Private Set Relations with Bloom Filters for Outsourced SLA Validation *

Louis Tajan[1], Dirk Westhoff[1], and Frederik Armknecht[2]

[1] Hochschule Offenburg, Offenburg, Germany firstname.name@hs-offenburg.de
[2] University of Mannheim, Mannheim, Germany armknecht@uni-mannheim.de

Abstract. In the area of cloud computing, judging the fulfillment of service-level agreements on a technical level is gaining more and more importance. To support this we introduce privacy preserving set relations as *inclusiveness* and *disjointness* based on Bloom filters. We propose to compose them in a slightly different way by applying a keyed hash function. Besides discussing the correctness of the set relations, we analyze how this impacts the privacy of the sets content as well as providing privacy on the sets cardinality. Indeed, our solution proposes to bring another layer of privacy on the sizes. We are in particular interested how the *overlapping bits* of a Bloom filter impact the privacy level of our approach. We concretely apply our solution to a use case of cloud security audit on access control and present our results with real-world parameters.

Keywords: Bloom filters · set operations · set relations · outsourced computation

1 Introduction

When investigating the area of distributed systems, and in particular service provisioning cloud systems based on Service-Level Agreements (SLAs) between user and service provider we observe a strong tendency to involve a third party [7, 14,17,22]. The third party is in role of auditing and judging in a semi-automated manner whether the offered services have indeed been successfully provided for a given epoch in the past. We are interested in dependable and privacy-preserving building blocks for a third party entity, let it be an auditor or other dependable third parties. Besides other digital evidences which are out of the scope of the work at hand, by using a kind of a whitelist approach an auditor would have to store i) a whitelist and ii) some log-lists for a given duration of the past to validate access to an offered service from different users respectively their identities e.g. ip-addresses [21]. More concretely, we are interested in building blocks which can be used to verify with a high level of confidence and in a fully privacy preserving manner whether indeed users from a given group have

* The presented work was funded by the Baden-Württemberg Stiftung within the project PAL SAaaS Building Triangular Trust for Secure Cloud Auditing.

accessed the service, or not. Thus, our objective is to provide a derivate of private set operations, namely private set relations on sets (let it be a whitelist, or, a general SLA) as well as multisets or sequences (let it be a simple loglist, or, more general, a contractual intermediate state). In many related works, set operations and relations are computed by multiparty based protocols, as in [13] where the authors propose a method that enables two parties to construct a decision tree without learning anything on each other's data. All the parties involved are sharing a set or a subset of their data with each other such that they all contribute to the computations. However, in the work at hand, we face different protocol requirements. We want all the parties that own data to upload it to an outsourced third party that will perform solely by itself the required computations. Such scenario has gained relevance with the increased usage of cloud infrastructures. Contrary to multiparty based solutions, we believe that preferably such a protocol class should be non-interactive. E.g. for an indeed practical cloud auditing setting it is required that the user does not have to be available for any validation step after having used a given cloud service. We argue that in particular this non-interactive character is valuable, making our approach beneficial compared to competing approaches.

Our Contributions. The concrete contributions of our work are threefold:

- We allow a third party entity to compute set relations namely, *inclusiveness* and *disjointness* in an outsourced model to perform a security auditing use case. To do so, we tune the Bloom filter approach by enhancing its privacy with respect to the sets content. Such an approach is also providing privacy on the sets cardinality.
- We present an attack to gain the sequences cardinality in the present configuration. By analyzing the behavior of *overlapping bits* in the Bloom filter environment we show what amount of information such an attack may provide.
- We implemented our solution and present our results obtained for the concrete cloud security audit on access control use case with real-world parameters.

One may argue that a simple pseudonymization could be sufficient for the above sketched scenario, as solely apply a keyed hash function on the sets [4]. However, even if the pseudonymization function remains private to any other party than the Bloom filter owners, one may directly gain knowledge on the number of common elements of two Bloom filters. Such a naive approach will also reveal which pseudonym is present in none, one or both sets. On the contrary, Bloom filter representation has the particular feature of adding obfuscation to the sets.

Organization of the Paper. In Section 4 we present the selected use case scenario along with its security model. In Section 3 we provide an overview of the existing solutions in the literature and we argue why our approach using

Bloom filters is relevant. In Section 2 we provide a short introduction to the subsequently discussed set relations and Bloom filters. Section 5 presents our proposed protocols and the analysis of its correctness properties for different parameters choices. In Section 6 we analyze the privacy fulfilled by our solution by proposing an attack based on the overlapping bits characteristic, as we coined it. In Section 7 we present our results obtained by our implementation before concluding in Section 8.

2 Preliminaries

In this section we introduce the type of operations we are performing on sets and how we represent them using the Bloom filters approach. In appendix A we summarize the used notation.

Set Operations and Relations. Multiple types of operations could be performed on sets. In this work, we aim to test set relations and some of them could be reduced to compute the cardinality of some operations. For instance, being able to compute the cardinality of the intersection of two sets indicates whether they have elements in common or if one set is included in the other one. For privacy concerns it could be of interest to solely reveal its cardinality instead of the intersection itself. Therefore, we propose a solution to solve two kinds of set relations namely the *inclusiveness* and the *disjointness* and define this as follows:

Definition 1 (Inclusiveness). *Let $\mathcal{A}$ and $\mathcal{B}$ be finite sets. $\mathcal{A}$ is included in $\mathcal{B}$, i.e. $\mathcal{A} \subset \mathcal{B}$, iff all elements from $\mathcal{A}$ are included in $\mathcal{B}$: $\forall a \in \mathcal{A} : a \in \mathcal{B}$.*

Definition 2 (Disjointness). *Let $\mathcal{A}$ and $\mathcal{B}$ be finite sets. $\mathcal{A}$ and $\mathcal{B}$ are disjoint iff none of the elements from $\mathcal{A}$ are included in $\mathcal{B}$. In other words, $\mathcal{A} \cap \mathcal{B} = \emptyset$: $\forall a \in \mathcal{A} : a \notin \mathcal{B}$.*

In the remainder of this work, we will propose privacy preserving protocols to perform the test of these two relations on sets.

Bloom Filters. A Bloom filter is a data structure introduced by Burton Howard Bloom in 1970 [2]. It is used to represent a set of elements. With a Bloom filter representing a certain set, one can verify whether an element is a member of this set. Such a data structure consists of a tabular of m bits which is associated to n_{key} public hash functions. At first, all the m bits are initialized to 0. Moreover two functions namely *add()* and *test()* are available. To add an element to the Bloom filter, one has to compute the hashes of this element with each of the respective n_{key} hash functions. Then, set the bit to 1 for each position corresponding to a hash value. To test whether one element is included in the Bloom filter with the *test()* function, one has, by the same manner, to compute the respective hash values of this element and verify if the respective bits are set to

1. If at least one of these bits is set to 0, then with certainty the tested element is not a member of the set represented by the Bloom filter (i.e. no false negative for *test()* could append). On the contrary, with some probability, the *test()* function could retrieve a false positive. Indeed, even if all the bits that have been verified are set to 1, the tested element may not be part of the set represented by the Bloom filter. There exist multiple applications for this approach. It is used by web browser or web site to optimize the cache or the site recommendations. With database query operations it could also for example reduce the disk lookups for non-existent rows or columns. Finally we could also mention its relevancy with respect to cryptocurrencies by for instance accelerating the wallet synchronization of Bitcoin or fasten the finding of logs on the Ethereum blockchain.

To express the probability of having a false positive when performing function *test()* we introduce the notion of an *overlapping bit*. It corresponds to the case where a certain bit of the Bloom filter has to be set to 1 due to the adding of an element but this specific bit is already set to 1. The probability of having an overlapping bit is null when the Bloom filter is still blank, and it grows along with the number of inserted elements. We express this probability as following with $X_{BF_{\mathcal{A}}}$ the amount of bits already set to 1 in $BF_{\mathcal{A}}$ at a specific point in time:

$$\mathcal{P}_{ob} = \frac{X_{BF_{\mathcal{A}}}}{m} \tag{1}$$

We could then express the average amount of different bits added to the Bloom filter when adding one new element to it:

$$X_{add} = n_{key} + \sum_{i=1}^{n_{key}} \left((-1)^i \cdot \frac{\sum_{j=i}^{n_{key}-1} \binom{j}{i}}{m^i} \right) \tag{2}$$

And we could generalize it to the average amount of bits added to the Bloom filter when adding N new elements to it:

$$X_{add}(N) = n_{key} \cdot N + \sum_{i=1}^{n_{key} \cdot N} \left((-1)^i \cdot \frac{\sum_{j=i}^{(n_{key} \cdot N)-1} \binom{j}{i}}{m^i} \right) \tag{3}$$

By observing the current state of a Bloom filter representing finite set $\mathcal{A}$ of $n_{\mathcal{A}}$ elements, one can express the exact amount of overlapping bits as the value $Y_{BF_{\mathcal{A}}}$:

$$Y_{BF_{\mathcal{A}}} = (n_{\mathcal{A}} \cdot n_{key}) - X_{BF_{\mathcal{A}}} \tag{4}$$

3 Related Work

Adding privacy to Bloom filters has been yet investigated in many works. The first way of doing that consists of directly encrypting the Bloom filter with homomorphic encryption as Kerschbaum did in [10] and [9] and how it has been developed in [1]. In [15] the authors add privacy by using two approaches: blind

signature schemes and oblivious pseudorandom functions. The second type of approaches, as we suggest in the current work, consists of replacing the traditional hash functions of the Bloom filter by HMAC functions. We explain in what way these existing solutions [6, 8, 12, 16, 18, 19] do not fit our requirements and therefore our solution brings novelty. First of all, we highlight the fact that none of the following solutions provide privacy on the sets cardinality. In [6] Goh associates Bloom filters with a keyed pseudo-random function to allow a private member testing in the Bloom filter. This is in particular one aspect we do not want the auditor be able to do. In [8] the authors propose a solution that approximately computes the dot product protocol using Bloom filters. Contrary to our outsourced requirement, this solution is a two-parties interactive protocol. They implemented their solution and show results for relatively small parameters (less than 5 keys and no Bloom filters larger than 5000 bits). They conclude that their solution roughly supports the use of more than four hash functions. In [12] the authors expose a construction of Bloom filters along with HMAC protocol in a wireless sensor aggregation scenario. Their approach is somehow similar but the base station (equivalent to the auditor here) shares HMAC keys directly with each of the nodes. Therefore, the merging of Bloom filters from different nodes does not allow any operation since different keys are used. In [16], still by combining the Bloom filter approach with a keyed hash function, the authors propose a solution to compute the membership of elements in a set. Therefore, they manipulate Bloom filters of unique elements that leads to data leakage regarding the amount of elements and could be very costly, especially when considering thousands of them. In [18], authors compute sets intersections but in such protocols, the parties (here the data sources) which perform the intersection need the secret key to return the result. Such a construction does not fit our requirements where in particular the active party (the auditor) should not hold the key for privacy reasons. In [19] the authors present some experimental results produced from relatively small Bloom filters parameters that shows limitations on having a large amount of bits in Bloom filters to keep the set sizes protected. They only motivate the use of HMAC by arguing that it adds an additional security layer to their protocol. We notice that, works that provide results by testing their solution are space limited. Other works [3, 5, 11], that deserve attention, compute set relations and operations using the Bloom filters approach but in a multiparty model that does not fit our requirements. In Table 1 we recap what the aforementioned approaches provide compared to our proposed solution. For each approach we precise if it can be used in an outsourced model where a malicious third party performs the computations, if functions *add()*, *test()* and the set relations *inclusiveness* (INC) and *disjointness* (DIS) are computable by the third party.

4 A Cloud Security Auditing Use Case

Next, we present a use case of a cloud security audit on access control based on [21]. In addition to the trendy increase of auditing in cloud environment, we

Table 1. Comparison of the approaches.

Approach	Outs.	*add()*	*test()*	INC/DIS	Card. privacy
Encrypted BF [1, 9, 10]	yes	yes	yes	yes (obfuscated results)	no
Blind Signature [15]	no	-	-	no	no
BF with HMAC [6]	yes	no	yes	no	no
- [8]	no	-	-	no	no
- [12]	yes	yes	yes	no	no
- [16]	yes	no	yes	yes	no
- [18]	yes	yes	yes	no	no
- [19]	yes	no	no	no	no
PSI [3, 5, 11]	no	-	-	no	yes
Our approach	yes	no	no	yes	yes

highlight the relevancy of this use case in terms of managing the data at several sides, namely clients, $\mathcal{CSP}$ and auditor along with its privacy requirements. Such a third party involvement should be classified as *malicious* with respect to a solid threat modeling as described below:

Scenario. This use case consists of users trying to connect and access a service provided by a cloud service provider ($\mathcal{CSP}$). To be authorized, one user should be an employee from the specific company that required the service and in addition, should be connecting to the service from an authorized IP address. An auditor will have to verify that the $\mathcal{CSP}$ correctly performed the access control on the company's respective service by testing if the IP addresses of the successful users are part of the authorized IP addresses set. On the contrary, the auditor will also have to verify that no authorized user has been denied to connect.

With respect to the given privacy requirements, the IP addresses from the successful and non-successful sequences as well as from the whitelist have to remain hidden from the auditor. As we have seen in Section 2, by using Bloom filters to represent any group of elements we lose the aspects of multiplicity and order of this group. Therefore, in the remainder of this work we will only use the notion of set. We define sets $\mathcal{W}$, $\mathcal{L}_1$ and $\mathcal{L}_2$ we are considering in the scenario:

- $\mathcal{W} = \{w_1, \ldots, w_{n_\mathcal{W}}\}$ which corresponds to the *whitelist*, the set of the authorized IP addresses.
- $\mathcal{L}_1 = \{l_1, \ldots, l_{n_{\mathcal{L}_1}}\}$ which corresponds to the logfile of the IP addresses that successfully connected.
- $\mathcal{L}_2 = \{l'_1, \ldots, l'_{n_{\mathcal{L}_2}}\}$ which corresponds to the logfile of the IP addresses that failed to connect.

As presented in Figure 1, we consider three parties; a cloud service provider $\mathcal{CSP}$, a client (company) $\mathcal{C}$ and an auditor $\mathcal{AD}$. $\mathcal{W}$ represents the set of the authorized IP addresses respectively to a specific client. Therefore, this set is generated and protected by the client himself. Its content is originally sent from the $\mathcal{C}$ to the $\mathcal{CSP}$ in a non-protected version such that the $\mathcal{CSP}$ could perform the access control. Nevertheless, it should remain hidden from the auditor $\mathcal{AD}$ during the

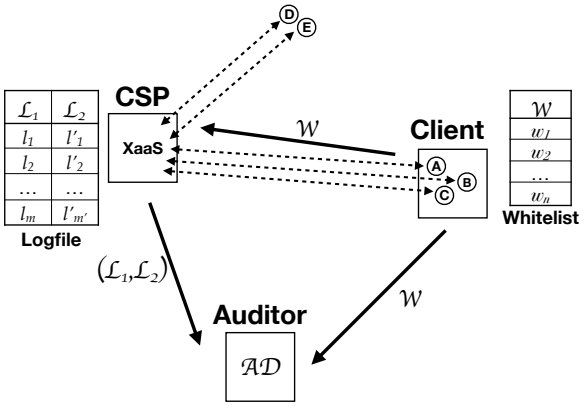

Fig. 1. Generic framework of the cloud security auditing. Users A, B and C should be authorized to connect by the $\mathcal{CSP}$ while users D and E should not.

whole auditing process and thus being protected. $\mathcal{L}_1$ and $\mathcal{L}_2$ represent the sets of the IP addresses for any connection attempts during the epoch $[t_a; t_b]$, for which the auditing is required. They respectively represent the set of the IP addresses for all the successful connections and non-successful connections. Both contain sensitive information therefore, their contents should also remain hidden from the $\mathcal{AD}$ during the whole process.

Set Relations. To verify that the $\mathcal{CSP}$ performed correctly the access control, the $\mathcal{AD}$ has to perform two types of set relations. First of all, to verify that only users with authorized IP addresses succeeded to connect, the $\mathcal{AD}$ could perform an *inclusiveness* relation between sets $\mathcal{L}_1$ and $\mathcal{W}$, i.e. to test if all the elements from $\mathcal{L}_1$ are included in $\mathcal{W}$. Secondly, to verify that the $\mathcal{CSP}$ has rejected only users connecting from non-authorized IP addresses, the $\mathcal{AD}$ could perform a *disjointness* relation between $\mathcal{W}$ and $\mathcal{L}_2$.

Security Model. These set relations could be considered as hard-to-solve if we want to achieve a considerable privacy level at the same time. Taking privacy into account means guaranteeing privacy on the elements of the sets regarding any external party and, in particular, the party which is performing the protocols of *inclusiveness* and *disjointness*. In such scenario, we consider on the one hand that the $\mathcal{C}$ and the $\mathcal{CSP}$ behave as *honest-but-curious* and on the other hand the $\mathcal{AD}$ may act as a *malicious* party. Indeed, the $\mathcal{CSP}$ already knows the whitelist content while the $\mathcal{C}$ will not have access to the logfiles during the auditing protocol since we are not in a multi-party computation model. The auditor $\mathcal{AD}$ will have access to an obfuscated version of the sets and we thus consider the case where it could try to retrieve information about their content or cardinality.

We highlight three security requirements that should be considered:

SR1. The $\mathcal{AD}$ should not be able to modify any part of the Bloom filters $BF_{\mathcal{W}}$, $BF_{\mathcal{L}_1}$ or $BF_{\mathcal{L}_2}$ in an undetectable manner and should not be able to generate its own Bloom filter and use it to perform the set relations protocols. I.e., performing the protocol with a Bloom filter from a client along with one generated by the third party should result in a dummy outcome.

SR2. The $\mathcal{AD}$ should not be able to learn from $BF_{\mathcal{W}}$, $BF_{\mathcal{L}_1}$ or $BF_{\mathcal{L}_2}$ if:
1) $w_i = l_j$ for some i, j 2) $l_i = l_j$ for some $i \neq j$
In other words, the $\mathcal{AD}$ should not be able to learn in how many occurrences an element is present in a set and if a specific element from $\mathcal{W}$ is also in $\mathcal{L}_1$ or $\mathcal{L}_2$. We require a certain level of obfuscation over the computation on the sets. Solely the results of the set relations should be revealed to the $\mathcal{AD}$.

SR3. The $\mathcal{AD}$ should not learn the sets cardinalities $n_{\mathcal{W}}$, $n_{\mathcal{L}_1}$ and $n_{\mathcal{L}_2}$ from $BF_{\mathcal{W}}$, $BF_{\mathcal{L}_1}$ and $BF_{\mathcal{L}_2}$ or any results from the protocols.

SR1 and **SR2** are considered as mandatory while **SR3** could be considered as rather optional. Indeed, one may argue that even if the auditor knows the sets cardinalities, he could still not guess which concrete IP addresses are in the sets. But one could easily imagine that firstly, in some use cases knowing the set cardinality is by itself a leak of privacy and secondly, there definitively do exist cases where for example, knowing that a set contains few elements will lead the auditor to infer which ones. Therefore, we will show that our solutions fulfill the third privacy requirement to a certain level λ_n which corresponds to the amount of possible size candidates of a set that the auditor will get after performing our protocols.

5 Protocols

For simplicity of presentation, we present our protocols with only one client involved. We remark that one can easily adapt it to a use case with multiple clients using the same cloud provider. We also emphasize the fact that, to process the two set relations, the considered Bloom filters should be similarly generated, namely with the same size m, keyed hash function and set of keys K. First we recall the two privacy enhancements from tuning the classical use of Bloom filter. Then we present the *inclusiveness* and *disjointness* set relations before explaining how the parameters should be selected to guarantee a certain level of correctness on these two relations.

Privacy Enhancements. Using Bloom filters for privacy-sensitive scenarios is not as common as its usage for the classical applications. Our approach to make such a technique fitting privacy-sensitive use cases, is based on the use of a public keyed collision-resistant hash function (e.g. MAC) with a set of n_{key} private keys instead of the n_{key} public hash functions. W.l.o.g. we use an HMAC function to solve the current use case. That being said, any party that does not hold the keys cannot use the *test()* function to directly verify if a specific element is included in the Bloom filter. The other security benefit when using an HMAC function is

that even if the function is publicly released, any party that does not hold the keys cannot add additional elements. More formally, we define a Bloom filter of size m of a set $\mathcal{A} = \{a_1, \ldots, a_{n_\mathcal{A}}\}$, with a set of n_{key} keys $K = \{k_1, \ldots, k_{n_{key}}\}$ and an HMAC function $h_k : \{0,1\}^* \rightarrow \{1, \ldots, m\}$ with $k \in K$ as:

$$BF\big(\mathcal{A}, (h_k)_{k\in K}\big) = bf_\mathcal{A}[j]_{1\leqslant j\leqslant m} \tag{5}$$

$$where \;\; bf_\mathcal{A}[j] = 1 \; if \; \exists \; (i,\kappa) \; s.t. \; h_{k_\kappa}(a_i) = j$$

$$bf_\mathcal{A}[j] = 0 \; otherwise$$

In the remaining parts of this work, we use the simplified notation $BF_\mathcal{A}$ to represent the Bloom filter of set $\mathcal{A}$. The second privacy enhancement we add to the use of Bloom filters corresponds to keep parameter n_{key} private to avoid revealing the sets' cardinalites. Having parameter n_{key} publicly released requires too many overlapping bits in the Bloom filter. Indeed, the naive technique to retrieve the cardinality of the set by looking at its respective Bloom filter would be to divide its amount of bits set to one by parameter n_{key}. There exists an optimized technique introduced by Swamidass and Baldi [20] which computes $n^*_\mathcal{A}$ an approximation of the number of distinct elements inserted in $BF_\mathcal{A}$ with $X_{BF_\mathcal{A}}$ the amount of bits set to 1 in the Bloom filter:

$$n^*_\mathcal{A} = -\frac{m}{n_{key}} ln\Big[1 - \frac{X_{BF_\mathcal{A}}}{m}\Big] \tag{6}$$

Such a technique requires even more overlapping bits to mislead the attacker. We see that by making parameter n_{key} private, one could not be able to compute $n^*_\mathcal{A}$ anymore. We give examples in Table 2 for several parameter configurations, we show the estimations of the cardinality of the sets with the naive technique and the one from Swamidass and Baldi (S&B) when we simulate the generation of 10^3 Bloom filters $BF_\mathcal{A}$.

By referring to the results in Table 2 obtained with the S&B technique, we could agree on the necessity of having n_{key} private to save privacy on the set cardinality. However, in Section 6 we show how an attacker could still gain information on the set cardinality even with such a precaution. One may argue the complexity of keeping the size of K private or the effort to store a large amount of keys. We could then slightly modify the protocol to have a unique key k. Indeed, the outcome of $h_k(x)$ will be divided in n_{key} equal size fragments and each indicates an index of the Bloom filter to increment. As it is suggested in [12], in case of fragment too small to contain information of size m, we may perform multiple rounds of HMAC operations with different salts. Of course, in such protocol modification, parameter n_{key} remains secret to bring privacy over the sets size.

Initialization.

$\mathbf{h}, \mathbf{n_{key}}, \mathbf{m}, \mathbf{K} \leftarrow$**Setup:** The client $\mathcal{C}$ should first choose and generate the Bloom filter parameters: the dimension m, the HMAC function h, the amount of

Table 2. Set cardinality estimations with the naive and Swamidass and Baldi (S&B) technique for different parameters configurations.

$n_{\mathcal{A}}$	m	n_{key}	$X_{BF_{\mathcal{A}}}$	$n^*_{\mathcal{A}}$ with naive	$n^*_{\mathcal{A}}$ with S&B
100	$1.44 \cdot 10^3$	10	$[682 - 758]$	[68.2 - 75.8]	[92.4 - 107.6]
100	$6.5 \cdot 10^7$	1700	$[169824 - 171718]$	[100.0- 101.1]	[99.9 - 100.0]
1000	$2 \cdot 10^9$	950	$[949730 - 949822]$	[999.7 - 999.8]	[999.9 - 1000.1]

keys n_{key} and the set of keys $K = \{\kappa_1, \ldots, \kappa_{n_{key}}\}$. $\mathcal{C}$ generates parameters by performing the following protocol:

- randomly choose $n_{key} \in [n^L_{key}; n^U_{key}]$ with n_{key}, n^L_{key} and n^U_{key} integers.
- set m such that $X_{\cap=\emptyset} < n^L_{key}$.

Values n^L_{key} and n^U_{key} are public and we determine them later considering correctness and privacy in Sections 5 and 6. The restriction on parameter m corresponds to a correctness consideration which we explain in more details in Section 5. $\mathcal{C}$ then selects the public HMAC function h, generates its n_{key} respective keys and privately shares parameters $\{h, n_{key}, m, K\}$ with $\mathcal{CSP}$.

$\mathbf{BF}_{\mathcal{W}}, \mathbf{BF}_{\mathcal{L}_1}, \mathbf{BF}_{\mathcal{L}_2} \leftarrow$Create($\mathcal{W}, \mathcal{L}_1, \mathcal{L}_2$): $\mathcal{C}$ (resp. $\mathcal{CSP}$) generates the Bloom filter of its data $\mathcal{W} = \{w_1, \ldots, w_{n_{\mathcal{W}}}\}$ (resp. $\mathcal{L}_1 = \{l_1, \ldots, l_{n_{\mathcal{L}_1}}\}$ and $\mathcal{L}_2 = \{l'_1, \ldots, l'_{n_{\mathcal{L}_2}}\}$).

$$
\begin{aligned}
BF_{\mathcal{W}} &= BF\big(\mathcal{W}, (h_\kappa)_{\kappa \in K}\big) = bf_{\mathcal{W}}[j]_{1 \leqslant j \leqslant m} \\
BF_{\mathcal{L}_1} &= BF\big(\mathcal{L}_1, (h_\kappa)_{\kappa \in K}\big) = bf_{\mathcal{L}_1}[j]_{1 \leqslant j \leqslant m} \\
BF_{\mathcal{L}_2} &= BF\big(\mathcal{L}_2, (h_\kappa)_{\kappa \in K}\big) = bf_{\mathcal{L}_2}[j]_{1 \leqslant j \leqslant m}
\end{aligned}
$$

Inclusiveness Protocol. This operator allows to verify if one set is included in another. It performs directly on the Bloom filters of the respective sets. We define this operator as $\mathbf{BF}_{\mathcal{A} \subseteq \mathcal{B}} \leftarrow \mathbf{INC}(\mathbf{BF}_{\mathcal{A}}, \mathbf{BF}_{\mathcal{B}})$:

$$
\begin{aligned}
&bf_{\mathcal{A} \subseteq \mathcal{B}}[j]_{1 \leqslant j \leqslant m} \leftarrow INC(BF_{\mathcal{A}}, BF_{\mathcal{B}}) \\
&where \;\; 0 \leftarrow bf_{\mathcal{A} \subseteq \mathcal{B}}[j] \; if \; (bf_{\mathcal{A}}[j] = 1 \;\wedge\; bf_{\mathcal{B}}[j] = 0) \\
&1 \leftarrow bf_{\mathcal{A} \subseteq \mathcal{B}}[j] \; otherwise.
\end{aligned}
\tag{7}
$$

We remark that this operator is equivalent to the bitwise binary operator combination:

$$
INC(BF_{\mathcal{A}}, BF_{\mathcal{B}}) \equiv \neg(BF_{\mathcal{A}}) \; OR \;\; BF_{\mathcal{B}} \tag{8}
$$

$\mathcal{AD}$ firstly computes the inclusion protocol on the two respective Bloom filters of sets $\mathcal{L}_1$ and $\mathcal{W}$ to test if $\mathcal{L}_1 \subseteq \mathcal{W}$, namely if all the authorized connections have been made from authorized IP addresses:

$$
INC(BF_{\mathcal{L}_1}, BF_{\mathcal{W}}) = BF_{\mathcal{L}_1 \subseteq \mathcal{W}} = bf_{\mathcal{L}_1 \subseteq \mathcal{W}}[j]_{1 \leqslant j \leqslant m}
$$

Then $\mathcal{AD}$ expresses $X_{\mathcal{L}_1 \subseteq \mathcal{W}}$ which corresponds to the number of bits set to 1 in the resulting Bloom filter:

$$X_{\mathcal{L}_1 \subseteq \mathcal{W}} = \sum_{j=1}^{m} bf_{\mathcal{L}_1 \subseteq \mathcal{W}}[j] \tag{9}$$

$\mathcal{AD}$ tests if $X_{\mathcal{L}_1 \subseteq \mathcal{W}} = m$ and can conclude that $\mathcal{L}_1 \subseteq \mathcal{W}$ if no false positive occurred. Otherwise we have $\mathcal{L}_1 \nsubseteq \mathcal{W}$ with certainty. In Appendix B, we provide a toy example of this protocol.

Disjointness Protocol. This set relation allows to verify that no elements from one set are included in another set. In other words, this allows to claim that two sets are disjoint. This test function is not trivial, indeed, if we use Bloom filters it is not sufficient to highlight the cases where a bit 1 has been inserted at the same index for the two respective Bloom filters. We define this operator as $\mathbf{BF}_{\mathcal{A} \cap \mathcal{B} = \emptyset} \leftarrow \mathbf{DIS}(\mathbf{BF}_{\mathcal{A}}, \mathbf{BF}_{\mathcal{B}})$:

$$\begin{aligned} &bf_{\mathcal{A} \cap \mathcal{B} = \emptyset}[j]_{1 \leqslant j \leqslant m} \leftarrow DIS(BF_{\mathcal{A}}, BF_{\mathcal{B}}) \\ &where \;\; 1 \leftarrow bf_{\mathcal{A} \cap \mathcal{B} = \emptyset}[j] \; if \; (bf_{\mathcal{A}}[j] = 1 \;\wedge\; bf_{\mathcal{B}}[j] = 1) \\ &0 \leftarrow bf_{\mathcal{A} \cap \mathcal{B} = \emptyset}[j] \; otherwise. \end{aligned} \tag{10}$$

We remark that this operator is equivalent to the bitwise logical-and operator:

$$DIS(BF_{\mathcal{A}}, BF_{\mathcal{B}}) \equiv BF_{\mathcal{A}} \; AND \; BF_{\mathcal{B}}. \tag{11}$$

To verify that no authorized user failed to connect to the service offered by $\mathcal{CSP}$, $\mathcal{AD}$ performs the disjointness relation on the respective Bloom filters of $\mathcal{W}$ and $\mathcal{L}_2$:

$$DIS(BF_{\mathcal{W}}, BF_{\mathcal{L}_2}) = BF_{\mathcal{W} \cap \mathcal{L}_2 = \emptyset} = \; bf_{\mathcal{W} \cap \mathcal{L}_2 = \emptyset}[j]_{1 \leqslant j \leqslant m}$$

Then $\mathcal{AD}$ expresses $X_{\mathcal{W} \cap \mathcal{L}_2 = \emptyset}$ which corresponds to the number of bits set to 1 in the resulting Bloom filter:

$$X_{\mathcal{W} \cap \mathcal{L}_2 = \emptyset} = \sum_{j=1}^{m} bf_{\mathcal{W} \cap \mathcal{L}_2 = \emptyset}[j] \tag{12}$$

$\mathcal{AD}$ compares it such that:

- if $X_{\mathcal{W} \cap \mathcal{L}_2 = \emptyset} < n_{key}^L$ then $\mathcal{W}$ and $\mathcal{L}_2$ are distinct
- if $X_{\mathcal{W} \cap \mathcal{L}_2 = \emptyset} \geqslant n_{key}^L$ then $\mathcal{W}$ and $\mathcal{L}_2$ have at least one element in common

Indeed for each element which is included in both sets, we get n_{key} times a bit set to 1 in the resulting Bloom filter. However we could still get such a bit set to 1 due to a bit set to 1 in $BF_{\mathcal{W}}$ and $BF_{\mathcal{L}_2}$ stemming from different elements originally added to the Bloom filters. We call such a case a false positive for the disjointness relation. We will discuss its probability of occurrence in the following sections. In Appendix C, we give a toy example of this protocol.

Correctness of the Set Relations. In this section we consider the correctness of our two proposed relations. We recall that the Bloom filter approach allows false positives but no false negative on the *test()* function. Nevertheless, in this work we are not considering the *test()* function and we thus focus on the overlapping bits of the Bloom filters resulting from our set relations.

Correctness of the Inclusiveness Relation. For the *inclusiveness* relation, we notice that only false positive could append and not false negative. Indeed, after performing $INC(BF_{\mathcal{L}_1}, BF_{\mathcal{W}})$, if there is an index j with $bf_{\mathcal{L}_1 \subseteq \mathcal{W}}[j] = 0$, we have $bf_{\mathcal{L}_1}[j] = 1$ and $bf_{\mathcal{W}}[j] = 0$, then with certainty, at least one element from $\mathcal{L}_1$ does not belong to $\mathcal{W}$. Concretely, if the outcome of the auditing process states that $\mathcal{L}_1 \nsubseteq \mathcal{W}$ then we have a probability of correctness of 1. On the other hand, if we get $\mathcal{L}_1 \subseteq \mathcal{W}$ as result, this outcome is not necessarily correct and we get a probability of correctness equals to $1 - \mathcal{P}_{FP}$ the probability of having a false positive. $\mathcal{P}_{FP}$ could be expressed in terms of parameters n_{key}, m and $n_{\mathcal{W}}$ denoting the amount of elements inserted in $BF_{\mathcal{W}}$. The probability that our *inclusiveness* relation outcomes a false positive whereas one element l_1 from $\mathcal{L}_1$ is not in $\mathcal{W}$ is equivalent to the one to have $test(\mathcal{W}, l_1)$ resulting true with the same parameters. We detail the value of $\mathcal{P}_{FP}$:

First, we denote the probability that after inserting $n_{\mathcal{A}}$ elements, a certain bit is equal to 1 is:

$$1 - (1 - \frac{1}{m})^{n_{key} \cdot n_{\mathcal{A}}} \tag{13}$$

If we consider that $Z_{\mathcal{L}_1,\mathcal{W}}$ elements from $\mathcal{L}_1$ are not included in $\mathcal{W}$, the probability of having a false positive after computing the *inclusiveness* relation is:

$$\mathcal{P}_{FP} \geqslant \left(1 - (1 - \frac{1}{m})^{n_{key} \cdot n_{\mathcal{A}}}\right)^{n_{key} \cdot Z_{\mathcal{L}_1,\mathcal{W}}} \tag{14}$$

$$\mathcal{P}_{FP} \simeq \left(1 - (1 - \frac{1}{m})^{X_{add}(n_{\mathcal{A}})}\right)^{X_{add}(Z_{\mathcal{L}_1,\mathcal{W}})} \tag{15}$$

In Appendix D, we give a toy example of a case of a false positive happening during the *inclusiveness* relation protocol. In Appendix E we give a validation of (14) by comparing its results with testing our implementation of the Bloom filter construction.

Correctness of the Disjointness Relation. For the *disjointness* relation, we have on the contrary no case of false positive but a case of false negative may append. Indeed, if we get $X_{\mathcal{W} \cap \mathcal{L}_2 = \emptyset} < n_{key}^L$ then it means that $BF_{\mathcal{W}}$ and $BF_{\mathcal{L}_2}$ have less than n_{key}^L (and thus less than n_{key}) indexes i where $bf_{\mathcal{W}}[i] = 1$ and $bf_{\mathcal{L}_2}[i] = 1$. It is then not possible that $\mathcal{W}$ and $\mathcal{L}_2$ have common elements. Regarding the false negative scenario, it could append if we get too many *overlapping bits* in the resulting Bloom filter. It consists of a case where $\mathcal{W}$ and $\mathcal{L}_2$ have no element in common but $\mathcal{AD}$ gets $X_{\mathcal{W} \cap \mathcal{L}_2 = \emptyset} \geqslant n_{key}^L$. To avoid such a case, we have to accurately tune the parameters such that in a case of distinct sets $\mathcal{W}$ and $\mathcal{L}_2$, the respective value $X_{\mathcal{W} \cap \mathcal{L}_2 = \emptyset}$ will never (with acceptable probability) be greater

than n^L_{key}. To do so, $\mathcal{C}$ has to carefully select the parameters n_{key} and m such that $X_{\cap=\emptyset} + \epsilon < n^L_{key}$. Value $X_{\cap=\emptyset}$ represents the expected value of $X_{\mathcal{W}\cap\mathcal{L}_2=\emptyset}$ when performing the *disjointness* protocol on two distinct sets $\mathcal{W}$ and $\mathcal{L}_2$. Value ϵ represents a default value that guarantees that $X_{\mathcal{W}\cap\mathcal{L}_2=\emptyset}$ is, with an acceptable probability, never greater than n^L_{key} when $\mathcal{W}$ and $\mathcal{L}_2$ are distinct. To express value $X_{\cap=\emptyset}$, we first give the probability of having a bit set to 1 for any index j in both Bloom filters $BF_{\mathcal{W}}$ and $BF_{\mathcal{L}_2}$, knowing that $\mathcal{W}$ and $\mathcal{L}_2$ are distinct:

$$\begin{aligned} p(bf_{\mathcal{W}}[j] = 1 \ \wedge \ bf_{\mathcal{L}_2}[j] = 1) &= p(bf_{\mathcal{W}}[j] = 1) \ \cdot \ p(bf_{\mathcal{L}_2}[j] = 1) \\ &= (1 - (1 - \frac{1}{m})^{n_{key} \cdot n_{\mathcal{W}}}) \ \cdot \ (1 - (1 - \frac{1}{m})^{n_{key} \cdot n_{\mathcal{L}_2}}) \end{aligned} \tag{16}$$

Finally, the expected amount of bits set to 1 in both $BF_{\mathcal{W}}$ and $BF_{\mathcal{L}_2}$ at the same index resulting from distinct set elements is:

$$X_{\cap=\emptyset} = m \ \cdot \ (1 - (1 - \frac{1}{m})^{X_{add}(n_{\mathcal{W}})}) \ \cdot \ (1 - (1 - \frac{1}{m})^{X_{add}(n_{\mathcal{L}_2})}) \tag{17}$$

When we have $Z'_{\mathcal{W},\mathcal{L}_2}$ common elements inserted in both Bloom filters, we get $X_{\mathcal{W}\cap\mathcal{L}_2=\emptyset} \simeq Z'_{\mathcal{W},\mathcal{L}_2} \cdot n_{key} + X_{\cap=\emptyset}$. Therefore, if $\mathcal{C}$ takes care that $X_{\cap=\emptyset}$ never gets greater or equal to n^L_{key}, then $\mathcal{AD}$ could notice when the two sets have common elements even in the case of $Z'_{\mathcal{W},\mathcal{L}_2} = 1$. In Appendix F we verify the correctness of (17).

Choosing Parameters Regarding Correctness. In the classical use of Bloom filters as presented in [2], some usage recommendations are made to generate parameters n_{key} and m:

$$m = -\frac{n_{\mathcal{A}} \cdot ln \ (\mathcal{P}_{FP})}{(ln \ 2)^2} \tag{18}$$

$$n_{key} = \frac{m}{n_{\mathcal{A}}} \cdot ln \ 2 \tag{19}$$

We recall that initially Bloom filters are not supposed to hold such relations testing as *inclusiveness* or *disjointness*. Therefore, the considerations on the generation of n_{key} and m are manifold. In Table 3 we provide some examples of correct parameters m and n_{key} such that the false positive probability $\mathcal{P}_{FP}$ of *inclusiveness* relation is acceptable and the expected amount $X_{\cap=\emptyset}$ is significantly smaller than, by instance, $n^L_{key} = 5$. We notice that in Table 3, parameters n_{key} and m are significantly smaller than the ones considered later but we recall that we care here solely about correctness of the set relations and not about privacy of the set cardinality.

6 Privacy Analysis

In this section we show how our solutions fulfill privacy in terms of content and cardinality. Since the *overlapping bits* distribution, as well as the privacy of the Bloom filter's content and cardinality are similar, no matter if they represent a whitelist or a logfile, we consider in this section generic sets $\mathcal{A}$ and $\mathcal{B}$.

Table 3. Appropriate parameters m and n_{key} considering $\mathcal{P}_{FP}$ and $X_{\cap=\emptyset}$ with fixed $n_{\mathcal{A}}$ and $n_{\mathcal{B}}$.

$n_{\mathcal{A}}$	$n_{\mathcal{B}}$	m	n_{key}	$\mathcal{P}_{FP}$	$X_{\cap=\emptyset}$
10^2	$5 \cdot 10^1$	10^6	10	$9.9 \cdot 10^{-31}$	0.50
10^2	10^3	10^7	10	$9.9 \cdot 10^{-41}$	0.99
10^3	10^3	$7.62 \cdot 10^9$	1861	~ 0	$4.54 \cdot 10^2$
10^4	$2 \cdot 10^2$	$9.47 \cdot 10^9$	1468	~ 0	$4.55 \cdot 10^2$

Distribution of the Overlapping Bits. In this section we analyze the characteristics of *overlapping bits* occurring throughout the basic step of Bloom filters generation. As it is illustrated in Appendix G, we obtain such a distribution by running the generation of 10^3 Bloom filters for each parameters configuration. From these distributions we could notice several characteristics. First, the more elements we add to the Bloom filter, the larger is the *overlapping bits* range. By instance, if we follow recommendations from (18) and (19), and we insert only 10 elements, we get a range of overlapping bits to approximately 10. When we have 100 inserted elements the range increases to approximately 40. Since our protocols use an HMAC function which generates a uniform random distribution, we could consider that the overlapping bits follow a normal distribution. If it is the case, to consider 99.7% of the possible overlapping bits values for a Bloom filter representing a specific set, one could define them in the range $[\mu - 3\sigma; \mu + 3\sigma]$ with μ the mean and σ the standard deviation. Setting the parameters in the objective to tune the distribution to get an acceptable overlapping bits range regarding the aiming level of privacy could be intended. In Table 4 we see how the standard deviation of the overlapping bits distribution varies depending on the parameters and the amount of inserted elements. As a second character-

Table 4. The average and standard deviation of the overlapping bits distribution in Bloom filters $BF_{\mathcal{A}}$ and $BF_{\mathcal{B}}$ for different parameters configurations.

$n_{\mathcal{A}}$	$n_{\mathcal{B}}$	m	n_{key}	$\mu_{\mathcal{A}}$	$\sigma_{\mathcal{A}}$	$\mu_{\mathcal{B}}$	$\sigma_{\mathcal{B}}$
10	100	$1.44 \cdot 10^3$	10	3.3	1.73	278.6	10.3
100	100	$2.2 \cdot 10^7$	1700	653.9	24.6	654.9	26.1
10	100	$2 \cdot 10^6$	1700	72.3	8.4	7023.4	77.9
100	1000	$6 \cdot 10^7$	500	20.8	4.44	2075.8	45.3

istic, we observe that when we have two sets with highly distant cardinalities $n_{\mathcal{A}} \ll n_{\mathcal{B}}$ (or resp. $n_{\mathcal{B}} \ll n_{\mathcal{A}}$), the number of overlapping bits in the Bloom filter of the smaller set $Y_{BF_{\mathcal{A}}}$ (resp. $Y_{BF_{\mathcal{B}}}$) substantially decreases and the one of the larger set substantially increases. Having too few overlapping bits in a Bloom filter could be problematic, especially if it could even be predictable by the attacker. By running tests we notice that no matter which n_{key} is picked or how many elements are inserted in the Bloom filters, if the ratio $\frac{n_{\mathcal{A}}}{n_{\mathcal{B}}}$ remains the

same, then firstly the expected amounts of overlapping bits in $BF_{\mathcal{A}}$ and $BF_{\mathcal{B}}$ remain approximately the same and secondly $Y_{BF_{\mathcal{A}}}$ (resp. $Y_{BF_{\mathcal{B}}}$) is quite small. Moreover, we see that it is even worse if we keep decreasing the ratio $\frac{n_{\mathcal{A}}}{n_{\mathcal{B}}}$. One solution to keep having an acceptable range of overlapping bits in the Bloom filter representations of the smaller set, even if we have a significant difference in the cardinalities, could be to use a greater domain $[n_{key}^{L}; n_{key}^{U}]$. Indeed, for the same ratio $\frac{n_{\mathcal{A}}}{n_{\mathcal{B}}}$, we get greater *overlapping bits* ranges.

Privacy on the Content. First, we claim that no attacker could determine which concrete IP address is included in the whitelist neither the log files. This holds by means of the Bloom filter construction. Indeed, each element from the sets are mapped with the HMAC function constructed from a cryptographic hash function and therefore benefits from its on-wayness characteristic. This means that the only straightforward manner to get any knowledge on the Bloom filter content would be to use the *test()* function which is only computable by $\mathcal{CSP}$ and $\mathcal{C}$. From this statement *SR1* and *SR2* are fulfilled:

SR1. If $\mathcal{AD}$ does not know the HMAC's keys $K = \{k_1, \ldots, k_{n_{key}}\}$, it cannot generate its own Bloom filter or add any element to an existing one and perform the set relations. Indeed, they require that all the considered Bloom filters are generated with the same keys.

SR2. These two sub-requirements are fulfilled thanks to the Bloom filter inherent characteristics. Indeed, the first one holds because of the fact that all elements inserted in a Bloom filter are mixed together and it is not possible, even from the same Bloom filter, to distinguish between them. The second sub-requirement holds since when the whitelist Bloom filter is created, even if an element occurs multiple times in the logfile, it will be added to the respective Bloom filter only once.

Privacy on the Cardinality. In this section, we focus on the ability of any attacker to retrieve the cardinality of the sets from one or multiple versions of the Bloom filter's representation of the set. We show that our solution fulfills the third security requirement:

SR3. The overlapping bits property of the Bloom filters allows to hide the exact number of elements in the whitelist or the logfile. However, $\mathcal{AD}$ is able to determine the amount of bits set to 1 in the Bloom filters. It could then deduct the following information: $n_{\mathcal{A}} \geqslant \frac{X_{BF_{\mathcal{A}}}}{n_{key}}$. By keeping parameter n_{key} secret to $\mathcal{AD}$, we consider the cardinalities obfuscated and SR3 could still be considered as fulfilled to a certain level.

We recall that there exists an optimized manner to get the cardinality of a set from its Bloom filter representation as explained in Section 5, the S&B technique. Without any overlapping bit, getting the result is therefore straightforward. On the contrary, having multiple overlapping bits will lead any non-authorized party

to misinterpret the cardinality. To ensure that, the ratio of the amount of overlapping bits over parameter n_{key} should be important. Another aspect to consider with such an approach is that anytime an attacker gets a different Bloom filter of the same set $\mathcal{A}$, he can get closer to its cardinality. Indeed, the cardinality of $\mathcal{A}$ will always be greater or equal to $\frac{X_{BF_\mathcal{A}}}{n_{key}}$. By accessing $X_{BF_\mathcal{A}}$ for multiple different Bloom filters of the same set $\mathcal{A}$, he therefore can refine his cardinality guessing.

We also notice that having an acceptable probability of false negative and an acceptable level of privacy for the set cardinalities are contradicting strategies. Indeed, our approach to solve the *disjointness* set relation is based on reducing the amount of overlapping bits to avoid confusion having common elements.

Sets Cardinalities Attack. We present here how an attacker, as the auditor, could aim to retrieve cardinalities $n_\mathcal{A}$ and $n_\mathcal{B}$. To do so, the $\mathcal{AD}$ will firstly try to determine parameter n_{key} used by the owner of $\mathcal{A}$ and $\mathcal{B}$. The $\mathcal{AD}$ knows that $n_{key} \in [n_{key}^L; n_{key}^U]$ and that n_{key} is a factor of the amount of bits inserted in both Bloom filters. The candidates list for n_{key} is represented as $L_{n_{key}} = \{l_1, \dots, l_{\lambda_{n_{key}}}\}$ with $\lambda_{n_{key}}$ a security parameter that represents the size of this list. We also consider the two sub-lists $L^\mathcal{A}_{n_{key}} = \{l_1, \dots, l_{\lambda_\mathcal{A}}\}$ and $L^\mathcal{B}_{n_{key}} = \{l_1, \dots, l_{\lambda_\mathcal{B}}\}$ which correspond to the lists of factors regarding $BF_\mathcal{A}$ and $BF_\mathcal{B}$ before the cross-checking that leads to $L_{n_{key}}$. We set $Y_{BF_\mathcal{A}} \in [ob_{\mathcal{A}_1}; ob_{\mathcal{A}_2}]$ and $Y_{BF_\mathcal{B}} \in [ob_{\mathcal{B}_1}; ob_{\mathcal{B}_2}]$ the amounts of overlapping bits in the Bloom filters. In each Bloom filter, some overlapping bits could have occurred, therefore the attacker knows that regarding $BF_\mathcal{A}$, n_{key} is a factor of $X_{BF_\mathcal{A}}$ or $(X_{BF_\mathcal{A}} + 1)$ or $(X_{BF_\mathcal{A}} + 2) \dots$ Similarly holds for $BF_\mathcal{B}$. It means that $L^\mathcal{A}_{n_{key}}$ (resp. $L^\mathcal{B}_{n_{key}}$) is composed by elements l_j which verify the two characteristics:

$$l_j \in [n_{key}^L; n_{key}^U] \text{ and } l_j | x_\mathcal{A} \text{ with } x_\mathcal{A} \in [X_{BF_\mathcal{A}} + ob_{\mathcal{A}_1}; X_{BF_\mathcal{A}} + ob_{\mathcal{A}_2}]$$
$$(\text{resp. } l_j | x_\mathcal{B} \text{ with } x_\mathcal{B} \in [X_{BF_\mathcal{B}} + ob_{\mathcal{B}_1}; X_{BF_\mathcal{B}} + ob_{\mathcal{B}_2}])$$

Finally, we have $L_{n_\mathcal{A}} = (l_i)_{i \in [1; \lambda_{n_\mathcal{A}}]}$ the list of candidates for $n_\mathcal{A}$ with $\lambda_{n_\mathcal{A}}$ the amount of elements in $L_{n_\mathcal{A}}$. Similarly we have $L_{n_\mathcal{B}} = (l_i)_{i \in [1; \lambda_{n_\mathcal{B}}]}$ the list of candidates for $n_\mathcal{B}$ with $\lambda_{n_\mathcal{B}}$ the amount of elements in $L_{n_\mathcal{B}}$. The first step of the attack consists of listing all the common factor of $\{X_{BF_\mathcal{A}}, (X_{BF_\mathcal{A}} + 1), (X_{BF_\mathcal{A}} + 2), \dots\}$ and $\{X_{BF_\mathcal{B}}, (X_{BF_\mathcal{B}} + 1), (X_{BF_\mathcal{B}} + 2), \dots\}$ to generate lists $L^\mathcal{A}_{n_{key}}$ and $L^\mathcal{B}_{n_{key}}$. Then, $\mathcal{AD}$ will intersect the two list to generate the candidates list $L_{n_{key}}$.

The second phase of the attack is to translate $L_{n_{key}}$ into lists $L_{n_\mathcal{A}}$ and $L_{n_\mathcal{B}}$. The $\mathcal{AD}$ could use the S&B technique [20] to approximate size $n_\mathcal{A}$ and since parameter m is public and value $X_{BF_\mathcal{A}}$ is directly computable, we have the following function:

$$n^*_\mathcal{A}(n_{key}) = -\frac{m}{n_{key}} ln\Big[1 - \frac{X_{BF_\mathcal{A}}}{m}\Big] \tag{20}$$

When we look closely to the list $L_{n_{key}}$, we could notice that if some elements are following, they are translated to the same candidate of $n_\mathcal{A}$. In other words,

multiple elements from $L_{n_{key}}$ correspond to the same element from $L_{n_{\mathcal{A}}}$, thus we get $\lambda_{n_{\mathcal{A}}} \leqslant \lambda_{n_{key}}$. In Appendix H we give a toy example of this *sets cardinalities attack*.

Attack on Three Cases. We implement the *sets cardinalities attack* and we run it with different levels of knowledge for the auditor. We differentiate three cases such that the *best case*, the *in between case* and the *worst case*. They could be seen as three attacker models where the level of knowledge is minimal in the *best case* and maximal in the *worst case*.

Best Case. The auditor knows nothing on the expected overlapping bits distribution. $\mathcal{AD}$ considers that any amount of overlapping bits could occur and thus the attack will not get it any valuable information on the sets cardinality. Indeed, the attack will return $\lambda_{\mathcal{A}} = \lambda_{\mathcal{B}} = \lambda_{n_{key}} = (n^U_{key} - n^L_{key})$. In such ideal case we get a perfect privacy on the sets cardinality.

In-between Case. In such a case we consider that the auditor has more knowledge on the overlapping bits that affect the Bloom filters. We assume that the auditor is not able to predict the exact overlapping bits distribution but could approximately estimate the overlapping bits ranges $[ob_{\mathcal{A}_1}; ob_{\mathcal{A}_2}]$ and $[ob_{\mathcal{B}_1}; ob_{\mathcal{B}_2}]$. $\mathcal{AD}$ will perform the attack with these maximized default ranges.

Worst Case. In the worst case assumption, the $\mathcal{AD}$ knows the exact overlapping bits distribution for both Bloom filters. With this knowledge, the auditor could associate a weight to any element from $L_{n_{key}}$ and thus element from $L_{n_{\mathcal{A}}}$ and $L_{n_{\mathcal{B}}}$. We set weight for each element from $L_{n_{key}}$ based on the overlapping distribution of each Bloom filter.

7 Implementation and Results

We have implemented our protocols of the set relations in Java. The following measurements have been made with CPU configuration of Intel Core i5 M520 2.40GHz x 4.

Results on the Cloud Auditing Use case. First of all, running the implementation allowed us to express a precise value of the false positive (resp. false negative) rate. We test our solution with parameters suiting the cloud security auditing use case meaning a whitelist of 10^3 or 10^4 elements and logfiles from 10^2 to 10^4 elements. To test the false positive case, we generate Bloom filters of a whitelist $\mathcal{W}$ and a logfile $\mathcal{L}_1$ with different amounts of IP addresses. The tested parameters configurations are displayed in Table 5. Every IP addresses inserted in $BF_{\mathcal{L}_1}$ are also inserted in $BF_{\mathcal{W}}$ except for one. For every parameters configuration we test the *inclusiveness* relation $INC(\mathcal{L}_1, \mathcal{W})$ 10^3 times. We performed the same experimentation protocol with the *disjointness* relation. In both cases we obtain 0.00% of false positives respectively false negatives.

Performance. In Table 5 we show the performance of our two set relations. We run 10^4 times each set operator for more accuracy and the computation times are expressed in seconds. We see that performance times decrease linearly depending on parameter m, indeed as presented, the set relations are equivalent to bit-wise operations on the Bloom filters. That being said, we also notice that the performance times considering the set cardinality privacy are by far acceptable especially in an auditing use case.

Table 5. Running times of the two set relations in seconds.

$n_{\mathcal{W}}$	$n_{\mathcal{L}_1}$	$n_{\mathcal{L}_2}$	m	n_{key}	n_{key}^L	n_{key}^U	Time for $INC(\mathcal{L}_1, \mathcal{W})$	Time for $DIS(\mathcal{L}_2, \mathcal{W})$
10^3	10^3	10^3	$1.18 \cdot 10^9$	733	500	2000	$2.57 \cdot 10^{-1}$	$2.16 \cdot 10^{-1}$
10^3	10^3	10^3	$7.62 \cdot 10^9$	1861	500	2000	$2.51 \cdot 10^{-1}$	$7.44 \cdot 10^{-1}$
10^4	$9 \cdot 10^3$	$2 \cdot 10^2$	$2.93 \cdot 10^9$	816	500	2000	$2.29 \cdot 10^{-1}$	$2.80 \cdot 10^{-1}$
10^4	$9 \cdot 10^3$	$2 \cdot 10^2$	$9.47 \cdot 10^9$	1468	500	2000	$2.16 \cdot 10^{-1}$	$8.67 \cdot 10^{-1}$

8 Conclusion

We showed how to compute two specific set relations namely *inclusiveness* and *disjointness* of outsourced sets of data in a privacy preserving manner. In addition to fulfill privacy on the content, we provided a certain level of privacy on the cardinality of the Bloom filter's data structure. Our implementation results validate that this level is acceptable for instance when applied to a cloud security audit on access control. Such an approach based on Bloom filters could be easily adapted also to other set relations or operations like *equality* or *relative complement*.

References

1. Bellovin, S.M., Cheswick, W.R.: Privacy-enhanced searches using encrypted bloom filters. Cryptology ePrint Archive, Report 2004/022 (2004), `https://eprint.iacr.org/2004/022`
2. Bloom, B.H.: Space/time trade-offs in hash coding with allowable errors. Commun. ACM **13**(7), 422–426 (Jul 1970). https://doi.org/10.1145/362686.362692, `http://doi.acm.org/10.1145/362686.362692`
3. Burkhart, M., fontas, X.D.: Fast private set operations with sepia (2012)
4. Churches, T., Christen, P.: Some methods for blindfolded record linkage. BMC Med. Inf. & Decision Making (2004). https://doi.org/10.1186/1472-6947-4-9, `https://doi.org/10.1186/1472-6947-4-9`
5. Egert, R., Fischlin, M., Gens, D., Jacob, S., Senker, M., Tillmanns, J.: Privately computing set-union and set-intersection cardinality via bloom filters. In: Foo, E., Stebila, D. (eds.) Information Security and Privacy - 20th Australasian Conference, ACISP 2015, Brisbane, QLD, Australia, June 29 - July 1, 2015, Proceedings. Lecture Notes in Computer Science, vol. 9144, pp. 413–430. Springer (2015), `http://dx.doi.org/10.1007/978-3-319-19962-7_24`

6. Goh, E.: Secure indexes. IACR Cryptology ePrint Archive **2003**, 216 (2003), http://eprint.iacr.org/2003/216
7. Han, S., Xing, J.: Ensuring data storage security through a novel third party auditor scheme in cloud computing. In: 2011 IEEE International Conference on Cloud Computing and Intelligence Systems, CCIS 2011, Beijing, China, September 15-17, 2011. pp. 264–268. IEEE (2011). https://doi.org/10.1109/CCIS.2011.6045072, https://doi.org/10.1109/CCIS.2011.6045072
8. Kantarcioglu, M., Nix, R., Vaidya, J.: An efficient approximate protocol for privacy-preserving association rule mining. In: Theeramunkong, T., Kijsirikul, B., Cercone, N., Ho, T.B. (eds.) Advances in Knowledge Discovery and Data Mining, 13th Pacific-Asia Conference, PAKDD 2009, Bangkok, Thailand, April 27-30, 2009, Proceedings. Lecture Notes in Computer Science, vol. 5476, pp. 515–524. Springer (2009). https://doi.org/10.1007/978-3-642-01307-2_48, https://doi.org/10.1007/978-3-642-01307-2_48
9. Kerschbaum, F.: Public-key encrypted bloom filters with applications to supply chain integrity. In: Li, Y. (ed.) Data and Applications Security and Privacy XXV - 25th Annual IFIP WG 11.3 Conference, DBSec 2011, Richmond, VA, USA, July 11-13, 2011. Lecture Notes in Computer Science, vol. 6818. Springer (2011). https://doi.org/10.1007/978-3-642-22348-8, https://doi.org/10.1007/978-3-642-22348-8_7
10. Kerschbaum, F.: Outsourced private set intersection using homomorphic encryption. In: Youm, H.Y., Won, Y. (eds.) 7th ACM Symposium on Information, Compuer and Communications Security, ASIACCS '12, Seoul, Korea, May 2-4, 2012. pp. 85–86. ACM (2012). https://doi.org/10.1145/2414456.2414506, http://doi.acm.org/10.1145/2414456.2414506
11. Lai, P.K.Y., Yiu, S., Chow, K.P., Chong, C.F., Hui, L.C.K.: An efficient bloom filter based solution for multiparty private matching. In: Arabnia, H.R., Aissi, S. (eds.) Proceedings of the 2006 International Conference on Security & Management, SAM 2006, Las Vegas, Nevada, USA, June 26-29, 2006. CSREA Press (2006)
12. Li, Z., Gong, G.: Efficient data aggregation with secure bloom filter in wireless sensor networks
13. Lindell, Y., Pinkas, B.: Privacy preserving data mining. In: Bellare, M. (ed.) Advances in Cryptology - CRYPTO 2000, 20th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 2000, Proceedings. Lecture Notes in Computer Science, vol. 1880, pp. 36–54. Springer (2000). https://doi.org/10.1007/3-540-44598-6_3, https://doi.org/10.1007/3-540-44598-6_3
14. Lins, S., Teigeler, H., Sunyaev, A.: Towards a bright future: Enhancing diffusion of continuous cloud service auditing by third parties. In: 24th European Conference on Information Systems, ECIS 2016, Istanbul, Turkey, June 12-15, 2016. p. Research Paper 130 (2016), http://aisel.aisnet.org/ecis2016_rp/130
15. Nojima, R., Kadobayashi, Y.: Cryptographically secure bloom-filters. Trans. Data Privacy **2**(2), 131–139 (2009), http://www.tdp.cat/issues/abs.a015a09.php
16. Qiu, L., Li, Y., Wu, X.: Preserving privacy in association rule mining with bloom filters. J. Intell. Inf. Syst. **29**(3), 253–278 (2007). https://doi.org/10.1007/s10844-006-0018-8, https://doi.org/10.1007/s10844-006-0018-8
17. Rizvi, S.S., Bolish, T.A., III, J.R.P.: Security evaluation of cloud service providers using third party auditors. In: Hamdan, H., Boubiche, D.E., Toral-Cruz, H., Akleylek, S., Mcheick, H. (eds.) Proceedings of the Second International Conference on Internet of things and Cloud Computing, ICC

2017, Cambridge, United Kingdom, March 22-23, 2017. pp. 106:1–106:6. ACM (2017). https://doi.org/10.1145/3018896.3025154, https://doi.org/10.1145/3018896.3025154

18. Schadow, G., Grannis, S.J., McDonald, C.J.: Discussion paper: Privacy-preserving distributed queries for a clinical case research network. In: Proceedings of the IEEE International Conference on Privacy, Security and Data Mining. pp. 55–65. CRPIT '14, Australian Computer Society, Inc., Darlinghurst, Australia (2002), http://dl.acm.org/citation.cfm?id=850782.850790
19. Schnell, R., Bachteler, T., Reiher, J.: Privacy-preserving record linkage using bloom filters. BMC Med. Inf. & Decision Making **9**, 41 (2009). https://doi.org/10.1186/1472-6947-9-41, https://doi.org/10.1186/1472-6947-9-41
20. Swamidass, S.J., Baldi, P.: Mathematical correction for fingerprint similarity measures to improve chemical retrieval. Journal of Chemical Information and Modeling **47**(3), 952–964 (2007). https://doi.org/10.1021/ci600526a, https://doi.org/10.1021/ci600526a, pMID: 17444629
21. Tajan, L., Westhoff, D., Reuter, C.A., Armknecht, F.: Private information retrieval and searchable encryption for privacy-preserving multi-client cloud auditing. In: 11th International Conference for Internet Technology and Secured Transactions, ICITST 2016, Barcelona, Spain, December 5-7, 2016. pp. 162–169. IEEE (2016). https://doi.org/10.1109/ICITST.2016.7856690, http://dx.doi.org/10.1109/ICITST.2016.7856690
22. Zhang, H., Ye, L., Shi, J., Du, X., Guizani, M.: Verifying cloud service-level agreement by a third-party auditor. Security and Communication Networks **7**(3), 492–502 (2014). https://doi.org/10.1002/sec.740, https://doi.org/10.1002/sec.740

A Notation

$\mathcal{AD}, \mathcal{C}, \mathcal{CSP}$	The auditor, the client and the cloud service provider
$\mathcal{W}$	The client's whitelist
$\mathcal{L}_1, \mathcal{L}_2$	The two sets of the logfile
$\mathcal{A}$, $\mathcal{B}$	Generic sets
$INC()$, $DIS()$	*Inclusiveness* and *disjointness* operators
$BF_{\mathcal{A}}, BF_{\mathcal{A}\cap\mathcal{B}}, BF_{\mathcal{A}\subseteq\mathcal{B}=\emptyset}$	The respective Bloom filter representation of $\mathcal{A}$, the *inclusiveness* and the *disjointness* relation between $\mathcal{A}$ and $\mathcal{B}$
$bf_{\mathcal{A}}[i]$	i^{th} index of $BF_{\mathcal{A}}$
$n_{\mathcal{A}}, N_{\mathcal{A}}$	Cardinality of set $\mathcal{A}$ and maximization of this size
h	HMAC function
K	Set of the keys used with function h
n_{key}	Amount of keys in K
n_{key}^L, n_{key}^U	Lower and upper bounds of n_{key}
m	Size of the Bloom filter
$\mathcal{P}_{FP}$	Probabilities of having a false positive to the *inclusiveness* operator
$\mathcal{P}_{FN}$	Probabilities of having a false negative to the *disjointness* operator
$X_{BF_{\mathcal{A}}}$	Number of bits set to 1 in $BF_{\mathcal{A}}$
$X_{\cap=\emptyset}$	Estimation of the number of bits set to 1 in $BF_{\mathcal{A}\subseteq\mathcal{B}=\emptyset}$ if $\mathcal{A}$ and $\mathcal{B}$ were disjoint
$Y_{BF_{\mathcal{A}}}$	Amount of overlapping bits in $BF_{\mathcal{A}}$
$Y^*_{BF_{\mathcal{A}}}$	Estimation of $Y_{BF_{\mathcal{A}}}$
$Z_{\mathcal{A},\mathcal{B}}$	Amount of elements from $\mathcal{A}$ which are not in $\mathcal{B}$
$Z'_{\mathcal{A},\mathcal{B}}$	Amount of elements in both $\mathcal{A}$ and $\mathcal{B}$
ρ, μ	Standard deviation and mean of the overlapping bits distribution
$L_{n_{key}}, L^{\mathcal{A}}_{n_{key}}$	Candidates lists of n_{key} used in general, in $BF_{\mathcal{A}}$
$L_{n_{\mathcal{A}}}$	Candidates list of $n_{\mathcal{A}}$
$\lambda_{n_{key}}, \lambda^{\mathcal{A}}_{n_{key}}, \lambda_{n_{\mathcal{A}}}$	Cardinalities of the candidates lists $L_{n_{key}}, L^{\mathcal{A}}_{n_{key}}$ and $L_{n_{\mathcal{A}}}$
$[ob^L_{\mathcal{A}}; ob^U_{\mathcal{A}}]$	Range of the overlapping bits distribution for $BF_{\mathcal{A}}$

B Toy Example: Inclusiveness Relation

We give an example of the *inclusiveness* relation with quite small sets and parameters. We test if $\mathcal{A}$ and $\mathcal{A}'$ are included in $\mathcal{B}$.
$\mathcal{A} = \{x_1, x_2\}$, $\mathcal{A}' = \{x_1, x_4\}$, $\mathcal{B} = \{x_1, x_2, x_3\}$
$m = 12$, $n_{key} = 3$, $K = \{k_1, k_2, k_3\}$

$h_{k_1}(x_1) = 10$, $h_{k_1}(x_2) = 1$, $h_{k_1}(x_3) = 5$, $h_{k_1}(x_4) = 9$
$h_{k_2}(x_1) = 11$, $h_{k_2}(x_2) = 10$, $h_{k_2}(x_3) = 12$, $h_{k_2}(x_4) = 2$
$h_{k_3}(x_1) = 2$, $h_{k_3}(x_2) = 5$, $h_{k_3}(x_3) = 8$, $h_{k_3}(x_4) = 1$
By computing the *inclusiveness* relation we see that $\mathcal{A} \subseteq \mathcal{B}$. On the contrary $\mathcal{A}' \nsubseteq \mathcal{B}$ since $bf_{\mathcal{A}'}[9] = 1$ and $bf_{\mathcal{B}}[9] = 0$.

BF_A	1	1	0	0	1	0	0	0	0	1	1	0
$BF_{A'}$	1	1	0	0	0	0	0	0	1	1	1	0
BF_B	1	1	0	0	1	0	0	1	0	1	1	1

$BF_{A \subseteq B}$	0	0	0	0	0	0	0	0	0	0	0	0
$BF_{A' \subseteq B}$	0	0	0	0	0	0	0	0	1	0	0	0

C Toy Example: Disjointness Relation

We give another toy example to test the *disjointness* relation between $BF_{\mathcal{A}}$, $BF'_{\mathcal{A}}$ and $BF_{\mathcal{B}}$.
$\mathcal{A} = \{x_4, x_5\}$, $\mathcal{A}' = \{x_1, x_4\}$, $\mathcal{B} = \{x_1, x_2, x_3\}$
$m = 12$, $n_{key} = 3$, $n^L_{key} = 3$, $n^U_{key} = 5$, $K = \{k_1, k_2, k_3\}$

$h_{k_1}(x_1) = 10$, $h_{k_1}(x_2) = 1$, $h_{k_1}(x_3) = 5$, $h_{k_1}(x_4) = 9$, $h_{k_1}(x_5) = 6$
$h_{k_2}(x_1) = 11$, $h_{k_2}(x_2) = 10$, $h_{k_2}(x_3) = 12$, $h_{k_2}(x_4) = 2$, $h_{k_2}(x_5) = 1$
$h_{k_3}(x_1) = 2$, $h_{k_3}(x_2) = 5$, $h_{k_3}(x_3) = 8$, $h_{k_3}(x_4) = 1$, $h_{k_3}(x_5) = 7$

BF_A	1	1	0	0	0	1	1	0	1	0	0	0
$BF_{A'}$	1	1	0	0	0	0	0	0	1	1	1	0
BF_B	1	1	0	0	1	0	0	1	0	1	1	1

$BF_{A \cap B}$	1	1	0	0	0	0	0	0	0	0	0	0
$BF_{A' \cap B}$	1	1	0	0	0	0	0	0	0	1	1	0

We get $X_{\mathcal{A} \cap \mathcal{B}} = 2$ and $X_{\mathcal{A}' \cap \mathcal{B}} = 4$. Since we have $X_{\mathcal{A} \cap \mathcal{B}} < n^L_{key}$, we definitively know that $\mathcal{A}$ and $\mathcal{B}$ are disjoint. On the contrary, we get $X_{\mathcal{A}' \cap \mathcal{B}} \geqslant n^L_{key}$, therefore $\mathcal{A}'$ and $\mathcal{B}$ might be disjoint. Indeed with such toy configuration we have $X_{\cap = \emptyset} = 2$ and we do not get $n^L_{key} << X_{\cap = \emptyset}$ and with $X_{\mathcal{A}' \cap \mathcal{B}} = 4$ we hesitate between either $\mathcal{A}'$ and $\mathcal{B}$ disjoint or having one element in common.

D Toy Example: False Positive with the Inclusiveness Relation

We give here an example of the *inclusiveness* relation between $BF_{\mathcal{A}}$ and $BF_{\mathcal{B}}$ and we obtain a false positive case.

$\mathcal{A} = \{x_1, x_5\}$, $\mathcal{B} = \{x_1, x_2, x_3\}$
$m = 12$, $n_{key} = 3$, $n_{key}^L = 3$, $n_{key}^U = 5$, $K = \{k_1, k_2, k_3\}$

$h_{k_1}(x_1) = 10$, $h_{k_1}(x_2) = 1$, $h_{k_1}(x_3) = 5$, $h_{k_1}(x_5) = 12$
$h_{k_2}(x_1) = 11$, $h_{k_2}(x_2) = 10$, $h_{k_2}(x_3) = 12$, $h_{k_2}(x_5) = 5$
$h_{k_3}(x_1) = 2$, $h_{k_3}(x_2) = 5$, $h_{k_3}(x_3) = 8$, $h_{k_3}(x_5) = 1$

BF_A	1	1	0	0	1	0	0	0	0	1	1	0
BF_B	1	1	0	0	1	0	0	1	0	1	1	1

$BF_{A \subset B}$	0	0	0	0	0	0	0	0	0	0	0	0

By performing the *inclusiveness* relation *INC* on $BF_{\mathcal{A}}$ and $BF_{\mathcal{B}}$ we get $\mathcal{A} \subseteq \mathcal{B}$ but this is not correct since $x_5 \notin \mathcal{B}$.

E False Positive Probability

We test our implementation of Bloom filters construction for multiple parameters configurations. We express in Table 6 the percentage of false positive obtained while performing 10^4 times the *inclusiveness* relation and we verify that it corresponds to the respective $\mathcal{P}_{FP}$ expressed in (14).

Table 6. Comparation of the False positive percentage resulting from the running of the inclusiveness relation's implementation and the probability from (14).

$n_{\mathcal{W}}$	n_{key}	m	$\mathcal{P}_{FP}$	$\%_{FP}$
10^2	10^1	$1 \cdot 10^3$	$1.02 \cdot 10^{-2}$	$9.00 \cdot 10^{-3}$
10^2	$7 \cdot 10^2$	$1 \cdot 10^4$	$5.28 \cdot 10^{-1}$	$5.38 \cdot 10^{-1}$
10^3	5	$5 \cdot 10^3$	$1.01 \cdot 10^{-1}$	$1.01 \cdot 10^{-1}$
10^3	10^2	$2 \cdot 10^4$	$5.09 \cdot 10^{-1}$	$5.12 \cdot 10^{-1}$
10^4	$5 \cdot 10^2$	$1 \cdot 10^6$	$3.40 \cdot 10^{-2}$	$4.00 \cdot 10^{-2}$
10^4	10^3	$1.5 \cdot 10^6$	$2.80 \cdot 10^{-1}$	$2.20 \cdot 10^{-1}$

F Predicting 0verlapping Bits for the Disjointness Relation

We compare the value of $X_{\cap=\emptyset}$ obtained by (17) with the value of $X_{\mathcal{W}\cap\mathcal{L}_2=\emptyset}$ obtained by running 10^4 times our implementation. We show our results in Table 7.

Table 7. Comparation of the overlapping bits on the disjointness relation between two distinct sets and its prediction from (17).

$n_{\mathcal{W}}$	$n_{\mathcal{L}_2}$	n_{key}	m	$X_{\cap=\emptyset}$	$X_{\mathcal{W}\cap\mathcal{L}_2=\emptyset}$
10^2	10^2	10	$1 \cdot 10^3$	$3.99 \cdot 10^2$	$4.00 \cdot 10^2$
10^2	10^2	10	$1 \cdot 10^6$	$9.99 \cdot 10^{-1}$	$1.02 \cdot 10^0$
10^2	10^2	700	$1 \cdot 10^4$	$9.99 \cdot 10^3$	$9.98 \cdot 10^3$
10^2	10^3	700	$1 \cdot 10^7$	$4.71 \cdot 10^3$	$4.72 \cdot 10^3$
10^3	10^3	5	$5 \cdot 10^7$	$5.00 \cdot 10^{-1}$	$5.02 \cdot 10^{-1}$
10^3	10^3	100	$2 \cdot 10^4$	$1.97 \cdot 10^4$	$1.97 \cdot 10^4$
10^3	10^3	1500	$1 \cdot 10^7$	$1.94 \cdot 10^5$	$1.94 \cdot 10^5$
10^3	10^4	1000	$1 \cdot 10^7$	$6.01 \cdot 10^5$	$6.02 \cdot 10^5$
10^4	10^4	1500	$1 \cdot 10^8$	$1.94 \cdot 10^6$	$1.94 \cdot 10^6$

G Overlapping Bits Distribution

To complete our analysis of the *overlapping bits* we run our Bloom filter implementation for several parameters configurations 10^3 times each. We start by illustrating in Figure 2 the distribution of $Y_{BF_{\mathcal{A}}}$ in two parameters configurations knowing $\{n_{\mathcal{A}} = 50,\ m = 1.3 \cdot 10^7,\ n_{key} = 200\}$ and $\{n_{\mathcal{A}} = 200,\ m = 1.4 \cdot 10^7,\ n_{key} = 100\}$.

In Figure 3, we show the distributions for Bloom filters of two sets with similar cardinalities ($n_{\mathcal{A}} \approx n_{\mathcal{B}}$) and the same parameters configuration. We also show the distribution of the amount of bits in the Bloom filter corresponding to the result of the disjointness operator between the two previous Bloom filter representing two distinct set. We have $\{n_{\mathcal{A}} = 1000, m = 1.44 \cdot 10^4, n_{key} = 10\}$ and $\{n_{\mathcal{B}} = 1100, m = 1.44 \cdot 10^4, n_{key} = 10\}$.

In Figure 4 we show a case where the parameters are generated following recommendations from (18) and (19) for set $\mathcal{B}$ and 100 distinct elements inserted in the Bloom filters. We use the same parameters configuration with set $\mathcal{A}$ containing only 10 distinct elements ($n_{\mathcal{A}} \ll n_{\mathcal{B}}$). We have $\{n_{\mathcal{A}} = 10, m = 1.44 \cdot 10^3, n_{key} = 10\}$ and $\{n_{\mathcal{B}} = 100, m = 1.44 \cdot 10^3, n_{key} = 10\}$.

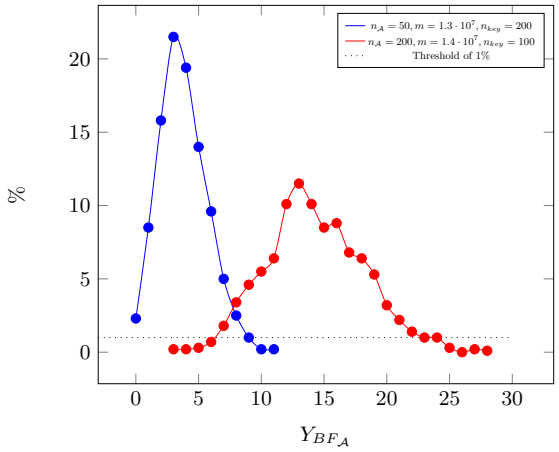

Fig. 2. The amounts of overlapping bits distribution for 10^3 Bloom filters of the same set generated.

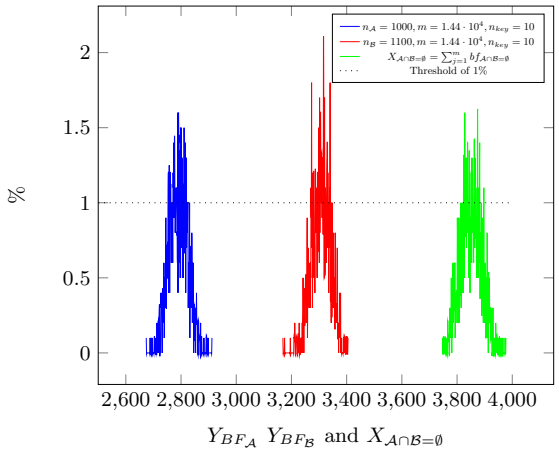

Fig. 3. The amounts of overlapping bits distribution for $n_{\mathcal{A}} \approx n_{\mathcal{B}}$.

H Set Cardinality Attack

We give a toy example of the attack obtained with our implementation of the protocol. The parameters selected by the clients are the following: $n_{key} = 17$ with $n_{key}^{L} = 10$ and $n_{key}^{U} = 30$, $m = 1.5 \cdot 10^8$, $n_{\mathcal{A}} = 1024$ and $n_{\mathcal{B}} = 2305$.

The auditor gets $BF_{\mathcal{A}}$ and $BF_{\mathcal{B}}$ and counts their respective bits set to 1; $X_{BF_{\mathcal{A}}} = 17407$ and $X_{BF_{\mathcal{B}}} = 39179$. He is then listing the factors as described in Section 6 by considering the overlapping bits amounts as $Y_{BF_A}, Y_{BF_B} \in [0; 10]$. The auditor selects the common factors from $L^{\mathcal{A}}_{n_{key}}$ and $L^{\mathcal{B}}_{n_{key}}$ and narrows the candidates list to $L_{n_{key}} = \{10, 11, 12, 13, 14, 15, 16, 17, 21, 23\}$. We note that the

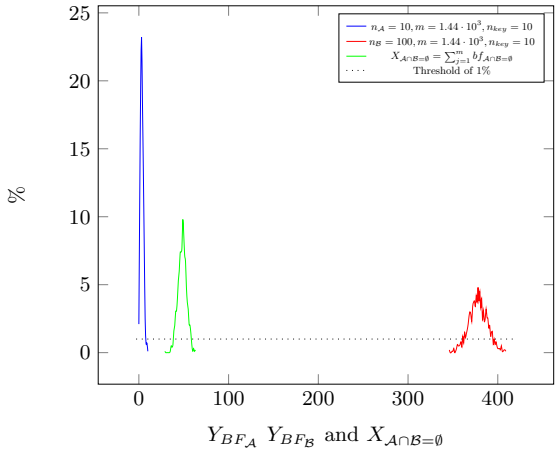

Fig. 4. The amounts of overlapping bits distribution for $n_{\mathcal{A}} \ll n_{\mathcal{B}}$.

$Y_{BF_{\mathcal{A}}}/Y_{BF_{\mathcal{B}}}$	$X_{BF_{\mathcal{A}}} + Y_{BF_{\mathcal{A}}}$	$L^{\mathcal{A}}_{n_{key}}$	$X_{BF_{\mathcal{B}}} + Y_{BF_{\mathcal{B}}}$	$L^{\mathcal{B}}_{n_{key}}$
0	1707	13	39179	29
1	1708	16, 17	39180	10, 12, 15, 20, 30
2	1709	21	39181	$\emptyset$
3	1710	10	39182	11, 13, 22, 26
4	1711	23	39183	$\emptyset$
5	1712	12	39184	16
6	1713	11	39185	17
7	1714	$\emptyset$	39186	14, 18, 21
8	1715	15, 27	39187	$\emptyset$
9	1716	14, 28	39188	$\emptyset$
10	1717	$\emptyset$	39189	$\emptyset$

auditor has reduced the candidates list by more than half. Indeed, before the attack we had $\lambda_{n_{key}} = 21$ and now $\lambda_{n_{key}} = 10$.

Then the auditor translates $L_{n_{key}}$ into $L_{n_{\mathcal{A}}}$ and $L_{n_{\mathcal{B}}}$ with the (S&B) technique:

$$L_{n_{\mathcal{A}}} = \{757, 829, 1024, 1088, 1161, 1243, 1339, 1451, 1583, 1741\}$$
$$L_{n_{\mathcal{B}}} = \{1704, 1866, 2305, 2449, 2612, 2799, 3014, 3265, 3562, 3918\}$$

We give more examples of the attack in particular for each of the three cases considered as attacker model.

H.1 Best Case

The auditor could only translate the information that n_{key} is any integer and $n_{key} \in [n^L_{key}; n^U_{key}]$. Therefore, the auditor reduces the candidates list to $L_{n_{\mathcal{A}}} = [\lceil n^*_{\mathcal{A}}(n^U_{key})\rceil; \lfloor n^*_{\mathcal{A}}(n^L_{key})\rfloor]$ and gets its cardinality as $\lambda_{n_{\mathcal{A}}} = \lfloor n^*_{\mathcal{A}}(n^L_{key})\rfloor -$

$\lceil n^*_{\mathcal{A}}(n^U_{key})\rceil + 1$ since in this case the attacker could not yet exclude any value from $L_{n_{\mathcal{A}}}$. We notice that in some cases, $\lambda_{n_{\mathcal{A}}}$ could be very small even without performing the attack. It is the case especially when $n_{\mathcal{A}}$ is quite small. In Table 8 we give some examples of $L_{n_{\mathcal{A}}}$ and $\lambda_{n_{\mathcal{A}}}$ without any attack.

Table 8. Respective $L_{n_{\mathcal{A}}}$ and $\lambda_{n_{\mathcal{A}}}$ for several parameter configurations.

$n_{\mathcal{A}}$	m	n^L_{key}	n^U_{key}	$X_{BF_{\mathcal{A}}}$	$L_{n_{\mathcal{A}}}$	$\lambda_{n_{\mathcal{A}}}$
10	$2 \cdot 10^6$	1500	2000	16935	$[9; 11]$	3
100	$2 \cdot 10^6$	1500	2000	162968	$[85; 113]$	29
100	$6 \cdot 10^7$	500	1000	49981	$[51; 100]$	50
1000	$6 \cdot 10^7$	500	1000	497936	$[501; 1000]$	500

H.2 In-between case

In Table 9 we show the results of the attack with such a limited knowledge.

Table 9. Results of the attack.

$n_{\mathcal{A}}$	$n_{\mathcal{B}}$	m	n_{key}	$[ob_{\mathcal{A}_1}; ob_{\mathcal{A}_2}]$	$[ob_{\mathcal{B}_1}; ob_{\mathcal{B}_2}]$	$\lambda_{\mathcal{A}}$	$\lambda_{\mathcal{B}}$	$\lambda_{n_{key}}$	$\lambda_{n_{\mathcal{A}}}$	$\lambda_{n_{\mathcal{B}}}$
100	1000	$6 \cdot 10^7$	500	[0 - 50]	[1500 - 2500]	38	315	38	38	38
10	100	$2 \cdot 10^6$	1700	[0 - 150]	[6000 - 8000]	46	425	46	3	3
100	100	$2.2 \cdot 10^7$	1700	[500- 1000]	[500 - 1000]	142	140	136	29	29

H.3 Worst case

For this attacker model we give examples with three different parameters configurations. In each case, we run 10^3 Bloom filter generations of each specific set to obtain a precise overlapping bits distribution. Then we provide all the retrieved candidates for parameter n_{key} and how they are translated into $n_{\mathcal{A}}$ and $n_{\mathcal{B}}$ candidates along with their respective weight. In particular we set a weight of 1.00 for the most probable candidate for parameter n_{key} and then, we set the other ones proportionately to this one.

We start by a parameters configuration of $\{n_{\mathcal{A}} = 100, n_{\mathcal{B}} = 1000, m = 6{\cdot}10^7, n_{key} = 500, n^L_{key} = 500, n^U_{key} = 1000\}$. Before the attack the auditor knows that $L_{n_{\mathcal{A}}} = [50; 100]$, $\lambda_{n_{\mathcal{A}}} = 51$, $L_{n_{\mathcal{B}}} = [500; 1000]$ and $\lambda_{n_{\mathcal{B}}} = 501$. In Table 10 we show the results of the attack obtained by running our implementation with the overlapping bits distribution obtained by generating 10^3 Bloom filters of the two sets. We see in Table 10 that the attacker gets greater weights for candidates which are not the correct ones. In particular we have six couples of candidates with an equal or greater probability than the correct couple $n_{\mathcal{A}}, n_{\mathcal{B}}$.

Table 10. Example of translating $L_{n_{key}}$ into $L_{n_{\mathcal{A}}}$ and $L_{n_{\mathcal{B}}}$.

Element from $L_{n_{key}}$	Weight	Element from $L_{n_{\mathcal{A}}}$ and $L_{n_{\mathcal{B}}}$	Weight
500	**0.67**	**100 and 1000**	**0.67**
505	1.00	99 and 990	1.00
532	0.02	94 and 940	0.02
625	0.67	80 and 800	0.67
633	0.02	79 and 790	0.02
641	0.94	78 and 780	0.94
658	0.02	76 and 760	0.02
685	0.14	73 and 730	0.14
862	0.85	58 and 580	0.85
877	0.43	57 and 570	0.43
893	0.02	56 and 560	0.02
909	1.00	55 and 550	1.00
926	0.17	54 and 540	0.17
1000	0.67	50 and 500	0.67

The second selected parameters configuration is $\{n_{\mathcal{A}} = 10, n_{\mathcal{B}} = 100, m = 2{\cdot}10^6, n_{key} = 1700, n^L_{key} = 1500$ and $n^U_{key} = 2000\}$. Before the attack the auditor knows that $L_{n_{\mathcal{A}}} = [8; 11]$, $\lambda_{n_{\mathcal{A}}} = 3$, $L_{n_{\mathcal{B}}} = [85; 113]$ and $\lambda_{n_{\mathcal{B}}} = 29$. In Table 11 we show the attack's results. With the attack, we get $L_{n_{\mathcal{A}}} = \{9, 10, 11\}$ and

Table 11. Example of translating $L_{n_{key}}$ into $L_{n_{\mathcal{A}}}$ and $L_{n_{\mathcal{B}}}$.

Element from $L_{n_{key}}$	Weight	Element from $L_{n_{\mathcal{A}}}$ and $L_{n_{\mathcal{B}}}$	Weight
1545, 1546, 1547	0.10, 0.53, 1.00	110 and 11	1.63
1700, 1701, 1702	**0.16, 0.71, 0.84**	**100 and 10**	**1.71**
1889, 1890	0.33, 0.71	90 and 9	1.04

$L_{n_{\mathcal{B}}} = \{90, 100, 110\}$ with $\lambda_{n_{\mathcal{A}}} = \lambda_{n_{\mathcal{B}}} = 3$. We also get the greater weight for the correct candidates.

We give a last example where $n_{\mathcal{A}} \approx n_{\mathcal{B}}$. The selected parameters are $\{n_{\mathcal{A}} = 100, n_{\mathcal{B}} = 100, m = 2.2{\cdot}10^7, n_{key} = 1700, n^L_{key} = 1500, n^U_{key} = 2000\}$. Before the attack the auditor knows that $L_{n_{\mathcal{A}}} = \lambda_{n_{\mathcal{A}}} = [85; 113]$ and $\lambda = \lambda_{n_{\mathcal{B}}} = 29$. In Table 12 we show the resulting $L_{n_{\mathcal{A}}}$ and $L_{n_{\mathcal{B}}}$. Finally we get $L_{n_{\mathcal{A}}} = L_{n_{\mathcal{B}}} = [85; 113]$ and $\lambda_{n_{\mathcal{A}}} = \lambda_{n_{\mathcal{B}}} = 29$ namely the same lists that before performing the attack. The auditor gets information on whether one candidate is more probable than another regarding the attached weight. We notice in this example that the correct value for $n_{\mathcal{A}}$ and $n_{\mathcal{B}}$ (100) is not the most probable, and even the auditor gets height candidates with an higher or equivalent probability of being the correct size. We see then that in some cases, the attack allows the auditor to reduce its lists $L_{n_{\mathcal{A}}}$ and $L_{n_{\mathcal{B}}}$ but in other cases it only provides approximation on the probability of being the actual size. We have shown in this section, depending on the considered security model, how much information any attacker could gain on the sets cardinality. We have seen that increasing the domain of parameters

Table 12. Example of translating $L_{n_{key}}$ into $L_{n_{\mathcal{A}}}$ and $L_{n_{\mathcal{B}}}$.

Element from $L_{n_{key}}$	Weight	Element from $L_{n_{\mathcal{A}}}$ and $L_{n_{\mathcal{B}}}$	Weight
1504	0.46	113	0.46
1518	0.61	112	0.61
1531	0.21	111	0.21
1545	0.54	110	0.54
1560	0.18	109	0.18
1574	0.93	108	0.93
1589	0.43	107	0.43
1604	0.64	106	0.64
1619	0.75	105	0.75
1634, 1635	0.18	104	0.36
1650	0.54	103	0.54
1666	0.21	102	0.21
1683	0.68	101	0.68
1700	**0.71**	**100**	**0.71**
1717	0.68	99	0.68
1734, 1735	0.21	98	0.42
1752	0.29	97	0.29
1771	0.61	96	0.61
1789	0.46	95	0.46
1808	0.46	94	0.46
1828	0.93	93	0.93
1848	0.61	92	0.61
1868	0.96	91	0.96
1889	0.71	90	0.71
1910	1.00	89	1.00
1932	0.61	88	0.61
1954	0.82	87	0.82
1976	0.18	86	0.18
2000	0.71	85	0.71

n_{key} and m could allow to reduce such information. We also recall that we assume in this use case that the most predominant privacy consideration is on the content of the sets. Preserving the cardinality's privacy is therefore secondary and providing a certain level of privacy is considered as sufficient.