Efficient and Confidentiality-Preserving Content-Based Publish/Subscribe with Prefiltering

Raphaël Barazzutti, Pascal Felber, *Member, IEEE*, Hugues Mercier, *Member, IEEE*, Emanuel Onica, and Etienne Rivière, *Member, IEEE*

Abstract—Content-based publish/subscribe provides a loosely-coupled and expressive form of communication for large-scale distributed systems. Confidentiality is a major challenge for publish/subscribe middleware deployed over multiple administrative domains. Encrypted matching allows confidentiality-preserving content-based filtering but has high performance overheads. It may also prevent the use of classical optimizations based on subscriptions containment.

We propose a support mechanism that reduces the cost of encrypted matching, in the form of a *prefiltering* operator using Bloom filters and simple randomization techniques. This operator greatly reduces the amount of encrypted subscriptions that must be matched against incoming encrypted publications. It leverages subscription containment information when available, but also ensures that containment confidentiality is preserved otherwise. We propose containment obfuscation techniques and provide a rigorous security analysis of the information leaked by Bloom filters in this case. We conduct a thorough experimental evaluation of prefiltering under a large variety of workloads. Our results indicate that prefiltering is successful at reducing the space of subscriptions to be tested in all cases. We show that while there is a tradeoff between prefiltering efficiency and information leakage when using containment obfuscation, it is practically possible to obtain good prefiltering performance while securing the technique against potential leakages.

Index Terms—Publish/subscribe, Confidentiality, Security, Encrypted processing, Bloom filters

1 Introduction

Publish/subscribe [1], or *pub/sub* for short, is an appealing communication paradigm for building large-scale applications composed of dynamic entities that produce and consume data events according to complex and unpredictable workflows. It offers an indirect, decoupled communication model between application components that act as either *publishers* of, or *subscribers* to, data. Publishers and subscribers do not need to know one another but only exchange data via some pub/sub middleware. Subscribers simply inform the system about what new elements of data they wish to receive by registering a *subscription*. Conversely, publishers send new events to the system in the form of *publications*. It is then the responsibility of the pub/sub system to appropriately route each publication towards all subscribers with matching interests.

Pub/sub systems are often classified according to the expressiveness of the subscriptions they allow. The model of interest in this article is *content-based filtering*. In this model, publications contain a descriptive *header* representing their content, generally as values over a set of *attributes*. Subscriptions are composed of *predicates* specifying *constraints* over these attributes. An emblematic example of content-based pub/sub is that of a financial network supporting automated trading. Stock quotes form a flow of publications, each describing the quotation for a given stock (name, price, volume of exchange, time, ...), while subscribers are companies, other agencies, or private investors wishing to invest or speculate on stocks. Only a subset of the quotes is of interest to these subscribers, who express their specific interests by means of subscriptions. For instance, as illustrated by Figure 1, a user monitoring the stock market might register a subscription $s_1 = \{symbol = \texttt{ACME} \wedge price \geq 100 \wedge currency = \texttt{USD}\}$ to be notified of quotes for company ACME with share price greater than or equal to $100. A publication quote with content $p = \{symbol = \texttt{ACME}, price = 102.23, volume = 210425, currency = \texttt{USD}\}$ matches the subscription s_1 and is forwarded to the subscriber in the form of a *notification*. The publication p also matches s_2, but not s_3.

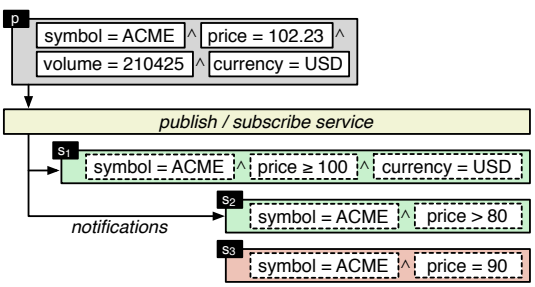


Fig. 1: A content-based pub/sub system.

1.1 Confidentiality-Preserving Filtering

Hosting publish/subscribe middleware on public or third party clouds yields many benefits, such as elasticity, costs reduction in personnel and infrastructure, ease of deployment, and availability. Pub/sub might even be offered as a service running on public clouds for connecting applications

Emanuel Onica is with Alexandru Ioan Cuza University of Iaşi, Romania; other authors are with the Institute of Computer Science, Université de Neuchâtel, Switzerland. emails: eonica@info.uaic.ro and first.last@unine.ch

through a high-throughput and scalable communication service [2]. However, cloud computing services raise major privacy protection concerns, especially in regard to data confidentiality. Recent research [3], [4] has shown that exploits at the hypervisor level, or exploits based on virtual machines collocation, could be used to gather private information. More general privacy issues have been raised regarding the non-uniform regulation access to stored data [5]. Cloud providers might store or migrate data in different locations, bound to different legislation granting different access rights to external entities (e.g., law enforcement agencies in the hosting countries). These rights and data migrations are not always transparent to the customer and can lead to unwanted access to the stored information. Data confidentiality is therefore one of the major hurdles for pub/sub middleware running on such infrastructures. As subscriptions' predicates must be compared against the headers of publications, the entity performing the routing must be trusted by all parties. Otherwise, the nature of subscriptions may reveal sensitive information about users. For instance, with the example of Figure 1, the availability of subscriptions may reveal investments strategies of the users.

A solution to address these privacy concerns is to assume that the pub/sub service runs on an untrusted environment by default and to use operations on encrypted publications and subscriptions to preserve their *confidentiality*. The filtering operation is thus performed without a decryption key. Unfortunately, existing encrypted filtering techniques have a prohibitive computational cost preventing their adoption for high-throughput content-based routing in untrusted environments despite their interesting properties. An interesting encrypted-filtering solution is asymmetric scalar-product preserving encryption (ASPE) [6]. ASPE relies on a distance-preserving but not distance-recoverable transformation, which was first used in the context of secure kNN query processing on databases [7]. The transformation performed by ASPE allows comparison between pairs of encrypted publications and subscriptions. Other schemes with comparable costs were also proposed [8], [9], [10], [11], [12].

1.2 Subscription Containment

Subscription containment is often used to speed up filtering [13], [14], [15]. We say that a subscription s_i *contains* another subscription s_j if any event that matches s_j also matches s_i, in other words if s_i is more general than s_j. For instance, in Figure 1, s_2 contains s_1 and s_3. When a subscription contains another subscription, any publication known not to match the former will never match the latter. Conversely, any publication known to match the latter will always match the former. Containment relationship creates a partial order on subscriptions, and by organizing them according to that order it is possible to quickly discard or accept many subscriptions at once. The ability to determine containment between subscriptions, even when encrypted, can pose a threat to confidentiality. For instance, in the example of Figure 1, three subscriptions for the same symbol name would be grouped together in the same containment subtree. Using domain-specific information on the relative popularity of stocks, an attacker may be able to derive information

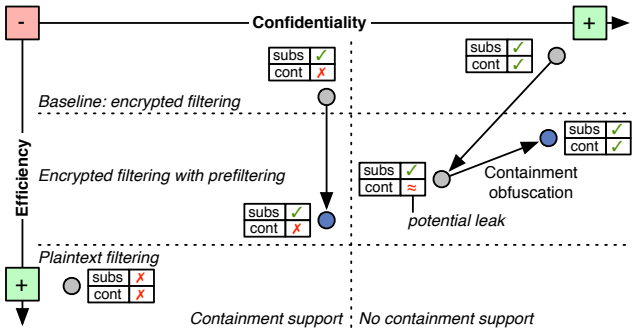


Fig. 2: High-level picture of the performance and confidentiality tradeoffs addressed in this paper (`subs` stands for subscriptions confidentiality and `cont` for containment confidentiality).

about the symbol associated to each subtree based on its size. Even without such domain-specific knowledge, containment information can be leveraged by an attacker to obtain the information that two particular subscribers have common interests. Similar observations were made in [8]. For this reason, some schemes completely disallow the possibility to determine containment [9], [12], whereas for other schemes containment determination is optional and only provided if requested by the application [6], [8], [10]. We name the ability of an encrypted matching scheme to prevent containment determination as *containment confidentiality*.

1.3 Our contributions

The main objective of this article is to reduce the prohibitive costs of encrypted filtering while preserving confidentiality. It is the extension of the work presented in [16], [17]. We provide a more comprehensive description of the security issues and motivations, complete the mathematical analysis, include all the proofs, expand numerical evaluations, discuss potential extensions, and more generally present our contributions in a cohesive fashion. Our main contributions are depicted in Figure 2 and summarized below. We consider both the case where containment determination is allowed by the scheme (left-hand side of Figure 2) and the case where containment information must remain confidential (right-hand side of Figure 2).

1.3.1 Prefiltering using Bloom filters

Our first contribution is an efficient *prefiltering* technique to significantly reduce the space of subscriptions that must be tested by the costly encrypted filtering engine during routing. Prefiltering followed by encrypted matching is a two-step process reminiscent of other work where an efficient preprocessing step imperfectly and quickly discards data to decrease the use of an accurate but costly processing task [18]. In a nutshell, the principle is to embed Bloom filters [19] inside publications and subscriptions. These Bloom filters encode the values carried by the publications and the equality constraints of the subscriptions. By testing the Bloom filters of subscriptions for inclusion in those of publications, one can efficiently determine the possibility for a message to match a subscription: if the test is negative, the message is guaranteed *not* to match; otherwise, it *might* match and must be further analyzed by the encrypted filtering engine. The prefiltering operation significantly improves the performance

of confidentiality-preserving encrypted matching, with or without the support for containment determination.

1.3.2 Containment obfuscation

We also consider the case where the base scheme provides containment confidentiality. Since prefiltering appends additional information to subscriptions (the Bloom filters), we analyze the power this additional information gives to an attacker. Indeed, despite their hash-based construction and their probabilistic nature, Bloom filters may still convey sensitive information. In particular, identical subscriptions yield equal Bloom filters, and contained subscriptions produce Bloom filters that are included in one another. As illustrated on the right-hand side of Figure 2, the use of prefiltering atop a scheme providing containment confidentiality may result in potential leaks of containment information–albeit strictly less so than by determining containment directly from encrypted subscriptions.

Our second contribution is a method for containment obfuscation as well as a comprehensive mathematical analysis of the impact of prefiltering on containment confidentiality. Containment obfuscation introduces randomness in the prefiltering process by randomly choosing a small number of hash functions from a larger set for each subscription equality constraint encoded in the Bloom filters. The main difficulty is the analysis of the dependencies introduced by hash functions collisions, which are cumbersome to model mathematically. We show that there is a fundamental tradeoff between the information that can be leaked to an attacker from the Bloom filters and the performance of the prefiltering operation. Fortunately, we provide evidence, both mathematically and using numerical simulations, that one can achieve efficient prefiltering and preserve containment confidentiality. More precisely, we show that prefiltering can improve the matching performance while providing almost no additional information to an attacker than what can be obtained from the encrypted subscriptions themselves.

1.4 Roadmap

This article is organized as follows. Our prefiltering algorithm is described in Section 2. The theoretical analysis of the performance-confidentiality tradeoff of the prefiltering scheme using containment obfuscation is presented in Section 3, with some of the proofs and the discussion of alternative attacks relegated to Appendices A and B. We present evaluation results on performance and security in Section 4, with additional evaluation figures in Appendix C. Related work is reviewed in Section 5. We conclude the paper in Section 6. Finally, we propose an extension to prefilter inequalities in Appendix D, which was never considered before.

2 Prefiltering Algorithm

In this section, we present a prefiltering operator based on Bloom filters embedded with subscriptions and publications. The operator can take advantage of containment relationships when available. The Bloom filters and the containment information can be used in isolation but provide the best performance when combined. Both aim at reducing the number of calls to the costly encrypted matching operation, which we denote as ENC-MATCH. They do so by quickly discarding some of the subscriptions that do not match (*negative matching*) and selecting some of the subscriptions that do match (*positive matching*) without calling ENC-MATCH. Our contributions are oblivious to the nature of the ENC-MATCH encrypted matching operation used.

2.1 Exploiting Containment Relations

We start by discussing how we can leverage *containment relationships* at the subscription level. This approach only works when the encrypted matching scheme does not preserve containment confidentiality and therefore allows determining containment relations from encrypted subscriptions.

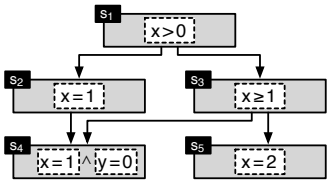


Fig. 3: Five subscriptions and their containment relationships (represented by arrows).

We follow a classical approach to exploit containment relationships as part of the filtering process [14], [15]. We organize subscriptions in a *partially ordered set* (or *poset* for short). Precedence relations in the poset correspond to containment relations between subscriptions. A poset constructed from a set of five subscriptions is shown in Figure 3. Note that a poset can be formed of several, non connected subgraphs. Upon filtering, when encountering a subscription s that matches a publication, we know that all its *containers* (ancestors in the containment poset) also match the publication and do not need to be sent to ENC-MATCH. Conversely, if s does not match the publication, all its *containees* (descendants in the poset) do not match either and can be discarded.

We leverage both positive and negative containment-based matching. As we describe in Section 2.3, our algorithm works by evaluating the complete set of subscriptions, split in several lists. The order in which subscriptions are tested is of great importance: a random evaluation order may lead, for instance, to containers being evaluated with positive matching before their containees, resulting in two calls to the costly ENC-MATCH function instead of a single one with the opposite evaluation order. Conversely, it is better to evaluate containers for negative matching before their containees. Determining positive matching requires calls to ENC-MATCH. Negative matching can also be determined with ENC-MATCH, but more efficiently based on the Bloom filters. We evaluate subscriptions in an order that respects the precedence relation in the poset to take full advantage of these early and fast negative matching decisions.

2.2 Negative Matching Using Bloom Filters

One of the key principles of prefiltering is to quickly identify subscriptions that are known *not* to match an incoming

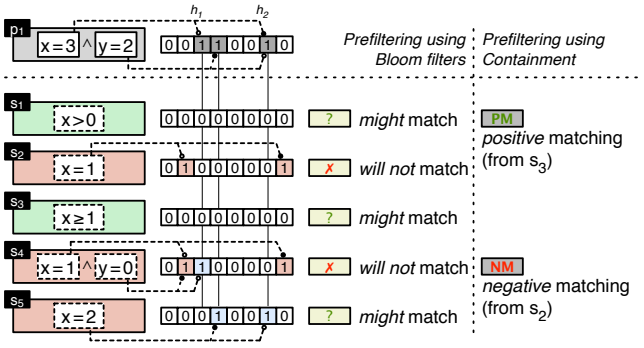


Fig. 4: Information used for prefiltering: Bloom filters embedded with subscriptions and publications, and containment-based matching.

publication (negative matching) without calling the costly ENC-MATCH function. To that end, we embed Bloom filters [19] in publications and subscriptions, when applicable, and use simple bit-wise operations to discard non-matching subscriptions.

Bloom filters are probabilistic data structures that allow for efficient testing of whether or not an item belongs to a set. A Bloom filter is essentially a bit array. When adding an item, one or several hash functions $h_1, \ldots, h_k$ are used to identify bit(s) of the Bloom filter that must be set to 1. To test whether an item belongs to the set, it is hashed again and, if all corresponding bits are set, it is likely a member the set; otherwise it is guaranteed not to be. Therefore, Bloom filters can yield false positives but no false negatives. The accuracy of the Bloom filter can be tuned by properly choosing its size and the number of hash functions.

When injecting a publication p in the system, besides encrypting it, we additionally hash the values of *all* its fields and insert them in a non-encrypted Bloom filter $B(p)$. On the other side, every subscription s also embeds a Bloom filter $B(s)$ containing the hashed values of its predicates with *equality* constraints. For instance, given a predicate *x*=2, the value 2 will be added to the Bloom filter. Figure 4 shows the same subscriptions as Figure 3, along with the information that can be used for prefiltering for a publication p_1. We assume for the illustration filters of 8 bits and two hash functions h_1, h_2. Upon matching, if $B(s) \not\subseteq B(p)$,[1] we know that s does not match p, irrespectively of the other (non-equality) predicates, and we can discard it. This is the case for s_2 where two bits set in $B(s_2)$ are not set in $B(p_1)$. Otherwise, s must be checked using the ENC-MATCH function (s_1, s_3 and s_5).

Non-matching subscriptions, i.e., false positives, may be sent to the ENC-MATCH function for three reasons. First, since Bloom filters $B(s)$ only encodes values associated to the equality constraints of s, a subscription may not match due to non-encoded constraints such as inequalities. Second, $B(s)$ includes values irrespective of the attribute they are associated with. For instance, $x = 5$ and $y = 5$ have the same Bloom filter representation. Finally, Bloom filters themselves produce false positives because distinct values can be associated to the same bits. The likelihood of such collisions can be controlled by carefully choosing the filter size and number of hash functions.

1. The operator $\subseteq$ denotes bitwise inclusion, i.e., $B(s) \subseteq B(p)$ if and only if all the bits in $B(s)$ are also set in $B(p)$.

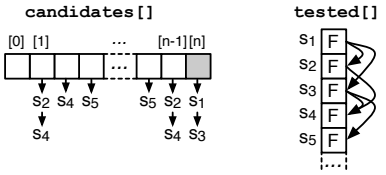


Fig. 5: Data structures used for prefiltering.

The hash functions are not known to the brokers, who only have access to encrypted data. Moreover, they are typically parameterized by an encryption key, and use a cryptographically secure scheme such as SHA-1 in a HMAC [20] construction.[2] Therefore, the Bloom filters do not reveal the values they contain. They may however allow determining a subset of containment relationships. We analyze this risk and provide a mitigation in Section 3.

The hash functions constitute a secret key, known only to the subscribers and publishers. The exchange of this secret key between the peers is the result of a preliminary key agreement protocol. The chosen key agreement protocol can be the same as the one required for the encrypted matching process itself, and is therefore independent of our prefiltering mechanism. An example could be an adaptation of OFT [21] as used for pub/sub key management in [22]. Key management in content-based systems is a challenging task and outside the scope of this article [6], [8], [12]. Most of the schemes used to secure the content of subscriptions and publications consider key management as an orthogonal issue and do not propose a solution.

2.3 Prefiltering Algorithm

The prefiltering algorithm takes advantage of Bloom filters for negative matching and, when available, containment relationships for negative and positive matching. Both reduce the number of calls to the costly ENC-MATCH encrypted matching function.

The prefiltering algorithm uses the data structures shown in Figure 5. We maintain an array (`candidates[]`) of $n + 1$ lists of candidate subscriptions, where n is the number of bits in Bloom filters. The i^{th} list for $i \in \{0, 1, \ldots, n-1\}$ contains subscriptions that have the i^{th} bit of their Bloom filter set. It follows that each subscription may belong to several lists. The last list, called *default list*, contains subscriptions that have no equality predicate, i.e., an empty Bloom filter.

In case containment relationships can be determined between encrypted subscriptions, we sort subscriptions in the candidate lists in such a way that containers appear before their containees. To that end, when inserting a new subscription, we place it just before its first containee appearing in the list, or at the end if none is found.

We additionally maintain an array (`tested[]`) to keep track of which subscriptions have already been tested. We also track known containment relationships between subscriptions (denoted by arrows on the right-hand side of Figure 5 but actually stored as a graph like in Figure 3).

We will consider several variants of the basic algorithm. For instance, if containment relationships are known, we

2. Even if SHA-1 (or other possible cryptographic hashes) output values over 160 bits, we can consider without loss of generality that a combination of these is used to set bits in the filter.

Algorithm 1: Prefiltering algorithm

```
1  PF-MATCH(p)
2  tested[] ← {false, ..., false}                    // No subscription tested yet
3  for i ← 0 ... n do                                // Check every list (including default)
4      if i = n ∨ B(p)[i] = 1 then                   // Default list or bit set in BF?
5          foreach s ∈ candidates[i] do              // Traverse list
6              if ¬tested[s] then                    // Not yet tested?
7                  m ← false                         // Initially not a match
8                  if i = n ∨ B(s) ⊆ B(p) then        // Possible match?
9                      m ← ENC-MATCH(p, s)           // Must do costly test
10                 if m then                         // Match?
11                     FORWARD(p, s)                 // Forward to subscriber
12                     if CB-POSITIVE-MATCH then     // Do positive match?
13                         foreach s' ∈ containers(s) do
14                             if ¬tested[s'] then   // Not yet tested?
15                                 tested[s'] ← true
16                                 FORWARD(p, s')    // Forward to subscriber
17                 else                              // No match
18                     if CB-NEGATIVE-MATCH then     // Do negative match?
19                         foreach s' ∈ containees(s) do
20                             tested[s'] ← true
21                 tested[s] ← true
```

may discard additional subscriptions when one is found that does not match, or conversely accept several subscriptions at once upon successful match, respectively denoted by `CB-NEGATIVE-MATCH` and `CB-POSITIVE-MATCH` in the code (`CB` stands for *containment-based*).

The prefiltering process is shown in Algorithm 1. Initially, the `tested[]` array is reset to indicate that no matching has taken place yet (Line 2). We then traverse the list `candidates[i]` associated with each bit i set in the publication's Bloom filter $B(p)$, as well as the default list (Lines 3–5). If the current subscription s has not yet been tested, as indicated by `tested[s]`, we check if there is a possible match (Lines 6–8). This can happen in two cases: s is part of the default list, i.e., $B(s)$ is empty; or $B(s) \subseteq B(p)$. If so, we must use the costly but accurate ENC-MATCH encrypted matching function that tells us with certainty if the publication must be forwarded to the corresponding subscriber (Line 9). In case of a match, we forward the publication to the interested subscriber (Line 11) and we optionally use containment-based positive matching to accept any yet-untested subscription (Lines 12–16). On the other end, if the publication does not match s, we may use containment-based negative matching to mark all the subscription's containees as having been tested, effectively discarding them for the rest of the filtering process (Lines 18–20).

Discussion. Let us first consider prefiltering with *containment obfuscation*. The very role of the Bloom filter is to perform aggressive negative matching without knowing containment relationships between subscriptions. During prefiltering, we only need to traverse the candidate lists associated with the bits set in the publication's Bloom filter, as well as the default list. All other subscriptions are discarded. The number of lists to traverse is therefore function of the number of fields (i.e., attributes) of the publication, which is expected to be much smaller than the size of the Bloom filters. In addition, no subscription is tested more than once thanks to the bookkeeping of the `tested[]` array.

Let us now consider *containment-based* prefiltering. If containment relationships are known, one can perform additional negative and positive matching. When testing a subscription that does not match, one can immediately discard all of its containees. However, it might happen that a subscription is tested before some of its containers, because they do not all belong to the same candidate lists. In such cases, when successfully matching a subscription, one can apply the positive matching optimization to immediately accept any yet-untested container.

The likelihood for any of these scenarios to occur depends on the nature and distribution of the publications and subscriptions. We experimentally evaluate the efficiency of the prefiltering variants in Section 4.

3 CONTAINMENT OBFUSCATION

We now focus on the case where the encrypted matching function does not allow determining containment relations between subscriptions. As depicted in Figure 2, we are interested in preserving containment confidentiality while still benefiting from the performance improvement obtained from prefiltering.

3.1 Adversary model and limitations

We consider as setting for our analysis a pub/sub overlay of brokers deployed in an untrusted environment. This environment can be for instance a public cloud which does not guarantee the confidentiality of persistently stored data, represented in our case by subscriptions. As discussed before, the stored subscriptions at the brokers and the flow of publications are encrypted, and the matching is done over encrypted data. We do not consider the case where a pub/sub entity can play multiple roles (e.g, a broker being also a subscriber), thus the brokers do not know the secret keys. We consider as attacker an external malicious entity capable to collect snapshots of a broker's persistent storage, thus gaining access to the registered subscriptions in encrypted form. We do not consider the case when the attacker colludes with one of the pub/sub system entities (publisher, subscriber or broker).

Protecting the subscribers' interests is usually the most critical confidentiality task. In fact, some broker-based matching schemes do not even encrypt the data that must be matched with the subscribers' interests [23]. For this reason, in our analysis we do not consider the case where the attacker can monitor and collect data from the dynamic flow of publications.

Our security analysis formally studies whether the addition of the non-encrypted Bloom filters to the encrypted subscriptions allows an attacker to determine *more information* about subscribers than otherwise possible (i.e., when observing the encrypted subscriptions only). We mention again that an evaluation of the security of the encrypted matching functions themselves is out of the scope of this article and that our contribution is oblivious to the encrypted matching scheme used. Under these assumptions, we consider the confidentiality level of the encrypted matching as the baseline when evaluating the security of our prefiltering, which adds a Bloom filter to every encrypted subscription.

3.2 Obfuscation mechanism

By using our prefiltering mechanism we observe that contained subscriptions produce Bloom filters that are included in one another. This may allow an attacker to derive approximate containment relationships between encrypted subscriptions, therefore leaking *more information* than without prefiltering. To counter this, we provide a *containment obfuscation* mechanism. The principle is to introduce additional randomness in the Bloom filters by randomly removing set bits from the subscriptions Bloom filters (*truncation*). This is illustrated in the top half of Figure 6. As can be seen from the figure, s_1 is more general than s_2[3], but after truncation their Bloom filters do not contain each other anymore. This makes it much harder for an attacker to derive accurate containment graphs.

If publications also require protection, bits can be randomly set to one in their associated Bloom filters (*pollution*). Pollution is illustrated in the bottom half of Figure 6. We note that polluting subscriptions and/or truncating publications would introduce false negatives, and prefiltering would not work. Since protecting the privacy of the publication flow is out of the scope of this article, we only analyze the impact of truncation and do not consider pollution.

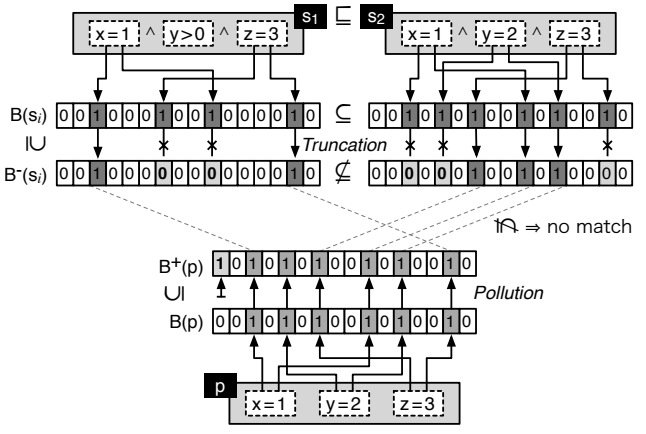


Fig. 6: Polluting publications Bloom filters and truncating subscriptions Bloom filters.

3.3 Security analysis

It should be clear that truncating subscriptions (and optionally polluting publications) increases the probability of false positive and does not introduce false negatives. In the following we study the tradeoff between prefiltering efficiency and containment information leaked to an attacker from the Bloom filters. We show that efficient prefiltering is possible while making containment information so noisy that it essentially becomes useless for potential attackers. The rest of the section is organized as follows. Notation, assumptions and the truncation hashing strategy are described in 3.3.1, and the subscription/publication domain in 3.3.2. The amount of containment information leaked to an attacker from the subscriptions Bloom filters and the efficiency of prefiltering are formally stated in Subsection 3.3.3 and solved in 3.3.4. In addition, we discuss and analyze two other potential but weaker attacks in Appendix A.

3. We write $s_1 \sqsubseteq s_2$ to say that s_1 is more general than s_2. This is justified and explained further in Section 3.3.1.

3.3.1 Notation and Assumptions

The notation is summarized in Table 1. Consider a subscription s_1 with $c_1 \geq 0$ different equality constraints encoded in a Bloom filter $B_\alpha(s_1)$ of size n using α out of k hash functions per constraint. Furthermore, let us consider a second subscription s_2 with $c_2 \geq c_1$ different equality constraints also encoded in a Bloom filter $B_\alpha(s_2)$ of size n with α out of k hash functions per constraint. We also consider a publication p with $c_p \geq c_1$ different values. The publication is encoded in a Bloom filter $B_k(p)$, that is, all k hash functions are used. We write $B_\alpha(s_1) \subseteq B_\alpha(s_2)$ if and only if all the nonzero bits in the Bloom filter of s_1 are also nonzero in the Bloom filter of s_2. Likewise, we write $B_\alpha(s_1) \subseteq B_k(p)$ if the Bloom filter of a subscription s_1 is included in the Bloom filter of a publication p.

s_i	Subscription i
c_i	Number of different equality constraints in s_i
p	Publication
c_p	Number of different values in p
k	Cardinality of the set of hash functions used to encode values in Bloom filters
α	Number of hashes randomly chosen out of k to encode a value in a subscription Bloom filter
$B_\alpha(s_i)$	Bloom filter representation of Subscription s_i using α hash functions per value
$B_k(p)$	Bloom filter representation of Publication p using all k hash functions per value
n	Bloom filter size
a	Number of attributes in the domain
v	Number of values per attribute in the domain
γ	Number of common values between two subscriptions (or between a subscription and a publication)
P_α	The probability that $s_1 \sqsubseteq s_2$ when $B_\alpha(s_1) \subseteq B_\alpha(s_2)$
$P_{\text{f\_pos}}$	The false positive probability that $B_\alpha(s_1) \subseteq B_k(p)$ when $s_1 \not\sqsubseteq p$

TABLE 1: Notation and definitions

We only consider the number of distinguishable values for subscriptions and publications since it is assumed that the same value appearing for multiple attributes is only hashed once. We only consider equality constraints, which are the only ones encoded in the Bloom filters. Thus, s_1, s_2 and p can formally be defined as sets of cardinality c_1, c_2 and c_p, respectively, and we use $A \sqsubseteq B$ to denote that A is a subset of B. The reason behind this nonstandard notation is to keep in mind that when $s_1 \sqsubseteq s_2$, the set s_1 is included in the set s_2, but the subscription s_1 is more general and is a container of s_2.

We assume that Bloom filters use a set of k independent and perfectly random hash functions. Our hashing strategy $\mathcal{H}_\alpha$ consists, for a fixed $\alpha \in \{1, 2, \ldots, k\}$, of randomly selecting α among the k hash functions for each subscription value to be encoded. We emphasize again that the hashing strategy is only applied to subscriptions; all the k hash functions must be used for publications, otherwise false negatives can occur.

3.3.2 Subscription/Publication Domain

We assume that there are a possible attributes $A_1, A_2, \ldots, A_a$ with equality constraints that can be used for each subscription and in publications, and that the set of v possible values

for attribute A_i is $V_i = \{v_1^i, v_2^i, \ldots, v_v^i\}$. It is assumed that when a subscriber or publisher selects an attribute to put in a subscription or publication, it chooses any of the a attributes with probability $\frac{1}{a}$. It is also assumed that the attribute takes each of its possible values with probability $\frac{1}{v}$. The domain we consider assumes that $V_i \cap V_j = \emptyset$ for $i \neq j$. Non-disjoint value sets can always be made disjoint by encoding each value with a small prefix representing its attribute.

It is assumed that an attacker knows the domain and its probability distribution and also k and α. We assume that when an attacker examines a subscription Bloom filter, it knows the number of different equality constraints that were encoded in it. This increases its power, but only slightly because the number of nonzero bits in a Bloom filter is highly correlated with the number of encoded values. Furthermore, this allows us to do the analysis without any assumption on the distribution of the number of equality attributes per subscription.

The domain uniformity is not a realistic assumption for most content-based systems, although two remarks must be made. Firstly, the domain uniformity allows us to analyze rigorously how an attacker can infer subscription containment from the Bloom filters as well as the performance of the prefiltering scheme. Secondly, the analysis can be carried out numerically for any content-based system for which an estimate of the statistics at the subscriber, publisher, or system level is available. One such example is presented in Section 4.

3.3.3 Problem Statement

Our objective is to study the tradeoff between the amount of information about subscriptions containment that can be inferred by an attacker and the performance of the prefiltering process. We formalize these notions below.

Containment Leaks. We want to prove that randomly selecting a small number of hash functions for each coded attribute of the Bloom filters restricts the attacker capacity to derive containment between the two subscriptions based on their attached Bloom filters. The expression of interest is

$$P_\alpha \triangleq \Pr[s_1 \sqsubseteq s_2 \mid B_\alpha(s_1) \subseteq B_\alpha(s_2)]. \quad (1)$$

The higher this probability is, the easier it is for an attacker to build an accurate containment graph when having access to a large number of subscription Bloom filters included in other subscription Bloom filters. Using Bayes' law, we can write

$$P_\alpha = \Pr[s_1 \sqsubseteq s_2] \cdot \frac{\Pr[B_\alpha(s_1) \subseteq B_\alpha(s_2) \mid s_1 \sqsubseteq s_2]}{\Pr[B_\alpha(s_1) \subseteq B_\alpha(s_2)]}. \quad (2)$$

When all the k hash functions are chosen,
$\Pr[B_k(s_1) \subseteq B_k(s_2) \mid s_1 \sqsubseteq s_2] = 1$.

Prefiltering Efficiency. Decreasing the amount of information to potential attackers is useless if it results in a prefiltering operation that fails to discard a significant fraction of the subscriptions with equality constraints that do not match an incoming publication. The expression of interest is the probability of false positive

$$P_{\text{f\_pos}} \triangleq \Pr[B_\alpha(s_1) \subseteq B_k(p) \mid s_1 \not\sqsubseteq p]. \quad (3)$$

It is the probability, when the equality constraints of a subscription are not included in the set of values of a publication, that the Bloom filter of the subscription is included in the Bloom filter of the publication.

3.3.4 Analysis with Collisions

We first present preliminary results followed by the exact expressions for P_α and $P_{\text{f\_pos}}$. We start by analyzing the number of common values between a pair of subscriptions. Let $\Pr_{\text{common}}[\gamma]$ be the probability that there are γ common values between subscriptions s_1 and s_2 for $0 \leq \gamma \leq c_1$.

Lemma 1.

$$\Pr_{\text{common}}[\gamma] = \frac{\binom{c_2}{\gamma} \cdot \sum_{\delta=0}^{c_1-\gamma} \left[\binom{c_2-\gamma}{\delta} \binom{a-c_2}{c_1-\gamma-\delta} (v-1)^\delta v^{c_1-\gamma-\delta} \right]}{\binom{a}{c_1} v^{c_1}}.$$

Proof: Supplementary material (Appendix B.1). □

Corollary 2.

$$\Pr[s_1 \sqsubseteq s_2] = \Pr_{\text{common}}[c_1] = \frac{\binom{c_2}{c_1}}{\binom{a}{c_1} v^{c_1}}.$$

Proof: The probability that $s_1 \sqsubseteq s_2$ is the probability that all c_1 values of s_1 also appear in s_2. □

We now focus on $\Pr[B_\alpha(s_1) \subseteq B_\alpha(s_2) \mid s_1 \sqsubseteq s_2]$ and $\Pr[B_\alpha(s_1) \subseteq B_\alpha(s_2)]$. Our main insight is to split the hashes into two categories. First, consider an attribute value common to subscriptions s_1 and s_2. If, for this value, there is a hash function h which is randomly chosen for both Bloom filters $B_\alpha(s_1)$ and $B_\alpha(s_2)$, we say there is a *good hash* between s_1 and s_2. Second, it is also possible that unrelated hash functions selected by both s_1 and s_2 randomly hash at the same position in the Bloom filters; this is called a *lucky hash*. Since we want $B_\alpha(s_1) \subseteq B_\alpha(s_2)$, all the hashes of s_1 in $B_\alpha(s_1)$ must be covered by the hashes of s_2 in $B_\alpha(s_2)$ by *good* and/or *lucky hashes*.

Let $\Pr_{\text{good}}[g, \gamma]$ be the probability that there are g *good hashes* between s_1 and s_2, where γ is the number of common values between both subscriptions. $\Pr_{\text{good}}[g, \gamma]$ is difficult to calculate for $\alpha > 1$ because, for instance, the probability that there are two good hashes for one common value between s_1 and s_2 is different from the probability that there are two common values with one good hash each. Thus, to evaluate $\Pr_{\text{good}}[g, \gamma]$ we must consider the integer partitions of g [24].

Let $\text{IntPart}(g, \gamma, \alpha_{min}, \alpha)$ be the set of all integer partitions of g into exactly γ parts with the additional constraint that the size of each part is at least α_{min} and at most α. We write $\lambda \vdash \text{IntPart}(g, \gamma, \alpha_{\min}, \alpha)$ to denote that λ is an integer partition of g with the desired properties. The frequency representation of a partition $\lambda \vdash \text{IntPart}(g, \gamma, \alpha_{\min}, \alpha)$ is $(\alpha_{\min}{}^{p_{\alpha_{\min}}} \alpha_{\min+1}{}^{p_{\alpha_{\min+1}}} \ldots \alpha^{p_\alpha})$, meaning that the value α_i appears p_{α_i} times in the partition.

Lemma 3.

$$\Pr_{\text{good}}[g, \gamma] = \sum_{\lambda \vdash \text{IntPart}(g,\gamma,\alpha_{\min},\alpha)} \left[\binom{\gamma}{p_{\alpha_{min}}, p_{\alpha_{min+1}}, \ldots, p_\alpha} \cdot \prod_{\beta=\alpha_{min}}^{\alpha} \left[\frac{\binom{\alpha}{\beta}\binom{k-\alpha}{\alpha-\beta}}{\binom{k}{\alpha}} \right]^{p_\beta} \right]$$

where $\alpha_{\min} = \max(0, 2\alpha - k)$.

Proof: Supplementary material (Appendix B.2). □

The formula is much simpler when $\alpha = 1$, since there is only one integer partition of g into γ parts when each part is either 0 or 1. This is described by the following corollary.

Corollary 4. When $\alpha = 1$,

$$\Pr_{\text{good}}[g, \gamma] = \binom{\gamma}{g} \left(\frac{1}{k}\right)^g \left(\frac{k-1}{k}\right)^{\gamma - g}.$$

Proof: When $\alpha = 1$, the problem is a direct application of the binomial distribution with g successes out of γ events and probability of success $\frac{1}{k}$. □

If $B_\alpha(s_1) \subseteq B_\alpha(s_2)$, all the Bloom filter bits derived from the hash functions randomly chosen for s_1 must somehow be covered by some of the hash functions randomly chosen for the Bloom filter of s_2. If we have g *good hashes* between s_1 and s_2, that leaves $c_1\alpha - g$ *lucky hashes* of s_1 that must be covered by some of the hash functions of s_2 randomly hashing at the same positions in the Bloom filter. Let $\Pr_{\text{lucky}}[h_1, h_2]$ be the probability that h_1 hashes of a subscription or publication are covered by some of the h_2 hashes of a second subscription or publication randomly hashing at the same positions in the Bloom filter.

Lemma 5.

$$\Pr_{\text{lucky}}[h_1, h_2] = \frac{1}{n^{h_1+h_2}} \sum_{i=0}^{h_2} \left[\binom{n}{i} i! \left\{ {h_2 \atop i} \right\} \sum_{j=0}^{i} \binom{i}{j} j! \left\{ {h_1 \atop j} \right\} \right].$$

where $\left\{ {k_1 \atop k_2} \right\}$ represent the Stirling numbers of the second kind [25].

Proof: Supplementary material (Appendix B.3). □

Lemma 6.

$$Pr[B_\alpha(s_1) \subseteq B_\alpha(s_2) \mid s_1 \sqsubseteq s_2] = \sum_{g=0}^{c_1\alpha} \left(\Pr_{\text{good}}[g, c_1] \Pr_{\text{lucky}}[c_1\alpha - g, c_2\alpha] \right).$$

Proof: Since $s_1 \sqsubseteq s_2$, there are c_1 common attribute values between s_1 and s_2. The probability that $B_\alpha(s_1) \subseteq B_\alpha(s_2)$ can be obtained by summing, for all values of g between 0 and $c_1\alpha$, the probability of having g *good hashes* times the probability that the $c_1\alpha - g$ *lucky hashes* of s_1 can be covered by the $c_2\alpha$ hash functions of s_2 randomly hashing at the same place in the Bloom filter of s_2. □

Lemma 7.

$$\Pr[B_\alpha(s_1) \subseteq B_\alpha(s_2)] = \sum_{\gamma=0}^{c_1} \left(\Pr_{\text{common}}[\gamma] \sum_{g=0}^{\gamma\alpha} \left(\Pr_{\text{good}}[g, \gamma] \Pr_{\text{lucky}}[c_1\alpha - g, c_2\alpha] \right) \right).$$

Proof: Suppose that there are γ common attribute values between s_1 and s_2. The probability that $B_\alpha(s_1) \subseteq B_\alpha(s_2)$ given γ can be obtained by summing, for all values of g between 0 and $\gamma\alpha$, the probability of having g *good hashes* times the probability that the $c_1\alpha - g$ *lucky hashes* of s_1 can be covered by the $c_2\alpha$ hash functions of s_2 randomly hashing at the same place in the Bloom filter of s_2. The probability that $B_\alpha(s_1) \subseteq B_\alpha(s_2)$ can be obtained by summing, for all values of γ between 0 and c_1, the probability that there are γ common attribute values between s_1 and s_2 times the probability that $B_\alpha(s_1) \subseteq B_\alpha(s_2)$ given γ. □

We are now ready to prove our main results.

Theorem 1.

$$P_\alpha = \frac{\left(\binom{a}{c_1} v^{c_1}\right)^{-1} \binom{c_2}{c_1} \sum_{g=0}^{c_1\alpha} \left(\Pr_{\text{good}}[g, c_1] \Pr_{\text{lucky}}[c_1\alpha - g, c_2\alpha] \right)}{\sum_{\gamma=0}^{c_1} \left(\Pr_{\text{common}}[\gamma] \sum_{g=0}^{\gamma\alpha} \left(\Pr_{\text{good}}[g, \gamma] \Pr_{\text{lucky}}[c_1\alpha - g, c_2\alpha] \right) \right)}.$$

Proof: The theorem can be derived by merging (2), Lemmas 1, 3, 5, 6, 7, and Corollaries 2 and 4. □

Theorem 2.

$$P_{\text{f\_pos}} = \sum_{\gamma=0}^{c_1-1} \left(\Pr_{\text{common}}[\gamma] \Pr_{\text{lucky}}[\alpha(c_1 - \gamma), c_p k] \right).$$

Proof: From (3), $P_{\text{f\_pos}} \triangleq \Pr[B_\alpha(s_1) \subseteq B_k(p) \mid s_1 \not\sqsubseteq p]$. Suppose that there are $\gamma < c_1$ common attribute values between s_1 and p. Since the publication uses all the k hash functions for each value in its Bloom filter, it follows that there are $\alpha\gamma$ *good hashes* for s_1, leaving $\alpha(c_1 - \gamma)$ *lucky hashes* that must be covered by the $c_p k$ hashes of p randomly hashing at the same positions in the Bloom filters. The probability of this occurring is given by $\Pr_{\text{lucky}}[\alpha(c_1 - \gamma), c_p k]$. The result can be obtained by summing, for all values of γ between 0 and c_1 ($s_1 \sqsubseteq p$ if $\gamma = c_1$), the probability that there are γ common attribute values between s_1 and p times the probability of $\alpha(c_1 - \gamma)$ *lucky hashes*. □

4 Evaluation and Discussion

In this section, we evaluate the performance of our prefiltering strategy under a wide range of scenarios. To the best of our knowledge, no existing work that we are aware of can be used for a meaningful comparison. We thus compare our prefiltering strategy with sensible and well-known algorithms and baselines. The evaluation when containment information is available is presented in Subsection 4.1; the baseline is the performance of a standard containment-based algorithm without prefiltering. In Subsection 4.2, we cover the performance-confidentiality tradeoffs with containment obfuscation. The performance baseline is the number of publications discarded by the brokers (i.e., publications that prefiltering could potentially discard), whereas the confidentiality baseline is the containment information leaked from random encrypted subscriptions based on their domain's probability distribution.

4.1 Prefiltering Performance

In this subsection, we experimentally study the efficiency of prefiltering by comparing its performance with a containment-based algorithm without prefiltering for different workloads described in Section 4.1.1. In Section 4.1.2, we focus on the reduction in the number of calls to the ENC-MATCH encrypted matching function with the different variants of prefiltering. In Section 4.1.3, we evaluate the performance gains on a complete implementation of the matching engine using ASPE [6] as the encrypted matching

Workload name	Number of equality predicates	Number of attributes	Distribution of values
e100	100% : 1 eq. pred.	8–11 (default)	Uniform
e80	20% : 0 eq. pred. 80% : 1 eq. pred.	8–11 (default)	
e80-2x		2× more	
e80-4x		4× more	
e+-2x	15% : 0 eq. pred. 60% : 1 eq. pred.	2× more	
e+-4x	15% : 2 eq. pred. 10% : 3 eq. pred.	4× more	
e80-z	20% : 0 eq. pred.	8–11 (default)	Zipf on symbol
e80-z+	80% : 1 eq. pred.		Zipf on all attributes
e100-z+	100% : 1 eq. pred.		

TABLE 2: Description of the workloads.

operation. Finally, as supplementary material in Appendix C, we study the effect of varying the size of the Bloom filters and the number of hash functions on the effectiveness of all prefiltering variants.

4.1.1 Workloads

We constructed an experimental workload inspired by the one used for the evaluation of the Meghdoot publish/subscribe system [26]. We gathered five years of quotes for 200 randomly selected stocks from the Yahoo! finance website (http://finance.yahoo.com/) which corresponds to a set of over 250,000 publications with between 8 and 11 attributes. We built synthetic subscriptions based on the same categories as in [26]. These subscriptions contain a variety of equality and range predicates on the attributes of stock quotes, namely the symbol, date, exchanged volume, and daily statistics on their price (open, close, high, low). By default, equality predicates are expressed on the symbol, while ranges apply to the volume and the various daily statistics. For workloads with additional equality predicates, we used extra attributes from the Yahoo! finance data: the stock exchange on which the stock is available; the trend indicating if the stock is higher or lower than the previous day; and the type of product (among 8 possible categories).

We also experimented with workloads containing more attributes than available in the original Yahoo! finance data. To that end, we simply created additional copies of the volume and daily statistics attributes by merging data from multiple quotes. We produced workloads with twice and four times more attributes on which subscriptions can express range predicates.

We finally experimented with different subscription distributions, by choosing predicate values either uniformly at random, or according to a Zipf's law with exponent $s = 1$.[4] Recall that publications are real data from Yahoo! finance and we did not modify their distribution in any way.

Table 2 describes the nine workloads used in our evaluation. We experimented with populations of up to 100,000 subscriptions of every type.

Table 3 presents characteristics of the workloads instrumental in the interpretation of experimental results. For each workload of 100,000 subscriptions, we created a containment graph. The second column indicates the number of nodes in the graph, with identical subscriptions being collapsed in a single node. Columns 3–5 respectively show the number of nodes that are root, leaf, or both (i.e., isolated nodes) in the graph. A node may be reachable from several roots and by multiple paths. Column 6 presents the average shortest (μ^-) and longest (μ^+) paths from some root to each node, as well as the maximum path length. Column 7 gives the average and maximum number of children at each node (excluding leaves), while Column 8 provides the same statistics for parents of each node (excluding roots). Columns 9, 10, and 11 respectively list the number of identical subscriptions in the workloads, the number of predicates in the subscriptions, and the number of equality predicates. Finally, the last column specifies the percentage of matching subscriptions for each of the considered workloads, averaged over a set of 1,000 publications.

4. Zipf's law and other power laws appear in a wide range of disciplines like finance, demography, biology, networks, ... [27]

4.1.2 Reducing the Number of Tests

We first study the number of calls to the ENC-MATCH encrypted matching function. We express this value as a "tests ratio" over the number of subscriptions. A value of 1 indicates that a call to ENC-MATCH must be performed for each subscription, as would be the case with a naive approach consisting in traversing the whole collection of subscriptions to test each one individually.

We evaluate the following algorithms:

- `CB` Containment-based strategy (baseline).
- `PF` Prefiltering with complete Bloom filters.
- `PF-TB` Prefiltering with truncated Bloom filters, using $\alpha = 1$ out of $k = 3$ hash functions.
- `PF+NM` PF combined with containment-based negative matching.
- `PF+NM+PM` PF combined with containment-based negative and positive matching.

We use Bloom filters of 128 bits and 3 hash functions. The reason for using Bloom filters of this dimension is that, as results show, this size is sufficient to provide good results for the considered workloads.

Results are shown in Figure 7 (note the logarithmic scales). Let us first consider the baseline `CB` strategy. One can observe that the most important savings can be obtained with workloads that have few matches (`e100`, `e80-4x`, `e+-4x`, and `e100-z+`, all of which report less than 1% of matches in Table 3) because many subscriptions can be discarded early while traversing the containment graph. With other workloads, the performance of `CB` still improves with the number of subscriptions, albeit at a lower rate. Both `PF` and `PF-TB` exhibit almost constant performance independent of the number of subscriptions. The reason is that savings depend in these cases only on negative matching obtained from subscriptions Bloom filters, which do not have an impact on the matching of other subscriptions as containment is not exploited. The only noticeable exception is with `e80-z+` and `e100-z+` where `PF` slightly improves. This is due to the fact that `PF` collapses identical subscriptions (which reach up to 1,000 in these workloads) into a single node in the lists, while `PF-TB` does not (recall that `PF-TB` is designed for containment obfuscation, thus it does not allow to determine subscriptions equality).

`PF` is significantly more efficient than `PF-TB` because truncation adds false positives. The difference between both strategies is more important for workloads that report few matches. It is also worth pointing out that `PF-TB` it more

Workload	Nodes	Roots	Leaves	R ∩ L	Depth			Children		Parents		Identical		Predicates			Equality			Matches
					μ^-	μ^+	max	μ	max	μ	max	μ	max	μ	min	max	μ	min	max	μ (%)
e100	89102	1688	3839	27	53.04	53.80	192	1.47	13	1.44	10	1.12	5	2.20	2	3	1.00	1	1	0.19
e80	79786	45	3504	0	78.41	298.29	2056	3.02	75	2.89	17	1.25	28	2.00	1	3	0.80	0	1	7.48
e80-2x	99944	1705	21812	320	3.90	27.73	106	10.64	336	8.46	49	1.00	2	3.20	2	5	0.80	0	1	2.78
e80-4x	100000	41315	78624	32229	1.61	2.09	9	16.52	307	6.02	63	1.00	1	5.60	4	9	0.80	0	1	0.40
e+-2x	99980	1956	36416	621	3.76	23.68	88	12.25	573	7.94	43	1.00	2	3.60	2	7	1.21	0	3	2.14
e+-4x	100000	46635	83596	38793	1.55	1.93	9	17.60	344	5.41	59	1.00	1	5.99	4	11	1.20	0	3	0.30
e80-z	63671	31	2821	0	118.70	309.75	2019	3.27	76	3.13	18	1.57	29	2.00	1	3	0.80	0	1	7.49
e80-z+	43059	23	2453	1	59.91	205.96	483	2.92	201	2.75	19	2.32	1015	2.00	1	3	0.80	0	1	8.93
e100-z+	46125	1148	2648	32	45.30	54.70	477	1.57	11	1.52	11	2.17	898	2.20	2	3	1.00	1	1	0.21

TABLE 3: Workload characteristics for 100,000 subscriptions (matches evaluated on 1,000 publications).

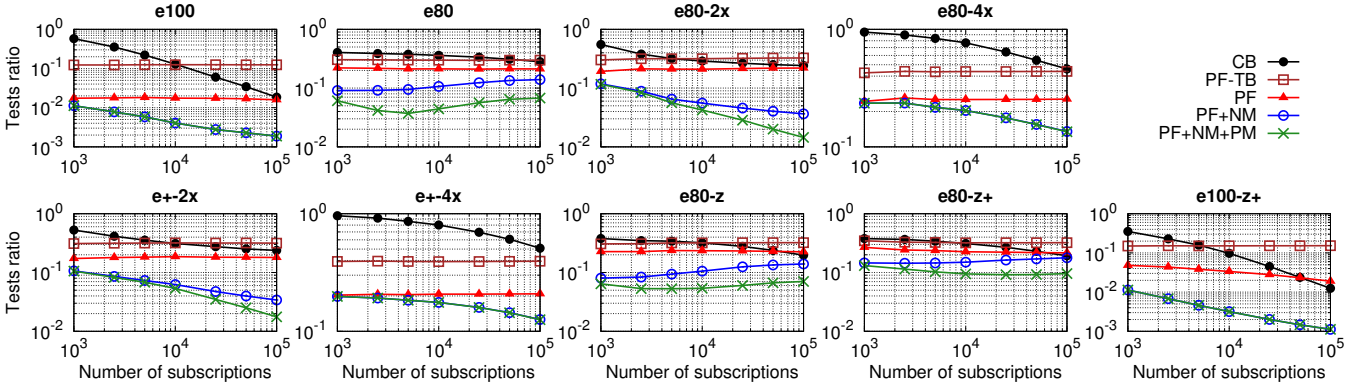


Fig. 7: Evolution of the number of tests, i.e., calls to the encrypted matching function, with different populations of subscribers (128-bit Bloom filters and 3 hash functions).

efficient that CB on most of the workloads up to large populations of subscriptions. This is notably the case of workloads with shallow containment graphs (e80-4x and e+-4x, with average node depth lower than 2).

Finally, the PF+NM and PF+NM+PM strategies perform best in all our experiments. They combine the benefits of PF and CB: like the former, they perform already well with few subscriptions; like the latter, they can take advantage of the large number of containment relationships available with many subscriptions. Unsurprisingly, positive matching is helpful with workloads that exhibit sufficient matches (e80, e80-2x, e+-2x, e80-z, and e80-z+, which report between 2% and 9% of matches). Workloads with high depth and a good proportion of subscriptions with no equality predicate (e80, e80-z, and e80-z+) exhibit an interesting trend, with performance initially improving and then slightly degrading as the number of subscription grows. The reason why PF+NM and PF+NM+PM do not follow the same slope as CB is that traversal is performed in containment order within the lists, but not globally in the whole graph. Consider two subscriptions s and s' with $s \sqsupseteq s'$ and s containing s'. The containee s' may have additional equality predicates and, therefore, belong to more lists in the prefilter structure than s. If s' is first encountered as part of a list to which s does not belong, the containee may be tested before its container against a publication p. If neither matches p, PF+NM would still test both subscriptions while CB would only test s as it strictly follows the global containment graphs. Note that despite the difference in slopes, CB does not catch up PF+NM nor PF+NM+PM in any of our workloads up for to 100,000 subscriptions, and the gains of the latter are in most cases close to one order of magnitude.

4.1.3 *Improving Filtering Time: Complete Implementation*

We present in this section the performance of privacy-preserving filtering on a complete implementation. The matching engine used is ASPE [6]. The goal is to verify how the reduction in calls to the ENC-MATCH function affects the actual filtering time. To that end, we have executed the same experiment as in Figure 7 using a single-threaded C++ implementation of our algorithms running on a dual 8-core 2.4 GHz Xeon with 64 GB RAM. Figure 8 reports the average time necessary to filter a publication with all considered workloads and algorithm variants, with the addition of a "naive" strategy that simply iterates through the set of subscriptions and tests them one by one using ASPE.

The gains expected from our evaluation of the tests ratio are mirrored in most of the graphs. While the behavior of the algorithms is consistent with results of Figure 7, PF+NM and PF+NM+PM strategies do not scale as well as anticipated on some workloads, notably when the average depth of nodes is high and many subscriptions have no equality predicate (e80, e80-z, and e80-z+). This can be explained by memory management and cache effects on our hardware, as well as the prohibitive size of the last list in the prefilter structure. More precisely, even though the ENC-MATCH function may not be actually called, the whole list must still be traversed; in contrast, exploration of the containment graph can be interrupted early. Despite these implementation artifacts, performance results clearly show that prefiltering provides a significant time reduction of the matching operation.

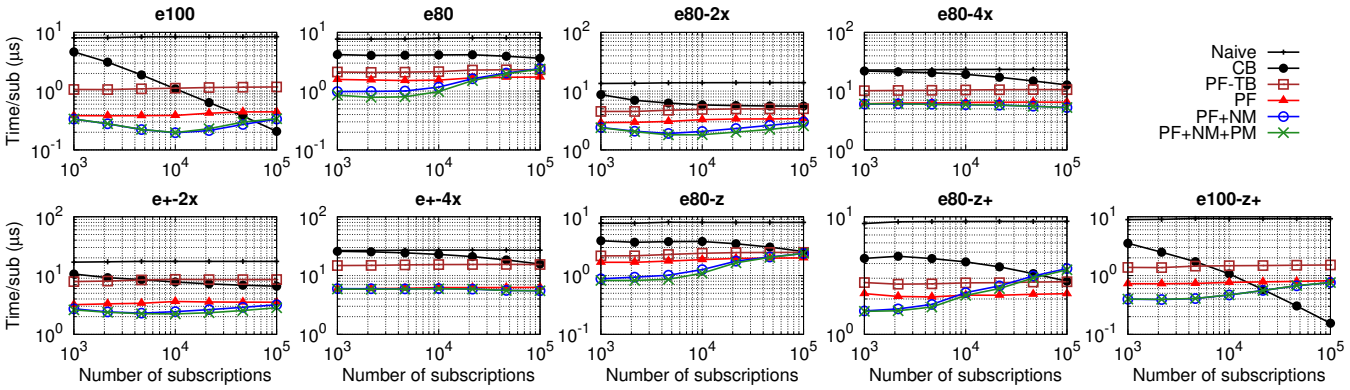
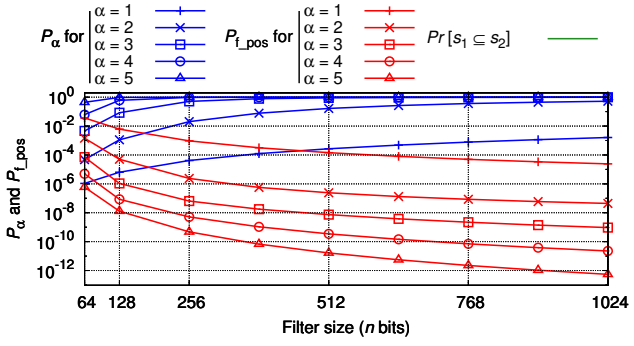
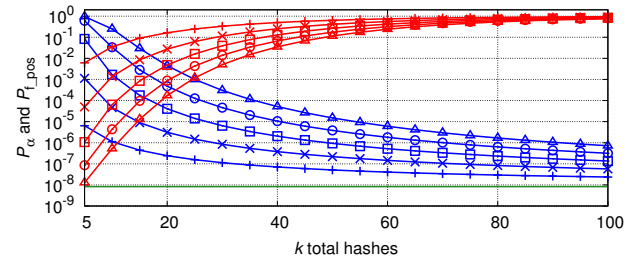


Fig. 8: Filtering performance using ASPE with different populations of subscribers (n =128-bit Bloom filters and k =3 hash functions).



(a) $P_\alpha / P_{\text{f\_pos}}$ tradeoff with $k = 5$ as n increases.



(b) $P_\alpha / P_{\text{f\_pos}}$ tradeoff with $n = 128$ as k increases.

Fig. 9: General containment obfuscation and prefiltering efficiency tradeoffs, using $a = 10$, $v = 100$, $c_1 = c_2 = 3$, $c_p = 5$.

4.2 Performance/Confidentiality Tradeoffs

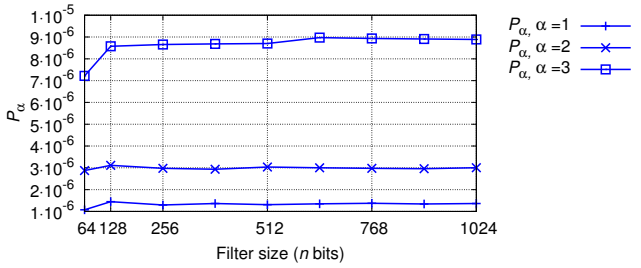
We now use Theorems 1 and 2 to study the fundamental tradeoffs between the efficiency and the containment confidentiality of prefiltering using truncation for containment obfuscation. We first consider as an example the uniform domain with $a = 10$ attributes and $v = 100$ values per attribute. We further assume that all the subscriptions contain $c_1 = c_2 = 3$ equality constraints, that all the publications contain $c_p = 5$ values, and that a set with $k = 5$ hash functions is used. Figure 9(a) shows how P_α and $P_{\text{f\_pos}}$ vary with the Bloom filter size. For both P_α and $P_{\text{f\_pos}}$, the five curves correspond to $\alpha \in \{1, 2, 3, 4, 5\}$, where α is the number of hash functions randomly selected out of k. It can be observed that P_α converges to 1 as the Bloom filter size increases. This can be explained by the fact that when Bloom filters are large, the probability that a pair of subscriptions that do not cover each other generates two Bloom filters such that one is included in the other approaches zero. We also notice that P_α converges much slower to 1 with small values of α. This indicates that using truncation used in conjunction with Bloom filters of reasonable size is an efficient strategy to decrease containment information leakage.

Figure 9(a) also shows that $P_{\text{f\_pos}}$ decreases as n and k increase. Figure 9(b) uses the same domain as Figure 9(a), except that it shows how P_α and $P_{\text{f\_pos}}$ vary with k with Bloom filters of 128 bits. It should be clear from both figures that in order to minimize the amount of information to an attacker, we can choose a small Bloom filter and select one hash function randomly out of a very large set. We can show from (2) and Theorem 1 that

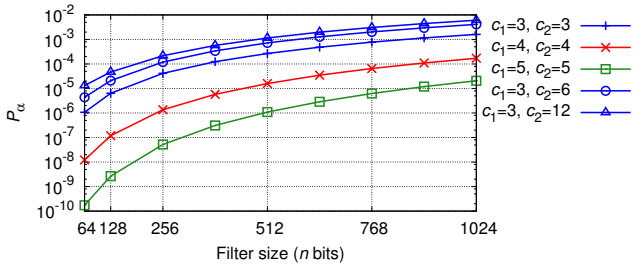
$$\begin{aligned}
&\lim_{k\to\infty} P_\alpha \\
&= \lim_{k\to\infty} \left(\Pr[s_1 \sqsubseteq s_2] \cdot \frac{\Pr[B_\alpha(s_1) \subseteq B_\alpha(s_2) \mid s_1 \sqsubseteq s_2]}{\Pr[B_\alpha(s_1) \subseteq B_\alpha(s_2)]} \right) \\
&= \Pr[s_1 \sqsubseteq s_2] \cdot \lim_{k\to\infty} \frac{\Pr[B_\alpha(s_1) \subseteq B_\alpha(s_2) \mid s_1 \sqsubseteq s_2]}{\Pr[B_\alpha(s_1) \subseteq B_\alpha(s_2)]} \\
&= \Pr[s_1 \sqsubseteq s_2].
\end{aligned} \tag{4}$$

With a very large value of k, there is no containment information leaked to an attacker who knows the domain probability space. However, since all the hash functions must be encoded in the Bloom filters for publications, the publications Bloom filters then have all their bits set to one. The probability of false positive then approaches 1 and prefiltering becomes useless. We illustrate these baselines in Figure 9(b), where $P_{\text{f\_pos}}$ approaches 1 and P_α approaches $\Pr[s_1 \sqsubseteq s_2] \approx 8.3 \cdot 10^{-9}$ as k increases.

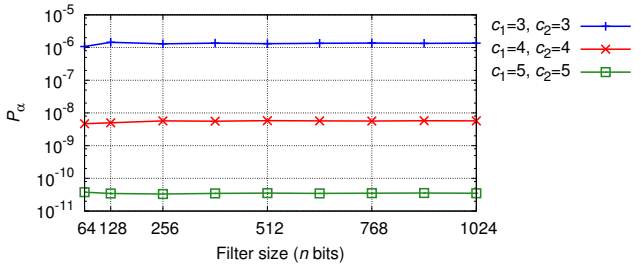
Figure 10(a) further illustrates the tradeoffs. In Figure 10(a), we choose k such that the false positive rate is always 3% ($P_{\text{f\_pos}} \approx 0.03$). We study how it affects P_α as α and the Bloom filter size vary. The observation is that for a fixed false positive rate, the containment confidentiality of the prefiltering is essentially fixed no matter how we vary the size of the Bloom filters and the cardinality of the hash functions set. One observation that can be drawn from this is that for complexity reasons we should set the Bloom filter size as small as we can get away with (for very small Bloom filters it is not possible to set the probability of false positive



(a) P_α when $P_{\text{f\_pos}} \approx 0.03$ ($c_1 = c_2 = 3$).



(b) P_α for varying n and different numbers of constraints ($\alpha = 1$).



(c) P_α when $P_{\text{f\_pos}} \approx 0.03$ ($c_p = 5$, $\alpha = 1$).

Fig. 10: Containment obfuscation and prefiltering efficiency tradeoffs, using $a = 10$, $v = 100$ and $k = 5$.

low enough). Typically, Bloom filters of size 64 or 128 are more than sufficient. Figure 10(a) also shows that the best tradeoff is reached when randomly selecting only $\alpha = 1$ hash function from the set. We mention that for some values of c_1 and c_2 we obtained a slightly better tradeoff for $\alpha = 2$ than for $\alpha = 1$, but we believe that the small gain, the increased complexity (much larger k) and infrequent occurrence do not really justify considering $\alpha > 1$. Despite the observed tradeoffs, the truncating strategy is a valuable mechanism. For instance, it can be observed from Figure 10(a) that for a 3% false positive rate, which is excellent, P_α is approximately $1 \cdot 10^{-6}$. This makes it challenging for an attacker to build an accurate containment graph and run statistical attacks using a set of encrypted subscriptions.

Figure 10(b) shows that the confidentiality of the prefiltering scheme increases dramatically when the number of equality constraints per subscription increases, but decreases slightly if an attacker tries to gain information by comparing a subscription with a small number of constraints with a second subscription with a large number of equality constraints. In Figure 10(c) we again choose k such that the false positive rate is always 3% ($P_{\text{f\_pos}} \approx 0.03$). It shows that the performance/confidentiality tradeoff is better when the number of subscription equality constraints approaches the number of publication values. Of course, in practical systems, subscribers do not necessarily generate subscriptions with a fixed number of equality attributes nor do publishers generate publications with a fixed number of values. When this occurs, the performance/confidentiality tradeoff must be approximated based on an estimation of the system workload.

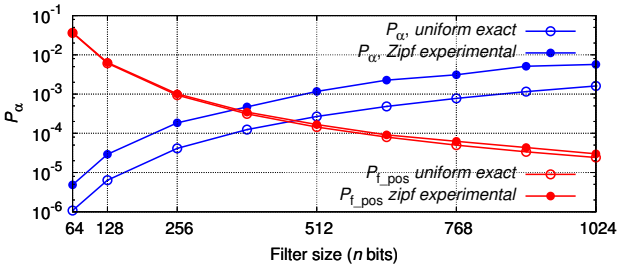


Fig. 11: Comparison of P_α and $P_{\text{f\_pos}}$ for uniform and non-uniform domains. Numerical simulations using $a = 10$, $v = 100$, $c_1 = c_2 = 3$, $c_p = 5$, $k = 5$ and $\alpha = 1$.

Finally, since real-world publication and subscription domains are rarely uniform, we also test our prefiltering scheme on a nonuniform domain. We use Zipf's law with exponential parameter 0, i.e., $\Pr[X = x] = \frac{\Pr[X=1]}{x}$ for $x = \{1, 2, 3, \dots\}$. We consider that the domain attributes follow Zipf's law; likewise, we assume that the possible values for each attribute also follow Zipf's law. We use $a = 10$, $v = 100$, 5 values per publication, 3 equality constraints per subscription, $k = 5$ and $\alpha = 1$. Non-uniform distributions like Zipf's law are harder to handle mathematically, thus we compute P_α and $P_{\text{f\_pos}}$ using numerical simulations. The results are shown in Figure 11. It can be observed that P_α is slightly higher when the domain follows Zipf's law, which is not surprising since the domain entropy is lower and the probability that the subscriptions share common values increases. $P_{\text{f\_pos}}$ is almost equal for small Bloom filters. This example provides evidence that our prefiltering scheme can be made secure and efficient with nonuniform real-life practical workloads.

5 Related Work

Our prefiltering operator usability relates with *encrypted matching schemes* in the context of privacy-preserving content-based publish/subscribe. Several such techniques have been described in the literature. We already presented ASPE [6] in our introduction. ASPE is adapted from secure kNN queries [7] used on databases, and only supports containment determination when encrypted subscriptions are augmented with additional reference vectors. C-CBPS [8] is another encrypted matching scheme. It proposes several distinct algorithms depending on the type of constraints and attributes. Most of these algorithms partition the attributes domain to build an approximation dictionary used as a set of reference points in the encryption process (the method can result in false positives). This shares similarities with the extension of prefiltering to range queries that we present in Appendix D. Containment evaluation can be supported by embedding extra information into the subscriptions, which adds a high overhead. The method described by Nabeel *et al.* [10] defines a matching scheme based on the public-key homomorphic Paillier cryptosystem. In order to support containment, the

scheme also requires additional encrypted values to be added to the subscriptions. The encrypted matching scheme in [9] is a commutative multiple encryption scheme, that uses the Pohlig-Hellman cryptosystem. It does not consider ranges nor coverage for subscriptions, and therefore does not support containment. In [12], Li *et al.* propose to support encrypted matching by transforming interval-matching into prefix-matching. The scheme then leverages prefix-preserving encryption, such that prefix matching can be performed on the encrypted subscriptions. The mechanism does not support containment. It presents an interesting approach but its security level is unclear according to [8]. Finally, in [11], Ion *et al.* present an encryption scheme that uses an attribute-based encryption approach, combined with an El Gamal-based searchable data encryption scheme for subscriptions. The scheme does not support containment, but was successfully leveraged for confidentiality-preserving filtering in the context of an e-Health application [28].

In [18], the authors use a two-step process for private record linkage. In a first preprocessing step, anonymized data sets quickly match or discard a large fraction of record pairs. This decreases the need of a costly secure multi-party computation protocol. Several performance/anonymity tradoffs are explored. The two-step process uses different techniques in a different context, but the main objective is the same as ours: improve performance without leaking information.

Our prefiltering technique makes use of Bloom filters. This data structure is also used in other work relating to confidentiality or security purposes in distributed systems. Kerschbaum [29] use Bloom filters to protect supply chains information against, for instance, malicious suppliers counterfeiting products. Shikfa *et al.* [30] use counting Bloom filters to securely compute a weighted matching ratio in a broker-based scenario. Perl *et al.* [31] use Bloom filters for a form of prefiltering of DNA search using homomorphic encryptions. They use pollution to obfuscate the filters, which is complementary to the pruning approach we use in this article. Goh [32] also use Bloom filter pollution in the context of a secure indexing technique allowing word searches over a collection of encrypted documents. Bellovin and Cheswick [33] briefly mention that randomizing the inputs could be beneficial for using Bloom filters for encrypted search, but they did not develop the idea further. Kuzu *et al.* [34] perform a cryptanalysis on Bloom filters used in private record linkage applications. This applies to keywords n-grams which are encoded in Bloom filters. Their work shares some characteristics with ours. Their Bloom filters are not encrypted, and they make similar assumptions on the information available to attackers (domain probability space and number of hash functions). Readers interested in other general uses of Bloom filters in distributed systems may refer to the survey [35]. In particular, Bloom filters have been used in content-based pub/sub for speeding up non-encrypted matching in [36].

6 Conclusion

Content-based routing middleware provides multiple advantages in terms of flexibility, scalability, and simplicity for the development of distributed applications. A major obstacle to the wider adoption of these techniques is their confidentiality: messages, individual subscriptions, and containment relationships among a set of subscriptions may reveal important information about a user or group of users. While encrypted routing hides the content of messages and subscriptions, it requires costly encrypted processing algorithms that make the filtering operation computationally prohibitive for high-throughput systems.

Prefiltering, and when available containment relationships between subscriptions, allow us to greatly reduce the performance gap between non-encrypted and encrypted filtering. By adding Bloom filters that encode publication values and subscription equality constraints, we can discard a large fraction of subscriptions before reaching the costly encrypted filtering operation. Evaluations confirmed that our mechanisms reduce the number of such costly operations required to filter an incoming publication by approximately one order of magnitude. Furthermore, when selecting a subset of the hash functions for encoding subscription equality constraints in the Bloom filters, we showed that it is still possible to prefilter a large fraction of the subscriptions while leaking very little containment information to potential attackers.

Several extensions of this work are possible. In Appendix D, we discuss how Bloom filters could be used to prefilter inequality constraints. We also believe that our results and techniques can be applied to many other problems in distributed systems where set inclusions are represented using a compact structure such as Bloom filters, and where confidentiality requirements are stringent. One such application is encrypted distributed databases [37], [38]. The rigorous mathematical analysis presented in this article could also be applied to other problems that were presented empirically, for instance the filter obfuscation technique presented by Perl *et al.* [31].

References

[1] P. Eugster, P. Felber, R. Guerraoui, and A.-M. Kermarrec, "The many faces of publish/subscribe," *ACM Computing Surveys*, vol. 35, no. 2, pp. 114–131, 2003.

[2] R. P. Barazzutti, P. Felber, C. Fetzer, E. Onica, M. Pasin, J.-F. Pineau, E. Rivière, and S. Weigert, "StreamHub: A massively parallel architecture for high-performance content-based publish/subscribe," in *7th ACM International Conference on Distributed Event-Based Systems*, ser. DEBS, July 2013.

[3] T. Ristenpart, E. Tromer, H. Shacham, and S. Savage, "Hey, you, get off of my cloud: exploring information leakage in third-party compute clouds," in *16th ACM conference on Computer and communications security*, ser. CCS, 2009.

[4] J. Somorovsky, M. Heiderich, M. Jensen, J. Schwenk, N. Gruschka, and L. Lo Iacono, "All your clouds are belong to us: security analysis of cloud management interfaces," in *Proceedings of the 3rd ACM workshop on Cloud computing security workshop*, ser. CCSW, 2011.

[5] S. Liston, "The Cloud: data protection and privacy – Whose cloud is it anyway?" in *12th Global Symposium for Regulators*, ser. GSR, 2012.

[6] S. Choi, G. Ghinita, and E. Bertino, "A privacy-enhancing content-based publish/subscribe system using scalar product preserving transformations," in *Database and Expert Systems Applications*, ser. Lecture Notes in Computer Science, P. Bringas, A. Hameurlain, and G. Quirchmayr, Eds., vol. 6261. Springer Berlin Heidelberg, 2010.

[7] W. K. Wong, D. W.-l. Cheung, B. Kao, and N. Mamoulis, "Secure kNN computation on encrypted databases," in *35th SIGMOD international conference on Management of data*, 2009.

[8] C. Raiciu and D. S. Rosenblum, "Enabling confidentiality in content-based publish/subscribe infrastructures," in *IEEE Securecomm*, 2006.

[9] A. Shikfa, M. Önen, and R. Molva, "Privacy-preserving content-based publish/subscribe networks," in *Emerging Challenges for Security, Privacy and Trust*, ser. IFIP Advances in Information and Communication Technology. Springer Boston, 2009.
[10] M. Nabeel, N. Shang, and E. Bertino, "Privacy-preserving filtering and covering in content-based publish subscribe systems," Purdue University, West Lafayette, Indiana, USA, Tech. Rep. 15, 2009.
[11] M. Ion, G. Russello, and B. Crispo, "Supporting publication and subscription confidentiality in pub/sub networks," in *6th Intl ICST Conference on Security and Privacy in Communication Networks*, ser. SecureComm, 2010.
[12] J. Li, C. Lu, and W. Shi, "An efficient scheme for preserving confidentiality in content-based publish-subscribe systems," College of Computing, Georgia Institute of Technology, Tech. Rep. GIT-CC-04-01, 2004.
[13] H. Jafarpour, S. Mehrotra, N. Venkatasubramanian, and M. Montanari, "MICS: an efficient content space representation model for publish/subscribe systems," in *3rd ACM International Conference on Distributed Event-Based Systems*, ser. DEBS, 2009.
[14] A. Carzaniga, D. S. Rosenblum, and A. L. Wolf, "Design and evaluation of a wide-area event notification service," *ACM Transactions on Computer Systems*, vol. 19, no. 3, 2001.
[15] H.-A. Jacobsen, A. Cheung, G. Lia, B. Maniymaran, V. Muthusamy, and R. S. Kazemzadeh, "The PADRES publish/subscribe system," in *Handbook of Research on Advanced Distributed Event-Based Systems, Publish/Subscribe and Message Filtering Technologies*, 2009.
[16] R. Barazzutti, P. Felber, H. Mercier, E. Onica, and E. Rivière, "Thrifty privacy: efficient support for privacy-preserving publish/-subscribe," in *6th ACM International Conference on Distributed Event-Based Systems*, ser. DEBS, 2012.
[17] H. Mercier, E. Onica, E. Rivière, and P. Felber, "Performance/security tradeoffs for content-based routing supported by Bloom filters," in *20th Intl Colloquium on Structural Information and Comm. Complexity*, ser. SIROCCO, Ischia, Italy, July 2013.
[18] A. Inan, M. Kantarcioglu, E. Bertino, and M. Scannapieco, "A hybrid approach to private record linkage," in *2014 IEEE 30th International Conference on Data Engineering*, 2008, pp. 496–505.
[19] B. H. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Comm. of the ACM*, vol. 13, no. 7, 1970.
[20] M. Bellare, R. Canetti, and H. Krawczyk, "Keying hash functions for message authentication," in *16th Annual Intl. Cryptology Conf. on Advances in Cryptology*, ser. CRYPTO, 1996.
[21] A. T. Sherman and D. A. McGrew, "Key establishment in large dynamic groups using one-way function trees," *IEEE Trans. on Software Engineering*, vol. 29, no. 5, 2003.
[22] J. Bacon, D. M. Eyers, J. Singh, and P. R. Pietzuch, "Access control in publish/subscribe systems," in *2nd intl. Conference on Distributed Event-Based Systems*, ser. DEBS, 2008.
[23] E. Shi, J. Bethencourt, T.-H. H. Chan, D. Song, and A. Perrig, "Multi-dimensional range query over encrypted data," in *Proceedings of the 2007 IEEE Symposium on Security and Privacy*, ser. SP, 2007, pp. 350–364.
[24] G. E. Andrews, *The Theory of Partitions*. Cambridge Mathematical Library, 1998.
[25] R. L. Graham, D. E. Knuth, and O. Patashnik, *Concrete Mathematics: A Foundation for Computer Science*, 2nd ed. Addison-Wesley Longman Publishing, 1994.
[26] A. Gupta, O. D. Sahin, D. Agrawal, and A. E. Abbadi, "Meghdoot: content-based publish/subscribe over p2p networks," in *5th ACM/IFIP/USENIX Intl. conference on Middleware*, 2004.
[27] M. E. J. Newman, "Power laws, Pareto distributions and Zipf's law," *Contemporary Physics*, vol. 46, pp. 323–351, 2005.
[28] M. Ion, G. Russello, and B. Crispo, "An implementation of event and filter confidentiality in pub/sub systems and its application to e-health," in *17th ACM conference on Computer and communications security*, ser. CCS, 2010.
[29] F. Kerschbaum, "Public-key encrypted Bloom filters with applications to supply chain integrity," in *Data and Applications Security and Privacy XXV*, ser. Lecture Notes in Computer Science, Y. Li, Ed., vol. 6818, 2011.
[30] A. Shikfa, M. Önen, and R. Molva, "Broker-based private matching," in *11th international conference on Privacy enhancing technologies*, ser. PETS, 2011.
[31] H. Perl, Y. Mohammed, M. Brenner, and M. Smith, "Fast confidential search for bio-medical data using Bloom filters and homomorphic cryptography," in *E-Science (e-Science), 2012 IEEE 8th International Conference on*, 2012, pp. 1–8.
[32] E.-J. Goh, "Secure indexes," *Cryptology ePrint Archive, Report 2003/216*, 2003.
[33] S. M. Bellovin and W. R. Cheswick, "Privacy-enhanced searches using encrypted Bloom filters," *Cryptology ePrint Archive, Report 2004/022*, 2004.
[34] M. Kuzu, M. Kantarcioglu, E. Durham, and B. Malin, "A constraint satisfaction cryptanalysis of Bloom filters in private record linkage," in *Privacy Enhancing Technologies*, ser. Lecture Notes in Computer Science, vol. 6794, 2011.
[35] S. Tarkoma, C. E. Rothenberg, and E. Lagerspetz, "Theory and practice of Bloom filters for distributed systems," *IEEE Communications Surveys Tutorials*, vol. 14, no. 1, 2012.
[36] Z. Jerzak and C. Fetzer, "Bloom filter based routing for content-based publish/subscribe," in *2nd international conference on Distributed event-based systems*, ser. DEBS, 2008.
[37] R. A. Popa, C. M. S. Redfield, N. Zeldovich, and H. Balakrishnan, "CryptDB: protecting confidentiality with encrypted query processing," in *23rd ACM Symposium on Operating Systems Principles*, ser. SOSP, 2011.
[38] M. Kurpicz, "Towards efficient privacy-preserving SQL processing in untrusted clouds," Master's thesis, University of Neuchâtel, July 2013.

Raphaël Barazzutti received his Master degree in Computer Science from the Swiss Federal Institute of Technology (EPFL) in 2009. He has worked as a development engineer at Swissquote Bank starting February 2009. Since October 2010 he is enrolled in a PhD programme at the University of Neuchâtel, Switzerland.

Pascal Felber received his M.Sc. and Ph.D. degrees in Computer Science from the Swiss Federal Institute of Technology (EPFL). He has then worked at Oracle Corporation and Bell-Labs in the USA, and at Institut EURECOM in France. Since 2004, he is a Professor of Computer Science at the University of Neuchâtel, Switzerland, working in the field of dependable, distributed, and concurrent systems. He has published over 120 research papers in various journals and conferences.

Hugues Mercier received the Ph.D. degree in electrical and computer engineering from the University of British Columbia in 2008. From 2008 to 2011, he was a postdoctoral research fellow at the Harvard School of Engineering and Applied Sciences, and at McGill University. Currently, he is a lecturer at the Université de Neuchâtel in Switzerland. His research interests include combinatorics and information theory.

Emanuel Onica is a lecturer at the Alexandru Ioan Cuza University of Iaşi, Romania. He received his Ph.D. degree in Computer Science from University of Neuchâtel, Switzerland in 2014, where he worked as scientific collaborator from 2010 to 2014. His current research interests lie in the area of distributed event based systems, with a focus on privacy.

Etienne Rivière is a lecturer at the University of Neuchâtel, Switzerland. He received his PhD in Computer Science from the University of Rennes, France in November 2007. In 2008 and 2009, Étienne has been a post-doctoral fellow under an "Alain Bensoussan" grant from ERCIM, which led him to the University of Neuchâtel and to NTNU Trondheim in Norway. His research interests lie in large-scale distributed systems and concurrent systems.

APPENDIX A
OTHER ATTACKS

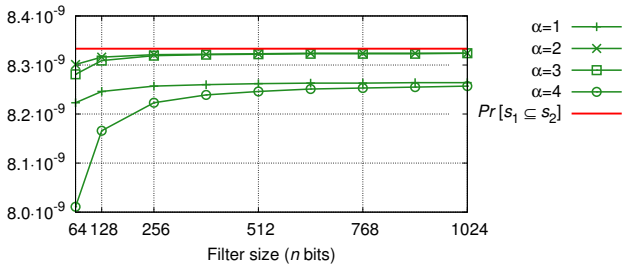


Fig. 12: $\overline{P}_\alpha$ for Bloom filters of varying size ($c_1 = c_2 = 3$).

In section 3, we studied how subscription containment information could be derived from Bloom filter containment. An attacker can also try to derive containment relationships based on mismatched Bloom filters, i.e., Bloom filters $B_\alpha(s_1)$ and $B_\alpha(s_2)$ such that $B_\alpha(s_1) \not\subseteq B_\alpha(s_2)$ and $B_\alpha(s_2) \not\subseteq B_\alpha(s_1)$. Let us consider the probability $\overline{P}_\alpha \triangleq \Pr[s_1 \sqsubseteq s_2 \mid B_\alpha(s_1) \not\subseteq B_\alpha(s_2)]$ which from Bayes' law can be rewritten as

$$\begin{aligned}\overline{P}_\alpha &= \frac{\Pr[s_1 \sqsubseteq s_2] \cdot \Pr[B_\alpha(s_1) \not\subseteq B_\alpha(s_2) \mid s_1 \sqsubseteq s_2]}{\Pr[B_\alpha(s_1) \not\subseteq B_\alpha(s_2)]} \\ &= \frac{\Pr[s_1 \sqsubseteq s_2] \cdot (1 - \Pr[B_\alpha(s_1) \subseteq B_\alpha(s_2) \mid s_1 \sqsubseteq s_2])}{1 - \Pr[B_\alpha(s_1) \subseteq B_\alpha(s_2)]}.\end{aligned} \tag{5}$$

$\overline{P}_\alpha$ can be evaluated using the formulas derived earlier in this section. It should be clear that $\overline{P}_k = 0$ and that $\overline{P}_\alpha > 0$ for $\alpha < k$, so at first sight randomly choosing some of the hash functions appears to provide more information to a potential attacker than selecting all the hash functions. This is misleading, since $\overline{P}_\alpha$ essentially increases from zero to $\Pr[s_1 \sqsubseteq s_2]$, providing no more information to an attacker than from random subscriptions. This is a welcome tradeoff considering how our hashing strategy decreases the amount of information leaked from contained subscriptions. An example comparing $\overline{P}_\alpha$ and $\Pr[s_1 \sqsubseteq s_2]$ is displayed in Figure 12.

Finally, although it is unclear what could be done with such information, an attacker could try to derive "noncontainment" attributes about pairs of encrypted subscriptions. In that case, the probability of interest is $\Pr[s_1 \not\sqsubseteq s_2 \mid B_\alpha(s_1) \not\subseteq B_\alpha(s_2)]$ which with Bayes' law and a little work can be rewritten as

$$\begin{aligned}&\Pr[s_1 \not\sqsubseteq s_2 \mid B_\alpha(s_1) \not\subseteq B_\alpha(s_2)] \\ &\quad = 1 - \frac{\Pr[s_1 \sqsubseteq s_2] \cdot \Pr[B_\alpha(s_1) \not\subseteq B_\alpha(s_2) \mid s_1 \sqsubseteq s_2]}{\Pr[B_\alpha(s_1) \not\subseteq B_\alpha(s_2)]}.\end{aligned} \tag{6}$$

Interestingly, $\Pr[s_1 \not\sqsubseteq s_2 \mid B_\alpha(s_1) \not\subseteq B_\alpha(s_2)] = 1$ with $\alpha = k$ and smaller than 1 when $\alpha < k$. This means that when $\alpha < k$, the attacker cannot even conclude with certainty that two subscriptions are not included into each other.

APPENDIX B
PROOFS FROM SECTION 3

We present in this appendix the proofs of Lemmas 1, 3 and 5 presented in Section 3.

B.1 Proof of Lemma 1

Proof: There are $\binom{a}{c_1}$ ways to choose c_1 attributes and v possible values for each attribute, thus the number of ways to choose the values for Subscription s_1 is given by

$$\binom{a}{c_1} v^{c_1}. \tag{7}$$

We now fix Subscription s_2 and count, among the possible ways to assign the values for s_1, how many of them have exactly γ common values with s_2. First, the number ways to select the attributes containing the values in s_1 which are also common to s_2 is given by

$$\binom{c_2}{\gamma}. \tag{8}$$

The $c_1 - \gamma$ values of s_1 that must not be in s_2 fall in one of the following two categories: either they are associated with an attribute that is also present in s_2 (but must take a different value) or they are from an attribute not present in s_2. Let δ be the number of values of s_1 that fall in the first category. Since there are $c_2 - \gamma$ attributes to choose from, each with $v-1$ allowed values, the number of ways to assign δ values of s_1 in the first category is

$$\binom{c_2 - \gamma}{\delta} (v-1)^\delta. \tag{9}$$

With γ common values and δ different values in the first category, there are $c_1 - \delta - \gamma$ values of s_1 that must be put in the second category. Since there are $a - c_2$ attributes to choose from, and since all v values per attribute are allowed, the number of ways to assign $c_1 - \gamma - \delta$ values in the second category is

$$\binom{a - c_2}{c_1 - \gamma - \delta} v^{c_1 - \gamma - \delta}. \tag{10}$$

From (8)-(10) and the fact that δ can vary between 0 and $c_1 - \gamma$, the number of ways to assign values to s_1 with γ common values with s_2 is

$$\binom{c_2}{\gamma} \cdot \sum_{\delta=0}^{c_1-\gamma} \left[\binom{c_2 - \gamma}{\delta} \binom{a - c_2}{c_1 - \gamma - \delta} (v-1)^\delta v^{c_1 - \gamma - \delta} \right]. \tag{11}$$

The probability that s_1 and s_2 have exactly γ common values follow from (11) and (7). □

B.2 Proof of Lemma 3

Proof: We first explain why we must consider all the partitions $\lambda \vdash \text{IntPart}(g, \gamma, \alpha_{\min}, \alpha)$ (we recall that $\text{IntPart}(g, \gamma, \alpha_{\min}, \alpha)$ is the set of all integer partitions of g into exactly γ parts with the additional constraint that the size of each part is at least $\alpha_{\min}$ and at most α). The g *good hashes* must be chosen among the hash functions of the γ common values between s_1 and s_2, thus we consider the partitions of g into exactly γ parts. Since there are α hashes per attribute value, each part of the partitions is at most α. Furthermore, when $\alpha > \frac{k}{2}$, it is not possible that the hashes chosen by s_1 and s_2 for a common value are mutually exclusive. More precisely, the number of good hashes per common value is at least $\alpha_{\min} = \max(0, 2\alpha - k)$, which explains why the size of each part in the partitions is at least $\alpha_{\min}$. The reason to consider all such partitions is that they have different probabilities of occurrence when $\alpha > 1$. For instance, selecting two good hashes for one common value between subscriptions s_1 and s_2 does not have the same probability as selecting one good hash for two different common values.

Consider a fixed partition $\lambda \vdash \text{IntPart}(g, \gamma, \alpha_{\min}, \alpha)$ using the frequency representation

$$(\alpha_{\min}{}^{p_{\alpha_{\min}}} \alpha_{\min+1}{}^{p_{\alpha_{\min+1}}} \dots \alpha^{p_\alpha}),$$

meaning that the value α_i appears p_{α_i} times in λ. The number of ways to assign the number of good hashes for each common value for our fixed λ corresponds to the number of different permutations of γ objects, where there are p_{α_i} indistinguishable objects of type α_i for $\alpha_{min+1} \leq \alpha_i \leq \alpha$. This is given by the multinomial coefficient

$$\binom{\gamma}{p_{\alpha_{\min}}, p_{\alpha_{min+1}}, \dots, p_\alpha} \triangleq \frac{\gamma!}{p_{\alpha_{\min}}! \cdot p_{\alpha_{min+1}}! \cdot \dots \cdot p_\alpha!}. \tag{12}$$

We now fix the number of *good hashes* for each attribute and calculate the probability of occurrence of this assignment. The probability of having β good hashes (and $\alpha - \beta$ lucky hashes) for a common value between both subscriptions is given by

$$\frac{\binom{\alpha}{\beta}\binom{k-\alpha}{\alpha-\beta}}{\binom{k}{\alpha}} \tag{13}$$

and it follows that the probability of having g *good hashes* for a given number of good hashes per common value is given by

$$\prod_{\beta=\alpha_{\min}}^{\alpha} \left[\frac{\binom{\alpha}{\beta}\binom{k-\alpha}{\alpha-\beta}}{\binom{k}{\alpha}}\right]^{p_\beta}. \tag{14}$$

From (13) and (14), the probability of having g *good hashes* for a fixed partition λ is

$$\binom{\gamma}{p_{\alpha_{\min}}, p_{\alpha_{min+1}}, \dots, p_\alpha} \prod_{\beta=\alpha_{\min}}^{\alpha} \left[\frac{\binom{\alpha}{\beta}\binom{k-\alpha}{\alpha-\beta}}{\binom{k}{\alpha}}\right]^{p_\beta} \tag{15}$$

and by considering all partitions $\lambda \vdash \text{IntPart}(g, \gamma, \alpha_{\min}, \alpha)$ we can conclude that

$$\Pr_{\text{good}}[g, \gamma] = \sum_{\lambda \vdash \text{IntPart}(g,\gamma,\alpha_{\min},\alpha)} \binom{\gamma}{p_{\alpha_{\min}}, p_{\alpha_{min+1}}, \dots, p_\alpha} \prod_{\beta=\alpha_{\min}}^{\alpha} \left[\frac{\binom{\alpha}{\beta}\binom{k-\alpha}{\alpha-\beta}}{\binom{k}{\alpha}}\right]^{p_\beta}. \tag{16}$$

□

B.3 Proof of Lemma 5

Proof: The probability that a Bloom filter of size n with c constraints and α hash functions per constraint has exactly i nonzero bits is

$$\frac{\binom{n}{i} \cdot i! \cdot \left\{ {c\alpha \atop i} \right\}}{n^{c\alpha}} \tag{17}$$

where $\left\{ {a \atop b} \right\}$ stands for the Stirling numbers of the second kind [25]. More precisely, there are $\binom{n}{i}$ distinct Bloom filters with i nonzero bits, each occurring with probability

$$\frac{i! \cdot \left\{ {c\alpha \atop i} \right\}}{n^{c\alpha}}. \tag{18}$$

We prove the lemma with two subscriptions s_1 and s_2. We remove the bits of $B_\alpha(s_1)$ obtained from the *good hashes* between s_1 and s_2 and only consider the h_1 *lucky hashes* that need to be randomly covered by the h_2 hashes of s_2. Let $B'_\alpha(s_1)$ be the Bloom filter of s_1 with only the h_1 lucky hash functions.

$$\Pr_{\text{lucky}}[h_1, h_2] = \sum_{i=1}^{h_2} \sum_{B'_\alpha(s_2) \text{ with } i \text{ nonzero bits}} \Pr[B'_\alpha(s_2)] \cdot \Pr[B'_\alpha(s_1) \subseteq B'_\alpha(s_2)] \tag{19}$$

and from (17) and (18) it follows that

$$\begin{aligned}
\Pr_{\text{lucky}}[h_1, h_2] &= \sum_{i=1}^{h_2} \sum_{\delta=1}^{\binom{n}{i}} \left[\frac{i! \cdot \left\{ {h_2 \atop i} \right\}}{n^{h_2}} \sum_{j=1}^{i} \frac{\binom{n}{j} \cdot j! \cdot \left\{ {h_1 \atop j} \right\}}{n^{h_1}} \cdot \frac{\binom{i}{j}}{\binom{n}{j}}\right] \\
&= \sum_{i=1}^{h_2} \left[\frac{\binom{n}{i} \cdot i! \cdot \left\{ {h_2 \atop i} \right\}}{n^{h_2}} \sum_{j=1}^{i} \frac{\binom{n}{j} \cdot j! \cdot \left\{ {h_1 \atop j} \right\}}{n^{h_1}} \cdot \frac{\binom{i}{j}}{\binom{n}{j}}\right] \\
&= \frac{1}{n^{h_1+h_2}} \sum_{i=1}^{h_2} \left[\binom{n}{i} \cdot i! \cdot \left\{ {h_2 \atop i} \right\} \sum_{j=1}^{i} \binom{i}{j} \cdot j! \cdot \left\{ {h_1 \atop j} \right\}\right].
\end{aligned} \tag{20}$$

□

APPENDIX C
ADDITIONAL EVALUATIONS: TUNING BLOOM FILTERS

We study in this section the influence of Bloom filter parameters on the efficiency of the prefiltering variants. This Section complements the evaluation presented in Section 4.1.2 of the main part of the paper, where we use a fixed setup of Bloom filters of 128 bits and 3 hash functions.

Figure 13 shows the tests ratios as function of the size of the Bloom filters (top) and the number of hash functions (bottom).

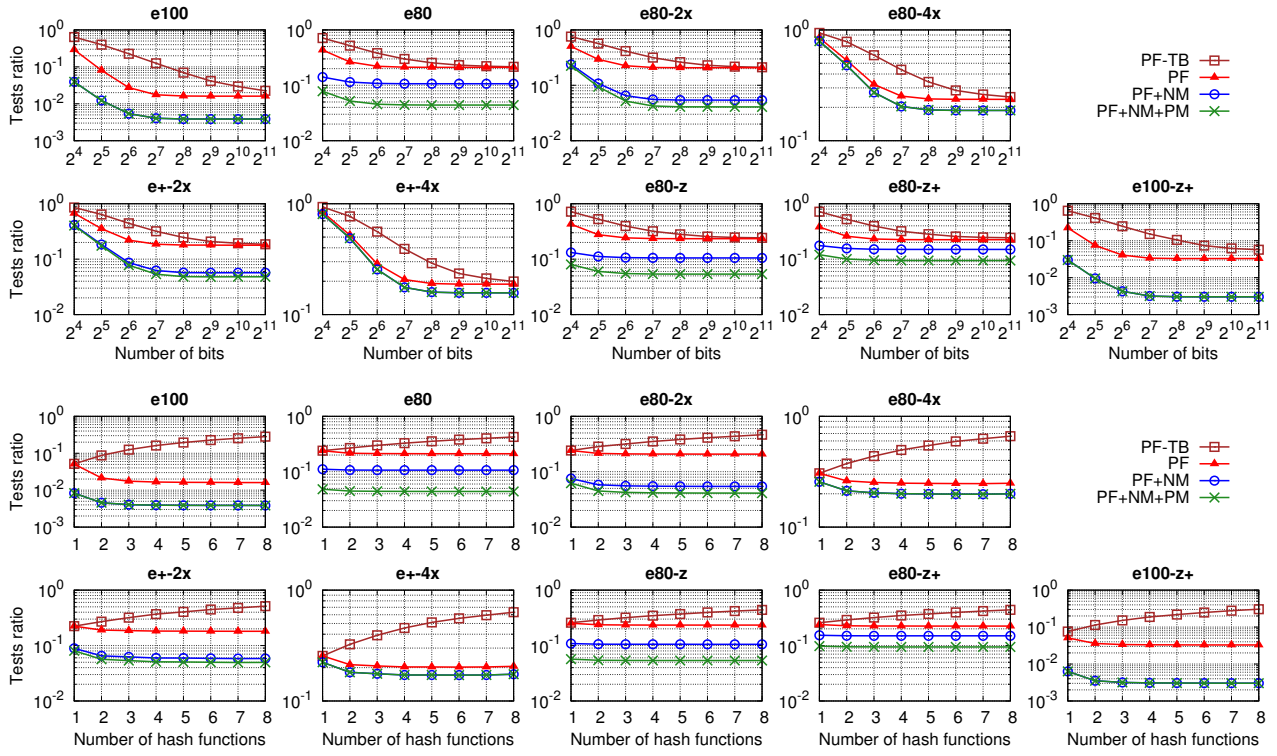


Fig. 13: Evolution of the number of tests when varying the size of the Bloom filters (top, using k =3 hash functions) and the number of hash functions (bottom, using n =128-bit Bloom filters).

A first observation is that the default values we have chosen (128-bit Bloom filters and 3 hash functions) perform well with all workloads and most algorithm variants. The benefits one can expect from larger Bloom filters and more hash functions are not significant, although for workloads with more attributes and equality predicates one should likely increase the size of the Bloom filters.

`PF-TB` is the exception: its efficiency continues to increase with larger Bloom filters, reaching almost the performance of `PF` with 2,048 bits. This shows that the higher rate of false positives resulting from truncation can be compensated by properly dimensioning the Bloom filters. Conversely, with more hash functions, the performance of `PF-TB` degrades faster than the performance of `PF` because the number of bits in the Bloom filters of the subscriptions does not change (only one hash function is used), while the Bloom filters of the publications become more densely populated, yielding additional false positives.

APPENDIX D
EXTENSION: PREFILTERING INEQUALITIES

In addition to the filtering of equality constraints, it would also be interesting to prefilter inequality constraints efficiently. We present an extension that is a first step in this direction. One way to achieve this is to adapt a technique using reference points for searching in dictionaries. Raiciu *et al.* [8] present an application to publish/subscribe systems of such a technique. The idea is to partition the domains in smaller intervals, and to use the boundaries of these intervals in inequality Bloom filters to approximate inequality constraints. An example is given by Figure 14. In this example, the domain $V = \{1, 2, \ldots, 100\}$ is partitioned into five intervals, each containing 20 values. If a subscription contains the constraint $v > 25$, the closest interval boundary covering the inequality, in this case ≥ 20, will be encoded in its inequality Bloom filter. Each incoming publication must encode all the interval boundaries covering its values. In the figure, the publication is $v = 30$. The interval bounds ≥ 1, ≥ 20, ≤ 100, ≤ 80, ≤ 60 and ≤ 40 thus must all be encoded in its inequality Bloom filter. The inequality Bloom filters of subscriptions and publications can then be compared exactly like the Bloom filters for equality constraints. This can provide a second prefiltering stage executed after the prefiltering on equality constraints. It should be noted that failure to encode all the covering interval bounds in publications may lead to false negatives. Interestingly, the truncation containment obfuscation technique discussed in Section 3 can also be applied to the inequality Bloom filters.

One additional drawback of prefiltering inequalities is the amount of leaked information from the Bloom filters. The analysis of containment information leaked from the subscriptions inequality filters is similar to the analysis done for equality constraints, with two differences: on the one hand, the size of the subscription domain is reduced to twice the number of interval boundaries, but on the other hand those boundaries do not correspond to the exact inequalities. Unfortunately, the size of the publication domain is also reduced to the number of interval boundaries, which can be very small.

We simulated inequality prefiltering with uniform publication and subscription domains of $a = 10$ attributes and

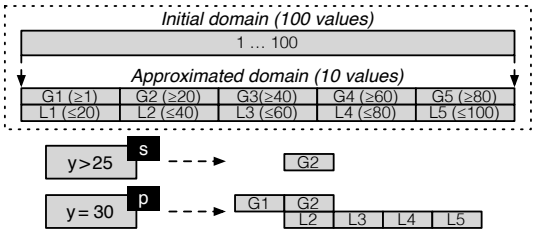


Fig. 14: Inequality prefiltering pre-mapping mechanism.

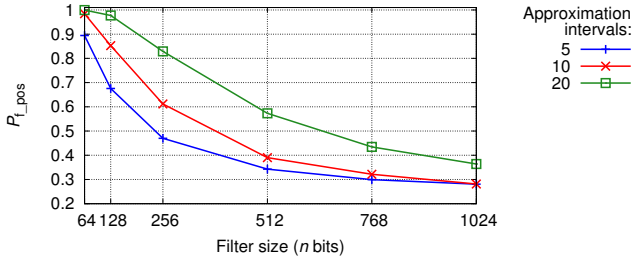


Fig. 15: Probability of false positives for inequality prefiltering as the size of the Bloom filters and the number of domain intervals vary, using $a = 10$, $v = 100$, $c_1 = 3$, $c_p = 5$, $k = 5$ and $\alpha = 1$.

$v = 100$ values per attribute, $c_1 = 3$ inequality constraints per subscription, $c_p = 5$ attributes per publication, and one hash function randomly chosen out of 5 for the subscriptions Bloom filters. The results are in Figure 15 for $\{5, 10, 20\}$ equal-size domain partitions. For a Bloom filter of 256 bits and 5 domain intervals, we can prefilter more than 50% of the non-matching publications. Unsurprisingly, the performance is significantly worse than for equality constraints. This is due to the difference between the real constraints and the interval boundaries, but more importantly to the large number of values that must be encoded in the publications Bloom filters.