: 1063-6692. DOI: 10.1109/90.851975. URL: http://dx.doi.org/10.1109/90.851975.

[13] A. L. Tatarowicz, C. Curino, E. P. C. Jones, and S. Madden. "Lookup Tables: Fine-Grained Partitioning for Distributed Databases". In: *2012 IEEE 28th International Conference on Data Engineering.* Apr. 2012, pp. 102–113. DOI: 10.1109/ICDE.2012.26.

[14] Lena Wiese. "Clustering-based fragmentation and data replication for flexible query answering in distributed databases". In: *Journal of Cloud Computing* 3.1 (Oct. 2014), p. 18. ISSN: 2192-113X. DOI: 10.1186/s13677-014-0018-0. URL: https://doi.org/10.1186/s13677-014-0018-0.

[15] C. Quantin, H. Bouzelat, and L. Dusserre. "A computerized record hash coding and linkage procedure to warrant epidemiological follow-up data security". In: vol. 43. cited By 9. 1997, pp. 339–342. DOI: 10.3233/978-1-60750-887-8-339. URL: https://www.scopus.com/inward/record.uri?eid=2-s2.0-0031317926&doi=10.3233%2f978-1-60750-887-8-339&partnerID=40&md5=3692dc6e670e489810d27afa3cc405f0.

[16] Rainer Schnell, Tobias Bachteler, and Jörg Reiher. "Privacy-preserving record linkage using Bloom filters". In: *BMC Medical Informatics and Decision Making* 9.1 (Aug. 2009), p. 41. ISSN: 1472-6947. DOI: 10.1186/1472-6947-9-41. URL: https://doi.org/10.1186/1472-6947-9-41.

[17] Mehmet Kuzu, Murat Kantarcioglu, Elizabeth Durham, and Bradley Malin. "A Constraint Satisfaction Cryptanalysis of Bloom Filters in Private Record Linkage". In: *Proceedings of the 11th International Conference on Privacy Enhancing Technologies*. PETS'11. Waterloo, ON, Canada: Springer-Verlag, 2011, pp. 226–245. ISBN: 978-3-642-22262-7. URL: http://dl.acm.org/citation.cfm?id=2032162.2032175.

[18] Frank Niedermeyer, Simone Steinmetzer, Martin Kroll, and Rainer Schnell. "Cryptanalysis of Basic Bloom Filters Used for Privacy Preserving Record Linkage". In: *Journal of Privacy and Confidentiality* 6 (Dec. 2014). DOI: 10.29012/jpc.v6i2.640.

[19] Peter Christen, Rainer Schnell, Dinusha Vatsalan, and Thilina Ranbaduge. "Efficient Cryptanalysis of Bloom Filters for Privacy-Preserving Record Linkage". In: Apr. 2017, pp. 628–640. ISBN: 978-3-319-57453-0. DOI: 10.1007/978-3-319-57454-7_49.

[20] Mohammad Alaggan, Sébastien Gambs, and Anne-Marie Kermarrec. "BLIP: Non-interactive Differentially-Private Similarity Computation on Bloom Filters". In: *14th International Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS 2012)*. Toronto, Canada, Oct. 2012. URL: https://hal.inria.fr/hal-00724829.

[21] Giuseppe Bianchi, Lorenzo Bracciale, and Pierpaolo Loreti. ""Better Than Nothing" Privacy with Bloom Filters: To What Extent?" In: *Proceedings of the 2012 International Conference on Privacy in Statistical Databases*. PSD'12. Palermo, Italy: Springer-Verlag, 2012, pp. 348–363. ISBN: 978-3-642-33626-3. DOI: 10.1007/978-3-642-33627-0_27. URL: http://dx.doi.org/10.1007/978-3-642-33627-0_27.

[22] Pierangela Samarati and Latanya Sweeney. *Protecting Privacy when Disclosing Information: k-Anonymity and Its Enforcement through Generalization and Suppression*. Tech. rep. 1998.

[23] F. S. Tabataba and M. R. Hashemi. "Improving false positive in Bloom filter". In: *2011 19th Iranian Conference on Electrical Engineering*. May 2011, pp. 1–1.

[24] Hyesook Lim, Nara Lee, Jungwon Lee, and Changhoon Yim. "Reducing False Positives of a Bloom Filter using Cross-Checking Bloom Filters". In: 8 (July 2014), pp. 1865–1877.

[25] Alex Davidson and Carlos Cid. "Computing Private Set Operations with Linear Complexities". In: *IACR Cryptology ePrint Archive* 2016 (2016), p. 108. URL: http://eprint.iacr.org/2016/108.

[26] Auguste Kerckhoffs. "La cryptographie militaire". In: *Journal des sciences militaires* IX (Jan. 1883), pp. 5–83.

[27] Udo Kroon. "Ma3tch: Privacy and knowledge: 'Dynamic networked collective intelligence'". In: *2013 IEEE International Conference on Big Data* (2013).

[28] C. E. Shannon. "A mathematical theory of communication". In: *The Bell System Technical Journal* 27.3 (July 1948), pp. 379–423. ISSN: 0005-8580. DOI: 10.1002/j.1538-7305.1948.tb01338.x.

[29] Catherine A. Meadows. "A More Efficient Cryptographic Matchmaking Protocol for Use in the Absence of a Continuously Available Third Party". In: *Proceedings of the 1986 IEEE Symposium on Security and Privacy, Oakland, California, USA, April 7-9, 1986*. 1986, pp. 134–137. DOI: 10.1109/SP.1986.10022. URL: https://doi.org/10.1109/SP.1986.10022.

[30] Michael J. Freedman, Kobbi Nissim, and Benny Pinkas. "Efficient Private Matching and Set Intersection". In: *Advances in Cryptology - EUROCRYPT 2004, International Conference on the Theory and Applications of Cryptographic Techniques, Interlaken, Switzerland, May 2-6, 2004, Proceedings*. Vol. 3027. Lecture Notes in Computer Science. Springer, 2004, pp. 1–19. DOI: 10.1007/978-3-540-24676-3_1. URL: https://iacr.org/archive/eurocrypt2004/30270001/pm-eurocrypt04-lncs.pdf.

[31] Moni Naor and Benny Pinkas. "Oblivious Transfer and Polynomial Evaluation". In: *Proceedings of the Thirty-first Annual ACM Symposium on Theory of Computing*. STOC '99. Atlanta, Georgia, USA: ACM, 1999, pp. 245–254. ISBN: 1-58113-067-8. DOI: 10.1145/301250.301312. URL: http://doi.acm.org/10.1145/301250.301312.

[32] Pascal Paillier. "Public-key Cryptosystems Based on Composite Degree Residuosity Classes". In: *Proceedings of the 17th International Conference on Theory and Application of Cryptographic Techniques*. EUROCRYPT'99. Prague, Czech Republic: Springer-Verlag, 1999, pp. 223–238. ISBN: 3-540-65889-0. URL: http://dl.acm.org/citation.cfm?id=1756123.1756146.

[33] Carmit Hazay and Kobbi Nissim. "Efficient Set Operations in the Presence of Malicious Adversaries". In: *Proceedings of the 13th International Conference on Practice and Theory in Public Key Cryptography*. PKC'10. Paris, France: Springer-Verlag, 2010, pp. 312–331. ISBN: 3-642-13012-7, 978-3-642-13012-0. DOI: 10.1007/978-3-642-13013-7_19. URL: http://dx.doi.org/10.1007/978-3-642-13013-7_19.

[34] Taher El Gamal. "A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms". In: *Proceedings of CRYPTO 84 on Advances in Cryptology*. Santa Barbara, California, USA: Springer-Verlag New York, Inc., 1985, pp. 10–18. ISBN: 0-387-15658-5. URL: http://dl.acm.org/citation.cfm?id=19478.19480.

[35] Florian Kerschbaum. "Public-key Encrypted Bloom Filters with Applications to Supply Chain Integrity". In: *Proceedings of the 25th Annual IFIP WG 11.3 Conference on Data and Applications Security and Privacy*. DBSec'11. Richmond, VA: Springer-Verlag, 2011, pp. 60–75. ISBN: 978-3-642-22347-1. URL: http://dl.acm.org/citation.cfm?id=2029896.2029906.

[36] Shafi Goldwasser and Silvio Micali. "Probabilistic Encryption &Amp; How to Play Mental Poker Keeping Secret All Partial Information". In: *Proceedings of the Fourteenth Annual ACM Symposium on Theory of Computing*. STOC '82. San Francisco, California, USA: ACM, 1982, pp. 365–377. ISBN: 0-89791-070-2. DOI: 10.1145/800070.802212. URL: http://doi.acm.org/10.1145/800070.802212.

[37] Florian Kerschbaum. "Outsourced Private Set Intersection Using Homomorphic Encryption". In: *Proceedings of the 7th ACM Symposium on Information, Computer and Communications Security*. ASIACCS '12. Seoul, Korea: ACM, 2012, pp. 85–86. ISBN: 978-1-4503-1648-4. DOI: 10.1145/2414456.2414506. URL: http://doi.acm.org/10.1145/2414456.2414506.

[38] H. Perl, Y. Mohammed, M. Brenner, and M. Smith. "Fast confidential search for bio-medical data using Bloom filters and Homomorphic Cryptography". In: *2012 IEEE 8th International Conference on E-Science*. Oct. 2012, pp. 1–8. DOI: 10.1109/eScience.2012.6404484.

[39] N. P. Smart and F. Vercauteren. "Fully Homomorphic Encryption with Relatively Small Key and Ciphertext Sizes". In: *Proceedings of the 13th International Conference on Practice and Theory in Public Key Cryptography*. PKC'10. Paris, France: Springer-Verlag, 2010, pp. 420–443. ISBN: 3-642-13012-7, 978-3-642-13012-0. DOI: 10.1007/978-3-642-13013-7_25. URL: http://dx.doi.org/10.1007/978-3-642-13013-7_25.

[40] Changyu Dong, Liqun Chen, and Zikai Wen. "When private set intersection meets big data: an efficient and scalable protocol". In: *Proceedings of the 2013 ACM SIGSAC conference on Computer communications security*. CCS '13. Berlin, Germany: ACM, 2013, pp. 789–800. ISBN: 978-1-4503-2477-9. DOI: 10.1145/2508859.2516701. URL: http://doi.acm.org/10.1145/2508859.2516701.

[41] Benny Pinkas, Thomas Schneider, and Michael Zohner. "Faster Private Set Intersection Based on OT Extension". In: *Proceedings of the 23rd USENIX Security Symposium, San Diego, CA, USA, August 20-22, 2014*. 2014, pp. 797–812. URL: https://www.usenix.org/conference/usenixsecurity14/technical-sessions/presentation/pinkas.

[42] Benny Pinkas, Thomas Schneider, and Michael Zohner. "Scalable Private Set Intersection Based on OT Extension". In: *IACR Cryptology ePrint Archive* 2016 (2016), p. 930. URL: http://eprint.iacr.org/2016/930.

[43] Rolf Egert, Marc Fischlin, David Gens, Sven Jacob, Matthias Senker, and Jörn Tillmanns. "Privately Computing Set-Union and Set-Intersection Cardinality via Bloom Filters". In: *Information Security and Privacy - 20th Australasian Conference, ACISP 2015, Brisbane, QLD, Australia, June 29 - July 1, 2015, Proceedings*. 2015, pp. 413–430. DOI: 10.1007/978-3-319-19962-7_24. URL: https://doi.org/10.1007/978-3-319-19962-7%5C_24.

[44] Oded Goldreich. *Foundations of Cryptography: Volume 2, Basic Applications*. New York, NY, USA: Cambridge University Press, 2004. ISBN: 0521830842.

[45] Joseph A. Akinyele, Christina Garman, Ian Miers, Matthew W. Pagano, Michael Rushanan, Matthew Green, and Aviel D. Rubin. "Charm: a framework for rapidly prototyping cryptosystems". In: *Journal of Cryptographic Engineering* 3.2 (2013), pp. 111–128. ISSN: 2190-8508. DOI: 10.1007/s13389-013-0057-3. URL: http://dx.doi.org/10.1007/s13389-013-0057-3.

[46] Bin Fan, Dave G. Andersen, Michael Kaminsky, and Michael D. Mitzenmacher. "Cuckoo Filter: Practically Better Than Bloom". In: *Proceedings of the 10th ACM International on Conference on Emerging Networking Experiments and Technologies*. CoNEXT '14. Sydney, Australia: ACM, 2014, pp. 75–88. ISBN: 978-1-4503-3279-8. DOI: 10.1145/2674005.2674994. URL: http://doi.acm.org/10.1145/2674005.2674994.

[47] Rasmus Pagh and Flemming Friche Rodler. "Cuckoo Hashing". In: *Algorithms — ESA 2001*. Ed. by Friedhelm Meyer auf der Heide. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pp. 121–133. ISBN: 978-3-540-44676-7.

[48] Hao Chen, Kim Laine, and Peter Rindal. "Fast Private Set Intersection from Homomorphic Encryption". In: *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017*. 2017, pp. 1243–1255. DOI: 10.1145/3133956.3134061. URL: https://doi.org/10.1145/3133956.3134061.