Privately Computing Set-Union and Set-Intersection Cardinality via Bloom Filters

Rolf Egert Marc Fischlin David Gens Sven Jacob Matthias Senker
Jörn Tillmanns

Technische Universität Darmstadt, Germany

Abstract. In this paper we propose a new approach to privately compute the set-union cardinality and the set-intersection cardinality among multiple honest-but-curious parties. Our approach is inspired by a proposal of Ashok and Mukkamala (DBSec'14) which deploys Bloom filters to approximate the size tightly. One advantage of their solution is that it avoids ample public-key cryptography. Unfortunately, we show here that their protocol is vulnerable to actual attacks. We therefore propose a new Bloom filter based protocol, also forgoing heavy cryptography, and prove its security.

1 Introduction

The set-union cardinality problem consists of n parties, each party holding a set X_i of data, and the task is to compute the size of the union of all sets, $s = |\bigcup X_i|$. Ideally, this should be done in a way such that only the final result s is revealed but nothing about the individual input sets. The problem is a close relative to other so-called privacy-preserving set operation problems, including the set-intersection cardinality problem where parties aim to compute $s = |\bigcap X_i|$ securely, and the more general set-union and set-intersection problems where the goal is to explicitly compute the union and the intersection, respectively, and not only their cardinality. There are also more relaxed versions of the problem, called disjointness and conjointness testing, where the parties only check whether the size of the intersection resp. union is 0 or not.

Protocols for privacy-preserving set operations have numerous applications, explaining the vast number of protocol proposals and publications (as discussed below). To name a few applications, De Cristofaro et al. [CGT12] point out that such protocols can be used for anomaly detection in network monitoring or for identifying common friends in social networks. Kerschbaum [Ker11] lists supply chain integrity protection as another application. De Cristofaro et al. [CGT12] also discuss that the cardinality versions can be specifically used in the areas of genomic computations, location sharing, or affiliation-hiding authentication.

1.1 The Ashok-Mukkamala Protocol

Ashok and Mukkamala [AM14] recently proposed a fast multi-party protocol for the set union cardinality problem using Bloom filters. A Bloom filter [Blo70] is an array-based representation for sets where each inserted element induces a set of 1's in the array, based on the results of hashing the element with a set of public hash functions.[1] Verifying membership can be then carried out by re-hashing the element and checking for 1-entries in the array. As such, Bloom filters have a small but tolerable false-positive rate, since one may accidentally hit 1's caused by other elements. The size of a Bloom filter is usually proportional to the number of elements, independently of the size of the domain of the elements.

[1]The hash functions do not need to be cryptographically secure, but can still be based on hash functions like SHA [TRL12].

The idea of the Ashok and Mukkamala (AM) protocol is now to use the fact that unions of Bloom filters correspond to the bit-wise OR of the arrays, which facilitates the computation of the union of the underlying sets. The size of the union then corresponds, with exponentially high accuracy [PSN10], to the number of 0's and 1's in the derived array. Unlike other solutions for the problem the AM protocol does not rely on expensive cryptographic operations and reduces the network communication overhead significantly.

Unfortunately, the authors of [AM14] only sketch some ideas why their protocol should indeed be secure, lacking a formal proof in the common simulation-based sense of secure multi-party computation. Our first result is to show that such a proof is elusive. We show how each party in the AM protocol can predict any other party's input set with very high precision, even if honestly following the protocol description, by inspecting the communication data afterwards. This clearly violates the notion of a secure protocol.

1.2 Secure Cardinality Protocols based on Bloom Filters

We adopt the approach [AM14] to use Bloom filters to devise protocols for the cardinality problems. Our first protocol is for the set-union cardinality problem and for three (or more) parties. We assume that each party first locally computes the Bloom filter of its input set. In the protocol two dedicated parties act as accumulators, receiving random shares of each party's Bloom filter. The sharing will be such that it is compatible with the union operation for Bloom filters, enabling each accumulator to compute the union over their Bloom filter shares. Both accumulators agree on a random permutation and forward the permuted arrays of their shares to the third dedicated party, the evaluator. The evaluator then assembles the permuted filters of the accumulators, computes the size of the union from the number of 1's, and announces the result to the other parties (if required).

Our protocol computes the size of the union with very small error (due to the negligible error caused by the Bloom filter approach and some term related to the secret sharing of the filters). It withstands honest-but-curious adversaries, as long as the adversary can only inspect the data of one of the three dedicated parties, and it provably leaks only the size of the union of the input Bloom filters of all parties.[2] The protocol is perfectly secure and does not rely on any cryptographic assumption. It requires little interaction between parties and the bit communication of each party is proportional to the size m of the Bloom filter. The latter is based on the observation that, for the cardinality problem, we can use "small" error-prone Bloom filter shares, as we are still able to approximate the number of errors sufficiently well and to subtract them out of the final result. We actually determine parameters enabling varying levels of communication complexity and accuracy.

Next we briefly discuss how to extent our protocol if more than one party can be accessed by the adversary. Basically, to tolerate t accesses of *dedicated* parties (and an arbitrary number of further parties) we need $\Omega(t^2)$ parties in total, building different layers of accumulators. For example, for $t = 2$ we need at least 6 participants. The protocol remains perfectly secure in the honest-but-curious setting and still provides a linear communication complexity for each party (linear in the size m of the Bloom filter).

We conclude with a two-party protocol for the cardinality problem, still based on Bloom filters. The protocol is a slight modification of the protocol of Kiayias and Mitrofanova [KM05] for testing disjointness, adopted to the case of union cardinality and the Bloom filter setting. Here we however need to revert to cryptographic means like homomorphic encryption. Our protocol is based on the ElGamal encryption scheme and computes the set-union cardinality problem in the honest-but-curious model. As opposed to similar protocols for the general set-union problem we take advantage of the fact that we merely need to compute the size of the union. This allows us to outsource the major computational effort to compute m

[2]Note that one can in principle compile protocols in the honest-but-curious case to thwart malicious attacks [Gol04], albeit this comes with an observable slow down.

encryptions and thus modular exponentiations for the Bloom filter array of size m to one party, the server. The client side, on the other hand, only computes a single encryption and m modular multiplications.

We finally discuss that all our protocols can be easily adapted to compute the set-intersection cardinality problem. Indeed, as pointed out in [CGT12], solutions in the two-party case for the private set-union cardinality problem in principle also give rise to protocols for the intersection problem, noting that $|X_1 \cap X_2| = |X_1| + |X_2| - |X_1 \cup X_2|$. However, this requires that the sizes of the individual sets X_1, X_2 are public, or that these values are incorporated privately into the computation. Also, the formula gets significantly more complicated for multiple parties.

Hence, we here instead use a direct approach for Bloom filters via De Morgan's law to adapt our protocols. This approach, surprisingly, went unnoticed in [AM14]. Given the Bloom filters each party first inverts its array bit-wise. Then they compute the cardinality of the union of these inverted filters with our protocol. This yields the bit-wise inverse of the AND of the original Bloom filters. The parties can thus derive the cardinality of the intersection by subtracting the cardinality of the obtained filter from the size of the Bloom filters.

1.3 Related Work

There are numerous works on privacy-preserving set operations, with various security and complexity properties. We list here only protocols specifically for set operation problems, neglecting general-purpose multi-party protocols. Still, we note that there are solutions which apply general methods like oblivious transfer to the set-operation setting. For a comprehensive overview about such approaches based on oblivious transfer, and their performance characteristics, see [PSZ14]. We also omit a comparison to assisted protocols in which (partially) trustworthy third parties support the evaluation; see [KMRS14] for an overview over these protocols.

The early proposals for set operation protocols by Freedman et al. [FNP04] and by Kiayias and Mitrofanova [KM05] used polynomials to represent input sets and jointly evaluated the polynomials via homomorphic encryption. Several subsequent approaches [KS05, HW06, Fri07, DMRY09, HN12, Haz15] adopted this idea in order to improve over security guarantees, efficiency, or functionality. Compared to our protocols these works often aim at the more general problems of set intersection and set union, and start by designing versions withstanding honest-but-curious adversaries before adding (efficient) zero-knowledge proofs to achieve security against malicious adversaries. At the same time, the proposals are usually more costly, even in the honest-but-curious model, in the sense that they deploy public-key cryptographic operations, and they often focus on the two-party case. This is also true for other public-key based approaches like [DKT10, DT10, CGT12].

Besides the oblivious polynomial evaluation approach, intersections can also be computed via oblivious pseudorandom functions [FIPR05]. Basically, this allows one party to evaluate a pseudorandom function on selected values without knowledge of the key, such that intersections can be spotted by comparing function values. This approach has been refined in several works [HL10, JL09, JL10], including also proposals with trustworthy hardware tokens [HL08, FPS$^+$11]. All these approaches are for two-parties only and, due to the current state of art of oblivious pseudorandom functions, based on public-key operations or additional hardware components.

Using Bloom filters for privacy-preserving set operations has been suggested before for example in [Ker11, Ker12]. The former work [Ker11] describes a Bloom filter based two-party protocol using homomorphic encryption to allow membership verification of single elements. This comes at the cost of a high communication complexity and reveals the element to be checked. The latter paper [Ker12] extends this idea and securely computes the set intersection via Bloom filters and homomorphic encryption, but adds extra computations on encrypted elements. Dong et al. [DCW13] used so-called garbled Bloom filters to compute intersections based on oblivious transfer. Garbled Bloom filters basically store the bits of the underlying sets in a randomly shared way in a *single* filter. This is in contrast to our approach in which the original filter is split into many shares. In addition, the oblivious transfer step in their protocol is inherent, even in the honest-but-curious model.

There are also approaches for secure set-operation protocols which are built on top of very basic multi-party computation protocols. For example, using the SEPIA library with secure protocols for addition and multiplication of bits, Many et al. [MBD12] propose to compute the union of Bloom filters via a bit-wise OR according to the formula $a \vee b = a + b - a \cdot b$. Especially the sub protocol for multiplication in this formula, however, adds a significant overhead. Similarly, Blanton and Aguiar [BA12] use basic comparison operations and oblivious sorting to implement set operations like the computation of the union in a modular way. Compared to our dedicated solution the overhead is again noticeable.

2 Preliminaries

2.1 Secure Multi-Party Computation

We follow the common approach to multi-party computations as in, e.g., [Gol04]. We assume a known number of n participants, following the terminology in [AM14] sometimes also called sites. Each party has some individual input and the joint goal is to privately compute a function f of these inputs. We usually consider the case that a single party, the *distributor*, first obtains the function output and that this party can then forwards the value to the other parties. We assume that the n parties are all connected point-wise, and that the protocol execution happens asynchronously in the sense that the parties' communication depends on incoming messages instead of global clocks and synchronous rounds of interaction.

We only sketch the formal security definitions of multi-party computations here; a more rigorous approach can be found in [Gol04]. In the *honest-but-curious* (aka. semi-honest) security model the efficient adversary receives the views of a subset of parties after a faithful protocol execution, consisting of the input and the internal randomness of the parties, the incoming and outgoing messages of all these parties, and the function value if the party is the distributor. This is in contrast to the *malicious* scenario where parties controlled by an adversary may arbitrarily deviate from the protocol, including aborts. As mentioned before, there are means to transform protocols in the honest-but-curious case into protocols withstanding malicious attacks [Gol04], at the expense of extra steps.

We let the honest-but-curious adversary inspect up to t of the n views of the parties for some parameter t. We call such adversaries *t-bounded*. We assume that the adversary selects the t parties and their views *non-adaptively*, at the outset of the protocol execution. Note that the adversary would usually also receive the protocol communication of the other parties sent over their public connections, but we may assume that secure channels are established through standard cryptographic means.[3]

Security in the honest-but-curious case now says that any adversary cannot deduce more from the views than what is known from their protocol inputs and the protocol's output anyway. This is formalized by saying that whatever the adversary can output based on the views, another algorithm could compute from the parties' inputs and the protocol's output without seeing the protocol execution. More formally, a protocol *privately computes a function* if for any efficient, t-bounded, non-adaptive adversary $\mathcal{A}$ as above, there exists an efficient algorithm $\mathcal{S}$, called the simulator, such that for any inputs the output of the adversary and the one of the simulator are computationally indistinguishable. Here, the simulator only receives the inputs of up to t parties and the function output. If this holds even for unbounded adversaries and statistical (or even perfect) indistinguishability, then we say that the protocol is *statistically (or perfectly) private.*

Needless to say that securely computing a function implicitly assumes that a benign execution of the

[3]Note that such cryptographic channels mainly rely on the faster symmetric-key cryptography.

protocol yields the correct value. We are also interested in sufficiently close approximations in this case, emphasizing that this only affects correctness but not security of the protocol. Hence, for a real-valued function f we say that a protocol *securely ϵ-approximates f* if security holds as above, and in addition for any inputs a faithful execution of the protocol yields with probability $99,9\%$ an output which is within the interval $v \pm \epsilon$ of the actual function value v.

The function we are abstractly interested in is the set-union cardinality problem where each party holds a set of data and their goal is to securely compute the size of the union of all their data. Following the terminology in [AM14] the parties are also called sites S_i, their sets X_i of data are sometimes called transaction identifiers (containing some item x) and are denoted by $L_i(x)$ there, and the union is occasionally also denoted by $L(x) = \bigcup_{i=1}^{n} L_i(x)$.

2.2 Using Bloom Filters for Multi-Party Computations

In this work we use Bloom filters to represent the sets X_i. Bloom filters, introduced by Burton Howard Bloom in the 1970's [Blo70], are data structures for membership checking on sets. Given a data set $X = \{x_1, x_2, \cdots, x_d\}$, a bit-array of size m, initialized with 0, and a set of public hash functions $H = \{h_1, h_2, \cdots, h_k\}$ with range $\{1, 2, \dots, m\}$, one creates the Bloom filter BF for the set X as follows. For each pair $x_i \in X$ and $h_j \in H$ compute $h_j(x_i)$ and set the corresponding bit in the array to 1. To check if a given element a is in the set X one computes all values $h_j(a)$ for $j = 1, 2, \dots, k$, looks up the corresponding entries in the previously generated array, and predicts membership of a in X if and only if all resulting array positions contain 1's. This can only result in false positives due to collisions, i.e., if the hash values $h_j(a)$ for some element $a \notin X$ all map to 1's which have been set by hashing other elements.

The error rate of Bloom filters directly corresponds to the size m of the bit array, the number k of hash functions and the size d of the data set. To get a good trade-off between size of the Bloom filter and error-rate, we choose the parameters such that the equation $k = \frac{m}{d} \ln 2$, taken from [TRL12], holds. As usual, this assumes that the hash functions behave like independent random functions. With this the Bloom filter should be filled to approximately 50%.

When using Bloom filters for private set operations one usually starts with the representation of the input sets of the parties as Bloom filters for some agreed-upon hash functions. The protocol then usually computes the union or intersection (or their cardinality) of the Bloom filters BF_i, neglecting the small errors due to false-positive rate of the filters. In our case we thus compute the (parameterized) function $(X_1, \dots, X_n) \mapsto |\bigvee_{i=1}^{n} \mathrm{BF}_i|$ securely, where the hash functions are given as randomly chosen parameters (for the function and to the protocol participants, including the adversary) and are used to derive the Bloom filters BF_i. Note that computing $m - |\bigvee_{i=1}^{n} \mathrm{BF}_i|$, as we actually do here, is equivalent.[4]

We remark that cryptographic protocols using Bloom filters implicitly assume an a-priori bound on the size of sets. The reason is that the common parameters for the Bloom filters are usually chosen to match these numbers. Else, if chosen independently of the data size, the error rate for functional correctness may increase significantly. We also make the assumption here, visualized by the fact that the k hash functions for the filter and the filters' size m are available to all parties in our protocol.

3 Attacks on the Ashok-Mukkamala Protocol

As mentioned before security of a protocol for the set-union cardinality problem should guarantee that participants cannot learn anything about other parties' inputs beyond the size of the union of all data. We now show that the scheme proposed in [AM14] violates this security property as an adversary can

[4]The protocol in [AM14] also seems to aim to compute this function securely, but this is not specified.

identify another party's input with high probability. This holds even if the adversary only inspects the communication data of a party after protocol completion.

We first provide a brief overview of the protocol proposed by Ashok and Mukkamala [AM14]. Then we present two attacks that can be applied individually or in combination. In the first attack we identify a candidate list of possible input data for the "victim" S_i based on the partial Bloom filter sent from S_i to our adversarially inspected site S_j in the protocol execution. This partial Bloom filter is built from S_i's input data, but only for a random subset of all hash functions. Our goal is to estimate the amount and configuration of shared data between S_i and S_j. We also simulate the computation of such a candidate list and present figures in order to showcase the effectiveness and precision of this attack. In the second attack we aim to reconstruct the random subset of used hash functions of site S_i, utilizing input data an adversary assumes is possessed by S_i with high probability. As noted in [AM14] the choice of the subset is critical to the security of the overall protocol. We conclude this section with a brief description of a combination of the previous two attacks and provide the results we obtained from our experiments.

3.1 The AM-Protocol in a Nutshell

Recall that the AM-Protocol is based on the approximation of the size of a set via the entries in a Bloom filter. That is, given k independent and agreed-upon hash functions $h_1, \ldots, h_k$ with range in $\{1, 2, \ldots, m\}$ —where we assume in the analysis that the functions are random— we build the Bloom filter of bit size m by hashing each element $x \in X$ via all functions $h_i(x)$ and setting the corresponding bit to 1. Then, given only the Bloom filter, we can approximate the size of X via the number z of zeros in the filter by:

$$|X| = \frac{\ln(z/m)}{k \ln(1 - 1/m)}$$

Furthermore, the approximation is quite tight [TRL12].

The AM-Protocol exploits that the bitwise OR of partial Bloom filters for any r sets of hash function indices $K_1, K_2, \ldots, K_r \subseteq \{1, 2, \ldots, k\}$ yields a Bloom filter for the hash functions with indices in the union $K_1 \cup K_2 \cup \cdots \cup K_r$. More precisely, letting $\mathrm{BF}|_K$ denote the Bloom filter created by applying only hash functions with indices from $K \subseteq \{1, 2, \ldots, k\}$ to the set X, we have

$$\bigvee_{i=1}^{r} \mathrm{BF}|_{K_i} = \mathrm{BF}|_{\cup_{i=1}^{r} K_i}.$$

This enables each site S_i to split the entire Bloom filter for its input data into partial ones, one for each other party S_j, by picking random subsets $K_{S_i \to S_j} \subseteq \{1, 2, \ldots, k\}$ of all hash functions $h_1, \ldots, h_k$, and creating the partial Bloom filters $\mathrm{BF}_{S_i \to S_j} = \mathrm{BF}|_{K_{S_i \to S_j}}$. Each partial filter is then handed to the site S_j for further processing. This is called the decomposition phase in [AM14].

For correctness it is necessary to ensure that all hash keys are used in some Bloom filter $\mathrm{BF}_{S_i \to S_j}$ of party S_i. The AM-Protocol therefore first lets S_i pick a random number $r_{S_i \to S_j}$ in some range $[a, b]$ for each partial filter, and then lets S_i include $r_{S_i \to S_j}$ random keys in the subset $K_{S_i \to S_j}$. The exact figures and the roles of a, b remain unspecified; it seems to us that they should prevent trivial subsets. To ensure that each key appears in at least one filter, the AM-protocol once more lets S_i randomly assign each hash key to one of the n partial Bloom filters.

Once a party S_j has received the partial Bloom filters $\mathrm{BF}_{S_i \to S_j}$ from all sites S_i, the party S_j computes the union of all these Bloom filters to create a filter BF'_{S_j} and sends out this value to all other sites. This is called the reconstruction phase in [AM14]. In the final merger phase each site S_i computes the Bloom filter union over all values BF'_{S_j} received from the n sites S_j. The final step is now to approximate the size of the union of the input data via the number z of 0-entries in the final Bloom filter.

3.2 Attack #1: Computing a Candidate List

Note that the underlying security idea of the AM-Protocol is that in the decomposition phase each party S_j only obtains a partial Bloom filter $\mathrm{BF}_{S_i \to S_j}$ from any other party S_i. In fact, Ashok and Mukkamala [AM14] argue about the number $r_{S_i \to S_j}$ of used hash functions being well distributed, neglecting the question of how much information about the original data is still contained in a partial Bloom filter. This observation is the leverage for our first attack, which allows us to use the partial Bloom filters to verify, with high accuracy, if the other party's input contains some individual data x.

To illustrate our attack assume that $r_{S_i \to S_j}$ hash functions with indices $K_{S_i \to S_j}$ have been used to construct the partial Bloom filter $\mathrm{BF}_{S_i \to S_j}$. Both the number $r_{S_i \to S_j}$ and the actual set $K_{S_i \to S_j}$ are only known to party S_i. Party S_j now tries to check whether some data x is part of S_i's input or not. To do so, it first computes the fraction $p \in [0, 1]$ of 1-entries in the partial Bloom filter. Then S_j hashes x with *all* hash functions $h_1, \ldots, h_k$ and counts how often it hits 1's in the filter $\mathrm{BF}_{S_i \to S_j}$. If x is not part of S_i's input, the hash values will, on the average, hit kp times; if x on the other hand, is contained in the set, the process will generate $r_{S_i \to S_j}$ hits for the $r_{S_i \to S_j}$ actually chosen hash functions in $K_{S_i \to S_j}$, plus an expected $(k - r_{S_i \to S_j})p$ hits for the remaining, unused hashed functions. On the average these are about $kp + (1 - p)r_{S_i \to S_j}$ hits, which exceeds the amount in the other case significantly for reasonable parameter choices. Although this gap and the number of samples is presumably too small to apply the Chernoff-Hoeffding bounds for estimating the deviation of expectations, our experiments and threshold choice below show that this gap suffices for a good prediction. Hence, S_j can decide membership of x in S_i's input with high probability.

We have run simulations on how good we can actually predict membership of elements. For this we created two Bloom filters for sets X_i, X_j with a given overlap $|X_i \cap X_j|$ for the respective parameters m, $|X_i| = |X_j|$, and k. Since we work with the Bloom filters exclusively and the actual data are irrelevant, we emulated ideal hash functions by inserting 1's for each element at random positions to create the Bloom filters (but with consistent position choice for the common elements, of course). For the partial Bloom filters we picked a random subset $r_{S_i \to S_j} \in [a, b]$ of the hash functions, where we chose $a = 3$ and $b = k - 3$. For each parameter set we repeated the experiment $1,000$ times, averaging the results.

To determine the candidate elements we first count the number of ones for each element. Then we compute the average and maximum count (AVG and MAX) over all elements. Finally we select all elements as candidates for which the count is equal or greater to the following bound:

$$\mathrm{AVG} + \max(0.6, p) \cdot (\mathrm{MAX} - \mathrm{AVG})$$

That is, if the hits exceeds the average number by about 60% of the gap to the maximum (or a p-fraction of the gap, where p is the ratio of 1's in the received Bloom filter) then we consider the value to be in the set. Notice that for reasonable parameter choices, p never exceeds 0.6 and $\max(0.6, p)$ can be safely replaced with 0.6. However in the third simulated scenarios we deliberately chose parameters that lead to very large p. In this scenario replacing $\max(0.6, p)$ with 0.6 would reduce the attack's effectiveness significantly.

Table 1 shows our simulation results. Here, *precision* (basically determining exactness) designates the percentage of correctly identified candidates out of all the chosen candidates. We mark the percentage of correctly identified candidates out of the actually shared elements as *recall* (basically determining completeness).

To see why the attack works recall that for $x \notin X_i$ we get on the average $E_{\notin} := kp$ hits, because each of the random hash functions maps the distinct value x to random and independent positions. In contrast, the expectation for $x \in X_i$ is $E_{\in} := kp + (1 - p)r$, because the remaining $k - r$ hash functions also map x to random positions in the array. Since $E_{\in}$ is always greater than $E_{\notin}$, it is possible to choose a bound that is greater than $E_{\notin}$ and smaller than $E_{\in}$. All elements with a number of ones greater than that

$|X_i| = |X_j| = 10,000,$
$m = 1,000,000, k = 70$

$\lvert X_i \cap X_j\rvert$	precision	recall
10	79.5 %	90.8 %
100	95.4 %	83.8 %
1000	99.2 %	71.7 %

$|X_i| = |X_j| = 10,000,$
$m = 2,000,000, k = 140$

$\lvert X_i \cap X_j\rvert$	precision	recall
10	92.2 %	94.9 %
100	97.6 %	90.9 %
1000	99.8 %	83.3 %

$|X_i| = |X_j| = 25,000,$
$m = 1,000,000, k = 70$

$\lvert X_i \cap X_j\rvert$	precision	recall
10	31.3 %	69.9 %
100	72.5 %	58.4 %
1000	93.9 %	45.3 %

Table 1: Simulation results for attack #1

bound will be considered a candidate. Since $E_\in$ depends on r, which is unknown to the attacker, an ideal bound cannot be calculated beforehand. It can however be estimated after counting the ones for every element. The average number of ones (AVG) will usually be close to $E_1 \notin$ while the maximum number of ones (MAX) will be closer to $E_\in$. Selecting a value between AVG and MAX as the bound is therefore reasonable. Extensive simulations with different approaches to calculate the bound led us to the given formula.

Note that this approach will only work if S_i and S_j actually share at least one element. Otherwise MAX will not be a good estimation of $E_\in$ and the calculated bound will be meaningless. If it is not known if both sites share any elements, a static bound may be used instead, that depends only on the known parameters as well as p. However, its effectiveness will vary greatly with r and potentially also with other parameters. Simulations have shown that a bound of $2kp$ may then be a reasonable choice in many scenarios.

3.3 Attack #2: Reconstructing the Set of Hash Functions

For the second attack we use a candidate list of common elements which could have been generated, for example, by our first attack. The attack here takes advantage of the fact that, for common elements, we can check if one of the k hash functions coincides on these elements. That is, for each of the k hash functions we check how many of the elements in the list hit a 1 in the partial Bloom filter we have obtained. We can roughly expect that only the actually chosen r hash functions will produce hits for all candidates. If we have a perfectly reliable candidate list of common elements, then we can even identify correctly left out hash functions, if the hash function maps a candidate to 0.

Once we have identified the used hash functions we can in principle use the partial Bloom filter to plot out the other party's elements, with an error which is higher than for the optimal choice for Bloom filter parameters, but still giving useful information to an attacker. We refrain from doing so here, though, as the importance to hide this choice has already been mentioned in [AM14].

We have again run experiments to verify the correctness of our idea. We have used the same parameters as for the first attack. We used the exact list of common elements as the candidate list. Our algorithm decided to include a hash function as chosen if the number of ones counted for that hash function is greater or equal to $(\text{AVG}_h + \text{MAX}_h)/2$ where AVG_h and MAX_h are the average and maximum number of ones for all hash functions. The results of our experiments are given in Table 2.

3.4 Combination of Attacks #1 and #2

As briefly mentioned before the attacks can also be combined. In the first step we use attack #1 to generate a list of candidates that, with high probability, are part of the other party's input. We subsequently use this candidate list as an input for attack #2 to reconstruct the hash function set. The results of the simulation of the combined attack are displayed in Table 3 where *success* denotes the likelihood of identifying the r chosen hash functions correctly. Note that we cannot expect to achieve the same success rate as in the

$|X_i| = |X_j| = 10,000,$
$m = 1,000,000, k = 70$

$\lvert X_i \cap X_j \rvert$	success
10	99.2 %
100	100 %
1000	100 %

$|X_i| = |X_j| = 10,000,$
$m = 2,000,000, k = 140$

$\lvert X_i \cap X_j \rvert$	success
10	97.3 %
100	100 %
1000	100 %

$|X_i| = |X_j| = 25,000,$
$m = 1,000,000, k = 70$

$\lvert X_i \cap X_j \rvert$	success
10	65.2 %
100	100 %
1000	100 %

Table 2: Simulation results for attack #2

case of the exact candidate list, but our experiments show that for sufficiently large intersections we still achieve overwhelming rates.

$|X_i| = |X_j| = 10,000,$
$m = 1,000,000, k = 70$

$\lvert X_i \cap X_j \rvert$	success
10	75.7 %
100	99.5 %
1000	99.9 %

$|X_i| = |X_j| = 10,000,$
$m = 2,000,000, k = 140$

$\lvert X_i \cap X_j \rvert$	success
10	90.1 %
100	99.9 %
1000	100 %

$|X_i| = |X_j| = 25,000,$
$m = 1,000,000, k = 70$

$\lvert X_i \cap X_j \rvert$	success
10	0.7 %
100	70.7 %
1000	99.7 %

Table 3: Simulation results for combination of attacks #1 and #2

4 Advanced Protocols for the Set-Union Cardinality Problem

In this section we present our new protocols. Although inspired by the general idea of using Bloom filters as in [AM14] our proposed solutions are substantially different in the way they deploy Bloom filters. We only describe here the set-union protocol; the set-intersection protocol can be derived by working over the inverted Bloom filters, as explained in the Introduction.

We start with a description of a protocol for the case of at least three participants. We additionally provide a security proof for this case. Three of the parties in this protocol will embrace a special role and we call them core nodes. Two of the core nodes will act as accumulators of intermediate results of other parties and forward their values to the third party, the evaluator, who will compute the final result (and possibly distribute it to all other parties). In this version of the protocol an arbitrary number of participants can be inspected by an adversary, but at most one of the three core nodes. We will subsequently explain how the protocol needs to be extended to guarantee security in case an adversary inspects more than one core node. Conclusively, we present an approach for the scenario where there are only two participants available for protocol execution.

4.1 The Three-Party Case

We first discuss the most basic scenario of our protocol with an arbitrary number of regular participants and three core nodes as described above. An overview over the proposed protocol is given in Figure 1. Each of the core nodes acts as a regular participant, too, but has to perform some additional work. The protocol consists of a general part which is executed by all members and a special part for the core nodes. For simplification we will refer to the core nodes as A, B and C for the rest of this section, where A and B are called accumulators and C is called the evaluator.

The idea of the first part of the protocol is that each party first locally computes its Bloom filter for its input. Then the party transforms the representation of its Bloom filter of 0's and 1's into b-bit values

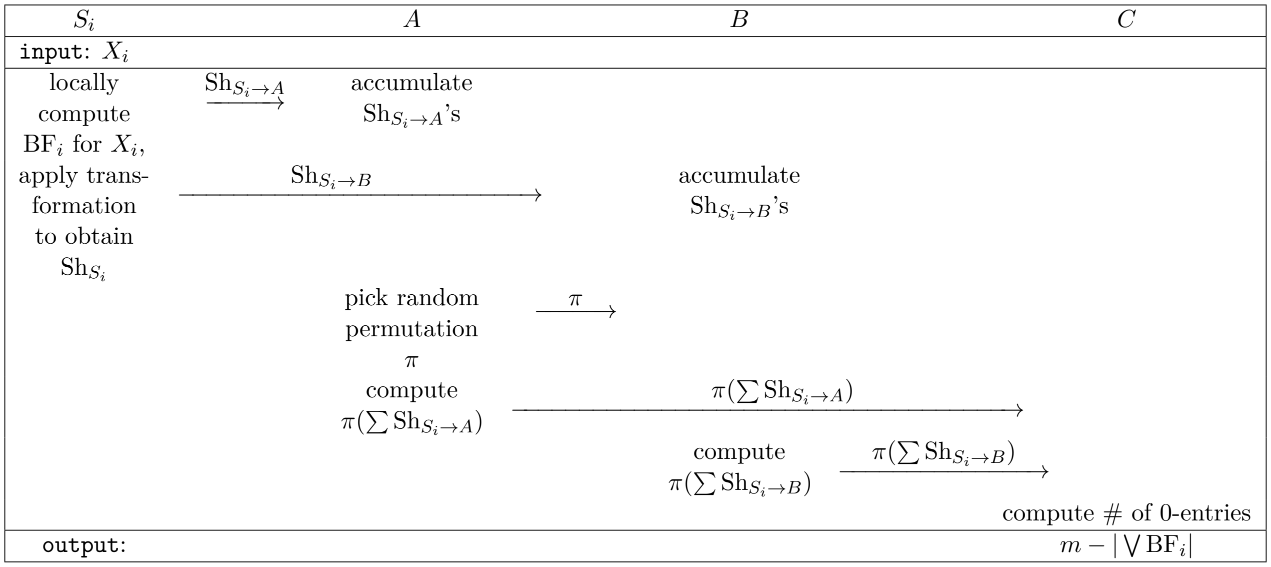
S_i
A
B
C
input: X_i
locally compute BF_i for X_i, apply transformation to obtain Sh_{S_i}
$\mathrm{Sh}_{S_i \to A}$
accumulate $\mathrm{Sh}_{S_i \to A}$'s
$\mathrm{Sh}_{S_i \to B}$
accumulate $\mathrm{Sh}_{S_i \to B}$'s
pick random permutation π
π
compute $\pi(\sum \mathrm{Sh}_{S_i \to A})$
$\pi(\sum \mathrm{Sh}_{S_i \to A})$
compute $\pi(\sum \mathrm{Sh}_{S_i \to B})$
$\pi(\sum \mathrm{Sh}_{S_i \to B})$
compute # of 0-entries
output:
$m - |\bigvee \mathrm{BF}_i|$

Figure 1: Protocol steps in the case of three or more parties

where

$$0 \mapsto 0 \bmod 2^b, \qquad 1 \mapsto \text{random element} \bmod 2^b.$$

Here $b = 4, 8$ or 16 may be appropriate choices, but even $b = 1$ is possible if one can tolerate slightly higher error rates. Subsequently the party splits its transformed Bloom filter representation into two equal-size shares $\mathrm{Sh}_{S_i \to A}$ and $\mathrm{Sh}_{S_i \to B}$ by mapping an entry $r \bmod 2^b$ to random $r_1, r_2 \bmod 2^b$ such that $r = r_1 + r_2 \bmod 2^b$ and placing r_1 in the first Bloom filter share and r_2 into the second one. Note that each of the two Bloom filter shares individually does not reveal any information about the original values. An example of this transformation process for $b = 8$ and thus $\mathbb{Z}_{256}$ is given in Figure 2.

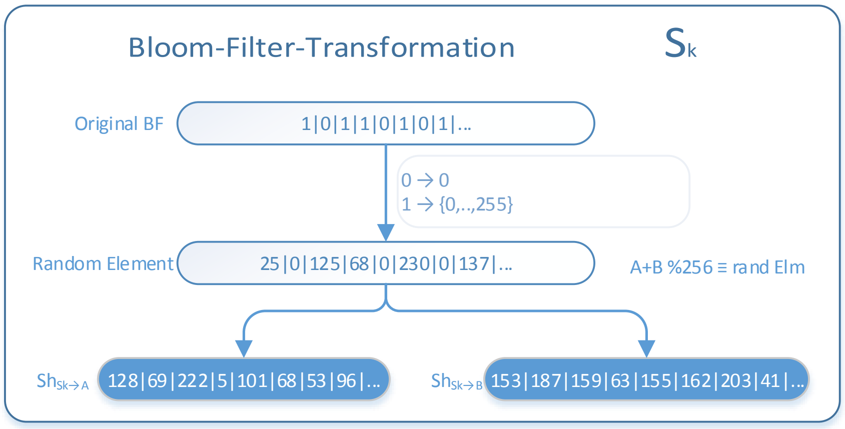

Figure 2: Local share computation of a node's Bloom filter

The parties now all send their first Bloom filter share to the accumulator A, and their second share to accumulator B. For each entry in the arrays, the accumulator simply sums up the corresponding entries in all received filter shares. At this point neither A nor B individually possess any useful information about the Bloom filters of the other parties. Still they hold a shared version of the combined Bloom filter of all parties. Now they agree on a random permutation π over $\{1, 2, \ldots, m\}$ and permute the entries of their shares according to π. This step is necessary to ensure that the evaluator C cannot reconstruct information about individual data from the combined filter. Both A and B then send their permuted filters to C who adds the entries component-wise and finally counts the number of 0-entries in the derived filter. Eventually C can compute an approximation the size of the union set as in [AM14]. An example of the calculations performed by the accumulators and the evaluator is given in Figure 3.

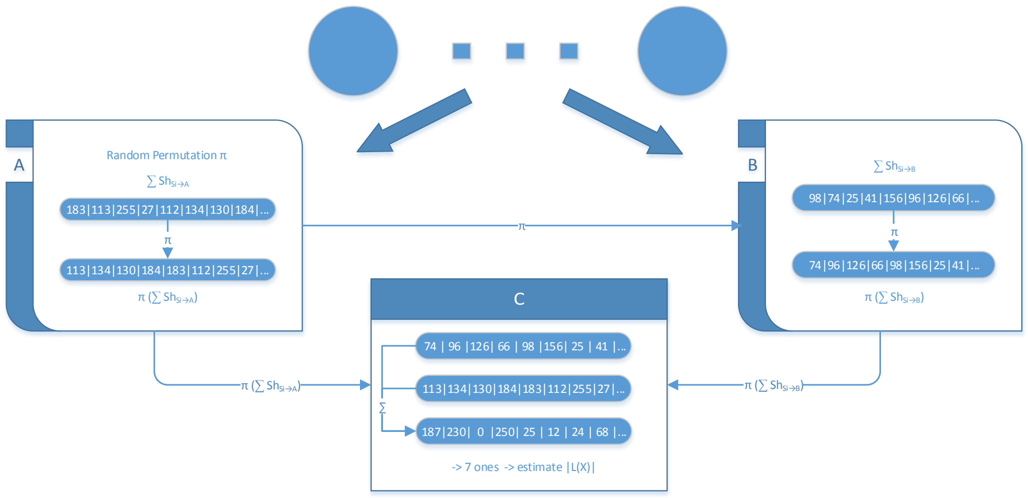

Figure 3: Accumulation and evaluation of the transmitted Bloom filters by the core nodes

Note that the final result, independently of the inaccuracies due to the set-size approximation technique via the zeros, contains a small error. This error is introduced by the representation of 1's as random elements in $\mathbb{Z}_{2^b}$, potentially picking 0 mod 2^b for a 1-entry.[5] However, we can actually estimate these false elements quite accurately: Let z_0 denote the number of 0-entries in the final Bloom filter, and z_1 the number of (false) 0-representations of original 1-entries, such that $z = z_0 + z_1$ for the eventually derived number z of zeros. Note that the expected value (over the random representations of the parties) of z_1 is $2^{-b}(m - z_0)$ since each of the $m - z_0$ non-zero entries is misrepresented with probability 2^{-b}. It follows that the expected number E_{z_0} of z_0 is given by

$$E_{z_0} = \frac{z - 2^{-b}m}{1 - 2^{-b}}.$$

Hence, given z (and the parameters m, b) we can approximate the actual value of z_0 easily.

Furthermore, since z_1 is given by the sum of independent variables, the number of errors in our approximation follows a binomial distribution. Suppose we take for z_1 a confidence interval of 99.9%. Then we can calculate the error bounds for the estimation of z_0 and thus of the final result for different values of b and $|\bigcup X_i|$, the actual size of the union. Approximating the binomial distribution through the Gaussian distribution we obtain that with probability at least 99.9% the result of an honest execution of the protocol lies within an interval ± 3.29 times the standard deviation $\sigma := \sqrt{(m - z_0) \cdot 2^{-b}(1 - 2^{-b})}$ around the mean. In this case, using the formula for estimating z_0, our approximation is in the interval $(m - z_0) \pm 3.29\sigma/(1 - 2^{-b})$ for the standard deviation σ with probability 99.9%.

[5]Excluding 0-representations for 1's potentially enables an attacker to trace the original 0-entries with some small, yet noticeable probability, such that we rather accept an error on the side of correctness.

For a concrete example consider a densely filled Bloom filter of size $m = 1,000,000$, with only 10% of 0-entries, and let $b = 1$. Then with probability of 99.9% our guess will be within $\pm 1,561$ elements of the expected 450,000 misrepresentations. Approximating z_0 as above with the same confidence we will output an estimation in the interval between 96,878 and 103,122 for the actual value 100,000. For the same value of m, but 50% of the Bloom filter being filled, and now with $b = 8$, our approximation of the actual value of 500,000 is within the range 498,854 and 500,146 with probability of at least 99.9%.

4.2 Proof of Security

We will now show security for our proposed protocol in the honest-but-curious security model, and under the additional assumption that the adversary is 1-bounded and is allowed to inspect at most one of the nodes. Recall that the notion of ϵ-approximation means that, in terms of correctness, the output value (for a benign execution) is within an interval $\pm\epsilon$ of the actual function value with probability 99.9%.

Theorem 4.1 *Let $n \geq 3$ and consider the n-party protocol in Section 4.1. The protocol ϵ-approximates the function $f_{m,h_1,\ldots,h_k}(X_1,\ldots,X_n) = m - |\bigvee_{i=1}^{n} BF_i|$ perfectly secure in the honest-but-curious case against 1-bounded non-adaptive adversaries, where $\epsilon = 3.29 \cdot (1-2^{-b})^{-1} \cdot \sqrt{2^{-b} \cdot (1-2^{-b}) \cdot |\bigvee_{i=1}^{n} BF_i|}$.*

We stress again that the parameter b only influences the correctness of the computation. The bound for ϵ has already been derived in the previous section. Note that for large b, like $b = 128$, and reasonable filter size the approximation is correct with probability 99.9%.

Proof. First note that if the party inspected by the adversary is different from a core node then security is trivial, as such parties only send out information. Such a view is easy to simulate because the simulator receives the party's input. We can thus focus on the case of the adversary accessing one of the core nodes. As before, let n denote the number of participants of the protocol, m the length of the Bloom filters, and b the number of bits that are used for representing a single bit by an element of the additive group $\mathbb{Z}_{2^b}$.

Consider first the case that an adversary inspects one of the accumulators A or B. We only consider the case of A here, as the case of B is analogous; the only difference is that A chooses the random permutation and transmits it. Because A is an accumulator, the adversary in the actual protocol execution learns a single share of every Bloom filter of every participant of the protocol. The entries of these shares $\mathrm{Sh}_{S_i \to A}$, $i = 1, 2, \ldots, n$ are determined by the randomness of the corresponding participant S_i. Each share individually consists of m random elements from $\mathbb{Z}_{2^b}$.

We can devise a simulator for A as follows: The simulator receives as input A's input set and the output of the protocol $|\bigvee_{i=1}^{n} \mathrm{BF}_i|$ (if this value is eventually distributed to all participants by the evaluator C). It creates a view of A by faithfully computing the first outgoing messages of A as an ordinary participant via A's input data. Then it simulates each incoming share $\mathrm{Sh}_{S_i \to A}$ by picking m random elements from $\mathbb{Z}_{2^b}$. The simulator also places a random permutation π in A's view, and the aggregated and permuted filter shares as the message sent to C. It is easy to see that our simulator perfectly simulates the view of A in an actual protocol execution.

Finally consider the case that the adversary asks to see the view of the evaluating node C. The node C in the actual protocol execution receives as input two randomly permuted vectors sent by A and B. When put together, these vectors add up to $m - |\bigvee \mathrm{BF}_i|$ presentations of 0-entries plus a random set of false 0-presentations of 1-entries, i.e., where the shares add up to 0 mod 2^b. We can build a simulator as follows. The simulator, receiving $m - |\bigvee \mathrm{BF}_i|$ and C's data set, first creates the random filter shares C would send as an ordinary participant to A and B. It next prepares an empty Bloom filter and randomly distributes $|\bigvee \mathrm{BF}_i|$ random elements from $\mathbb{Z}_{2^b}$ and leaves the other entries as 0. It next splits all the entries of this filter randomly into two Bloom filters as the protocol participants, and puts these shares into the view of C. This again perfectly simulates the actual view of C in a genuine protocol execution, because C in this case also receives randomly permuted entries of the same distribution. □

4.3 Extensions to the Multi-Party Case

The basic three-party protocol which we described previously is only secure if at most one of the three core nodes is inspected by an adversary. For an adversary being able to access more core nodes we need to adapt the previous approach. The basic idea of our extension is to use additional layers of accumulators. The i-th layer (where we count the evaluator as layer 1 and the two accumulators of the basic case as layer 2) will consist of i accumulators. To withstand t-bounded adversaries we will use $t+1$ of such layers where parties can only act as accumulators on one level. The set-up for $t = 2$ is given in Figure 4.

The execution starts by having each party compute its Bloom filter and splitting it up into $t + 1$ random shares over the group $\mathbb{Z}_{2^b}$ similar to the three-party case. i.e., 0-entries are random shares of 0, and 1-entries become random shares of random elements from $\mathbb{Z}_{2^b}$. Again all of the core nodes are also participating in the general Bloom filter distribution phase in the beginning, i.e., act both as accumulators as well as contributors. Then each party sends its $t + 1$ shares to the parties of the highest accumulator layer. The accumulators of a layer accumulate their shares and permute the accumulated Bloom filter shares randomly, where each layer picks a fresh permutation, e.g., chosen by one party and distributed to the accumulators of the same layer.

In the next step the accumulators forward their filters to the next lower layer. Here, allowing to reduce the communication overhead, it suffices that each lower-level accumulator receives at least two independent shares from the higher layer. In Figure 4 party B for example guarantees the two independent shares by splitting its share for the lower level, such that A and C can simply forward their share. When the evaluator eventually receives the shares it computes the output as before.

Security relies on the same observation as in the three-party case. First note that the adversary cannot obtain all individual shares of a participant sent to the $t + 1$ accumulators of the highest layer, as the adversary can only inspect up to t parties. The adversary could, potentially, inspect all t parties of the second highest layer, but then the data would correspond to randomly permuted shares of the overall result already. This is again easy to simulate, similar to the case of inspection of the evaluator C in the three party case.

If the adversary, on the other hand, inspects a party of the highest layer to learn the random permutation of that layer, then it must miss one of the t parties of the second highest layer and thus at least one of the shares. We can set this argument forth, noting that if the adversary misses the permutation of the Bloom filters (as the composition of all permutations of all t layers, except for layer 1) by not inspecting any party of some layer, then the t-bounded adversary cannot learn any information, besides inspecting the evaluator. But the t-bounded adversary can only learn the permutation if it inspects one accumulator for each layer. But then it misses the data of the evaluator and could only derive information about individual Bloom filters if it was able to access all shares of a layer. But by construction, since it inspects one party at each of the t layer levels, it must lack knowledge of a share on each level for every Bloom filter. It follows as in the three-party case that nothing is leaked except the function's output.

4.4 The Two-Party Case

In the two-party case it seems inevitable to rely on public-key cryptography and computations on encrypted data. Our solution here utilizes the homomorphic property of the ElGamal encryption scheme. Our protocol has the interesting feature that the major part of computational complexity lies on one side, supporting typical client-server scenarios: While one party has to perform $2m$ exponentiations to create the m ElGamal encryptions of the m Bloom filter entries, the other party merely needs to compute two

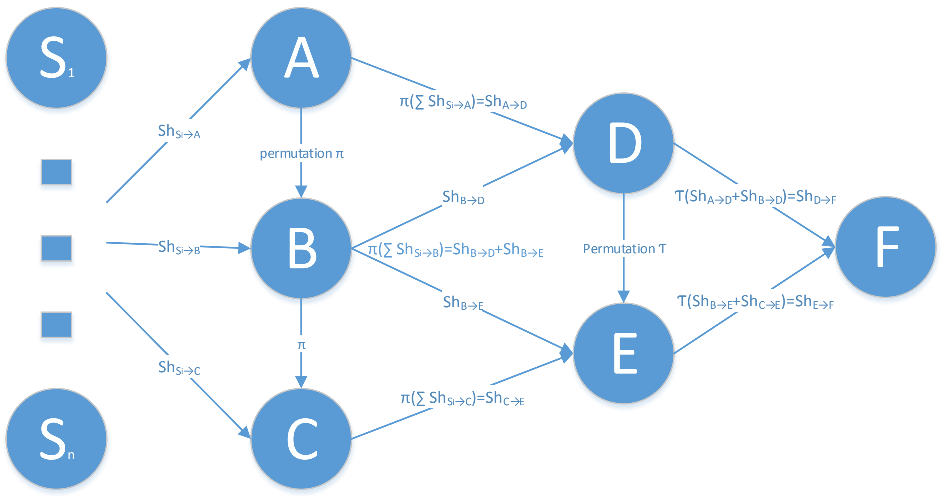

Figure 4: Accumulation and evaluation by the core nodes in the multi-party case

exponentiations and up to $2m + 2$ multiplications. In contrast to our previous protocols, the result of the protocol is perfectly correct.

The protocol in Figure 5 works as follows. First the parties will compute the intersection of the Bloom filters. For this party 1 encrypts the entries $\mathrm{BF}_1[i]$ of its Bloom filter bitwise as $g^{1-\mathrm{BF}_1[i]}$ to $(R_i, S_i) = (g^{r_i}, pk^{r_i} \cdot g^{1-\mathrm{BF}_1[i]})$ for random r_i. Among these m pairs, party 2 picks those for which its Bloom filter value $\mathrm{BF}_2[i]$ is 0, and multiplies them together (component-wise). To hide the choice of this subset, party 2 re-randomizes the result by multiplication with (g^s, pk^s). Note that, at this point, the obtained ciphertext (V, W) is an encryption of $g^{\sum_{i:\mathrm{BF}_2[i]=0}(1-\mathrm{BF}_1[i])}$, thus counting the number of entries i for which both filters are 0. Hence, if party 1 decrypts and determines this sum with at most m multiplications by comparing it to powers $g^0, g^1, g^2, \ldots g^m$, it obtains the size of the union of the filters by subtracting this sum from m.

We only sketch that the protocol securely computes $|\mathrm{BF}_1 \vee \mathrm{BF}_2|$ in the honest-but-curious model; a formal proof is straightforward. To argue security consider first the case that the adversary inspects the view of party 1 after an honest execution. In particular this means that a simulator needs to provide a consistent view, including the inputs, the randomness, and the incoming messages, given BF_1 and the final output $s = |\mathrm{BF}_1 \cup \mathrm{BF}_2|$. The simulator creates the view by picking the randomness faithfully as the honest party would, and by inserting a fresh ciphertext (V, W) of g^{m-s} into the transcript part of the view. Note that this view is identically distributed as a view of party 1 in an execution, showing that the protocol protects the data of party 1.

Next consider the case that the adversary inspects the view of party 2. In this case we again simulate the view given only the filter of party 2 and the size s of the union. To this end the simulator again faithfully picks the randomness for party 2 and simulates that the first, incoming message by creating a genuine key pk and m random ciphertexts (R_i, S_i) of 1's as (g^{r_i}, pk^{r_i}) for random r_i. The third protocol message is easy to simulate by simply passing s in clear. By the IND-CPA security of the ElGamal encryption scheme (under the decisional Diffie-Hellman assumption) it follows that the adversary cannot distinguish our simulated encryptions from genuine ones, and thus the views of the two cases.

```
Party 1                                                                 Party 2
input: Bloom filter $\mathrm{BF}_1$ of size $m$          Bloom filter $\mathrm{BF}_2$ of size $m$
pick $(sk, pk)$ for ElGamal scheme
over group $G = \langle g \rangle$ of prime order $q > m$

for $i = 1$ to $m$ do:
  pick $r_i \leftarrow \mathbb{Z}_q$
  set $(R_i, S_i) = (g^{r_i}, pk^{r_i} \cdot g^{1-\mathrm{BF}_1[i]})$
end
                    $\xrightarrow{pk, (R_i, S_i)_{i=1,2,\ldots,m}}$
                                                         pick $s \leftarrow \mathbb{Z}_q$
                                  compute $V = g^s \cdot \prod_{i:\mathrm{BF}_2[i]=0} R_i$
                                 compute $W = pk^s \cdot \prod_{i:\mathrm{BF}_2[i]=0} S_i$
                    $\xleftarrow{(V, W)}$
decrypt to $\Sigma = W \cdot V^{-sk}$
set $\sigma = 0$   //find $\sigma$ with $\Sigma = g^\sigma$
while $\Sigma \neq 1$ and $\sigma \leq m$ do
  set $\Sigma \leftarrow \Sigma \cdot g^{-1}$ and $\sigma \leftarrow \sigma + 1$
end
output: $m - \sigma$   //assume that party 1 sends the sum to party 2, too
```

Figure 5: Two-Party Protocol based on Bloom filters and the ElGamal encryption scheme

5 Conclusion

Our protocols adopt the basic idea of the AM protocol to devise secure multi-party protocols which are provably secure, lightweight on the crypto operations, and low on network communication. It remains an interesting open question how one can enhance our protocol to withstand malicious adversaries, ideally forgoing expensive zero-knowledge proofs. Also, a worthwhile effort would be to devise errorless versions of our protocol.

Acknowledgments

Marc Fischlin is supported by the Heisenberg grant Fi 940/3-2 and SPP 1736 grant Fi 940/5-1 of the German Research Foundation (DFG).

References

[AM14] Vikas G. Ashok and Ravi Mukkamala. A scalable and efficient privacy preserving global itemset support approximation using bloom filters. In *DBSec*, volume 8566 of *Lecture Notes in Computer Science*, pages 382–389. Springer, 2014. (Cited on pages 1, 2, 3, 4, 5, 6, 7, 8, 9, and 11.)

[BA12] Marina Blanton and Everaldo Aguiar. Private and oblivious set and multiset operations. In Heung Youl Youm and Yoojae Won, editors, *ASIACCS 12: 7th Conference on Computer and Communications Security*, pages 40–41, Seoul, Korea, May 2–4, 2012. ACM Press. (Cited on page 4.)

[Blo70] Burton H Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7):422–426, 1970. (Cited on pages 1 and 5.)

[CGT12] Emiliano De Cristofaro, Paolo Gasti, and Gene Tsudik. Fast and private computation of cardinality of set intersection and union. In *Cryptology and Network Security (CANS)*, volume 7712 of *Lecture Notes in Computer Science*, pages 218–231. Springer, 2012. (Cited on pages 1 and 3.)

[DCW13] Changyu Dong, Liqun Chen, and Zikai Wen. When private set intersection meets big data: an efficient and scalable protocol. In Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung, editors, *ACM CCS 13: 20th Conference on Computer and Communications Security*, pages 789–800, Berlin, Germany, November 4–8, 2013. ACM Press. (Cited on page 3.)

[DKT10] Emiliano De Cristofaro, Jihye Kim, and Gene Tsudik. Linear-complexity private set intersection protocols secure in malicious model. In Masayuki Abe, editor, *Advances in Cryptology – ASIACRYPT 2010*, volume 6477 of *Lecture Notes in Computer Science*, pages 213–231, Singapore, December 5–9, 2010. Springer, Berlin, Germany. (Cited on page 3.)

[DMRY09] Dana Dachman-Soled, Tal Malkin, Mariana Raykova, and Moti Yung. Efficient robust private set intersection. In Michel Abdalla, David Pointcheval, Pierre-Alain Fouque, and Damien Vergnaud, editors, *ACNS 09: 7th International Conference on Applied Cryptography and Network Security*, volume 5536 of *Lecture Notes in Computer Science*, pages 125–142, Paris-Rocquencourt, France, June 2–5, 2009. Springer, Berlin, Germany. (Cited on page 3.)

[DT10] Emiliano De Cristofaro and Gene Tsudik. Practical private set intersection protocols with linear complexity. In Radu Sion, editor, *FC 2010: 14th International Conference on Financial Cryptography and Data Security*, volume 6052 of *Lecture Notes in Computer Science*, pages 143–159, Tenerife, Canary Islands, Spain, January 25–28, 2010. Springer, Berlin, Germany. (Cited on page 3.)

[FIPR05] Michael J. Freedman, Yuval Ishai, Benny Pinkas, and Omer Reingold. Keyword search and oblivious pseudorandom functions. In Joe Kilian, editor, *TCC 2005: 2nd Theory of Cryptography Conference*, volume 3378 of *Lecture Notes in Computer Science*, pages 303–324, Cambridge, MA, USA, February 10–12, 2005. Springer, Berlin, Germany. (Cited on page 3.)

[FNP04] Michael J. Freedman, Kobbi Nissim, and Benny Pinkas. Efficient private matching and set intersection. In Christian Cachin and Jan Camenisch, editors, *Advances in Cryptology – EUROCRYPT 2004*, volume 3027 of *Lecture Notes in Computer Science*, pages 1–19, Interlaken, Switzerland, May 2–6, 2004. Springer, Berlin, Germany. (Cited on page 3.)

[FPS$^+$11] Marc Fischlin, Benny Pinkas, Ahmad-Reza Sadeghi, Thomas Schneider, and Ivan Visconti. Secure set intersection with untrusted hardware tokens. In Aggelos Kiayias, editor, *Topics in Cryptology – CT-RSA 2011*, volume 6558 of *Lecture Notes in Computer Science*, pages 1–16, San Francisco, CA, USA, February 14–18, 2011. Springer, Berlin, Germany. (Cited on page 3.)

[Fri07] Keith B. Frikken. Privacy-preserving set union. In Jonathan Katz and Moti Yung, editors, *ACNS 07: 5th International Conference on Applied Cryptography and Network Security*, volume 4521 of *Lecture Notes in Computer Science*, pages 237–252, Zhuhai, China, June 5–8, 2007. Springer, Berlin, Germany. (Cited on page 3.)

[Gol04] Oded Goldreich. *The Foundations of Cryptography*, volume 2. Cambridge University Press, 2004. (Cited on pages 2 and 4.)

[Haz15] Carmit Hazay. Oblivious polynomial evaluation and secure set-intersection from algebraic prfs, 2015. (Cited on page 3.)

[HL08] Carmit Hazay and Yehuda Lindell. Constructions of truly practical secure protocols using standardsmartcards. In Peng Ning, Paul F. Syverson, and Somesh Jha, editors, *ACM CCS 08: 15th Conference on Computer and Communications Security*, pages 491–500, Alexandria, Virginia, USA, October 27–31, 2008. ACM Press. (Cited on page 3.)

[HL10] Carmit Hazay and Yehuda Lindell. Efficient protocols for set intersection and pattern matching with security against malicious and covert adversaries. *Journal of Cryptology*, 23(3):422–456, July 2010. (Cited on page 3.)

[HN12] Carmit Hazay and Kobbi Nissim. Efficient set operations in the presence of malicious adversaries. *Journal of Cryptology*, 25(3):383–433, July 2012. (Cited on page 3.)

[HW06] Susan Hohenberger and Stephen A. Weis. Honest-verifier private disjointness testing without random oracles. In *Privacy Enhancing Technologies, 6th International Workshop, PET 2006, Cambridge, UK, June 28-30, 2006, Revised Selected Papers*, volume 4258 of *Lecture Notes in Computer Science*, pages 277–294. Springer, 2006. (Cited on page 3.)

[JL09] Stanislaw Jarecki and Xiaomin Liu. Efficient oblivious pseudorandom function with applications to adaptive OT and secure computation of set intersection. In Omer Reingold, editor, *TCC 2009: 6th Theory of Cryptography Conference*, volume 5444 of *Lecture Notes in Computer Science*, pages 577–594. Springer, Berlin, Germany, March 15–17, 2009. (Cited on page 3.)

[JL10] Stanislaw Jarecki and Xiaomin Liu. Fast secure computation of set intersection. In Juan A. Garay and Roberto De Prisco, editors, *SCN 10: 7th International Conference on Security in Communication Networks*, volume 6280 of *Lecture Notes in Computer Science*, pages 418–435, Amalfi, Italy, September 13–15, 2010. Springer, Berlin, Germany. (Cited on page 3.)

[Ker11] Florian Kerschbaum. Public-key encrypted bloom filters with applications to supply chain integrity. In *Data and Applications Security and Privacy XXV - 25th Annual IFIP WG 11.3 Conference, DBSec 2011, Richmond, VA, USA, July 11-13, 2011. Proceedings*, pages 60–75, 2011. (Cited on pages 1 and 3.)

[Ker12] Florian Kerschbaum. Outsourced private set intersection using homomorphic encryption. In Heung Youl Youm and Yoojae Won, editors, *ASIACCS 12: 7th Conference on Computer and Communications Security*, pages 85–86, Seoul, Korea, May 2–4, 2012. ACM Press. (Cited on page 3.)

[KM05] Aggelos Kiayias and Antonina Mitrofanova. Testing disjointness of private datasets. In Andrew Patrick and Moti Yung, editors, *FC 2005: 9th International Conference on Financial Cryptography and Data Security*, volume 3570 of *Lecture Notes in Computer Science*, pages 109–124, Roseau, The Commonwealth Of Dominica, February 28 – March 3, 2005. Springer, Berlin, Germany. (Cited on pages 2 and 3.)

[KMRS14] Seny Kamara, Payman Mohassel, Mariana Raykova, and Seyed Saeed Sadeghian. Scaling private set intersection to billion-element sets. In Nicolas Christin and Reihaneh Safavi-Naini, editors, *FC 2014: 18th International Conference on Financial Cryptography and Data Security*, volume 8437 of *Lecture Notes in Computer Science*, pages 195–215, Christ Church, Barbados, March 3–7, 2014. Springer, Berlin, Germany. (Cited on page 3.)

[KS05] Lea Kissner and Dawn Xiaodong Song. Privacy-preserving set operations. In Victor Shoup, editor, *Advances in Cryptology – CRYPTO 2005*, volume 3621 of *Lecture Notes in Computer Science*, pages 241–257, Santa Barbara, CA, USA, August 14–18, 2005. Springer, Berlin, Germany. (Cited on page 3.)

[MBD12] Dilip Many, Martin Burkhart, and Xenofontas Dimitropoulos. Technical Report TIK report no. 345, ETH Zurich, Switzerland, 2012. (Cited on page 4.)

[PSN10] Odysseas Papapetrou, Wolf Siberski, and Wolfgang Nejdl. Cardinality estimation and dynamic length adaptation for bloom filters. *Distributed and Parallel Databases*, 28(2-3):119–156, 2010. (Cited on page 2.)

[PSZ14] Benny Pinkas, Thomas Schneider, and Michael Zohner. Faster private set intersection based on OT extension. In *Proceedings of the 23rd USENIX Security Symposium, San Diego, CA, USA, August 20-22, 2014.*, pages 797–812. USENIX Association, 2014. (Cited on page 3.)

[TRL12] Sasu Tarkoma, Christian Esteve Rothenberg, and Eemil Lagerspetz. Theory and practice of bloom filters for distributed systems. *IEEE Communications Surveys and Tutorials*, 14(1):131–155, 2012. (Cited on pages 1, 5, and 6.)