PSImple: Practical Multiparty Maliciously-Secure Private Set Intersection

Abstract

Private set intersection (PSI) protocols allow a set of mutually distrustful parties, each holding a private set of items, to compute the intersection over all their sets, such that no other information is revealed. PSI has a wide variety of applications including online advertising (e.g., efficacy computation), security (e.g., botnet detection, intrusion detection), proximity testing (e.g., COVID-19 contact tracing), and more. PSI is a rapidly developing area and there exist many highly efficient protocols. However, almost all of these protocols are for the case of *two parties* or for *semi-honest* security. In particular, before our work, there has been no *concretely efficient*, *maliciously* secure *multiparty* PSI protocol.

We present *PSImple*, the first concretely efficient maliciously-secure multiparty PSI protocol. Our protocol is based on garbled Bloom filters, extending the 2-party PSI protocol of Rindal and Rosulek (Eurocrypt 2017) and the *semi-honestly* secure multiparty protocol of Inbar, Omri, and Pinkas (SCN 2018).

To demonstrate the practicality of the PSImple protocol, we implemented our protocol and ran experiments with on up to 32 parties and 2^{18} inputs. We incorporated several optimizations into our protocol, and compared our protocol with the 2-party protocol of Rindal and Rosulek and with the semi-honest protocol of Inbar et al.

Finally, we also revisit the parameters used in previous maliciously secure PSI works based on garbled Bloom filters. Using a more careful analysis, we show that the size of the garbled Bloom filters and the required number of oblivious transfers can be significantly reduced, often by more than 20%. These improved parameters can be used both in our protocol and in previous maliciously secure PSI protocols based on garbled Bloom filters.

1 Introduction

Private set intersection (PSI) protocols allow a set of mutually distrustful parties, each holding a private data set, to compute the intersection over all data sets. PSI has a wide variety of applications including online advertising (e.g., efficacy computation), security (e.g., botnet detection, intrusion detection), proximity testing (e.g., COVID-19 contact tracing), and more.

Indeed, PSI is a special case of secure multiparty computation (MPC), allowing a set of parties to compute some computational tasks over their private input, while guaranteeing several security properties, even in the face of adversarial behavior. Two of the most basic security properties are correctness and privacy, roughly requiring that the correct output is learned and that no other information is revealed. There exist two main adversarial models. *Semi-honest* adversaries are assumed to follow the prescribed protocol honestly, but may try to infer additional information seeing their view in the protocol execution. A more realistic adversarial model is that of *malicious* adversaries that may instruct the parties that they corrupt to deviate from the prescribed protocol in an arbitrary manner.

Our focus in this work is on the construction of *concretely efficient* PSI-tailored protocols (where by "concrete efficiency" we mean faster run-time in practice). It is instructive to note that a protocol may have very good asymptotic efficiency, but perform poorly in practical scenarios. This is usually due to extensive use of public-key operations, typically requiring exponentiation, which result in large constants. In particular, for 2-party malicious PSI protocols, Rindal and Rosulek [35] showed that the protocol of [9], which is based on Diffie-Helman, despite requiring significantly less communication, is more than an order of magnitude slower than their protocol, which is based on oblivious transfer[1] and garbled Bloom filters.

Concretely efficient *generic* MPC protocols (e.g., [1, 8, 14, 20, 37]) are less suitable for the PSI problem, as their complexity highly depends on the circuit size, which is large for PSI (typically, incurring a slowdown of two orders of magnitude, see [24]). Over the last decade, substantial research has

[1] While OT is based on public key operations, modern MPC protocols use OT *extension* [18], in which only a small amount of OTs require public key operations, and the rest are generated using symmetric-key primitives.

been dedicated to the construction of *concretely efficient* PSI protocols. However, these protocols were either restricted to the two-party setting (e.g., [9, 10, 21, 26, 30, 31, 33, 35, 36]) or were restricted to deal with semi-honest adversaries [17, 24].

1.1 Review of Related Previous Works

The PSImple protocol relies on two main primitives, oblivious transfer (OT) [32] and garbled Bloom filters (GBF) [10]. K-out-of-N oblivious transfer is a cryptographic primitive, allowing a receiver to interact with a sender, holding N strings $s_0, \dots, s_N$, such that the receiver learns some K of these strings, at its choice, but nothing else. The sender learns nothing (specifically, not which of the strings the receiver chose to learn). A Bloom filter (BF) [2] is a data structure used to encode a set S over some domain $\mathcal{D}$ of n elements as a Boolean array of length $N > n$. It is attributed with k hash functions $h_1, \dots, h_k$. An element $x \in \mathcal{D}$ is encoded into the BF by setting all indices $h_1(x), \dots, h_k(x)$ in the BF to be 1.

We next review the existing ideas for PSI protocols based on GBFs, starting with the two-party semi-honest construction of [10].

Two-party semi-honest PSI of [10]. Say that two parties $\mathsf{P}_0, \mathsf{P}_1$ wish to compute the intersection between their respective sets S_0 and S_1. Using the above primitive it is natural to consider the following idea. First, each party constructs the BF, according to its private set. Then, they engage in an OT protocol so that P_0 (as the receiver) learns the values from P_1's BF – only in the indices holding a 1 value in P_0's BF. Finally, by taking the bit-wise AND from both BFs, P_0 learned the BF of the intersection.

While the above protocol is correct, it is not secure, as P_0 may learn about 1 value indices in P_0's BF, even if they where set to one on account of elements that are not in the intersection (but are in P_1's set). To overcome this leakage, Dong et al. [10] introduced a variant of Bloom filters, called *garbled Bloom filters* (GBF). A GBF is attributed with same k hash functions as its respective BF. In each coordinate of the GBF there is a a σ long random string. The strings are chosen independently and uniformly, with the only requirement, that for any element $x \in \mathcal{D}$ in the underlying set, the XOR over all strings in indices $h_1(x), \dots, h_k(x)$ in the GBF equals some value y_x.

The protocol of [10] follows as before with the only difference that P_0 learns the desired coordinates (with value 1 in the Bloom filter of P_0) from the garbled Bloom filter of P_1. In the GBF variant proposed by [10], it is predetermined that $y_x = x$ for any x. Thus, given the appropriate strings, P_0 can test whether an element x is in the intersection by checking if the XOR over all strings in indices $h_1(x), \dots, h_k(x)$ (which it got from P_1's GBF) equals x. On the other hand, for any x' that does not belong to P_0's set, P_0 learns nothing but random and independent strings.

Two-party malicious PSI of [35]. The construction of [10] works only for semi-honest adversaries, as malicious parties can use an input set that is much larger than the allowed set size. Rindal and Rosulek [35] suggested an efficient translation to the malicious setting: First, to prevent P_0 from cheating, they needed to use a maliciously secure K-out-of-N OT protocol. Second, to prevent P_1 from cheating (pretending to have a larger set), they use the following idea. Rather than having a predetermined value y_x for every possible element x, the strings of the GBF are chosen uniformly and independently, and y_x is taken to be the result of these choices, i.e., y_x is simply the XOR over all strings in indices $h_1(x), \dots, h_k(x)$. Finally, after P_0 has learned t of the strings, P_1 needs to send to P_0 the values of y_x for every element x in its set. This, indeed, puts a limit on the number of elements P_1 can use. We note that in order for P_0 to find the intersection, given the (encoded) items of P_1, it is required to perform $\omega(N \log N)$ comparisons in the number of items in a data set (reduced from a quadratic number by sorting).

Unfortunately, there is no known concretely efficient K-out-of-N OT protocol with malicious security. Rindal and Rosulek [35] overcome this problem by constructing a concretely efficient, maliciously secure, *approximate random* oblivious transfer protocol. They show that this primitive suffices for the security of the above protocol, with a caveat that a malicious P_0 can use a slightly larger set than the bound for honest parties. For more details on this protocol, see Section 2.1.

Multiparty semi-honest PSI of [17]. Inbar et al. [17] extended the work of [10] to the multiparty setting for *augmented* semi-honest security.[3] To compute the intersection over a set of $t+1$ parties $\{\mathsf{P}_0, \mathsf{P}_1, \dots, \mathsf{P}_t\}$, they used the XOR secret sharing scheme. Specifically, each party P_i computes a GBF G_i for its set – using another variant of GBF, where $y_x = 0$ for every element x. Then, P_i shares its GBF G_i among all parties (making the share of each party P_j an independent uniform string s_j and making its own share the XOR of all other s_js with the GBF G_i, i.e., according to the XOR secret sharing scheme). Next each party locally XORs all the shares it got from all parties. It follows, by the GBF variant used, that the XOR all these shares is a valid GBF of the intersection. However, P_0 cannot learn this GBF, as it may extract information from it by removing the randomness incorporated by corrupt parties. Hence, P_0 uses a (semi-honest secure) OT to learn only the coordinates attributed to its subset. To compute the intersection, P_0 computes a cumulative GBF G^* by XORing all the GBFs it obtained from all other parties. Finally. P_0 outputs all elements x in its set, for which the XOR of the coordinates in G^* that are attributed with x is zero.

1.2 Contributions

Given the current state for concretely efficient PSI, the main problem we tackle in this work is:

Construct a concretely efficient multiparty protocol for computing private set intersection, secure against malicious adversaries, scaling well with the number of parties and with data set size.

Our main contributions can be summarized as follows.

1. We present *PSImple*, the first concretely efficient, multiparty PSI protocol that is secure against *malicious* adversaries, corrupting any subset of parties.

2. We implemented PSImple and incorporated several code optimizations. We ran experiments to show the practicality of PSImple, showing that PSImple is competitive even against similar protocols that are limited to 2-parties [35] or give a weaker security guarantee [17, 38].

3. We revisit the parameter analysis of previous works on efficient PSI based on garbled Bloom filters (GBF) in several ways. Performing a careful analysis, we are able to reduce the number of required oblivious transfer (OT) calls by up to 25%.

We next elaborate on each of these contributions.

The *PSImple* protocol – A multiparty PSI protocol in the malicious model. A key idea in the two-party malicious PSI protocol of [35] is to somehow "bind" each party to a restricted subset of the coordinates (of the computed GBF for the intersection) that will be correlated with the other party's GBF. For party P_0, this is obtained by the K-out-of-N OT it performs (as a receiver) with P_1. For P_1 the binding effect comes from the fact that P_1 can only send a fixed number of codewords to P_0.

Trying to combine the ideas from [35] and [17], we note that it is possible to let P_0 perform an (approximated, random) K-out-of-N OT with each of the other parties separately (i.e., the first part of the [35] protocol). After this phase, P_0 holds a garbled Bloom filter for every party, however, it does not know the appropriate codewords (i.e., y_xs). Obviously, the parties cannot just send the codewords to P_0, as P_0 is not allowed to learn the intersection of its set with the set any proper subset of the honest parties.

A possible idea for completing the protocol is to let the parties secret share their GBFs (using the XOR scheme) and then reveal the codewords of these XORed shares to P_0. This is indeed secure, and it allows P_0 to find the intersection by finding codewords, one from each party, that sum to its own codeword. However, all known algorithms for finding these codewords grow exponentially in the number of parties. Therefore, we take a different path, revisiting the construction of [35].

A new two-party malicious PSI. In our two-party malicious construction, the parties start in the same way as in [35], by P_0 (as the receiver) performing an (approximated, random) K-out-of-N OT with P_1 (as the sender), letting P_0 learn the appropriate parts in the GBF G_1 of P_1. As in the construction of [35], this binds P_0 to choose a bounded subset of coordinates from P_1's that may affect the output GBF. To similarly bind P_1, in a second phase, the parties switch roles and perform an (approximated, random) K-out-of-N OT with P_1 as the receiver and P_0 as the sender, letting P_1 learn the appropriate parts in the GBF G_0 of P_0.

Now, if P_0 XORs G_0 with its part of G_1 and also P_1 XORs G_1 with its part of G_0, then they would both hold GBFs that agree on the codewords of elements in the intersection, that is, if they XOR these two GBFs, then the result would be a GBF, where for every element x in the intersection, the codeword y_x is 0. However, P_1 cannot just send this GBF to P_0, as it may reveal additional information about elements in P_1's set that are not in the intersection.

To solve the above issue, we introduce the notion of *re-randomizing a GBF*. That is, given a GBF G for a set S, selecting a uniformly random GBF G' that agrees with G on all codewords for elements in S. One way to implement this operation is to XOR G with a random GBF for S with all codewords y_x being 0, for every $x \in S$. A more direct algorithm to rerandomize a GBF appears in Appendix C. We can thus complete the protocol by P_1 sending to P_0 a re-randomized version of the GBF it obtained.

We note that this alternative protocol more than doubles the communication complexity compared with [35], but reduces the number of comparisons to be linear in the size of each set. While this has little effect in the two-party case, in the case of more than two parties, the saving is drastic – in PSImple the number of comparisons remains linear in the size of P_0's set. In contrast, the above direct extension of [35] to the multiparty setting would require finding codewords (one from each party) that XOR to 0. To the best of our knowledge, the best solution to this problem is still exponential in the number of parties.

A new multiparty PSI protocol. In the two party construction, to impose on each party P_i a restriction on the size of the data set it uses when interacting with P_j, we let P_i act as the receiver in a (approximated, random) K-out-of-N OT execution with P_j. Since in the multiparty setting we allow any subset of the parties to be corrupted, it is natural to assume that it is necessary to have every pair of parties perform two executions of the K-out-of-N OT protocol (with the roles being reversed at each time). We prove, however, that it suffices for security to only have P_0 perform two executions of the approximated, random K-out-of-N OT protocol with each of the other parties.

Indeed, to generalize our two-party protocol to the multiparty case, we first let P_0 perform two approximated, random K-out-of-N OT execution with each party P_i. Then, P_0 XORs all $2t$ GBFs it obtained in these executions. Let G_0 be the resulting (cumulative) GBF of P_0. Similarly, let G_i be the GBF obtained by party P_i as the XOR of the two GBFs it saw in the interaction with P_0. As before, each P_i needs to rerandomize G_i before sending it to P_0. Let G_i^* be the rerandomized version of G_i.

It follows that $G_0 \oplus \bigoplus_{i\in[t]} G_i^*$ is a GBF for the intersection of all the parties, where the codeword for every element x in the intersection is the zero string. This is because any codeword for x that appeared in any of the original GBFs (say in the interaction of P_0 with party P_i), appears twice in the above summation, once for P_0 and once for P_i. So the idea is to allow P_0 to learn the above summation. However, if each P_i simply sends G_i^* to P_0, then security is breached as P_i can compute the intersection with each party separately. To overcome this, we let the parties first share their G_i^* in a t-out-of-t additive secret sharing scheme, and then locally sum all the shares they received. Finally, by sending the summed shares to P_0, they allow P_0 to reconstruct the GBF of the intersection $G_0 \oplus \bigoplus_{i\in[t]} G_i^*$, but nothing else.

Implementation, Code Optimizations, and Experiments. We implemented PSImple and incorporated several code optimizations that significantly reduced the communication and the required memory, and also allowed us to move much of the computation to the offline phase (i.e., can be done before the inputs are known to the parties). We ran experiments with 2 to 32 parties and input size of 2^8 to 2^{18}, in order to demonstrate the practicality of PSImple, and analyzed the runtime to understand the asymptotics and the cost of the various steps. We compare our results with existing protocols that are based on GBFs, in particular the 2-party maliciously secure PSI protocol of [35] and the multiparty PSI protocols of [17, 38] that give a significantly weaker security guarantee. As PSImple is specifically designed as a multiparty PSI protocol with malicious security, we expected PSImple to be significantly slower than these protocols. However, somewhat surprisingly, our experimental results show that PSImple is quite competitive and in some cases even faster.

Improved analysis and choice of parameters. The two-party maliciously-secure PSI protocol of [35] uses the cut-and-choose technique over N_{OT} executions of random oblivious transfer (ROT) – to allow two parties to jointly select a GBF of length $N_{\mathrm{BF}} < N_{\mathrm{OT}}$ of the intersection. Specifically, some $N_{cc} < N_{\mathrm{OT}} - N_{\mathrm{BF}}$ of the strings selected in the ROTs are revealed to prove honest behavior. Performing a more careful analysis of the Bloom filter and cut-and-choose parameters we are able to reduce the number of required ROTs N_{OT} by up to 25% compared to [35], as can be seen in Table 1. Since ROT is often the bottleneck in PSI protocols based on GBFs, this improvement has a great effect on the overall efficiency of these protocol. In particular, our analysis directly improves PSImple, as well as the protocols of [35, 38].

1.3 Additional Related Work

Currently, the state-of-the-art in two-party maliciously secure PSI are the protocols of [36] and [26], both concretely-efficient, have quasi-linear and linear communication complexities, respectively, and are almost as efficient as the fastest semi-honest PSI protocol [23]. The benchmarks made in [26] suggest that it is currently the fastest two-party, maliciously secure, PSI protocol. We remark that [26] uses a primitive called *PaXoS*, of which garbled Bloom filters is a special case. Following this work, it is interesting to see if the techniques of [26] can also be extended to the multiparty setting.

Apart from [17], an additional PSI protocol in the *semi-honest multiparty* setting is the protocol of [24], which is based on symmetric-key techniques. The protocol of [24] is significantly faster than [17] for a small amount of parties. However, it does not scale as well with the number of parties, and we do not know if it can be *efficiently* extended to the malicious setting.

Regarding maliciously-secure multiparty PSI protocols, the works of [15] and [11] both have very good asymptotic communication complexity. However, both of these protocols are not *concretely efficient* and, therefore, have not been implemented. We remark that [11] achieve a stronger security guarantee than PSImple and [15], because in the protocol of [11] all the parties output the intersection.

Zhang et al. [38] recently made an interesting attempt to build a concretely efficient maliciously secure protocol extending the protocol of [35]. However, their solution is in a non-standard security model, as it assumes that the adversary either does not corrupt P_0 or does not corrupt another designated party P_1. If these parties do collude, then the corrupt parties may learn the intersection of the Bloom filters of the honest parties. In particular, in the three party setting, this implies leaking the BF of the honest party. Furthermore, this leakage occurs even in the semi-honest setting. Hence, the security model they dealt with is significantly more relaxed than the standard malicious security model we assume.

Additionally, there is a line of work that is based on circuits [16, 27–29]. In [28], the authors managed to reduce the size of the circuit to linear in a number of items, vs. quadratic for the naïve solution and quasi-linear in the sorting solution of [16]. However, these works are for the semi-honest two-party case, and the techniques are not easily extendable.

2 Background and Definitions

In this section we give the necessary definitions and notations, and briefly describe the cryptographic primitives that we use in our protocol. The formal definitions of the corresponding functionalities appear in Appendix A. Additionally, we provide here a brief description of the PSI protocols of Rindal and Rosulek [35] and Inbar, Omri and Pinkas [17].

Notations. We denote the computational security parameter by σ, and the statistical security parameter by λ. In our implementation, $\sigma = 128$ and $\lambda = 40$. For $l \in \mathbb{N}$, $[l]$ denotes

n	2^8		2^{12}		2^{16}		2^{20}	
Protocol	Our MPSI	[35]	Our MPSI	[35]	Our MPSI	[35]	Our MPSI	[35]
k	147	94	134	94	131	91	129	90
N_{BF}	64,733	88,627	851,085	1,121,959	12,660,342	16,579,297	197,052,485	257,635,123
N_{OT}	74,379	99,372	901,106	1,187,141	12,948,963	16,992,857	198,793,103	260,252,093

Table 1: Comparison of our Π_{AppROT} parameters k, N_{BF} and N_{OT} with [35] for set size n, statistical security $\lambda = 40$, computational security $\sigma = 128$.

the set $\{1, \ldots, l\}$.

We use the notation $\mathcal{P} = (\mathsf{P}_0, \ldots, \mathsf{P}_t)$ for the set of parties, where P_0 is the evaluating party, and the remaining t parties are non-evaluating parties. The size of the input set of any honest party is bounded by n, and $\mathcal{D}$ is the domain of the input items.

Private Set Intersection. In a private set intersection protocol, a set of parties $\mathcal{P} = (\mathsf{P}_0, \ldots, \mathsf{P}_t)$, each having up to n items from domain $\mathcal{D}$ as their private inputs, compute the intersection of their input sets. As a result of the protocol, the evaluating party P_0 learns (only) the intersection of those sets, and all other parties learn nothing.

We assume a malicious adversary that may corrupt up to t parties (i.e., all parties but one). The adversary has full control over these parties, and may instruct them to arbitrarily deviate from the prescribed protocol. Following the real vs. ideal paradigm for proving security, our goal is to prove that such an adversary cannot do more harm than a very limited (ideal-world) adversary. In particular, the ideal-world adversary may only choose the inputs of the corrupted parties. We note that the size of these adversarial input sets n' could be slightly larger than the prescribed bound, i.e., $n' > n$ (it is possible to show that using our parameters n' must be smaller than $4n$). Indeed, this is less standard in secure computation, however, we inherit this assumption from the work of [35]. The ideal functionality $\mathcal{F}_{MPSI}$ is given in Appendix A.

Additive Secret Sharing. An additive secret sharing scheme enables a set of t parties to share a secret S such that no proper subset of them can learn any information about S (apart from its length). However, all t parties together are able to reconstruct the secret. Each party P_i receives a value S_i of length $|S|$, called P_i's *share* of S, such that $S = S_1 \oplus \ldots \oplus S_t$. To obtain such a sharing, $t-1$ of the shares can be selected uniformly at random, and the last share is set to be the XOR of these $t-1$ shares and the secret S.

Additive secret-sharing is linear, so in particular, given two additive sharings of secrets S_1 and S_2, parties can locally compute shares of the sum $S_1 \oplus S_2$ by XORing their own shares of S_1 and S_2.

Bloom Filters and Garbled Bloom Filters. A Bloom filter is a compact data structure [2] to store a set of items that allows efficient probabilistic membership testing. It consists of N_{BF} bits and associated with k independent random hash functions h_1,...,h_k: $\{0,1\}^* \to [N_{\text{BF}}]$. Initially, all the bits of the Bloom filter are set to 0. To add an item x to the Bloom filter, the bits at indices $h_1(x)$,...,$h_k(x)$ are set to 1 (regardless of whether their current value is 0 or 1).

If all the bits in the Bloom filter at indices $h_1(x)$,..., $h_k(x)$ equal 1, then this is interpreted as if x is a member of the set. Note that this might be a false positive result (i.e., x is misidentified as being represented in the Bloom filter), if other elements of the set turn the Bloom filter bits at indices $h_1(x)$,...,$h_k(x)$ to 1's.

We denote by p_{False} the false-positive probability for a given Bloom filter, i.e., the probability of a positive result for some randomly chosen item. This probability depends (apart from the length of the Bloom filter N_{BF} and the number of hash-functions k) on the number of items currently stored in the Bloom filter (more precisely, on the number on 1's in Bloom filter). The analysis of the false positive probability of a Bloom filter is less trivial than it may initially seem [3, 9, 13, 25]. In this paper we used the refined formula from [13].

Garbled Bloom filters (GBF) were introduced by Dong et al. [10] as the garbled version of a Bloom filter, obtained by expanding each bit in the original BF to a σ-long bit string. The compactness of the original Bloom filter is somewhat compromised in a GBF for the sake of obtaining an obliviousness property. As before, any element x is attributed with k coordinates in the GBF, i.e., the hash-values $h_1(x), \ldots, h_k(x)$. Intuitively, this obliviousness property means that for a given element x, it is impossible to learn anything on whether x is in the data set without querying the GBF on *all* k coordinates attributed to x. On the other hand, given the strings in all coordinates attributed with x, we compute the *codeword* y_x as follows.

$$y_x = \bigoplus_{i \in h_*(x)} \text{GBF}[i], \tag{1}$$

where $h_*(x) \stackrel{def}{=} \{h_j(x) : j \in [k]\}$. [2] If the GBF and x are given, and it is known what the codeword of x should be, then it is possible to check if x is in the GBF using Equation (1).

Fixing the length and hash functions of a Bloom filter, a set of items uniquely determines the associated Bloom filter. In contrast, there usually exist many distinct garbled Bloom

[2] We stress that the summation in (1) is performed over the set of indices without repetitions. Pinkas et al. [30] showed that the probability of a collision $h_i(x) = h_j(x)$ is noticeable, which may lead to the elimination of the corresponding GBF string from the sum.

filters for any given set, even if the codewords are fixed as well. Based on the GBF construction algorithm of Dong et al. [10], we construct an efficient algorithm to *rerandomize* a garbled Bloom filter G for a fixed set X. That is, to select a uniformly random GBF G' that agrees with G on the codewords of the elements X. For completeness, this algorithm is given in Appendix C.1.

Observe that if BF_1 and BF_2 are two Bloom filters (of the same length and using the same hash-functions) of sets X_1 and X_2, then $\mathrm{BF}_1 \wedge \mathrm{BF}_2$ (i.e, the bitwise AND of the arrays) is a Bloom filter of $X_1 \cap X_2$. Similarly, if GBF_1 and GBF_2 are two garbled Bloom filters (of the same length and using the same hash-functions) of sets X_1 and X_2, then $\mathrm{GBF}_1 \oplus \mathrm{GBF}_2$ is the GBF of $X_1 \cap X_2$, where codewords for any element is XOR of its codewords in Y_{X_1} and Y_{X_2}. In particular, if the equal items in X_1 and X_2 have the same codewords in GBF_1 and GBF_2, then all the items from $X_1 \cap X_2$ have all-zero codewords in $\mathrm{GBF}_1 \oplus \mathrm{GBF}_2$.

Random Oblivious Transfer. In a 1-out-of-2 random oblivious transfer (ROT) protocol there are two parties: a sender and a receiver. The private input of the receiver is its choice bit b, while the sender has no input. As a result of the ROT, the sender receives two random values: m_0 and m_1, and the receiver receives m_b. The sender learns no information about b, and the receiver learns nothing about m_{1-b}.

In a K-out-of-N random OT protocol, the private input of the receiver is its set of choice indices of size K, denoted by $J = \{j_1, ..., j_K\}$, where $j_i \in [N]$, $i \in [K]$, while the sender has no input. As a result of the K-out-of-N ROT, the sender receives N random values: $M = \{m_1, ..., m_N\}$, and the receiver receives only the values indexed by J, namely $M_J = \{m_{j_1}, ..., m_{j_K}\}$. The sender learns no information about I, and the receiver learns nothing about $M \setminus M_J$.

Cut-and-Choose. Cut-and-choose is a common technique to ensure that secret data has been constructed according to an agreed method. The high-level idea is that after the secret data has been created, a random part of the data is opened and checked. If the checked part has been constructed honestly, the rest of the data, which remains secret, is assumed to be constructed honestly as well, and used in the protocol. Note that this implies that the amount of secret data initially generated needs to be larger than the required secret data needed for the protocol.

2.1 The Two-Party Protocol of Rindal and Rosulek [35]

The starting point of our protocol is the maliciously secure two-party PSI protocol of Rindal and Rosulek [35]. We describe the main ideas of their protocol here, as they are important for understanding our protocol.

The high-level idea of [35] for finding the intersection between the data sets of two parties is to let the parties construct GBFs that agree on the codewords of their joint items. This can be achieved using a K-out-of-N random OT protocol, by allowing P_0 to learn K of the N random strings that P_1 learns. P_0 then chooses a permutation over $[N]$ that relocates these K strings in indices that are equal 1 in P_0's BF. To turn these N (permuted) strings into a GBF, all P_1 needs to do is compute the resulting codewords attributed with its elements (i.e., to obtain the codeword for an element x, it computes $h_1(x) \oplus, \ldots, h_k(x)$). Finally, P_1 sends these codewords to P_0.

Approximate K-out-of-N Random OT Π_{AppROT}. Unfortunately, to the best of our knowledge, no concretely-efficient maliciously-secure K-out-of-N OT protocol exists. Instead, [35] implement an *approximate* K-out-of-N Random OT, allowing the receiver to request slightly more than K strings. Rindal and Rosulek [35] show that their PSI protocol remains secure with a proper choice of parameters, which guarantee that the false-positive probability (p_{False}) of the resulting Bloom filter is still negligible. This, in turn, means that it is impossible for the adversary to test the intersection for items that were not part of its input set.

The approximate K-out-of-N subprotocol of Rindal and Rosulek, which we denote by Π_{AppROT} (formally described in Figure 1), implements the functionality $\mathcal{F}_{AppROT}$, appearing in Appendix A. We next give a rough overview of the Π_{AppROT} protocol. In the first phase, the two parties invoke N_{OT} parallel executions of a maliciously secure two-party 1-out-of-2 random OT, where in a known fraction of these execution the receiver P_0 requests to learn the string m_1, and in all others to learn m_0. In the second phase, the parties use the cut-and-choose technique to verify that P_0 behaved honestly. Specifically, the sender P_1 asks P_0 to reveal a random subset of its choices and verifies that the right fraction of 0-string choices appear. If not, P_1 aborts the execution.

Finally, the unrevealed choice strings are reordered by P_0 so that they are attributed to its desired locations. Because the cut-and choose set is chosen by the sender, the receiver initially forms the requests sequence at random. Therefore, the reordering at the final stage is necessary. The full Rindal and Rosulek's PSI protocol is given in Appendix B.

Remark. We mention that in the PSI protocol of [35] the possible input set size of the adversary n' may be larger than n, and the PSImple protocol inherits this property. This happens because in the cut-and-choose check only the number of 1's in the Bloom filter is bounded, but not n itself. It is shown in [35] that $n' < 2N_{\mathrm{BF}}/\sigma$ (the authors stress that this is a very rough bound given for the worst case); we refer the reader to [35] for the detailed analysis.

We note that with the choice of parameters in [35], $N_{\mathrm{BF}} < 3n\sigma$, so $n' < 6n$. With our choice of parameters , since N_{BF} is comparatively lower (see Tables 1 & 4), $N_{\mathrm{BF}} < 1.73n\sigma$ so

Protocol Π_{AppROT}

Parties: A sender and a receiver.
Inputs (used only in the online phase): The receiver inputs its choice bit-array B; the sender has no input.
Offline phase $\Pi^{Offline}_{AppROT}$:

1. **[random OTs]** The sender performs N_{OT} random OTs with the receiver. The receiver chooses bits $c_1,...,c_{N_{\text{OT}}}$ with N^1_{OT}s '1's among them, and $N_{\text{OT}} - N^1_{\text{OT}}$ '0's (randomly permuted). As a result, in the jth ROT, the sender learns random strings m_{j0}, m_{j1} (of length σ) chosen by the functionality $\mathcal{F}_{ROT}$, while the receiver uses its choice bit c_j and learns $m_{j*} = m_{jc_j}$.
2. **[cut-and-choose challenge]** The sender randomly chooses the set $C \subseteq [N_{\text{OT}}]$ of size N_{cc} and sends C to the receiver.
3. **[cut-and-choose response]** The receiver checks that $|C| = N_{cc}$, then computes and sends to the sender the set $R = \{j \in C | c_j = 0\}$. To prove that it used the choice bit '0' in the OTs indexed by R, it also sends $r^* = \bigoplus_{j\in R} m_{j*}$. The sender aborts if $r^* \neq \bigoplus_{j\in R} m_{j0}$ or if $|C| - |R| > N_{maxones}$, where $N_{maxones}$ is the maximal number of '1's allowed in the cut-and-choose OTs.

Online phase Π^{Online}_{AppROT}:

4. **[permute unopened OTs]** The receiver chooses a random injective function $\pi : [|B|] \to ([N_{\text{OT}}] \setminus C)$ such that $B[j] = c_{\pi(j)}$, and sends π to the sender.

   The receiver permutes its random values m_{j*} according the π, and the sender permutes m_{j1} according to π.

Outputs: The receiver outputs $M_* = \{m_{j*}\}_{j\in[|B|]}$; the sender outputs $M = \{m_{j1}\}_{j\in[|B|]}$.

Figure 1: Protocol Π_{AppROT}

$n' < 3.46n$. Thus, it seems that our improved parameters also give a slightly better security guarantee.

2.2 The Semi-Honest Multiparty PSI Protocol of Inbar et al. [17]

We briefly review the *augmented* semi-honest secure multiparty PSI protocol of Inbar et al. [17].[3] Using a semi-honest K-out-of-N OT, the evaluating party P_0 might receive the appropriate K coordinates from the GBFs of all other parties (such that the codeword of any encoded item is 0). The difficulty is that if each party now sends its GBF to P_0, then P_0 would not only recover the intersection of all the parties' inputs, but also the intersection of its input set with the input set of each other party separately.

To make sure P_0 learns the intersection and nothing else, each non-evaluating party begins by additively sharing its GBF among all other parties. The *cumulative GBF*, i.e., the sum of all the shares P_i got, is then used as the inputs the sender P_i in a K-out-of-N OT interaction with P_0 as the receiver. Note that the XOR of all these cumulative GBFs is a GBF of the intersection of all parties but P_0 (this follows by the linear secret sharing and the fact that all codewords equal 0). However, as P_0 only selected the K coordinates in accordance with its input set, by summing (XORing) all the GBFs it received, P_0 computes the GBF of the intersection of the sets of all parties, including P_0 itself. Note that this protocol is not secure in the malicious setting as there is no mechanism to prevent the parties from using all-ones Bloom filters, which represents the entire domain of the items.

[3] An augmented semi-honest adversary is an adversary that may choose to change its input, but then follows the prescribed protocol honestly. We note that the semi-honest secure protocol of [17] is less efficient as every pair of parties need to perform a K-out-of-N OT protocol.

3 The PSImple Protocol

In this Section we explain in detail our multiparty maliciously-secure PSI protocol, PSImple, and the underlying techniques we use. As mentioned above, we build on the techniques of [35] and [17] (both works, in turn, extend [10]).

As a warm up, we first describe in Section 3.1 the protocol for the two-party case. We do not suggest to use PSImple in the two-party scenario, because it is less efficient than the protocol of [35].[4] For more than two parties, however, PSImple is the only concretely efficient PSI protocol secure against malicious adversaries. The full fledged multiparty PSImple protocol is described in Section 3.2. The formal description of the PSImple protocol appears in Figure 2.

Before moving on to describe the protocol, let us first consider what may seem as the direct way to extend the protocol of [35] to the multiparty case, using the ideas of [17], and why it does not work for us. The idea is to have P_0 to perform Π_{AppROT} with each other party P_i independently and then have P_0 XOR all the GBFs received from the Π_{AppROT}'s to compute its cumulative GBF, denote it by G^*.

Recall that the codeword for an element x with respect to a GBF G with hash functions $h_1,\dots,h_k$ is the XOR over the strings in coordinates $h_1(x),\dots,h_k(x)$. Now, if each party P_i

[4] This is because our protocol uses two instances of Π_{AppROT}, which is the heaviest part of the protocol in terms of communication complexity, whereas the protocol of Rindal and Rosulek uses only a single instance of Π_{AppROT}.

sends to P_0 the codewords attributed to its set X_i and its GBF, then P_0 can compute the intersection of all parties, however, it can also compute the intersection with party separately, which is not allowed. To avoid this leakage, each P_i can additively share the its GBF among all parties P_j ($j \in [t]$), and then compute the cumulative GBF G_i^* as the XOR of all the shares it holds. After that, each party P_i sends to P_0 the codewords attributed to its set X_i and its cumulative GBF G_i^*. Finally, P_0 concludes that an item x in its input set with codeword y_x is in the intersection, if there exist codewords $y^1, \ldots, y^t$ received from $\mathsf{P}_1, \ldots, \mathsf{P}_t$, respectively, such that $y_x = y^1 \oplus \ldots \oplus y^t$.

The above is indeed correct and secure. However, an exhaustive search for a combination of codewords that sum to y_x grows exponentially with number of parties, and we do not know of any solution that is not exponential in the number of parties.

3.1 PSImple, Two-Party Case

One of the key points of the PSI protocol of [35] protocol is in some sense to "bind" each of the two parties to a Bloom filter of a restricted size set. By this we mean that there is a stage in the protocol in which each party must choose a limited number of coordinates (of the resulting GBF) that may become correlated with the BF of the other party, whereas all other coordinates remain independent of the other party's BF. It is important to note that these choices are made before the party learns any meaningful information in the protocol. In the protocol of [35], such a binding is achieved for P_0 by participating in Π_{AppROT} as the receiver. The binding for P_1 is achieved when it sends the codewords that correspond to its elements to P_0.

As explained above, when moving to the multiparty setting, the amount of work done by P_0 to find a sum of these codewords that match its own grows exponentially in the number of parties. Thus, one of the key points of PSimple is to achieve this binding without sending the codewords. To this end, the parties execute a second instance of Π_{AppROT}, with the parties playing reversed roles. In this way, the binding of P_1 is achieved similarly to the binding of P_0.

As a result, each party P_i receives two garbled Bloom filters, one from each execution of Π_{AppROT}. Put differently, P_i holds its own full GBF, and the k coordinates it chose from the GBF of P_{1-i} (padded with random strings to complete a GBF). By XORing these GBFs locally, P_i obtains the cumulative garbled Bloom filter GBF^i. It follows that $\mathrm{GBF}^{\mathrm{IS}} = \mathrm{GBF}^0 \oplus \mathrm{GBF}^1$ (i.e., the XOR of the two cumulative GBFs) is a GBF that has the zero string on all coordinates that where chosen by both P_0 as a receiver and P_1 as a receiver, and has a random string in all other coordinates.

On the positive side, we have that for any element x in the intersection, it holds that the codeword of x with respect to $\mathrm{GBF}^{\mathrm{IS}}$ is 0. Thus, the intersection could now be reconstructed as follows: P_1 sends GBF^1 to P_0, who concludes that $x \in X_0$ is in the intersection if x has the 0-codeword in $\mathrm{GBF}^{\mathrm{IS}}$. On the negative side, however, this method is insecure, as it allows P_0 to identify coordinates that were queried by P_1, even if they are not coordinates of an element in the intersection. This occurs if both parties choose the same coordinate s, as in this case $\mathrm{GBF}^{\mathrm{IS}}[s] = 0$.

To avoid this, P_1 rerandomizes its cumulative GBF. Denote the result by GBF^{1*}. Recall that the codewords of GBF^{1*} are equal to those of GBF^1 for items in the set X_1, but there is no longer a connection between the individual indices $\mathrm{GBF}^0[s]$ and $\mathrm{GBF}^{1*}[s]$, for any s.

Next, P_1 sends GBF^{1*} to P_0. Since rerandomization does not affect the codewords, it follows that if x is in $X_0 \cap X_1$, then it has the same codeword in both GBF^0 and GBF^{1*}. In other words, $\mathrm{GBF}^* = \mathrm{GBF}^0 \oplus \mathrm{GBF}^{1*}$ is a garbled Bloom filter of the set $X_1 \cap X_2$, in which the codewords of the items are equal to zero. Therefore, P_0 can check, for each item x_{0j} in its input, if x_{0j} is in the intersection, by testing $\bigoplus_{s \in h_*(x_{0j})} \mathrm{GBF}^*[s] = 0$.

3.2 PSImple, Multiparty Case

In this section we explain how to extend PSImple to the multiparty setting. Similarly to the two-party case, the parties achieve a "binding" of each party to its Bloom filter by executing Π_{AppROT}. Initially, it would seem that each party needs to perform two instances of Π_{AppROT}, in reverse roles, with each other party. However, we show in the proof, that to achieve this binding, it suffices that each party only performs two instances of Π_{AppROT} with P_0.

After the executions of Π_{AppROT}, the protocol proceeds as in the 2-party case, with each party XORing the garbled Bloom filters it received from its executions of Π_{AppROT}, and rerandomizing them. Recall that the rerandomization operation is done to hide coinciding requested coordinates in the executions of Π_{AppROT}, while preserving the property that codewords for joint elements are equal.

Let GBF^0 be the cumulative GBF of P_0, i.e., the sum of all the $2t$ GBFs it saw in its $2t$ interactions in Π_{AppROT}. Let GBF^{i*} be the rerandomized version of the cumulative GBF obtained by P_i in the two interactions of Π_{AppROT} it had with P_0. The idea is to let P_0 learn $\mathrm{GBF}^* = \mathrm{GBF}^0 \oplus_{i \in [t]} \mathrm{GBF}^{i*}$, which corresponds to a garbled Bloom filter of the intersection of all parties, with all-zero codewords. Then, P_0 would be able to compute the intersection similarly to the two-party case: For each element x_{0j} of its input set, it outputs x_{0j} as the member of the intersection, if $\bigoplus_{s \in h_*(x_{0j})} \mathrm{GBF}^*[s] = 0$.

However, we cannot simply let each party P_i send GBF^{i*} to P_0 as in the 2-party case, since this would be identical to t independent 2-party PSImple executions. Hence, P_0 would be able to recover its intersection with each party P_i independently, which is not secure.

To avoid this, the parties first additively share their GBFs and let P_0 reconstruct the sum. From the linear property of additive secret-sharing, it follows that P_0 recovers the sum of these GBFs, i.e., GBF*, and from the secrecy property it follows that P_0 learns nothing but GBF*.

The PSImple multi-party protocol is described formally in Figure 2. Following the offline/online paradigm, we divide our MPSI protocol Π_{MPSI} into two phases: an offline-phase $\Pi_{MPSI}^{Offline}$, which can be executed by the parties before they know their inputs, and an online-phase Π_{MPSI}^{Online}, which is executed after the parties learn their inputs.

Asymmetric Set Sizes. In the PSImple description above, we considered n as the exact set size for all the honest parties. However, n should be treated as an upper bound on set sizes, allowing honest parties to have only $n_i \leq n$ items. To this end, each party P_i computes its Bloom filter BF_i from its input set X_i and $n - n_i$ additional random dummy items, to perform Π_{AppROT}'s. This way, the behaviour of P_i with n_i items is indistinguishable (from the point of view of the adversary) from its behavior with n items. In the rerandomization step, P_i uses its n_i-elements to compute GBF^i (dummy items are not treated as items in the rerandomization procedure). As shown in the proof of Theorem 1 (App. F), the withdrawal of items while computing GBF^{i*} doesn't affect the view of the adversary.

3.3 Security and Correctness

In proving the security of a protocol via the real vs. ideal paradigm (specifically, in the UC model, as we do in this work), there is no separation into security properties, and in particular, privacy and security follow immediately. In Appendix F, we provide a complete proof for the security of the PSImple protocol, proving the following theorem.

Theorem 1. *Assume oblivious transfer protocols exist. Then, the Π_{MPSI} protocol of Figure 2 securely realizes the functionality $\mathcal{F}_{MPSI}$ with computational UC-security with abort in presence of a static, non-uniform computationally bounded, malicious adversary $\mathcal{A}$ corrupting any number of parties in the random oracle model, where the Bloom filter hash functions are modeled as (non-programmable) random oracles, and the other protocol parameters are chosen as described in subsection 4.*

We next sketch the ideas behind the proof, referring to it correctness and privacy separately.

Correctness. Our goal here is to prove that in an honest execution of the protocol the output of P_0 is indeed the intersection. Let x be an item in the intersection of all sets. It follows by construction, as previously explained, that P_0 outputs x as part of the intersection. Specifically, for every $i \in [t]$ in the interactions between P_0 and P_i, both parties are going to request the coordinates attributed with x from the other party. Thus, both codewords for x (for both P_0 and P_i) are going to be summed into the cumulative GBF of each of them. In addition, P_i will still keep these codewords in the rerandomization process. Finally, as all GBFs are XORed by P_0, these all codeword will cancel out.

If x is not in the intersection, then there exists a party P_i for $i \in [t] \cup \{0\}$ whose set does not contain x. Thus, except for the overwhelmingly small probability of a false positive in the underlying BF (i.e., probability p_{False}), in the OT interaction as a receiver P_i is not going to request all coordinates for x. Thus, the codeword for x from this interaction for P_i will be a completely independent uniform σ-long string. Hence, the probability that x is in the output is $2^{-\sigma}$, which is negligible. A proof of the consistency appears in Appendix F.1.

Malicious Security of Π_{MPSI}. Proving the security of MPC protocols is a delicate task, requiring a rigorous analysis. We next present a very high level overview of the security proof of our protocol. The main goal of the proof is to construct a simulator for the adversary, i.e., an ideal world adversary that interacts with the honest parties via the ideal PSI functionality and simulates a view that is closely distributed to the view real-world adversary in an execution of the protocol.

The first challenge of the simulator is to extract the effective inputs of the corrupted parties (i.e., the inputs that they actually use). To this end, we use the fact the hash functions are random oracles. Once this is done, the simulator can go to the ideal functionality with the intersection of all sets of malicious parties (this is possible, as the ideal adversary is allowed to change the inputs of corrupted parties). In addition all OT interactions with the malicious parties, the simulator acts as honest parties would, holding arbitrarily chosen inputs. Finally, as the simulator receives the output intersection from the functionality (in the case P_0 is corrupt), all the simulator needs to do is to send GBF shares that are consistent with this output and with all the random shares that the corrupted parties have seen so far. By the properties of secret sharing schemes, this is indeed possible.

3.4 Asymptotic Complexity

In Table 2, we compare the communication complexity of PSImple with that of the multiparty PSI protocols of [11, 15, 17, 22]. The distribution of the communication complexity in the offline and online phases is given in Table 3.

We observe that the overall communication complexity is asymptotically almost the same as [17], which is only semi-honestly secure. The communication complexity is slightly worse than in [15] and [11]. However, we recall that these protocols are not concretely efficient. Additionally, PSImple scales somewhat better with respect to the number of parties.

We note that the workload in PSImple is not balanced: the majority of communication is with the evaluating party P_0, while for each other party it is t times less.[5] A detailed

Protocol of Malicious-secure Multiparty PSI Π_{MPSI}

Parameters:
σ - computational security parameter; λ - statistical security parameter; N_{BF} - size of the Bloom filter;
$N_{\mathrm{OT}} > N_{\mathrm{BF}}$ - number of random OTs to perform; N^1_{OT}, N_{cc}, $N_{maxones}$ – parameters for Π_{AppROT} computed as in Sec. 4.
Inputs: Each party P_i, $i \in \{0,...,t\}$, inputs its set of items $X_i = \{x_{i1}, x_{i2}, ..., x_{in_i}\}$, $n_i \leq n$, $x_{ij} \in \mathcal{D}$.

Offline-phase $\Pi^{Offline}_{MPSI}$:

1. **[hash seeds agreement]**
   Parties run a coin-tossing protocol to agree on random hash-functions h_1, h_2, ... ,h_k: $\{0,1\} \to [N_{\mathrm{BF}}]$.
2. **[symmetric approximate ROT-offline]** Parties perform in parallel (with parameters N_{OT}, N^1_{OT}, N_{cc}, and $N_{maxones}$):
   (a) P_0 as a receiver performs $\Pi^{Offline}_{AppROT}$ with each P_i, $i \in [t]$.
   (b) Each P_i, $i \in [t]$, as a receiver performs $\Pi^{Offline}_{AppROT}$ with P_0.
3. **[random shares]** Each P_i, $i \in [t]$, sends $S^{il} = (s^{il}_1, ..., s^{il}_{N_{\mathrm{BF}}})$ to any P_l, $l \in [t] \setminus \{i\}$, where $s^{il}_r \xleftarrow{R} \{0,1\}^\sigma$, $r \in [N_{\mathrm{BF}}]$.

Online-phase Π^{Online}_{MPSI}:

4. **[compute Bloom filters]** Each party P_i, $i \in [t] \cup \{0\}$, locally computes the Bloom filter BF_i of its input set X_i. If $n_i < n$, then P_i computes the Bloom filter of the joint set X_i with $(n - n_i)$ random dummy items.
5. **[symmetric approximate ROT-online]**
   (a) Using BF_0 as its input, P_0 performs Π^{Online}_{AppROT} with every other party to finish Π_{AppROT}s started on Step 2a. As a result, it receives t arrays M^i_*, P_i learns M^i, where M^i_* and M^i are N_{BF}-size arrays of σ-bit values.
   (b) Using BF_i as its input, every party P_i performs Π^{Online}_{AppROT} with P_0 to finish Π_{AppROT}s started on Step 2b. As a result, P_i learns $\hat{M}^i_*$, and P_0 receives $\hat{M}^i$'s, where $\hat{M}^i$ and $\hat{M}^i_*$ are N_{BF}-size arrays of σ-bit values.
   (c) P_0 computes $\mathrm{GBF}^0 = \bigoplus_{i\in[t]} \left(M^i_* \oplus \hat{M}^i\right)$. Each P_i, $i \in [t]$, computes $\mathrm{GBF}^i = M^i \oplus \hat{M}^i_*$.
6. **[re-randomize GBFs]** Each P_i locally re-randomizes its garbled Bloom filter GBF^i for items and corresponding codewords only from X_i (without dummy items) (Algorithm **ReRandGBF** C.1) .
7. **[secret-sharing of GBFs]** Each P_i, $i \in [t]$, locally computes
   $$\mathrm{GBF}^{i*} = \mathrm{GBF}^i \bigoplus_{l\in[t]\setminus\{i\}} \left[S^{li} \oplus S^{il}\right]$$
   and sends GBF^{i*} to P_0.
8. **[reconstructing the GBF of the intersection]** P_0 computes $\mathrm{GBF}^* = \bigoplus_{i\in[t]} \mathrm{GBF}^{i*} \bigoplus \mathrm{GBF}^0$. Recall that this corresponds to a GBF of the intersection with codewords 0 for all items in the intersection.
9. **[output]** For each $x_{0j} \in X_0$, P_0 outputs x_{0j} as a member of the intersection, if
   $$\bigoplus_{r\in h_*(x_{0j})} \mathrm{GBF}^*[r] = 0.$$

Figure 2: The PSImple Multiparty protocol

Protocol	Communication complexity	
	overall	P_0
semi-honest security		
IOP18 [17]	$O(tn\sigma k)$	$O(tn\sigma k)$
malicious security		
KS05 [22]	$O(t^3 n\sigma)$	$O(t^2 n\sigma)$
HV17 [15]	$O(t^3\sigma + tn\sigma\log(n))$	$O(t^2\sigma + tn\sigma\log(n))$
GN19 [11]	$O(t^3\sigma + tn\sigma)$	$O(t^2\sigma + tn\sigma)$
PSImple	$O(tn\sigma^2 + tn\sigma\log(n\sigma))$ [5]	$O(tn\sigma^2 + tn\sigma\log(n\sigma))$

Table 2: Comparison of communication complexity.

Party	offline	online
P_0	$O(tn\sigma^2)$	$O(tn\sigma(\log(n\sigma)+\sigma))$
P_i	$O(n\sigma^2)$[5]	$O(n\sigma(\log(n\sigma)+\sigma))$

Table 3: Theoretical communication complexity of PSImple in offline and online phases.

analysis of the asymptotic communication and compuation complexity of PSImple is given in Appendix G.

4 Protocol Parameters

In this section, we revisit the parameter analysis of [35] for the parameters of Π_{AppROT}. We show that the size of the garbled Bloom filters and the number of required OTs can, in some cases, be reduced by 23-25% (see Table 1). These improved parameters can used in our protocol, as well as in previous PSI protocols based on GBFs and cut-and-choose such as [35,38].

The first difference from the analysis of [35] is the following: The number of OTs depends on the number of 1's that would be necessary to build the Bloom filter of the receiver. For n items in the input set of the receiver, with k hash-functions, the upper bound of required 1's is nk (k indices per each of n items), which is sufficient even if each item has a separate set of hash-indices. However, the probability of collisions is quite high, and the number of 1's in a Bloom filter has a Poisson distribution with very low deviation. Thus, instead of requiring a sufficient number of 1's after the cut-and-choose to build any Bloom filter, as done in [35], we require this number to be sufficient to build almost all Bloom filters (this implies that the GBF can later be constructed in the protocol with overwhelming probability.) This change significantly reduces the total number of required 1's from the OT, and consequently, the total number of required OTs.

The second difference from the analysis of [35] is technical: in [35], the sender chooses bits to check with the probability p_{chk}, whereas in our version of Π_{AppROT}, the size of the checked set is deterministic. Fixing the size of the checked set simplifies the protocol instructions and the simulation in the security proof.

Below we give a brief explanation about the restrictions which allow us to build the optimization problem, and the algorithm for the calculating the parameters. Additionally, we give the optimal parameters for several input sizes, and compare them with the parameters used in [35].

GBF parameters:
N_{OT} – number of ROTs in Π_{AppROT};
N_{BF} – size of the Bloom filter of the receiver;
N^1_{OT} – number of ones that an honest receiver should have among N_{OT} choice bits;
k – number of Bloom filter hash functions;
N_{cc} – number of bits to choose for the cut-and-choose check;
$N_{maxones}$ – the maximal number of 1's among the N_{cc} choice bits allowed in order to pass the cut-and-choose check.

As before, σ is the computational security parameter and λ is the statistical security parameter. Informally, we can formulate the parameter requirements as follows:

- *After the cut-and-choose, the receiver has enough ones and zeroes to build the Bloom filter.*
- *A malicious receiver has too few ones to find a false positive.*
- *An honest receiver passes the cut-and-choose check with overwhelming probability.*

Recall that p_{False} is the probability of a false-positive in the Bloom filter of the receiver. The second condition requires that p_{False} is negligible, even if the receiver is malicious. I.e. that $Pr[p_{False} \geq 2^{-\sigma}] \leq 2^{-\lambda}$.

Fixing n, σ and λ, we have to set k, N_{BF}, N_{cc}, N_{OT}, $N_{maxones}$ and N^1_{OT}. As we have three conditions and six variables, three of parameters are free. It is reasonable to take k and N_{BF} free and find the values of the other parameters that minimize N_{OT}, because the number of random OT is the heaviest part of the protocol.[6]

We prove that for any positive n, σ and λ there exist k, N_{BF}, N_{cc}, N_{OT}, $N_{maxones}$ and N^1_{OT} that meet the above requirements, and construct an algorithm to find the optimal parameter values. Based on the view of constrains, the feasible region of the parameters is bounded from below by k and N_{BF}, and the minimum of N_{OT} is located near their minimum values. For reasons of the guaranteed existence of a solution for $k > \sigma$ and that $N_{\mathrm{BF}} = O(nk)$, we heuristically adopted the search boundaries $k_{min} = \sigma$, $k_{max} = 2\sigma$, $N_{\mathrm{BF,min}} = nk$ and $N_{\mathrm{BF,max}} = 3nk$. The algorithm then works by going over all the possible values of N_{BF} and k in this region and taking the parameters which result in the minimal N_{OT}. The full parameter analysis and a formal description of the algorithm appears in Appendix D.

[5] With a standard optimization of generating the secret shares from preshared random seeds (see Section 5). Without it, the communication complexity for P_i is $O(tn\sigma^2 + n\sigma\log(n\sigma))$, and the overall communication complexity is $O(t^2 n\sigma^2 + tn\sigma\log(n\sigma))$.

[6] We note that for the online-phase, N_{BF} is more critical. However, trying to optimize N_{BF} results in a very poor N_{OT}. More details can be found in Appendix D.

Parameters	$n=2^8$	$n=2^{10}$	$n=2^{12}$	$n=2^{14}$	$n=2^{16}$	$n=2^{18}$	$n=2^{20}$
k	147	139	134	132	131	130	129
N_{BF}	64,733	229,055	851,085	3,253,782	12,660,342	49,786,942	197,052,485
N_{OT}	74,379	250,684	901,106	3,373,092	12,948,963	50,491,817	198,793,103
N_{OT}^1	32,885	116,132	428,425	1,637,989	6,376,614	25,026,621	98,728,744
N_{cc}	7,473	17,748	42,882	105,863	262,924	655,322	1,644,397
$N_{maxones}$	3,627	8,719	21,160	52,620	131,385	327,830	821,450

Table 4: Optimal (in N_{OT}) Π_{AppROT} parameters for set size n, statistical security $\lambda = 40$, and computational security $\sigma = 128$.

Running the constructed algorithm with $\lambda = 40$, $\sigma = 128$, we obtained the parameters presented in Table 4. A comparison with the parameters used in [35] is given in Table 1.

5 Implementation, Code Optimizations, and Experimental Results

We wrote our code in C++, using the LibOTe library [34] for the cryptographic primitives and the maliciously secure OT extension of Keller et al. [19]. The hash functions are computed using fixed-key AES and taking modulus.[7] We separate the protocol into two phases: an offline phase, which can be run before the parties know their inputs, and an online phase, which is run after the parties know their inputs. In many scenarios, the parties can communicate much before they require to find the intersection. In such cases, it is often preferable that the online time is as short as possible, while the offline phase can be significantly longer.

We have made the following code optimizations: As suggested by Hazai et al. [14], we have performed the additive secret-sharing in the offline phase, and using seeds.[8] This way, generating shares can be done locally. Additionally, we have moved memory allocation to the offline phase and reduced the required amount of memory by XORing results directly into the cumulative GBF on the fly. As we executed the experiments on machines with only 2 cores (see below), we chose to use a single thread for each instance of AppROT, which implies that the total number of threads in P_0 is double the number of parties; for stronger machines, it makes sense to use more threads per party. Our code will be available on Github.

To benchmark PSImple, we ran experiments on Amazon Web Server using t3.xlarge machines (2 cores and 16 GB RAM) with Unix OS, running on a LAN network with 1ms latency and 5Gb bandwidth. We tested the protocol with 2-32 parties, and with input size of 2^8-2^{18} per party. The results are given in Table 5. We observed that for large inputs and large amount of parties the code crashed due to insufficient memory. Thus, we plan to rerun the experiments on stronger machines.

parties	$\mathbf{n=2^8}$	$\mathbf{n=2^{10}}$	$\mathbf{n=2^{12}}$	$\mathbf{n=2^{14}}$	$\mathbf{n=2^{16}}$	$\mathbf{n=2^{18}}$
2	0.315	0.466	0.956	3.052	10.717	42.701
4	0.558	0.736	1.310	3.891	14.242	56.875
8	1.188	1.529	2.414	6.334	23.240	*
12	1.826	2.335	3.571	9.280	32.820	*
16	2.474	3.124	5.549	12.983	41.965	*
32	5.066	6.420	9.804	24.815	*	*

Table 5: The runtime of PSImple in seconds for 2-32 parties and input size 2^8-2^{18}. * signifies that the protocol crashed due to insufficient memory.

As can be observed from the results, the running time of PSImple grows approximately linearly both in the number of parties and in the number of inputs.[9] This is illustrated in Figures 3 and 4, respectively.

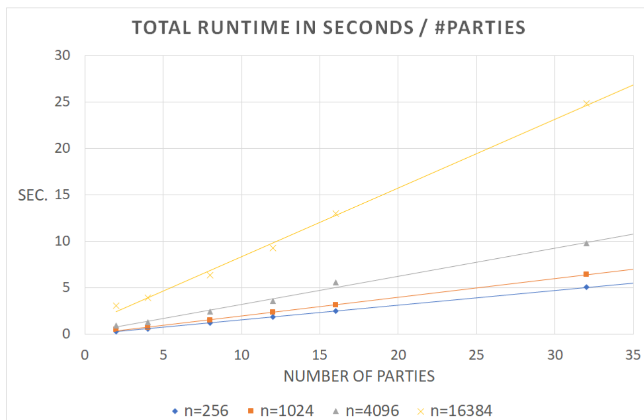

Figure 3: Total time for different number of parties

We next looked at the cost of the various steps of PSImple. An interesting aspect of the runtime is that, for a small amount of parties (e.g., 2, 4), the main bottleneck is the rerandomization step. However, the runtime of the rerandomization step remains constant with the number of parties. As a result, for a small amount of parties, the runtime is dominated by the rerandomization (in the online phase), while for a large amount of parties the runtime is dominated by the OTs (in the offline phase). The computation complexity of the rerandomization algorithm **ReRand** is $O(nk)$. Thus, it is possible that

[7] This restricts the input items' domain to 120 bits, as it requires 8 bits for the hash function selection. Note also that this makes some assumptions on the randomness of AES, and that taking modulus already implies that this is not fully random. However, our experiments suggest that this is random enough.

[8] We remark that this optimization requires the random oracle assumption, which we require for our proof.

[9] The offline complexity of PSImple is linear in N_{OT} and the online complexity is (quasi-)linear in Nbf. N_{OT} and Nbf are (asymptotically) linear in the input size.

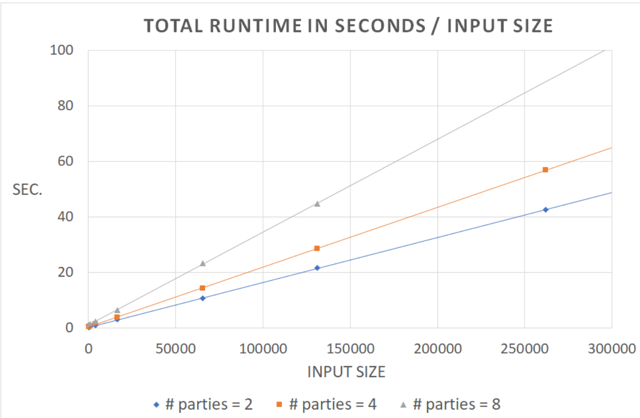

Figure 4: Total time for different size inputs

for a small amount of parties, PSImple would run faster using a different set of parameters, in which k is smaller at the cost of increasing the number of OTs. An interesting question is if the **ReRand** algorithm can be computed more efficiently using parallel computation, as this would drastically improve the runtime of this step. We plan to look into this question further.

We next compare PSImple's runtime with the reported runtimes of IOP [17] (multiparty augmented semi-honest protocol), Zhang et al. [38] (multiparty, non-standard security model), and [35] (two-party malicious protocol). We chose to compare with these protocols as they are similarly based on GBFs. Note that a more fair comparison would be with a maliciously secure multiparty PSI protocol, but previous maliciously secure multiparty PSI protocols were not concretely efficient and, therefore, were not implemented.

Comparison with IOP [17] and Zhang et al. [38]. In Tables 6 and 7 we compare PSiImple's runtime with the reported runtimes of the multiparty PSI protocols of [17] and [38], respectively. Recall that [17] only achieves augmented semi-honest security and therefore should be significantly faster than PSImple as it requires only semi-honest OT, no cut-and-choose, and there is no need to perform OT in both directions. The protocol of [38] is in a very non-standard security model, since it makes the assumption that two dedicated parties, P_0 and P_1, are not simultaneously corrupted. This relaxation of the security model allows them to have a significantly simpler protocol, which is insecure in the standard malicious model.

Surprisingly, despite the fact that PSImple achieves much stronger security guarantees, the runtime of PSImple in our experiments is not significantly slower, and in some cases even faster, than the reported runtimes of [17] and [38]. We attribute this to the fact that [17] wrote their code in Java, and that both [17] and [38] ran their experiments on slightly weaker testing platforms. Nevertheless, this shows that moving to maliciously security using PSImple does not incur a very high penalty.

		sec. model	$n = 2^8$	$n = 2^{16}$	$n = 2^{18}$
4 parties	[17]	semi-honest	2.08	25.03	91.48
	PSImple	malicious	0.558	14.242	56.875
12 parties	[17]	semi-honest	2.29	38.53	195.51
	PSImple	malicious	1.826	32.82	*

Table 6: Comparison of PSImple with [17] in total runtime.

		sec. model	$n = 2^8$	$n = 2^{12}$	$n = 2^{16}$
4 parties	[38]	non-standard malicious	0.76	2.95	40.91
	PSImple	malicious	0.558	1.310	14.242

Table 7: Comparison of PSImple with [38] in total runtime.

Comparison with RR [35]. Recall that the maliciously secure protocol of [35] is limited to two-parties, while PSImple is intended as a multiparty protocol. As explained, directly extending [35] to the multiparty scenario would make the computation complexity grow exponentially with the number of parties, while PSImple only grows approx. linearly with the number of parties. Thus, even for modest parameters such as 4 parties and input size $n = 2^{16}$, we expect the direct multiparty extension of [35] to be barely practical, while this takes less than 10 seconds using PSImple. For a larger number of parties, we expect the direct multiparty extension of [35] to be completely impractical, while PSImple's runtime grows linearly in the number of parties.

For the two-party scenario, PSImple requires two instances of Π_{AppROT}, while [35] requires only a single instance. Additionally, PSImple's online phase requires both more computation, due to the additional rerandomization step, and more communication, since it requires two executions of Π_{AppROT}^{Online} and in the last communication round the entire GBF is sent, while [35] requires a single execution of Π_{AppROT}^{Online} and only the codewords are sent in the last round. Furthermore, [35] benchmarked their protocol using stronger machines.

For the above reasons, we expected PSImple to be outperformed by [35] in the 2-party scenario. Although our comparison in Table 8 confirms our expectation, we observe that the total runtime is not significantly slower than [35]. In contrast, the online time of PSImple, dominated by the rerandomization step, is significantly slower than the online time reported by [35].

		$n = 2^8$	$n = 2^{12}$	$n = 2^{16}$
total time	[35] (single thread)	0.2	0.9	9.7
	[35] (4 threads)	0.17	0.63	4.3
	PSImple (2 threads)	0.315	0.956	10.717
online time	[35]	0.003	0.04	0.7
	PSImple	0.027	0.487	8.054

Table 8: Comparison of 2-party PSImple with [35].

References

[1] Aner Ben-Efraim, Michael Nielsen, and Eran Omri. Turbospeedz: Double your online SPDZ! improving SPDZ using function dependent preprocessing. In Robert H. Deng, Valérie Gauthier-Umaña, Martín Ochoa, and Moti Yung, editors, *Applied Cryptography and Network Security - 17th International Conference, ACNS 2019, Bogota, Colombia, June 5-7, 2019, Proceedings*, volume 11464 of *Lecture Notes in Computer Science*, pages 530–549. Springer, 2019.

[2] Burton H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Commun. ACM*, 13(7):422–426, 1970.

[3] Prosenjit Bose, Hua Guo, Evangelos Kranakis, Anil Maheshwari, Pat Morin, Jason Morrison, Michiel Smid, and Yihui Tang. On the false-positive rate of bloom filters. *Information Processing Letters*, 108(4):210–213, 2008.

[4] Ran Canetti. Security and composition of multiparty cryptographic protocols. *Journal of CRYPTOLOGY*, 13(1):143–202, 2000.

[5] Charalambos A Charalambides. *Enumerative combinatorics*. CRC Press, 2002.

[6] Ken Christensen, Allen Roginsky, and Miguel Jimeno. A new analysis of the false positive rate of a bloom filter. *Information Processing Letters*, 110(21):944–949, 2010.

[7] Vasek Chvátal. The tail of the hypergeometric distribution. *Discrete Mathematics*, 25(3):285–287, 1979.

[8] Ivan Damgård, Marcel Keller, Enrique Larraia, Valerio Pastro, Peter Scholl, and Nigel P. Smart. Practical covertly secure MPC for dishonest majority - or: Breaking the SPDZ limits. In Jason Crampton, Sushil Jajodia, and Keith Mayes, editors, *Computer Security - ESORICS 2013 - 18th European Symposium on Research in Computer Security, Egham, UK, September 9-13, 2013. Proceedings*, volume 8134 of *Lecture Notes in Computer Science*, pages 1–18. Springer, 2013.

[9] Emiliano De Cristofaro, Jihye Kim, and Gene Tsudik. Linear-complexity private set intersection protocols secure in malicious model. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 213–231. Springer, 2010.

[10] Changyu Dong, Liqun Chen, and Zikai Wen. When private set intersection meets big data: An efficient and scalable protocol. In *the ACM Conference on Computer and Communications Security, CCS'13*, pages 789–800. ACM, 2013.

[11] Satrajit Ghosh and Tobias Nilges. An algebraic approach to maliciously secure private set intersection. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 154–185. Springer, 2019.

[12] Oded Goldreich. *The Foundations of Cryptography, volume 2*. Cambridge University Press, 2004.

[13] Fabio Grandi. On the analysis of bloom filters. *Information Processing Letters*, 129:35–39, 2018.

[14] Carmit Hazay, Peter Scholl, and Eduardo Soria-Vazquez. Low cost constant round MPC combining BMR and oblivious transfer. In Tsuyoshi Takagi and Thomas Peyrin, editors, *Advances in Cryptology - ASIACRYPT 2017 - 23rd International Conference on the Theory and Applications of Cryptology and Information Security, Hong Kong, China, December 3-7, 2017, Proceedings, Part I*, volume 10624 of *Lecture Notes in Computer Science*, pages 598–628. Springer, 2017.

[15] Carmit Hazay and Muthuramakrishnan Venkitasubramaniam. Scalable multi-party private set-intersection. In *IACR International Workshop on Public Key Cryptography*, pages 175–203. Springer, 2017.

[16] Yan Huang, David Evans, and Jonathan Katz. Private set intersection: Are garbled circuits better than custom protocols? In *19th Annual Network and Distributed System Security Symposium, NDSS 2012, San Diego, California, USA, February 5-8, 2012*. The Internet Society, 2012.

[17] Roi Inbar, Eran Omri, and Benny Pinkas. Efficient scalable multiparty private set-intersection via garbled bloom filters. In *International Conference on Security and Cryptography for Networks*, pages 235–252. Springer, 2018.

[18] Y. Ishai, J. Kilian, K. Nissim, and E. Petrank. Extending oblivious transfers efficiently. In D. Boneh, editor, *Advances in Cryptology – CRYPTO 2003*, volume 2729 of *Lecture Notes in Computer Science*, pages 145 – 161. Springer, 2003.

[19] Marcel Keller, Emmanuela Orsini, and Peter Scholl. Actively secure ot extension with optimal overhead. In *Annual Cryptology Conference*, pages 724–741. Springer, 2015.

[20] Marcel Keller, Valerio Pastro, and Dragos Rotaru. Overdrive: Making SPDZ great again. In Jesper Buus Nielsen and Vincent Rijmen, editors, *Advances in Cryptology - EUROCRYPT 2018 - 37th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tel Aviv, Israel, April 29 - May 3, 2018 Proceedings, Part III*, volume 10822 of *Lecture Notes in Computer Science*, pages 158–189. Springer, 2018.

[21] Myungsun Kim, Hyung Tae Lee, and Jung Hee Cheon. Mutual private set intersection with linear complexity. In *International Workshop on Information Security Applications*, pages 219–231. Springer, 2011.

[22] Lea Kissner and Dawn Song. Privacy-preserving set operations. In *Annual International Cryptology Conference*, pages 241–257. Springer, 2005.

[23] Vladimir Kolesnikov, Ranjit Kumaresan, Mike Rosulek, and Ni Trieu. Efficient batched oblivious prf with applications to private set intersection. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 818–829, 2016.

[24] Vladimir Kolesnikov, Naor Matania, Benny Pinkas, Mike Rosulek, and Ni Trieu. Practical multi-party private set intersection from symmetric-key techniques. In *the ACM Conference on Computer and Communications Security, CCS'17*, 2017.

[25] Michael Mitzenmacher and Eli Upfal. *Probability and computing: Randomization and probabilistic techniques in algorithms and data analysis*. Cambridge university press, 2017.

[26] Benny Pinkas, Mike Rosulek, Ni Trieu, and Avishay Yanai. Psi from paxos: Fast, malicious private set intersection. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 739–767. Springer, 2020.

[27] Benny Pinkas, Thomas Schneider, Gil Segev, and Michael Zohner. Phasing: Private set intersection using permutation-based hashing. In Jaeyeon Jung and Thorsten Holz, editors, *24th USENIX Security Symposium, USENIX Security 15, Washington, D.C., USA, August 12-14, 2015*, pages 515–530. USENIX Association, 2015.

[28] Benny Pinkas, Thomas Schneider, Oleksandr Tkachenko, and Avishay Yanai. Efficient circuit-based psi with linear communication. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 122–153. Springer, 2019.

[29] Benny Pinkas, Thomas Schneider, Christian Weinert, and Udi Wieder. Efficient circuit-based PSI via cuckoo hashing. In Jesper Buus Nielsen and Vincent Rijmen, editors, *Advances in Cryptology - EUROCRYPT 2018 - 37th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tel Aviv, Israel, April 29 - May 3, 2018 Proceedings, Part III*, volume 10822 of *Lecture Notes in Computer Science*, pages 125–157. Springer, 2018.

[30] Benny Pinkas, Thomas Schneider, and Michael Zohner. Faster private set intersection based on OT extension. In *Proceedings of the 23rd USENIX Security Symposium*, pages 797–812. USENIX Association, 2014.

[31] Benny Pinkas, Thomas Schneider, and Michael Zohner. Scalable private set intersection based on ot extension. *ACM Transactions on Privacy and Security (TOPS)*, 21(2):1–35, 2018.

[32] M. O. Rabin. How to exchange secrets by oblivious transfer. TR-81, Harvard, 1981.

[33] Amanda C Davi Resende and Diego F Aranha. Faster unbalanced private set intersection. In *International Conference on Financial Cryptography and Data Security*, pages 203–221. Springer, 2018.

[34] Peter Rindal. libOTe: an efficient, portable, and easy to use Oblivious Transfer Library. `https://github.com/osu-crypto/libOTe`.

[35] Peter Rindal and Mike Rosulek. Improved private set intersection against malicious adversaries. In *Advances in Cryptology – EUROCRYPT 2017*, pages 235–259, 2017.

[36] Peter Rindal and Mike Rosulek. Malicious-secure private set intersection via dual execution. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 1229–1242, 2017.

[37] Xiao Wang, Samuel Ranellucci, and Jonathan Katz. Global-scale secure multiparty computation. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017*, pages 39–56. ACM, 2017.

[38] En Zhang, Feng-Hao Liu, Qiqi Lai, Ganggang Jin, and Yu Li. Efficient multi-party private set intersection against malicious adversaries. In *Proceedings of the 2019 ACM SIGSAC Conference on Cloud Computing Security Workshop*, pages 93–104, 2019.

A Basic Functionalities

$\mathcal{F}_{MPSI}$

$t+1$ – number of parties;
$\mathsf{P}_0, \mathsf{P}_1, ..., \mathsf{P}_t$ - parties; P_i holds $X_i = \{x_{i1}, x_{i2}, ..., x_{in_i}\}$, $n_i \leq n$ – the private input of the party;
n – the maximal size of the input set of any honest party; $n' \geq n$ – the maximal allowed size of the input set of any malicious party;
$\mathcal{D}$ – the domain of the items in the set of each party;
σ – computational security parameter; λ – statistical security parameter.
Inputs: X_i from each party P_i, n, σ.
Computation: If size of the corrupt party input set is bigger than n', then functionality aborts. Else it computes the intersection of all data sets: $X = \cap_{i=0}^{t} X_i$.
Outputs: Party P_0 receives X from the functionality, all other parties receive no output.

Figure 5: $\mathcal{F}_{MPSI}$ – Ideal multiparty private set intersection functionality

$\mathcal{F}_{ROT}$

Parties: the sender, the receiver.
Parameters: σ is the length of the OT strings.
Inputs: $b \in \{0,1\}$ from the receiver.
Computation: Functionality samples uniformly random messages m_0, $m_1 \leftarrow \{0,1\}^{\sigma}$.
Outputs: Give output m_b to the receiver and give (m_0, m_1) to the sender.

Figure 6: $\mathcal{F}_{ROT}$ – Ideal random 1-out-of-2 OT functionality

$\mathcal{F}_{AppROT}$

Parties: a sender, a receiver.
Parameters:
σ – computational security parameter; λ – statistical security parameter;
k – number of hash-functions in Bloom filter;
N – number of items to create.
Inputs:
From the receiver: $I = \{i_j\}_{j\in[K]}$; from the sender: no input.
Outputs:
Upon receiving I from the sender, samples the uniformly random $M = \{m_1, m_2, ..., m_N\}$, where m_is are σ-bit strings, and computes $M_* = \{m_{i_1}, m_{i_2}, ..., m_{i_K}\}$. Gives M to the sender, gives M_* to the receiver.

If the adversary corrupts the receiver
The functionality is working in two steps:

1. The corrupt receiver chooses K'' and sends it to the ideal functionality.
   Upon receiving K'' from the receiver, computes p_{False} – the false-positive probability in the Bloom filter of length N, with k hash-functions and K'' cells consisting 1's. If $p_{False} \geq 2^{-\sigma}$, then gives $\perp$ to the sender and to the receiver. Otherwise samples and gives to the receiver the uniformly random $M' = \{m'_1, m'_2, ..., m'_{K''}\}$, where m'_is are σ-bit strings.
2. After the response from the functionality, the receiver chooses and sends either $\perp$ or a partial injective mapping $I' = \begin{pmatrix} i_1 & i_2 & \dots & i_{K'} \\ j_1 & j_2 & \dots & j_{K'} \end{pmatrix}$, where $i_s \in [K'']$, $j_s \in [N]$, $s \in [K']$, $K'' \geq K' \geq K$.
   Upon receiving $\perp$ from the receiver, gives $\perp$ to the sender.
   Upon receiving I' from the receiver, computes
   $$m_j = \begin{cases} m'_{i_s}, & \text{if } j = j_s,\, s \in [K']; \\ \text{fresh random}, & \text{else.} \end{cases}$$
   The functionality gives $M = \{m_i\}_{i\in[N]}$ to the sender.

If the adversary corrupts the sender.

The functionality is working in two steps:

1. Upon receiving I from the receiver, samples and gives to the sender the uniformly random $M'' = \{m''_1, m''_2, ..., m''_{N_{\mathrm{OT}}}\}$, where m''_is are σ-bit strings.
2. Receives from the corrupt sender either $\perp$ or $C \subseteq [N_{\mathrm{OT}}]$: $|C| = N_{cc}$. Upon receiving $\perp$ from the sender, gives $\perp$ to the receiver.
   Upon receiving C from the sender, samples the uniformly random N-permutation $\psi : [N_{\mathrm{OT}}] \setminus C \to [N]$, computes $M = \psi(M'') = \{m_1, m_2, ..., m_N\}$ and $M_* = \{m_{i_1}, m_{i_2}, ..., m_{i_K}\}$. Gives M to the sender and M_* to the receiver.

Figure 7: $\mathcal{F}_{AppROT}$ – Ideal approximate K-out-of-N Random OT functionality

B Rindal and Rosulek Malicious-Secure Two-Party PSI Protocol

Protocol [35] (Malicious-secure Two-Party PSI):

Parameters:

X is Alice's input, Y is Bob's input. N_{BF} is the required Bloom filter size; k is the number of Bloom filter hash functions; N_{OT} is the number of OTs to generate. H is modeled as a random oracle with output length σ. α is the fraction of ones, p_{chk} is the probability of choosing each particular bit in cut-and-choose, $N_{maxones}$ is the maximal number of ones recovered in the cut-and-choose set to pass the check.

1. **[setup]** The parties perform a secure coin-tossing subprotocol to choose (seeds for) random Bloom filter hash functions $h_1,...,h_k\colon \{0,1\}^* \to [N_{\text{BF}}]$.
2. **[random OTs]** Bob chooses a random string $b = b_1,\ldots,b_{N_{\text{OT}}}$ with an α fraction of 1s. Parties perform N_{OT} OTs of random messages (of length σ), with Alice as sender. In the ith OT, Alice learns random strings $m_{i,0}, m_{i,1}$ chosen by the functionality. Bob uses choice bit b_i and learns $m_i^* = m_{i,b_i}$.
3. **[cut-and-choose challenge]** Alice chooses a set $C \subseteq [N_{\text{OT}}]$ by choosing each index with independent probability p_{chk}. She sends C to Bob. Bob aborts if $|C| > N_{\text{OT}} - N_{\text{BF}}$.
4. **[cut-and-choose response]** Bob computes the set $R = \{i \in C | b_i = 0\}$ and sends R to Alice. To prove that he used choice bit 0 in the OTs indexed by R, Bob computes $r^* = \bigoplus_{i\in R} m_i^*$ and sends it to Alice. Alice aborts if $|C| - |R| > N_{maxones}$ or if $r^* \neq \bigoplus_{i\in R} m_{i,0}$.
5. **[permute unopened OTs]** Bob generates a Bloom filter BF containing his items Y. He chooses a random injective function $\pi : [N_{\text{BF}}] \to ([N_{\text{OT}}] \setminus C)$ such that $\text{BF}[i] = b_{\pi(i)}$, and sends π to Alice.
6. **[randomized GBF]** For each item x in Alice's input set, she computes a summary value

$$K_x = H\left(x || \bigoplus_{i\in h_*(x)} m_{\pi(i),1}\right),$$

where $h_*(x) = \{h_i(x) : i \in [k]\}$. She sends a random permutation of $K = \{K_x | x \in X\}$.

7. **[output]** Bob outputs $\left\{y \in Y | H(y || \bigoplus_{i\in h_*(y)} m^*_{\pi(i)}) \in K\right\}$.

Figure 8: Malicious-secure two-party PSI protocol of Rindal and Rosulek

C Algorithms for the Garbled Bloom Filter

C.1 Re-randomization Algorithm for the Garbled Bloom Filter

Algorithm ReRandGBF $(X, Y, H^*, n, N_{\text{BF}}, \sigma)$

Input:

The set of items $X = (x_1, ..., x_n)$;
the set of codewords $Y = (y_1, ..., y_n)$: $|y_i| = \sigma$, $(i \in [n])$;
family of hash-indices $H^* = (h_*(x_1), ..., h_*(x_k))$: $h_*(x_i) = \left\{s | h_j(x_i) = s, j \in [k]\right\}$, $(i \in [n])$.

Algorithm:

GBF = *empty* N_{BF}*-size array of* σ*-long strings*
for i=1 ***to*** n ***do***
    finalInd=–1
    finalShare=y_i
    for each $j \in h_*(x_i)$ ***do***
        if GBF$[j]$ ***is*** *empty* ***then***
            if finalInd==–1 ***then***
                finalInd= j
            else
                GBF$[j] \xleftarrow{R} \{0,1\}^\sigma$
                finalShare=finalShare$\oplus$GBF$[j]$
        else
            finalShare=finalShare$\oplus$GBF$[j]$
    GBF[finalInd] =finalShare [10]
for $i = 0$ ***to*** $N_{\text{BF}} - 1$ ***do***
    if GBF$[i]$ ***is*** *empty* ***then***
        GBF$[i] \xleftarrow{R} \{0,1\}^\sigma$
return GBF

Output: GBF – garbled Bloom filter of set X with codewords from Y with hash-functions $h_1, ..., h_k$.

C.2 Algorithm for Computation of the Hash-Indices Set $h_*(x)$

Algorithm HashIndicesGBF(x, H, N_{BF})

Input:

Item x;
N_{BF} – length of GBF;
family of hash-functions $H = (h_1, ..., h_k)$: $h_i : \{0,1\}^* \to \{0,1\}^{N_{\text{BF}}}$, $(i \in [k])$.

Algorithm:

$h_*(x)$ = *empty 0-size array*
for i=1 ***to*** k ***do***
    if $h_i(x) \notin h_*(x)$ ***then***
        add $h_i(x)$ *to* $h_*(x)$

Output: $h_*(x)$ – set of indices of item x from the family of hash-functions $H = \{h_1, ..., h_k\}$.

C.3 Algorithm for Computation of the Codeword from the Garbled Bloom Filter

Algorithm CodewordGBF(GBF$, x, h_*(x), N_{\text{BF}}, \sigma)$

Input:

x – item;
GBF – random garbled Bloom filter;
N_{BF} – length of GBF;

[10] Note, that the probability of fail in this algorithm, that can appear in case finalInd==-1, is the probability of false- positive for one of n items. According (7), $p_{False} < 2^{-\sigma}$, so the union bound over all $x \in X$ is $n2^{-\sigma}$, which is still negligible in σ.

σ – bitlength of string in GBF;
$h_*(x)$ – set of hash-indices of x; $\forall i \in h_*(x)$, $i \in [N_{\mathrm{BF}}]$.

Algorithm:
1: y=0
2: ***for each*** $i \in h_*(x)$ ***do***
3: y=$y \oplus \mathrm{GBF}[i]$
Output: y – codeword for x in garbled Bloom filter GBF indexed by $h_*(x)$.

D Parameters of Π_{AppROT}

In this appendix we give the formulas for the parameters of the Π_{AppROT} protocol in Claim D.1 We made this analysis instead of what was performed in [35] to reach the improvement in the number of required OT instances and in the Bloom filter size. The experimental results show a 23-25% reduction in the number of required ROT in comparison with [35]. Also, we proved the existence of the suitable parameters for any positive n, λ, σ in Claim D.2.

We use the concept of the negligible (in security parameter) function. A function $\mu : \mathbb{N} \to \mathbb{N}$ is **negligible** if for every positive polynomial $p(\cdot)$ and all sufficiently large x it holds that $\mu(x) < 1/p(x)$. In our circumstances, we imply that the negligible in λ (or σ) value is less than $2^{-\lambda}$ ($2^{-\sigma}$ respectively).

The choice of parameters according to this requirement depends on the false positive probability for the Bloom filter size, that we denote as p_{False}. Before the year 2008, it was believed $p_{False} = p^k$ [25], where p is the proportion of ones, and k is the number of the Bloom filter hash-functions. But in the Year 2008 in [3] was shown that this formula is incorrect and is just a lower bound for p_{False}. Despite the new precise formula was drawn there, as well as the upper bound for the false-positive probability, the precise value was hardly computable, and the upper bound is quite rough and may be implemented only in limited cases. In the year 2010 in [6], a new algorithm for computing the false-positive probability was developed, and the authors conducted the series of experiments to show that the difference between the old formula and the precise value is decreasing as N_{BF} grows. However, the precise value of p_{False} is hard to use, and no asymptotic were found. In the year 2018 in [13], finally, was deduced the Taylor's second-order approximation of the false-positive probability, which we can use to obtain (in Appendix E) the upper bound for p_{False} (6), which is used in this appendix.

In the proof of Claim D.1 we use several tail inequalities for different distributions. The Azuma-Hoeffding inequality [25, p 355] for the distribution of zeroes in the Bloom filter is connected with the problem of the number of empty bins in the Balls and Bins model, and sounds as follows. Suppose we are throwing m balls independently and uniformly at random into n bins. Let F be the number of empty bins after m balls are thrown. Then $Pr[|F - E(F)| \geq \varepsilon] \leq 2e^{-\frac{2\varepsilon^2}{m}}$. In our case, the number of bins is nk, and we are interested only in the right tale of distribution.

Also, we use the tail inequality, obtained by V. Chvatal in [7] for hypergeometric distribution. Namely, for $HG(M,N,n)$ by $0 < t < pn$, we have $Pr[X \geq (p+t)n] \leq \left(\left(\frac{p}{p+t}\right)^{p+t} \left(\frac{1-p}{1-p-t}\right)^{1-p-t} \right)^N \leq e^{-2t^2 n}$.

Using the relations proved in Claim D.1, we built the algorithm in Figure 9 and found the optimal in N_{OT} parameters for several values of n, when $\sigma = 128$ and $\lambda = 40$. The parameters are given in Table 4.

Algorithm for computing parameters for Π_{AppROT}

1. Set $k = k_{\min}$.
2. Set $N_{\mathrm{BF}} = N_{\mathrm{BF,min}}$.
3. If $N_{\mathrm{BF}} > N_{\mathrm{BF,max}}$ then set $k = k+1$ and go to Step 2.
4. Calculate

$$N_1 = \left\lceil N_{\mathrm{BF}}\left(1 - e^{-\frac{nk}{N_{\mathrm{BF}}}}\right) + \sqrt{\frac{nk\lambda\ln 2}{2}} \right\rceil, \quad N = \left\lceil N_{\mathrm{BF}} + \sqrt{2nk\lambda\ln 2} \right\rceil, \ \alpha = N_1/N.$$

5. Calculate $a = \frac{\alpha}{\min(\alpha, 1-\alpha)}\sqrt{\frac{\lambda ln2}{2}}$, $c = \sqrt{2\lambda\ln 2}N$, $b = \frac{N_{\mathrm{BF}}}{2^{\sigma/k}}\left[1 + \frac{k(k-1)}{2N_{\mathrm{BF}}}\left(\frac{1}{\alpha} - 1\right)\right]^{-\frac{1}{k}} - \alpha N - \frac{\lambda\ln 2}{\min(\alpha,1-\alpha)}$, $D = b^2 - 4ac$.
6. If $D \leq 0$ or $b \leq \sqrt{D}$ then $N_{\mathrm{BF}} = N_{\mathrm{BF}} + 1$, and go to Step 3.
7. Take the minimal integer N_{cc}: $\frac{(b-\sqrt{D})^2}{4a^2} \leq N_{cc} \leq \frac{(b+\sqrt{D})^2}{4a^2}$ such that $N_{maxones} - N_{maxhonest} > 0$, where

$$N_{maxones} = \left\lfloor \frac{b + \frac{\lambda\ln 2}{\min(\alpha,1-\alpha)} + \alpha N}{N + N_\Delta} N_{cc} - \sqrt{\frac{\lambda\ln 2 N_{cc}}{2}} \right\rfloor, \quad N_{maxhonest} = \left\lceil \alpha N_{cc} + \sqrt{\frac{\lambda\ln 2 N_{cc}}{2}} \right\rceil, \ N_\Delta = \left\lceil \frac{a}{\alpha}\sqrt{N_{cc}} \right\rceil.$$

If there is no appropriate N_{cc} then set $N_{\mathrm{BF}} = N_{\mathrm{BF}} + 1$, and go to Step 3.
8. Calculate $N_{\mathrm{OT}} = N + N_\Delta + N_{cc}$, $N^1_{\mathrm{OT}} = \lceil \alpha N_{\mathrm{OT}} \rceil$, and save the tuple ($k$, N_{BF}, N_{cc}, N_{OT}, N^1_{OT}, $N_{maxones}$), if N_{OT} value is less than before.
9. If $k < k_{\max}$ then set $k = k+1$, and go to Step 2.
10. Output the tuple with minimal N_{OT}.

Figure 9: Numerical algorithm for computing parameters for Π_{AppROT}

For the online-phase, N_{BF} is more critical. In Table 9 we give the optimal by N_{BF} parameters, where advantage in N_{BF} is from 10,4% for $n = 2^8$, to 1,4% for $n = 2^{20}$ in comparison with the optimization of N_{OT}. However, the cost of it is the much higher disadvantage in the number of OTs (from 20,6% for $n = 2^8$ to 46,7% for $n = 2^{20}$) and in the size of the cut-and-choose set (from 287% for $n = 2^8$ to 5793,9% for $n = 2^{20}$), which entails an increase of the overall cost.

Parameters	$n=2^8$	$n=2^{10}$	$n=2^{12}$	$n=2^{14}$	$n=2^{16}$	$n=2^{18}$	$n=2^{20}$
k	139	133	130	129	129	128	128
N_{BF}	57,993	210,014	797,706	3,107,680	12,265,989	48,735,894	194,288,832
N_{OT}	89,772	320,253	1,206,819	4,681,730	18,439,057	73,183,339	291,592,668
N^1_{OT}	41,257	152,908	587,848	2,310,232	9,183,449	36,421,202	145,456,131
N_{cc}	28,994	104,963	398,851	1,553,817	6,132,901	24,367,378	97,143,998
$N_{maxones}$	13,960	51,323	196,635	771,384	3,063,672	12,145,310	48,495,359

Table 9: Optimal (in N_{BF}) Π_{AppROT} parameters for set size n, statistical security $\lambda = 40$, and computational security $\sigma = 128$.

GBF parameters:

N_{OT} : number of ROTs in Π_{AppROT};

N_{BF} : size of the Bloom filter of the receiver;

N^1_{OT} : number of 1's among N_{OT} choice bits of the receiver;

k : number of Bloom filter hash functions;

N_{cc} : number of bits to choose for the cut-and-choose check;

$N_{maxones}$: the maximal number of 1's among the N_{cc} choice bits allowed in order to pass the cut-and-choose check.

Informal requirements:

- *After the cut-and-choose, the receiver has enough ones and zeroes to build the Bloom filter.*
- *Both an honest and a malicious receiver have too few ones to find false positive.*
- *An honest receiver passes cut-and-choose with the overwhelming probability.*

All the random OTs and cut-and-choose check are performed in $\Pi^{offline}_{AppROT}$, when the receiver may not know its Bloom filter. The first requirement means, that the number of ones and zeroes among input bits of the receiver after the cut-and-choose check should be such that the receiver can construct from them almost any Bloom filter of the size N_{BF} for n items using k hash-functions. The probability of fail should be negligible in λ.

The second and the third requirements are connected with the inequality for the p_{False} which discussed in Section 2.1. Namely, the sender rejects the cut-and-choose response, if $Pr[p_{False} \geq 2^{-\sigma}] > 2^{-\lambda}$.

In the claim below we describe the constraints in a more precise way.

Claim D.1. *By choosing the parameters of Π_{AppROT} under the following constraints*

$$N_0 = \left\lceil N_{\mathrm{BF}} e^{-\frac{nk}{N_{\mathrm{BF}}}} + \sqrt{\frac{nk\lambda \ln 2}{2}} \right\rceil,\ N_1 = \left\lceil N_{\mathrm{BF}} \left(1 - e^{-\frac{nk}{N_{\mathrm{BF}}}}\right) + \sqrt{\frac{nk\lambda \ln 2}{2}} \right\rceil,\ N = N_0 + N_1, \alpha = N_1/N; \tag{2}$$

$$N_{cc} > 0 : \ N_{maxones} - N_{maxhonest} > 0, \ \ \textit{where} \tag{3}$$

$$N_{maxones} = \left\lfloor \frac{N_{\mathrm{BF}} N_{cc}}{N + N_\Delta} 2^{-\frac{\sigma}{k}} \left[1 + \frac{k(k-1)}{2N_{\mathrm{BF}}} \left(\frac{1}{\alpha} - 1\right)\right]^{-\frac{1}{k}} - \sqrt{\frac{\lambda N_{cc} \ln 2}{2}} \right\rfloor,\ N_{maxhonest} = \left\lceil \alpha N_{cc} + \sqrt{\frac{\lambda N_{cc} \ln 2}{2}} \right\rceil, \ \textit{and} \tag{4}$$

$$N_\Delta = \left\lceil \frac{1}{\min(\alpha, 1-\alpha)} \sqrt{\frac{\lambda N_{cc} \ln 2}{2}} \right\rceil,\ N_{\mathrm{OT}} = N + N_\Delta + N_{cc},\ N^1_{\mathrm{OT}} = \lceil \alpha N_{\mathrm{OT}} \rceil, \tag{5}$$

the following requirements hold:

1. *the honest receiver after cut-and-choose has enough ones to build Bloom filter (with parameters N_{BF}, k, n) with the probability at least $1 - 2^{-\lambda+1}$;*

2. *$Pr[p_{False} \geq 2^{-\sigma}] \leq 2^{-\lambda}$ with either honest or malicious receiver in Π_{AppROT};*

3. *an honest receiver passes the cut-and-choose check successfully with probability at least $1-2^{-\lambda}$.*

Proof. Consider every constraint one by one.

1. *After the cut-and-choose, the receiver has enough ones and zeroes to build the Bloom filter.*

   Compute the number of ones and zeroes required to build the Bloom filter. Denote the number of zeroes in the Bloom filter by N_0, and of ones by N_1. The probability of every given bit in Bloom filter to be 0 is $q = (1-1/N_{\text{BF}})^{nk} \approx e^{-\frac{nk}{N_{\text{BF}}}}$ and to 1 is $p = 1-q$ [25, p 116]. Using the Azuma-Hoefding inequality [25, p 355], we get $Pr[N_0 - E(N_0) \geq \varepsilon] \leq e^{-\frac{2\varepsilon^2}{nk}}$. We require that the number of zeroes is not enough to build the Bloom filter with the negligible in λ probability. Hence $e^{-\frac{2\varepsilon^2}{nk}} \leq 2^{-\lambda}$; solving this inequality, we get $\varepsilon \geq \sqrt{\frac{nk\lambda\ln 2}{2}}$. Therefore we should have at least $N_0 = \left\lceil N_{\text{BF}}q + \sqrt{\frac{nk\lambda\ln 2}{2}} \right\rceil$ zeroes after cut-and-choose.

   By implementation of the Azuma-Hoeffding inequality to the number of ones in the Bloom filter, we get $N_1 = \left\lceil N_{\text{BF}}p + \sqrt{\frac{nk\lambda\ln 2}{2}} \right\rceil$ ones needed after cut-and-choose, which gives (2). Denote $N = N_0 + N_1$. Then $\alpha = N_1/N$ is the proportion of ones in the choice sequence of the receiver.

   We need to be sure that after the cut-and-choose, the receiver has at least N_1 ones. This requires some extra supply of ones and zeroes $N_\Delta = N_{\text{OT}} - N$. Let the honest receiver chooses N_{OT} bits, with $N^1_{\text{OT}} = \lceil \alpha N_{\text{OT}} \rceil$ ones among them. The other party chooses N_{cc} arbitrary bits to check. We require that among the remaining $N + N_\Delta$ bits be at least N_1 ones with probability $\geq 1-2^{-\lambda}$. It means, that among N_{cc} cut-and-choose bits are no more than $N^1_{\text{OT}} - N_1 = \alpha N_{\text{OT}} - N_1 = \alpha(N + N_\Delta + N_{cc}) - \alpha N = \alpha(N_\Delta + N_{cc})$ bits.

   The number of ones in the cut-and-choose set is distributed hypergeometrically $HG(\alpha N_{\text{OT}}, N_{\text{OT}}, N_{cc})$. Using the V. Chvatal inequality for hypergeometric distribution [7], we get

   $Pr[X \geq \alpha(N_\Delta + N_{cc})] = Pr\left[X \geq \left(\alpha + \frac{\alpha N_\Delta}{N_{cc}}\right) N_{cc}\right] \leq e^{-2\frac{(\alpha N_\Delta)^2}{N_{cc}}} \leq 2^{-\lambda}$. Hence we need to have $N_\Delta \geq \frac{1}{\alpha}\sqrt{\frac{N_{cc}\lambda\ln 2}{2}}$ extra bits in random OT.

   Analogically, we require that the number of zeroes after cut-and-choose remain at least N_0 with the probability at least $1-2^{-\lambda}$. Hence, $N_\Delta \geq \frac{1}{1-\alpha}\sqrt{\frac{N_{cc}\lambda\ln 2}{2}}$. Consequently, $N_\Delta = \left\lceil \frac{1}{\min(\alpha, 1-\alpha)}\sqrt{\frac{N_{cc}\lambda\ln 2}{2}} \right\rceil$, which is (5).

2. *Both an honest and a malicious receiver have too few ones to find false positive.*

   Denote by N_{1rest} number of 1's left by the receiver after the opening of N_{cc} cut-and-choose bits. The choice of parameters according to this requirement depends on the false positive probability for the Bloom filter size of N_{BF} with N_{1rest} bits set to 1 and k hash-functions, that we denote as p_{False}. The upper bound for it (in our conditions of the experiment) is obtained in in Appendix E from [13]:

   $$p_{False} < p^k \left[1 + \frac{k(k-1)}{2N_{\text{BF}}}\left(\frac{1}{p} - 1\right)\right]. \tag{6}$$

   Turning to the cut-and-choose in Π_{AppROT}, the sender doesn't see the actual number of items n nor the actual number of ones (that is N_{1rest} in our notation). With known N_{1rest}, one can express $p = N_{1rest}/N_{\text{BF}}$. With the probability at least $1-2^{-\lambda}$, because N_{1rest} is greater or equal to N_1 with the such a probability (according to the first statement in this claim), holds $1/p = N_{\text{BF}}/N_{1rest} < N_{\text{BF}}/N_1 < N/N_1 = 1/\alpha$. Hence, with probability at least $1-2^{-\lambda}$, from (6),

   $$p_{False} < \left(\frac{N_{1rest}}{N_{\text{BF}}}\right)^k \left[1 + \frac{k(k-1)}{2N_{\text{BF}}}\left(\frac{1}{\alpha} - 1\right)\right]. \tag{7}$$

   We require that $Pr[p_{False} \geq 2^{-\sigma}] \leq 2^{-\lambda}$. Using (7), we can rewrite the expression in parentheses as $N_{1rest} \geq N_{\text{BF}} 2^{-\frac{\sigma}{k}} \left[1 + \frac{k(k-1)}{2N_{\text{BF}}}\left(\frac{1}{\alpha} - 1\right)\right]^{-1/k}$.

   Suppose that the malicious party chooses more ones than required, and the proportion of ones now is $\hat{\alpha} > \alpha$, and the sender observes $\tilde{\alpha}$ proportion of ones among N_{cc} opened bits. Latter is the unbiased estimator of $\hat{\alpha}$ with $Pr\left[\hat{\alpha}N_{cc} \geq (\tilde{\alpha} + \tilde{\delta})N_{cc}\right] \leq e^{-2\tilde{\sigma}^2 N_{cc}} \leq 2^{-\lambda}$. Hence $\tilde{\sigma} \geq \sqrt{\frac{\lambda\ln 2}{2N_{cc}}}$.

Then, using (7), the number of ones in the remained set with the overwhelming probability is $N_{1rest} < (\tilde{\alpha}+\tilde{\sigma})(N+N_\Delta) \leq (\tilde{\alpha}+\sqrt{\frac{\lambda \ln 2}{2N_{cc}}})(N+N_\Delta) \leq N_{\mathrm{BF}} 2^{-\frac{\sigma}{k}} / \left[1+\frac{k(k-1)}{2N_{\mathrm{BF}}}\left(\frac{1}{\alpha}-1\right)\right]^{\frac{1}{k}}$. As the number of observed ones is $\tilde{\alpha}N_{cc}$, we have the inequality for the maximal number of opened ones as $N_{maxones} \leq \tilde{\alpha}N_{cc} \leq \frac{N_{\mathrm{BF}}N_{cc}}{N+N_\Delta} 2^{-\frac{\sigma}{k}}\left[1+\frac{k(k-1)}{2N_{\mathrm{BF}}}\left(\frac{1}{\alpha}-1\right)\right]^{-\frac{1}{k}} - \sqrt{\frac{\lambda N_{cc}\ln 2}{2}}$. Equality (4) follows.

3. *An honest receiver passes cut-and-choose with the overwhelming probability.*

   Denote by $N_{maxhonest}$ the maximal number of ones in the cut-and-choose set in the case of an honest receiver, and $Pr\left[N_{maxhonest} \geq (\alpha+\delta)N_{cc}\right] \leq e^{-2\sigma^2 N_{cc}} \leq 2^{-\lambda}$, hence $N_{maxhonest} = \left\lceil \alpha N_{cc} + \sqrt{\frac{\lambda N_{cc}\ln 2}{2}} \right\rceil$. If $N_{maxones} - N_{maxhonest} > 0$, then the honest receiver passes the cut-and-choose check.

   Considering $N_\Delta = \frac{1}{\min(\alpha,1-\alpha)}\sqrt{\frac{N_{cc}\lambda \ln 2}{2}}$, this expression is transformable to the following square inequality:

   $$\frac{\alpha}{\min(\alpha,1-\alpha)}\sqrt{\frac{\lambda \ln 2}{2}}N_{cc} - \left(\frac{N_{\mathrm{BF}}}{2^{\sigma/k}}\left[1+\frac{k(k-1)}{2N_{\mathrm{BF}}}\left(\frac{1}{\alpha}-1\right)\right]^{-\frac{1}{k}} - \alpha N - \frac{\lambda \ln 2}{\min(\alpha,1-\alpha)}\right)\sqrt{N_{cc}} + \sqrt{2\lambda \ln 2}N < 0. \quad (8)$$

   Thus, if a suitable N_{cc} exists, its value lies between the squares of non-negative roots of the corresponding square equation. If both roots are negative, or if there are no roots, then there are no suitable parameters in this case.

   According to the desire to have as few OTs as possible, we have to take the minimal non-negative value of N_{cc} from the interval determined by this inequality. Nevertheless, because of roundings in parameter calculations, the actual interval is, as a rule, narrower. Therefore we should, besides, check that indeed $N_{maxones} - N_{maxhonest} > 0$ by this particular N_{cc}. (3) follows.

□

The existence of suitable parameters for any n, σ, λ follows from the next claim.

Claim D.2. *For any choice of positive n, σ, λ there exist positive k, N_{BF}, N_{cc}, N_{OT}, $N_{maxones}$, N^1_{OT} such that (2)-(5) hold.*

Proof. Considering the asymptotic when $N_{\mathrm{BF}} \to \infty$, we compute the following limits:

$$\lim_{N_{\mathrm{BF}}\to\infty} N_{\mathrm{BF}}\left(1-e^{\frac{-nk}{N_{\mathrm{BF}}}}\right) = \lim_{N_{\mathrm{BF}}\to\infty} N_{\mathrm{BF}}\left(1-\left(1+\frac{1}{N_{\mathrm{BF}}}\right)^{-nk}\right) = \lim_{N_{\mathrm{BF}}\to\infty} N_{\mathrm{BF}}\left(1-\left(1+\frac{nk}{N_{\mathrm{BF}}}\right)^{-1}\right) = \lim_{N_{\mathrm{BF}}\to\infty}\frac{N_{\mathrm{BF}}nk}{N_{\mathrm{BF}}+nk} = nk;$$

$$\lim_{N_{\mathrm{BF}}\to\infty}\alpha = \lim_{N_{\mathrm{BF}}\to\infty}\frac{N_1}{N} = \lim_{N_{\mathrm{BF}}\to\infty}\frac{N_{\mathrm{BF}}\left(1-e^{\frac{-nk}{N_{\mathrm{BF}}}}\right)+\sqrt{nk\lambda \ln 2/2}}{N_{\mathrm{BF}}+\sqrt{2nk\lambda \ln 2}} = \lim_{N_{\mathrm{BF}}\to\infty}\frac{nk+\sqrt{nk\lambda \ln 2/2}}{N_{\mathrm{BF}}+\sqrt{2nk\lambda \ln 2}} = \lim_{N_{\mathrm{BF}}\to\infty}\frac{nk+\sqrt{nk\lambda \ln 2/2}}{N_{\mathrm{BF}}} = 0. \quad (9)$$

Due to (9), in the asymptotics we can only consider the case $\alpha \leq 0.5$, and hence $\min(\alpha,1-\alpha) = \alpha$.

After fixing k and N_{BF}, the rest of the parameters can be computed directly, with the exception of N_{cc}, which is derived from Inequality (8). So, the question of the existence of suitable parameters is the question of existence of positive roots in the square equation $ax^2 - bx + c = 0$, where, taking $\min(\alpha,1-\alpha) = \alpha$, $a = \sqrt{\frac{\lambda ln2}{2}}$, $b = \frac{N_{\mathrm{BF}}}{2^{\sigma/k}}\left[1+\frac{k(k-1)}{2N_{\mathrm{BF}}}\left(\frac{1}{\alpha}-1\right)\right]^{-\frac{1}{k}} - \alpha N - \frac{\lambda \ln 2}{\alpha}$, $c = \sqrt{2\lambda \ln 2}N$. For the existence of two positive roots, it is sufficient to have $b > 0$, $D = b^2 - 4ac > 0$. Let fix some value of $k > \sigma$ and prove that there exists some N_{BF} such that those conditions hold. From (9),

$$\lim_{N_{\mathrm{BF}}\to\infty} b = \lim_{N_{\mathrm{BF}}\to\infty} \frac{N_{\mathrm{BF}}}{2^{\sigma/k}} \left[1 + \frac{k(k-1)}{2N_{\mathrm{BF}}}\left(\frac{1}{\alpha}-1\right)\right]^{-\frac{1}{k}} - \alpha N - \frac{\lambda \ln 2}{\alpha} =$$

$$= \lim_{N_{\mathrm{BF}}\to\infty} \frac{N_{\mathrm{BF}}}{2^{\sigma/k}} \left[1 + \frac{k(k-1)}{2N_{\mathrm{BF}}} \frac{N_{\mathrm{BF}}}{nk+\sqrt{nk\lambda \ln 2/2}}\right]^{-\frac{1}{k}} - \frac{nk+\sqrt{nk\lambda \ln 2/2}}{N_{\mathrm{BF}}}\left(N_{\mathrm{BF}} + \sqrt{2nk\lambda \ln 2}\right) - \frac{\lambda \ln 2 N_{\mathrm{BF}}}{nk+\sqrt{nk\lambda \ln 2/2}} =$$

$$= \lim_{N_{\mathrm{BF}}\to\infty} N_{\mathrm{BF}} \left(2^{\sigma}\left[1 + \frac{k(k-1)}{2nk+\sqrt{2nk\lambda \ln 2}}\right]\right)^{-\frac{1}{k}} - nk - \sqrt{nk\lambda \ln 2/2} - N_{\mathrm{BF}} \frac{\lambda \ln 2}{nk+\sqrt{nk\lambda \ln 2/2}} =$$

$$= \lim_{N_{\mathrm{BF}}\to\infty} N_{\mathrm{BF}} \left[\left(2^{\sigma}\left[1 + \frac{k(k-1)}{2nk+\sqrt{2nk\lambda \ln 2}}\right]\right)^{-\frac{1}{k}} - \frac{2\lambda \ln 2}{nk+\sqrt{2nk\lambda \ln 2}}\right].$$

The asymptotic of b is linear in N_{BF}. The value in outer square parentheses is constant by fixed n, λ, k, and σ. If $k > \sigma$, then $\left(2^{\sigma}\left[1 + \frac{k(k-1)}{2nk+\sqrt{2nk\lambda \ln 2}}\right]\right)^{-\frac{1}{k}}$ tends to 1 when k grows, while $\frac{2\lambda \ln 2}{nk+\sqrt{2nk\lambda \ln 2}}$ tends to 0. Thus, for sufficiently large $k = k_1$, $\lim_{N_{\mathrm{BF}}\to\infty} b = \lim_{N_{\mathrm{BF}}\to\infty} C_1 N_{\mathrm{BF}}$, where $C_1 > 0$. Hence there exists a sufficiently large N_{BF} such that $b > 0$.

Now consider the asymptotic of the discriminant when $k \geq k_1$:
$\lim_{N_{\mathrm{BF}}\to\infty}(b^2 - 4ac) = \lim_{N_{\mathrm{BF}}\to\infty}\left((C_1 N_{\mathrm{BF}})^2 - 4N\lambda ln2\right) = \lim_{N_{\mathrm{BF}}\to\infty}\left((C_1 N_{\mathrm{BF}})^2 - 4N_{\mathrm{BF}}\lambda \ln 2 - 4\sqrt{2nk}(\lambda \ln 2)^{3/2}\right) = \lim_{N_{\mathrm{BF}}\to\infty}(C_1 N_{\mathrm{BF}})^2$.
Again, by the sufficiently large k there exists the sufficiently large N_{BF} such that $b^2 - 4ac > 0$. That implies that two positive roots of the square equation can be found, and therefore there exists N_{cc} that satisfies Equation (8). □

E False-Positive Probability of a Bloom Filter

In this appendix we obtain the upper bound for p_{False} – false-positive probability for the Bloom filter in the form, suitable to use in Π_{AppROT}. As the sender in this protocol should evaluate this probability without knowing n – the number of items, directly, it has to use the expression which includes p – the proportion of 1's, instead.

The second-order Taylor's approximation of the false-positive probability in the Bloom filter is derived in [13]. We give it as it was presented in the original paper:

$$p_{False} = \left(\frac{E[X]}{m}\right)^k + \frac{\sigma_X^2}{2}\frac{k(k-1)}{m^2}\left(\frac{E[X]}{m}\right)^{k-2}, \tag{10}$$

where m is the length of Bloom filter (in our notation, it is N_{BF}), k is the number of hash-functions, X is the number of ones presented in Bloom filter, $E[X] = m\left[1-(1-1/m)^{kn}\right]$ is the expectation of the number of ones, and

$$\sigma_X^2 = m\left(1-\frac{1}{m}\right)^{kn}\left[1-m\left(1-\frac{1}{m}\right)^{kn} + (m-1)\left(1-\frac{1}{m-1}\right)^{kn}\right]$$

is the standard deviation of the number of ones. In all those equations, n is the number of items already presented in the Bloom filter. For the malicious receiver the number of items can be higher than n (which is the event that the sender is trying to prevent in cut-and-choose). For the convenience of using (10) in our conditions we switch from the expression of p_{False} through n (number of items) to the expression through p (probability of 1's). We take $\left[1-(1-1/m)^{kn}\right] = p$, then $E[X] = mp$. Also notice that $1-\frac{1}{m-1} < 1-\frac{1}{m}$. Then we can rewrite σ_X^2 as

$$\sigma_X^2 = m(1-p)\left[1-m(1-p)+(m-1)\left(1-\frac{1}{m-1}\right)^{kn}\right] < m(1-p)\left[1-m(1-p)+(m-1)(1-p)\right] = mp(1-p). \tag{11}$$

From (10) and (11), we get

$$p_{False} < p^k + \frac{p(1-p)}{2}\frac{k(k-1)}{m}p^{k-2} = p^k\left[1+\frac{k(k-1)}{2m}\frac{1-p}{p}\right] = p^k\left[1+\frac{k(k-1)}{2m}\left(\frac{1}{p}-1\right)\right]. \tag{12}$$

Replacing m by N_{BF} according to our notation, we got (6).

F Security Proof

In this appendix we give necessary lemmas and calculations behind security statements in Section 3.3. The proof of the consistency F.1 relates to the correctness statement.

F.1 Consistency of PSImple

Consistency follows from next: consider GBF^* that P_0 learns on the step 8.

$$\mathrm{GBF}^* = \bigoplus_{i\in[t]} \mathrm{GBF}^{i*} \bigoplus \mathrm{GBF}^0 = \bigoplus_{i\in[t]\cup\{0\}} \mathrm{GBF}^i \bigoplus_{i,l\in[t],i\neq l} \left(S^{li} \oplus S^{il}\right) \bigoplus_{i\in[t]} \left(M_*^i \oplus \hat{M}^i\right) = \bigoplus_{i\in[t]\cup\{0\}} \mathrm{GBF}^i \bigoplus_{i\in[t]} \left(M_*^i \oplus \hat{M}^i\right).$$

GBF^i is the re-randomised $M^i \oplus \hat{M}_*^i$, and the re-randomization performed by P_i doesn't affect codewords of its items, therefore for any $x_{ij} \in X_i$ the codeword $y_{ij} = \oplus_{s\in h_*(x)} \mathrm{GBF}^i[s] = \oplus_{s\in h_*(x)} \left(M^i[s] \oplus \hat{M}_*^i[s]\right) = \oplus_{s\in h_*(x)} \left(M_*^i[s] \oplus \hat{M}^i[s]\right)$.

Hence, for every item $x \in \cap_{i\in[t]} X_i$ with codewords y_is, we have

$$\bigoplus_{s\in h_*(x)} \mathrm{GBF}^*[s] = \bigoplus_{s\in h_*(x)} \left(\mathrm{GBF}^i[s] \bigoplus_{i\in[t]} \left(M_*^i[s] \oplus \hat{M}^i[s]\right)\right) = \bigoplus_{i\in[t]} (y_i \oplus y_i) = 0.$$

F.2 Security

Notation. Let $W = \{w_1, ..., w_n\}$ be a set of n elements. A partial injective and onto mapping $\xi : A \to B$ where $|B| = k$, and $|A| = n$ is called a k-permutation from n [5, p. 40]. We usually denote such mappings by a $k-tuple$ over A. For such a permutation ξ, given a vector X indexed by elements of A, we denote by $Y = \xi(X)$ a vector indexed by elements of B, where $y_i = x_{\xi^{-1}(i)}$ for each $i \in B$. For some ordering I of B, we denote by $J = \xi(I)$ the ordering For indices we write $j = \xi(i)$ if we use the permutation to define the mapping.

Definition 1. *A function $\mu : \mathbb{N} \to \mathbb{N}$ is **negligible** if for every positive polynomial $p(\cdot)$ and all sufficiently large x it holds that $\mu(x) < 1/p(x)$.*

Definition 2. *We say that two distribution ensembles $\{X(\kappa,a)\}_{\kappa\in N, a\in\{0,1\}^*}$ and $\{Y(\kappa,a)\}_{\kappa\in N, a\in\{0,1\}^*}$ are **statistically close**, denoted $\{X(\kappa,a)\} \overset{s}{\approx} \{Y(\kappa,a)\}$, if for every non-uniform distinguisher D there exists a negligible function μ such that for all a and κ, $|Pr[D(X(\kappa,a)) = 1] - Pr[D(Y(\kappa,a)) = 1]| \leq \mu(\kappa)$.*

Model. We prove our results in the standard UC model [4], assuming private authenticated channels between the parties (in fact, authentication of sender's identity is already guaranteed due to the fact that the adversary may only reorder messages, delay or delete messages sent between a pair of parties but not modify them). The latter can be modeled by assuming communication via ideal channel functionalities that allow for the suitable adversarial behavior (allowing interventions as above, but adding secrecy of message content), see [4] for more details - the following is a brief recap of the setting, which is identical to the standard setting of [4], except of assuming channels as above, instead of the 'bare bones' network.

The parties, the adversary $\mathcal{A}$ and the environment $\mathcal{Z}$ are modeled as polynomial-time non-uniform ITMs. All parties, including $\mathcal{Z}$ (for which it is an only input) have a public parameter 1^k provided as input. Furthermore, it is known that for defining UC security, it suffices to consider a 'dummy adversary', which merely relays messages between $\mathcal{Z}$, parties and instances of idealized functionalities. That is, at the beginning it corrupts a set of parties $\mathcal{Z}$ instructs it to corrupt. Then, every time a party p sends it a message m (intended for party p'), it sends $\mathcal{Z}$ a message that 'p requested to send a message to p''. Note that as we assume private channels, it does not see the content of m unless p is corrupt, in which case it also reports to $\mathcal{Z}$ the content of the message. It also receives commands from $\mathcal{Z}$ to relay a message waiting to be sent, or send a given message m' from a corrupted party p to some party p' or as a message to an idealized functionality. We also assume $\mathcal{Z}$ has access to the entire state of $\mathcal{A}$, including the state of all parties corrupted by it so far. In some more detail:

Real World execution Very briefly, as standard in the UC setting, at the beginning of a protocol execution $\mathcal{Z}$ is initialized with a public security parameter and invokes other machines - the adversary $\mathcal{A}$ and protocol participants which are also give 1^k as a security parameter. $\mathcal{Z}$ provides the inputs to the protocol participants by writing to their input tapes. More precisely, a party P_i in a (sub)protocol Π (one of possibly many to be executed by $\mathcal{Z}$) starts its execution of a given protocol, by having an ITM identified by some ID playing its role activated by $\mathcal{Z}$ writing (SID_Π, x) the string x as its intended input in a protocol Π (identified by session id SID_Π). Other parties' roles are played by other ITMs, running the program intended for the same session ID. [11] The scheduling of messages is asynchronous and is controlled by the adversary (to the extent explained above). The execution by an uncorrupted P_i in a given protocol is resumed once it receives the next prescribed message according to the protocol (on its communication tape). At the end of a protocol's execution, each honest party writes its output to $\mathcal{Z}$'s 'subroutine output tape', making their outputs part of $\mathcal{Z}$'s view.

Corruption is modeled by special 'corrupt' messages, and F is immediately informed regarding the corruption. Since we always consider the dummy adversary whose program is only to perform instructions from $\mathcal{Z}$, we consider $Real_{\Pi,\mathcal{Z}}(k)$ (instead of $Real_{\Pi,\mathcal{Z},\mathcal{A}}(k)$), omitting the specification of $\mathcal{A}$.Corruptions are modeled by having an adversary send a special 'corrupt' message to the newly corrupted party. By $Ideal_{\mathcal{F},\mathcal{Z},\mathcal{S}}(k)$ we denote the output distribution of $\mathcal{Z}$'s output in an ideal world implementation of the functionality F, in the presence of a simulator $\mathcal{S}$, when running with a public security parameter 1^k. By $Ideal_{\mathcal{F},\mathcal{Z},\mathcal{S}}$ we denote the ensemble $\{Ideal_{\mathcal{F},\mathcal{Z},\mathcal{S}}(1^k)\}_{k\in\mathbb{N}}$, by $Real_{\Pi,\mathcal{Z}}$ – the ensemble $\{Real_{\Pi,\mathcal{Z}}(1^k)\}_{k\in\mathbb{N}}$.

Ideal World - evaluating a functionality $\mathcal{F}$ For our purposes, the setting is conceptually similar to the ideal world in the stand alone setting [12]. In particular, the distinguishing entity (i.e., $\mathcal{Z}$) cannot observe the content of messages between the ideal functionality $\mathcal{F}$ and corrupted parties. One difference is that we use an extended notion of a functionality $\mathcal{F}$, which specifies an additional interface with the adversary (possibly of an interactive form), giving it power beyond the prescribed output in case these parties were honest.

Definition 3. *A protocol Π is said to computationally* ***UC-securely compute*** *a given ideal functionality $\mathcal{F}$ against a static adversary, if there exists an ideal-world adversary $\mathcal{S}$ (a simulator), such that for every nonuniform polynomially bounded environment $\mathcal{Z}$ running in the presence of a dummy adversary $\mathcal{A}$, corrupting parties at the outset of the protocol (before sending or receiving any other messages via $\mathcal{A}$),*

$$Ideal_{\mathcal{F},\mathcal{Z},\mathcal{S}} \overset{c}{\approx} Real_{\Pi,\mathcal{Z}}.$$

Similarly, for statistical and perfect security, the indistinguishability notion above is $\overset{s}{\approx}$ and $\overset{p}{\approx}$ respectively.[12] [13]

Definition 4. *A protocol Π is said to realize F with type* ***security with abort*** *(type is either computational, statistical, or perfect) if it type-securely realizes F', defined exactly as F, but it additionally allows the adversary to send $\perp$ after receiving the prescribed output of the corrupted parties, and before sending outputs to honest parties. If $\perp$ was sent, the functionality sends $\perp$ to each honest party, instead of its prescribed output [12, Chapter 7] and [4].*[14]

F.3 Approximate K-out-of-N ROT

Parties in our *PSImple* protocol use Π_{AppROT} – approximate K-out-of-N random OT protocol to obtain garbled Bloom filters for the Bloom filter of length $N = N_{\mathrm{BF}}$ which consists K 1's. We give this protocol in $\mathcal{F}_{AppROT}$-hybrid model in Figure 10. To make the security proof clearer, we explicitly define the default values for the cut-and-choose challenge, and the $\perp$- and "continue"-replies, which are omitted in the main text. The functionality $\mathcal{F}_{ROT}$ is given in Figure 7.

Lemma 1. *The protocol Π_{AppROT} realizes the functionality $\mathcal{F}_{AppROT}$ with statistical UC-security with abort in the presence against static malicious (non-uniform polynomial-time) adversaries in the $\mathcal{F}_{ROT}$-hybrid model.*[15]

Proof. In the following analysis, we do not need to explicitly deal with delaying or deleting messages by $\mathcal{Z}$. This is because in the real protocol, if a message is delayed, then the other party simply waits. Since, this is a two-party protocol, in the ideal-world, the simulator $\mathcal{S}$ can simply emulate this behavior, i.e., wait for the delayed message. Also, we may assume wlog. that the input of the uncorrupted party are provided by $\mathcal{Z}$ before any messages are sent in the protocol. This is so because it is a 2PC protocol,

[11] Note that a given ITM may participate in several sessions concurrently. To distinguish which protocol belongs to which protocol, every message delivered in Π is labeled by SID_Π.

[12] Note that the above requirement is only for real executions which terminate, in the sense that all parties wrote some value to the output tape of $\mathcal{Z}$. Nothing is guaranteed before the execution has terminated.

[13] The standard notion for statistical and perfect security allows the simulator run in polynomial time in the runtime of $\mathcal{A}$, and does not require that $\mathcal{A}$ and $\mathcal{Z}$ are unbounded. In our case we need to limit the number of accesses to the RO of the adversary for security to hold, so for simplicity, we limit $\mathcal{A}, \mathcal{Z}$ to be polynomially bounded. In fact, we could handle somewhat super-polynomial adversaries.

[14] F' is a specific kind of a generalized F functionalities, allowing for extra 'abort after seeing output' capabilities for the adversary.

[15] In fact, this protocol is even secure in the more standard statistical UC-model, where the adversary may be unbounded, and simulator is polynomial in adversaries' runtime.

Protocol Π_{AppROT} in $\mathcal{F}_{ROT}$-hybrid model

Parties: Sender, Receiver.
Parameters: σ – length of the OT strings (computational security parameter); λ – statistical security parameter;
$N = N_{\text{BF}}$ is the number of OTs required; $N_{\text{OT}} > N$ is the number of random OTs to generate;
N^1_{OT}, N_{cc}, $N_{maxones}$ are parameters of cut-and-choose described in Section 4.
K is the number of 1's in Bloom filter of the receiver.
Inputs: B is the choice vector of the receiver ($B[i] \in \{0,1\}$, $i \in [N_{\text{BF}}]$) of size $N = N_{\text{BF}}$ consisting K 1's.

1. **[random OTs]** The sender and the receiver call $\mathcal{F}^{N_{\text{OT}}}_{ROT}$ performing N_{OT} random OTs with Receiver in parallel. The receiver chooses requests $c_1, ..., c_{N_{\text{OT}}}$ with N^1_{OT}s ones among them, and $N_{\text{OT}} - N^1_{\text{OT}}$ zeroes (randomly permuted). As a result, in the jth ROT, the sender learns random strings m_{j0}, m_{j1} (of length σ) chosen by the functionality $\mathcal{F}_{ROT}$. Receiver uses choice bit c_j and learns $m_{j*} = m_{jc_j}$.
2. **[cut-and-choose challenge]** The sender chooses set $C \subseteq [N_{\text{OT}}]$ of size N_{cc} uniformly random and sends C to Receiver.
3. **[cut-and-choose response]** The receiver checks if $|C| = N_{cc}$; if $|C| > N_{cc}$, then truncates it, if $|C| < N_{cc}$, then adds indices by default (for example, $1, 2, ...$). Receiver computes and sends to Sender the set $R = \{j \in C | c_j = 0\}$. To prove that he used choice bit 0 in the OTs indexed by R, it also sends $r^* = \bigoplus_{j \in R} m_{j*}$. The sender replies with $\perp$ if $|C| - |R| > N_{maxones}$ or if $r^* \neq \bigoplus_{j \in R} m_{j0}$, and with "continue" otherwise.
4. **[permute unopened OTs]** Receiver chooses random injective function $\pi : [N] \to ([N_{\text{OT}}] \setminus C)$ such that $B[j] = c_{\pi(j)}$, and sends π to Sender.
   The receiver permutes its random values m_{j*} according the π, and the sender permutes m_{j1} according to π. If π is formed incorrectly (not from the domain $[N_{\text{OT}}] \setminus C$ or not to $[N]$), then use a default value (N consecutive values from $[N_{\text{OT}}] \setminus C$).

Outputs: the receiver has output m_{j*} ($j \in [N]$ such that $B[j] = 1$); the sender has m_{j1}, ($j \in [N]$).

Figure 10: Protocol Π_{AppROT} in $\mathcal{F}_{ROT}$-hybrid model

and the first message received by the corrupted party comes from $\mathcal{F}_{AppROT}$, which requires participation of both parties (but the honest one is waiting).

Security in face of a corrupted receiver. the simulator $\mathcal{S}$, once activated by $\mathcal{Z}$, emulates the protocol towards $\mathcal{Z}$.[16]

1. In the 1st step of the protocol, when emulating $\mathcal{F}_{ROT}$ and obtaining the input $c_1, c_2, ..., c_{N_{\text{OT}}}$ from $\mathcal{Z}$, $\mathcal{S}$ randomly chooses a set $C \subseteq [N_{\text{OT}}]$, where $|C| = N_{cc}$, and computes
$$K'' = \sum_{i \in [N_{\text{OT}}] \setminus C} c_i.$$
   Then, $\mathcal{S}$ passes K'' as an input of the receiver to $\mathcal{F}_{AppROT}$ and receives either $M' = \{m'_1, m'_2, ..., m'_{K''}\}$ or $\perp$.

   If the simulator received $\perp$ from $\mathcal{F}_{AppROT}$, it passes to $\mathcal{Z}$ $\{m_{j*}\}_{j \in N_{\text{OT}}}$ of uniformly random σ-bit strings as the output of $\mathcal{F}_{ROT}$ of the 2nd step of the protocol, sends to it C as the cut-and-choose request and, upon getting the answer, passes $\mathcal{Z}$ $\perp$ as sender's reply, appends $\{m_{j*}\}_{j \in N_{\text{OT}}}$, and halts.

   If $\mathcal{S}$ receives M', for any $j \in [N_{\text{OT}}] \setminus C$, with $c_j = 1$ it computes
$$\psi(j) = \sum_{i \in [N_{\text{OT}}] \setminus C,\ i \leq j} c_i$$
   and the function $\phi \colon [K''] \to [N_{\text{OT}}] \setminus C$, as the inverse of ψ. I.e., $\phi(i) = j$, whenever $\psi(j) = i$ for $i \in [K'']$, $j \in [N_{\text{OT}}] \setminus C$ such that $c_j = 1$. The simulator constructs the injective mapping For any $j \in [N_{\text{OT}}]$ it constructs $m_{j*} = m'_{\psi_j}$, if $j \in [N_{\text{OT}}] \setminus C$ and $c_j = 1$, or $m_{j*} \xleftarrow{R} \{0,1\}^{\sigma}$, otherwise. Simulator gives $\{m_{j*}\}_{j \in [N_{\text{OT}}]}$ to $\mathcal{Z}$ as the output of $\mathcal{F}_{ROT}$ in 1st step.

   The simulator sends C to $\mathcal{Z}$ as the message of 2nd step of the protocol.

2. $\mathcal{S}$ waits for a message from the receiver in 2nd step of the protocol. Upon receiving the response $(R, r*)$ it checks that $|C \setminus R| \leq N_{maxones}$, $r* = \oplus_{j \in R} m_{j*}$, and $c_j = 0\ \forall j \in R$. If not, then passes $\perp$ to the ideal functionality as an input of the

[16] As mentioned above, $\mathcal{A}$ is fixed to just relays messages from $\mathcal{Z}$ to the parties and back. Intuitively, $\mathcal{S}$ attempts to do the same.

receiver and sends $\mathcal{Z}$ $\perp$ as sender's round-3 message. If the receiver passes the check, then sends him Continue and waits for the permutation.

Upon receiving permutation π from $\mathcal{Z}$ in 4th step, $\mathcal{S}$ checks, that $\pi : [N_{\mathrm{OT}}] \setminus C \to [N]$. If not, then uses a default value for π (N consecutive values from $[N_{\mathrm{OT}}] \setminus C$).

If yes, then computes $I' = \begin{pmatrix} i_1 & i_2 & \dots & i_{K'} \\ \pi(\phi(i_1)) & \pi(\phi(i_2)) & \dots & \pi(\phi(i_{K'})) \end{pmatrix}$ where $i_s \in [K'']$ such that $\exists \pi(\phi(i_s))$ and gives it to the ideal functionality as the input of the receiver.

Proof of security in face of a corrupted receiver. Consider an environment running on some fixed public parameters 1^{σ}, 1^{λ}, k, N as input. Let x denote the input vector to all parties given at the outset by $\mathcal{Z}$ to all parties.In step 1, $\mathcal{Z}$ asks $\mathcal{S}$ to send $\mathcal{F}_{ROT}$ the set Q of the requests to $\mathcal{F}_{ROT}$, where $Q' \subseteq Q$ is the set of 1-requests, and the rest are 0-requests. It receives a sequence m_{j*} ($j \in [N_{\mathrm{OT}}]$) of random strings in response (by specification of $\mathcal{F}_{ROT}$). By definition of $\mathcal{S}$, the distributions m_{j*} in the real and ideal worlds are identical (as the inputs x were fixed by $\mathcal{Z}$ to be the same). In more detail, Q' is identical in both worlds. As to m_{j*}, $\mathcal{S}$ picks C and sends K'' to the ideal functionality, where $K'' = |Q' \cap ([N_{\mathrm{OT}}] \setminus C)|$. The emulated m_{j*}'s at locations $j \in Q' \cap ([N_{\mathrm{OT}}] \setminus C)$ are taken from the functionality's step-1 output to receiver if it does not equal $\perp$, and random independent values picked by $\mathcal{S}$ otherwise. In both cases, the other values $\mathcal{S}$ sends emulating replies of $\mathcal{F}_{ROT}$ are random values independent of all others. Now, in the real world, the sender chooses C, and either

$$|C \setminus R| > N_{maxones} \tag{13}$$

holds or not for R induced by C and Q' (for the honest receiver). Since the simulator and sender pick C according to the same distribution in both worlds, that does not depend of receiver's view so far, the probability that (13) is satisfied is identical in both. Then $\mathcal{Z}$ responds with the same R (identical in both worlds). Consider the case when the inequality (13) holds:

- In the real world, the receiver either reported the correct R in which case sender certainly aborts, or reported a larger R so that the equation $|C \setminus R| > N_{maxones}$ no longer holds. In the latter case, there is at least one value $j \in R$, for which m_{j0} is not known to the receiver. Thus guessing r^* expected by the sender occurs with probability at most $2^{-\sigma}$ over the sender's randomness. Overall, the sender aborts in step 3 with probability at least $1 - 2^{-\sigma}$ (over the choice of r). $\mathcal{S}$ also appends $\perp$ to the simulated view as the sender's message.

- In the ideal world, the simulator sets K'' as the number of 1-OT requests in $[N_{\mathrm{OT}}] \setminus C$ on behalf of the receiver in step 1 (of the adversary's interaction with the functionality). With our choice of parameters, according Claim D.1, the evaluation of p_{False} computed for the Bloom filter of length N with K'' 1's in it, is larger than $2^{-\sigma}$ except with negligible (in λ) probability, since (13) holds. Therefore, the ideal functionality sends $\perp$ to both parties and aborts by the end of step 1.

To summarize, the joint view of the adversary and the sender's output is this case is at statistical distance at most $2^{-\sigma} + neg(\lambda)$.

$$Ideal_{\mathcal{F},Z,\mathcal{S}} \overset{s}{\approx} Real_{\Pi,Z} \overset{s}{\approx} (D, \perp). \tag{14}$$

Here D is the distribution over the receiver's view up until step 2 in the real world, as described above.

Now, consider the case when (13) is not satisfied in the real world. If $\mathcal{Z}$ sends R^* (which differs from R induced by its $\mathcal{F}_{ROT}$'s inputs) so that (13) is satisfied for C, R^*, or $r^* \neq \bigoplus_{j \in R} m_{j0}$ the sender outputs $\perp$ and halts immediately. By construction of $\mathcal{S}$, it sends $\perp$ in step 2 as receiver's input, and replies with $\perp$ to both parties. $\mathcal{S}$ again appends $\perp$ as sender's message to the simulated view. Thus, if (14) holds with 0-error (in particular, over the entire support of $Real_{\Pi,Z}$, the sender's output is $\perp$). Otherwise, in the real world, $\mathcal{Z}$ proceeds by picking π (based on its entire view so far), and sends it to the sender, who permutes the values m_{j1} it picked previously according to π. In particular, the m_{j1}'s for the K' j's for which $\pi(j) \in Q'$ are also output to the sender at positions $\pi(j)$, and all other $m_{j'1}$'s output to sender are random values, independent of receiver's view so far (as it never received these values). In the ideal world, $\mathcal{S}$ receives (the same) π from $\mathcal{Z}$, and sets I' sent to the ideal functionality in step 2 in a way that ensures the sender's output at positions $\pi(j)$ for j with $\pi(j) \in Q'$, equal the m_{j1} at this position from the receiver's view. The rest are random independent values, as initially generated by the functionality. We conclude that in this latter case $Ideal_{\mathcal{F},Z,\mathcal{S}} \overset{s}{\approx} Real_{\Pi,Z}$ with 0-error.

Overall, we get a statistical distance of at most $neg(\lambda) + 2^{-\sigma}$ between $Ideal_{\mathcal{F},Z,\mathcal{S}}$ and $Real_{\Pi,Z}$.

Security in face of a corrupted sender. As in the previous case, following the delivery of inputs to *all* the parties by $\mathcal{Z}$ (written by it to their input tapes), the simulator $\mathcal{S}$, once activated by $\mathcal{Z}$, operates as follows.

1. In the first step of the protocol, $\mathcal{S}$ calls the ideal functionality $\mathcal{F}_{AppROT}$ (with no input on behalf of the sender) and receives $M'' = \{m''_1, m''_2, ..., m''_{N_{\mathrm{OT}}}\}$. Then samples the uniformly random $M_0 = \left\{m_{j0}\right\}_{j\in[N_{\mathrm{OT}}]}$, and computes $M_1 = \left\{m_{j1}\right\}_{j\in[N_{\mathrm{OT}}]} = M''$. It gives M_0 and M_1 to $\mathcal{Z}$ as the sender's reply from emulated $\mathcal{F}_{ROT}$s.

2. In 2nd step of the protocol, $\mathcal{S}$ waits for the message C from $\mathcal{Z}$. If $|C| > N_{cc}$, then truncates it, if $|C| < N_{cc}$, then adds indices by default $(1,2,...)$. It samples the number of 1's $|C \setminus R|$ hypergeometrically $HG(N_{\mathrm{OT}}, N^1_{\mathrm{OT}}, |C|)$ (N^1_{OT} is determined in Section 4), computes $|R| = |C| - |C \setminus R|$ and distributes $|R|$ 0-indices uniformly random over the indices from C to build the set $R \subset C$ as the set of indices of 0. Then $\mathcal{S}$ computes $r^* = \bigoplus_{j\in R} m_{j0}$. It gives R and r^* to $\mathcal{Z}$ as the message in 3rd step of the protocol.

   If the simulator receives $\perp$ from $\mathcal{Z}$ in 3rd step, if gives it to $\mathcal{F}_{AppROT}$ and halts. Else gives C to $\mathcal{F}_{AppROT}$ as an input of the sender.

   Upon receiving M, M_* from $\mathcal{F}_{AppROT}$, computes the N-permutation $\pi : [N_{\mathrm{OT}}] \setminus C \to [N]$ at random such that $M = \pi(M'')$ and gives it to $\mathcal{Z}$ as a message in 4th step of the protocol.

Proof of security in face of a corrupted sender. Consider an environment $\mathcal{Z}$ running on some fixed public parameters 1^σ, 1^λ, k, N. Let x denote the input vector to all parties given at the outset by $\mathcal{Z}$ to all parties, as in the previous case. The environment $\mathcal{Z}$ (via $\mathcal{S}$) receives from $\mathcal{F}_{ROT}$ two sets $M_0 = \{m_{j0}\}_{j\in[N_{\mathrm{OT}}]}$ and $M_1 = \{m_{j1}\}_{j\in[N_{\mathrm{OT}}]}$, whose distributions are identical and independent on x in both real and ideal (from $\mathcal{F}_{ROT}$'s emulated by $\mathcal{S}$) worlds by the construction of the simulator.

Then $\mathcal{Z}$ responds with the set $C \subset [N_{\mathrm{OT}}]$ such that $|C| = N_{cc}$ based on (x, M_0, M_1) and receives (R, r^*) in response. R is distributed identical in the real and ideal world by $\mathcal{S}$ construction. As for r^*, it deterministically depends on M_0 and R and therefore is also identical in the real and ideal worlds.

The environment $\mathcal{Z}$ responses with either $\perp$ or $\mathsf{Continue}$, which it chooses basing on his view (x, M_0, M_1, R, r^*), which is, in its turn, depend only on x. If it sends $\perp$, then it receives nothing in response, the execution stops in both real and ideal worlds, and the adversary has $\perp$ as an output. If $\mathcal{Z}$ sends $\mathsf{Continue}$, it receives the N-permutation $\pi : [N_{\mathrm{OT}}] \setminus C \to [N]$. This permutation is random and has the same distribution in both protocol and simulation. In both worlds it is constructed so that it places m_{j1}'s over input indices of the receiver j to the same positions in outputs of the sender and of the receiver.

We conclude, that the joint view of the adversary and the receiver's output is statistically indistinguishable, with 0-error. □

Consistency of $\mathcal{F}_{AppROT}$. Now we show that for the honest sender and honest receiver, Π_{AppROT} protocol realizes $\mathcal{F}_{AppROT}$ ideal functionality. The honest receiver sends to the protocol the choice bits $c_1, ..., c_{N_{\mathrm{OT}}}$, recovers the subset $C \in [N_{\mathrm{OT}}]$ received from the sender and sends the permutation $\pi : [N_{\mathrm{OT}}] \setminus C \to [N]$ such that $\pi(c_1, ..., c_{N_{\mathrm{OT}}}) = B$.

First, note that, if the receiver is honest, with our choice of parameters it passes the cut-and-choose check with the overwhelming probability. Also with the overwhelming probability, it succeed in finding the suitable π, as there is enough 1's among the choice bits remaining after cut-and-choose.

After $\mathcal{F}^{N_{\mathrm{OT}}}_{ROT}$, the sender has $(m_{11}, ..., m_{N_{\mathrm{OT}}1})$ and $(m_{10}, ..., m_{N_{\mathrm{OT}}0})$. The receiver has $\left(m_{1c_1}, ..., m_{N_{\mathrm{OT}}c_{N_{\mathrm{OT}}}}\right)$. Applying the permutation π to $(m_{11}, ..., m_{N_{\mathrm{OT}}1})$ and $\left(m_{1c_1}, ..., m_{N_{\mathrm{OT}}c_{N_{\mathrm{OT}}}}\right)$ accordingly, the sender gets $M = \left(m_{\pi(i_1)1}, ..., m_{\pi(i_N)1}\right)$, and the receiver gets $M_* = \left(m_{\pi(i_1)c_{\pi(i_1)}}, ..., m_{\pi(i_N)c_{\pi(i_N)}}\right) = \left(m_{\pi(i_1)B[1]}, ..., m_{\pi(i_N)B[N]}\right)$, where $i_1 = \min([N_{\mathrm{OT}}] \setminus C)$, $i_N = \max([N_{\mathrm{OT}}] \setminus C)$. Thus, the elements of M and M^* at the indices i such that $B[i] = 1$ collude, and at the other indices differ. As the set of indices i such that $B[i] = 1$, $i \in N$ defines I – the input set of the honest receiver to $\mathcal{F}_{AppROT}$, the receiver has the values from the sender's output set at indices from I, as described by the functionality $\mathcal{F}_{AppROT}$.

F.4 PSI protocol in the hybrid model

Figure 11 describes the PSImple protocol in random oracle, $\mathcal{F}_{AppROT}$-hybrid model. Note that as the hash functions are modeled by the random oracle, the coin-tossing step for hash-seed agreement (Step 1 in Figure 2) is omitted in Figure 11. Additionally, the ideal functionality $\mathcal{F}_{AppROT}$ is not separated into offline and online phases. Furthermore, we need to add a padding after $\mathcal{F}_{AppROT}$, since this functionality does not provide a padding for the receiver's garbled Bloom filter. Note that this padding does not affect security, since the strings in the padding are replaced in the following rerandomization step. For clarity, in Figure 11 we explicitly describe all the $\perp$-replies that parties can send (as we consider security with abort and asynchronous execution).

Protocol of Malicious-secure Multiparty PSI Π_{MPSI} in the $\mathcal{F}_{AppROT}$-hybrid model

Parameters:
n - the maximal size of the input set of the party; σ - computational security parameter; λ - statistical security parameter;
N_{BF} - size of the Bloom filter; $\mathcal{D}$ – a domain of input items;
Inputs: P_i inputs $X_i = \{x_{i1}, x_{i2}, ..., x_{in_i}\}$, $n_i \leq n$ – the set of items from $\mathcal{D}$ ($i \in \{0, ..., t\}$).

Offline Phase:

1. **[(R0) random shares]** Each P_i, $i \in [t]$, sends $S^{il} = (s_1^{il}, ..., s_{N_{\mathrm{BF}}}^{il})$ to any P_l, $l \in [t] \setminus \{i\}$, where $s_r^{il} \xleftarrow{R} \{0,1\}^{\sigma}$, $r \in [N_{\mathrm{BF}}]$.

Online Phase:

2. **[(R1) compute Bloom filters]** P_i ($i \in [t] \cup \{0\}$) computes Bloom filter BF_i of its items from X_i. If $n_i < n$, then P_i computes the Bloom filter of the joint set X_i with $(n - n_i)$ random dummy items.
3. **[(R1) symmetric approximate ROTs]** Parties perform in parallel:
   (a) Using BF_0's 1's indices set J as input, P_0 calls $|J|$-out-of-N_{BF} $\mathcal{F}_{AppROT}$ as the receiver with each of the other parties P_i ($i \in [t]$). As a result, it receives t sets of string $M_*^i[j]$ for each $j \in J$, P_i learns M^i. P_0 let $M_*^i[j] = 0$ for $j \in [N_{\mathrm{BF}}] \setminus J$.
   (b) Using BF_i 1's indices set J as input, each P_i ($i \in [t]$) calls $|J|$-out-of-N_{BF} $\mathcal{F}_{AppROT}$ as the receiver with P_0. As a result, P_i learns $\hat{M}_*^i[j]$ for each $j \in J$, and P_0 receives $\hat{M}^i$. P_i sets $\hat{M}_*^i[j] = 0$ for $j \in [N_{\mathrm{BF}}] \setminus J$.
4. **[(R2) compute and re-randomize GBFs]** If P_0 did not receive $\perp$ from $\mathcal{F}_{AppROT}$, it computes $\mathrm{GBF}^0 = \bigoplus_{i \in [t]} \left(M_*^i \oplus \hat{M}^i\right)$. If P_i did not receive $\perp$ from $\mathcal{F}_{AppROT}$, it computes $\mathrm{GBF}^i = M^i \oplus \hat{M}_*^i$, codewords $y_{ij} = \bigoplus_{r \in h_*(x_{ij})} \mathrm{GBF}^i[r]$ ($j \in [n_i]$) and re-randomizes GBF^i from X_i and codewords y_{ij} ($j \in [n_i]$) according to algorithm **BuildGBF** from C.1.
5. **[(R2) cumulative GBFs of P_is]** If P_i ($i \in [t]$) did not receive $\perp$ from $\mathcal{F}_{AppROT}$, it computes and sends to P_0 the cumulative garbled Bloom filter:
$$\mathrm{GBF}^{i*} = \mathrm{GBF}^i \bigoplus_{l \in [t] \setminus \{i\}} \left[S^{li} \oplus S^{il}\right].$$
   Else it sends $\perp$.
6. **[(R2) cumulative GBF of P_0]** If P_0 did not receive $\perp$ from $\mathcal{F}_{AppROT}$ or from P_i in the previous step, it computes $\mathrm{GBF}^* = \bigoplus_{i \in [t]} \mathrm{GBF}^{i*} \bigoplus \mathrm{GBF}^0$.
7. **[(R2) output]** If P_0 did not receive $\perp$ from $\mathcal{F}_{AppROT}$ or from P_i in the 6th step, it outputs x_{0j} as a member of the intersection, if
$$\bigoplus_{r \in h_*(x_{0j})} \mathrm{GBF}^*[r] = 0, j \in [n_0].$$
   Else it outputs $\perp$.

Figure 11: The PSImple multiparty protocol in the $\mathcal{F}_{AppROT}$-hybrid model

In Lemma 2 and consequently in Lemma 3 and in Theorem 1 we require a non-uniform polynomial-time adversary in sense of polynomially-bounded requests to the Random Oracle. This follows from the next: the union bound of the probability of having at least one false-positive result over $|Q|$ requests is $|Q|p_{False} < |Q|2^{-\sigma}$. To keep it negligible, $|Q| = poly(\sigma)$. In the case of polynomially-bounded (in σ) domain $\mathcal{D}$, this requirement is fulfilled automatically, otherwise (for example, it the typical case of an exponential-size domain) we require a computationally bounded (in σ) adversary in Theorem 1.

Lemma 2. *The protocol Π_{MPSI} securely realizes the functionality $\mathcal{F}_{MPSI}$ with statistical UC-security with abort in presence of static (non-uniform polynomial-time) malicious adversary corrupting up to t parties in the $\mathcal{F}_{ROT}, \mathcal{F}_{AppROT}$-hybrid model, where the Bloom filter hash functions are non-programmable random oracles, and the other protocol parameters are chosen as described in subsection 4.*

Proof. In our protocol and functionality, we take n' such that for the Bloom filter consisting n' or less elements, $p_{False} \leq 2^{-\sigma}$, and for the Bloom filter with $n'+1$ and more elements, $p_{False} > 2^{-\sigma}$. It means, that the malicious receiver in $\mathcal{F}_{AppROT}$ receives $\perp$ from the first step of $\mathcal{F}_{AppROT}$ functionality if and only if its effective Bloom filter consists more than n' items.

Consider the case when evaluating party P_0 is honest, and some subset of other parties $I \subseteq \{\mathsf{P}_1, ..., \mathsf{P}_t\}$ are corrupt.

Simulator description. The simulator $\mathcal{S}$, once activated by $\mathcal{Z}$, emulates $\mathcal{F}_{AppROT}$ towards $\mathcal{Z}$. We stress, that all the corrupt parties are emulated asynchronously, according to the message scheduling decided by $\mathcal{Z}$ - one message at a time. We only describe the simulation by order of steps in the protocol for convenience. In step 1 (preprocessing), $\mathcal{S}$ sends to $\mathcal{Z}$ uniformly random shares S^{il}, as honest P_i's would do according the protocol, to any corrupt party $\mathsf{P}_l \in I$, and learns S^{li}s from $\mathcal{Z}$.

In step 2, $\mathcal{S}$ make queries $Q_i = \{q_{ij} | j \in [n_i]\}$ on behalf of corrupt parties $\mathsf{P}_i \in I$ as requested by $\mathcal{Z}$ (where $n_i \geq n$ is polynomially bounded, as $\mathcal{Z}$ is), to the random oracle to compute Bloom filter's hash-indices.[17] Denote $Q = \cup_{i|\mathsf{P}_i \in I} Q_i$ – the joint query set of corrupt parties.

In step 3, $\mathcal{S}$ plays $\mathcal{F}_{AppROT}$ functionality towards $\mathcal{Z}$ for any corrupt party. Once both inputs of $\mathcal{F}_{AppROT}$ have been requested by $\mathcal{Z}$ to be delivered:

- For P_i acting as a corrupt sender, the simulator samples uniformly at random M''^i of length N_{OT} and gives as the first part of the output to $\mathcal{Z}$. Upon receiving from P_i either $\perp$ or set C, gives to $\mathcal{Z}$ for P_i the vector M^i or $\emptyset$.[18]
- For any P_i acting as a corrupt receiver, upon receiving K'' the simulator samples uniformly at random and gives $\hat{M}'^i$ of length K'', or gives $\perp$, then the simulator receives set of indices J_i (and then can extract the effective Bloom filter $\tilde{\mathrm{BF}}_i$) or $\perp$.

First, all of them receive M''^i's, and then each of them in its turn sends its inputs to $\mathcal{F}_{AppROT}$'s.

The simulator remembers each of the M^i's and computes $\hat{M}^i_*$'s (from $\hat{M}'^i$ and J_i as $\mathcal{F}_{AppROT}$ would compute M in the case of corrupt receiver) for all $\mathsf{P}_i \in I$, and extracts a set of Bloom filters $\tilde{\mathrm{BF}}_i$ for any P_i (the second string of the mapping J_i is the sequence of 1's indices of $\tilde{\mathrm{BF}}_i$), if all the calls to emulated $\mathcal{F}_{AppROT}$ are completed successfully (without $\perp$'s).

In step 4, parties have no interaction, so $\mathcal{S}$ does nothing.

At the 5th step, once all round-1 and round-2 executions have completed, $\mathcal{S}$ observes GBF^{i*}s or $\perp$s sent by $\mathcal{Z}$ on behalf of corrupt $\mathsf{P}_i \in I$.

- If there were no $\perp$'s as an outputs of the simulated $\mathcal{F}_{AppROT}$s or as the messages of the 5th step, $\mathcal{S}$ computes the sum $\mathrm{GBF}^*_I = \bigoplus_{i \in I} \mathrm{GBF}^{i*}$. Now $\mathcal{S}$ can subtract all the secret shares sent and received to corrupt parties on behalf of honest and vice versa:
$$\mathrm{GBF}_I = \mathrm{GBF}^*_I \bigoplus_{\mathsf{P}_i \in I} \bigoplus_{\mathsf{P}_l \in \mathcal{P} \setminus (I \cup \mathsf{P}_0)} \left(S^{il} \oplus S^{li} \right).$$
GBF_I is the effective value of $\bigoplus_{\mathsf{P}_i \in I} \mathrm{GBF}^i$. Now the simulator extracts the effective input of corrupt parties as
$$\tilde{X}_I = \left\{ q \in Q | \bigoplus_{r \in h_*(q)} \mathrm{GBF}_I[r] = \bigoplus_{\substack{\mathsf{P}_i \in I \\ r \in h_*(q)}} (M^i[r] \oplus \hat{M}^i_*[r]) \right\},$$
sends it to the ideal functionality, and receives either $\tilde{X}$ or $\perp$ as the output of $\mathcal{F}_{MPSI}$.
- Else, the simulator sends the effective input of the adversary $\perp$ to the ideal functionality and receives $\perp$ as its output.

[17] For the simplicity, we suppose that $\mathcal{Z}$ makes queries to the random oracle right before the computation of the Bloom filter. After **R1**, when all the $\mathcal{F}_{AppROT}$'s done, the adversary can also make queries, but the probability of the new item is in the existing Bloom filter is p_{False}, which is negligible.

[18] Recall, that according the $\mathcal{F}_{AppROT}$ specification, $M^i = \psi_i(M''^i)$, where $\psi_i : [N_{\mathrm{OT}}] \setminus C \to [N_{\mathrm{BF}}]$ is the uniformly random N_{BF}-permutation.

Simulator Analysis. Consider an environment $\mathcal{Z}$ running on some fixed public parameters 1^σ, 1^λ, t, n, k, N_{BF}. We assume first that *all* parties parties receive inputs from $\mathcal{Z}$ (written by it to their input tapes), at the onset of the execution. We will later show how to get rid of this assumption. Denote by $\mathcal{X} = \{X_i\}_{\mathsf{P}_i \in \mathcal{P} \setminus (I \cup \mathsf{P}_0)}$ – inputs of honest parties. We prove indistinguishability by induction on the message graph of $\mathcal{Z}$ - sent messages to the various parties, and to $\mathcal{F}_{AppROT}$ throughout the execution, starting with the inputs provided, and messages received from honest parties (emulated by $\mathcal{S}$ in the ideal world) are statistically indistinguishable. The induction is on the message number according to the order of message delivery by $\mathcal{Z}$ in the real world (which $\mathcal{S}$ follows). As P_0 is honest, we have to also prove the indistinguishability of the joint view of $\mathcal{Z}$ with the output of the honest P_0 in the simulation and in the real-world execution of the protocol (conditioned in $\mathcal{Z}$'s view, for an overwhelming fraction of the views, as we shall show).

At the start, the (partial) view of $\mathcal{Z}$ is clearly the same in both worlds (as $\mathcal{Z}$ and other parties receive the same public parameters) at the onset of the execution. Clearly, step 1 any value sent and received by an honest party from $\mathcal{Z}$, or sent from an honest party to $\mathcal{Z}$ (a random share of 0) are identically distributed.

$\{\hat{M}'^i/\perp\}_{\mathsf{P}_i \in I}$, $\{M''^i\}_{\mathsf{P}_i \in I}$, $\{M^i/\emptyset\}_{\mathsf{P}_i \in I}$: these messages to $\mathcal{Z}$ are identically distributed to the values received by the corrupted sender/receiver in the real world protocol, by definition of $\mathcal{F}_{AppROT}$, and the fact that at any step of interaction of the $\mathcal{F}_{AppROT}$ instances, the view of each emulated P_i is distributed identically to the real world. Note that in particular, these values do not depend on $\mathcal{X}$.

Let us compare the output distribution of P_0. In the ideal world, $H = \tilde{X}_I \bigcap \left(\bigcap_{\mathsf{P}_i \in \mathcal{P} \setminus I \cup \{\mathsf{P}_0\}} X_i \right)$ is the output of P_0, or $\perp$ if the simulator sent $\perp$ to the ideal functionality.

In case $\mathcal{S}$ did not send $\perp$ to the ideal functionality, H is a subset of the real-world output of P_0, as the honest parties act honestly, and the contribution of the malicious parties does not 'spoil' the equality verified, for each of the items in X_0 that P_0 checks the condition in step 7 for (GBF re-randomization in step 5 by honest parties does not take place in the ideal world, but does not affect their codewords y_{ij}'s, and thus does not affect the condition in step 7). Now, malicious parties may have chosen 1-items in their $\mathcal{F}_{AppROT}$ executions, at locations outside of the query set Q. However, then they either query too many 1's in that $\mathcal{F}_{AppROT}$ execution, in which case, in the ideal world as well, $\mathcal{S}$ notices it, and sends $\perp$ on behalf of $\mathsf{P}_i \in I$ as input to the ideal functionality (and thus we are in a different case than assumed). Otherwise, each corrupted receiver, requests sufficiently few 1's adding any element in the intersection of the honest parties sets $H_1 = \bigcap_{\mathsf{P}_i \in \mathcal{P} \setminus I \cup \{\mathsf{P}_0\}} X_i$ with the probability at most p_{False} (for instance, by using the received $M_i \oplus \hat{M}_i^*$ at all 1-positions in GBF^i, without re-randomizing) for each given party. Since elements not known to all of them are complemented by P_0 by a random string, the probability of adding an element is upper bounded by the probability of a fixed corrupted party P_i adding it, and the result follow. By union bound, adding an element in H_1 by P_i occurs with probability $\leq n \cdot p_{False}$. Assuming no extra elements not covered by Q were added by all malicious parties, for each $x \in X_0 \setminus H$, let j denote some index for which some $\mathsf{P}_l \in I$ did not learn $\hat{M}^i[j]$. The probability of P_0 adding x_{0s} to the output due to passing verification in the step 7 is at most $2^{-\sigma}$, which is the probability of P_i guessing $\hat{M}^i[j]$ by the adversary.

In the simulation, the ideal functionality receives $\perp$ if and only if either it is initialized by the adversary (which has the same probability for any adversarial input) or if the corrupt party's input is larger than n' (which is equivalent because of $\mathcal{F}_{AppROT}$).

Summarizing the above, with statistical distance between real and ideal worlds is at most $neg(\sigma)$.

Finally, consider a situation when the inputs are not provided by $\mathcal{Z}$ at the onset of the protocol, but rather at some intermediate point. We are still able to preserve our indistinguishability invariant, since all $\mathcal{Z}$ sees before deciding to provide an input to some honest P_i are random independent values: either shares picked in step 1, or $\mathcal{F}_{AppROT}$ replies from step 3. Correlating honest parties' inputs with these values does not break the indistinguishability invariant for our protocol and $\mathcal{S}$ above in any way.

Now consider the case of corrupt P_0, and some number of other parties (including zero) $I \subset \{\mathsf{P}_1, ..., \mathsf{P}_t\}$ are corrupt. Note that at least one party is honest.

Simulator description. As before, the simulator interacts with $\mathcal{Z}$ through the dummy adversary. It simulates the replies of $\mathcal{F}_{AppROT}$ and of the honest parties towards $\mathcal{Z}$, by order of its requests. Recall that $\mathcal{Z}$ is responsible for giving corrupted parties their input, and these do not go through $\mathcal{S}$.

In step 1 (preprocessing), $\mathcal{S}$ sends to corrupt parties $\mathsf{P}_l \in I$ uniformly random values S^{il}, as honest P_is would do, and gets S^{li}s to be delivered by corrupted parties P_l from $\mathcal{Z}$.

In step 2, corrupt parties make queries $Q_i = \{q_{ij} | j \in [n_i]\}$ ($\mathsf{P}_i \in I \cup \{0\}$), where $n_i \geq n$ is polynomially bounded, to the random oracle to compute Bloom filter hash-functions.[19] The simulator observes them and computes $Q = \cup Q_i$ – the joint query set of corrupt parties.

[19] In fact, they could make them at the later round 2 as well, for that particular party.

In step 3, the simulator plays $\mathcal{F}_{AppROT}$ functionality for P_0 in its interaction with any honest P_i. Simulating its outputs as follows.

- for P_0 acting as a corrupt receiver in the simulated interaction with any honest P_i, upon receiving K'' the simulator samples M'^i uniformly random of length K'', or $\perp$ as $\mathcal{F}_{AppROT}$ would. then the simulator receives set of indices J_i (and then can extract the effective Bloom filter $\tilde{\mathrm{BF}}_{0i}$) or $\perp$;
- for P_0 acting as a corrupt sender in the simulated interaction with any honest P_i, the simulator samples uniformly at random $\hat{M''}^i$ of length N_{OT} and gives as the first part of the output. Upon receiving from P_0 either $\perp$ or set C, gives to P_0 the vector $\hat{M}^i$ or $\emptyset$.

We stress, that $\mathcal{S}$ delivers messages asynchronously, by the order $\mathcal{Z}$ sends messages to parties of $\mathcal{F}_{AppROT}$, and the above presentation is written (reporting messages from honest parties is done once an honest party requests to deliver a given message).

If the simulated $\mathcal{F}_{AppROT}$'s completed successfully (without $\perp$'s, and in particular did complete at all), the simulator extracts a set of Bloom filters $\tilde{\mathrm{BF}}_{0i}$ from any instance between corrupt P_0 and honest P_i and computes M^i ($\mathsf{P}_i \in \mathcal{P} \setminus \{I \cup \mathsf{P}_0\}$) as the functionality $\mathcal{F}_{AppROT}$ would compute the output for the honest sender (from M'^i and J_i). If and once all $\mathcal{F}_{AppROT}$ executions are completed, $\mathcal{S}$ extracts an effective input of the adversary as $\tilde{X}_I = \left\{q \in Q | \forall s \in h_*(q), \forall j \notin I, \tilde{\mathrm{BF}}_{0j}[s] = 1\right\}$ – queries which are presented in all the extracted Bloom filters of P_0.

After both steps 3a,3b are emulated, when the effective malicious input $\tilde{X}_I$ is extracted, $\mathcal{S}$ sends either it or $\perp$ to the ideal functionality as the input of each of the corrupted parties $\mathcal{F}_{MPSI}$, and receives either $\tilde{X}$ or $\perp$ as the output. In the above, $\perp$ is sent if and only if at least one of the emulated $\mathcal{F}_{AppROT}$'s ended with $\perp$, the simulator gives $\perp$, which is the effective input of the adversary, to $\mathcal{F}_{MPSI}$ and receives $\perp$ from there.

To simulate a step-5 message by an honest party $\mathsf{P}_i \notin I$ replied to $\mathcal{Z}$, right after all the $\mathcal{F}_{AppROT}$'s of P_i are completed, even if there are another running $\mathcal{F}_{AppROT}$'s for other parties:

- If P_i received $\perp$ from $\mathcal{F}_{AppROT}$, $\mathcal{S}$ sends $\perp$ to $\mathcal{Z}$ as the message for P_0.
- If P_i's $\mathcal{F}_{AppROT}$'s are completed successfully, and P_i is not the last honest party whose $\mathcal{F}_{AppROT}$'s done, $\mathcal{S}$ sends a uniformly random GBF^{i*} to $\mathcal{Z}$ as the message for P_0.
- If all the $\mathcal{F}_{AppROT}$'s are completed, but there were at least one $\perp$, and P_i is the last honest party whose $\mathcal{F}_{AppROT}$'s done (without $\perp$), $\mathcal{S}$ sends a uniformly random GBF^{i*} to $\mathcal{Z}$ as the message for P_0.
- If all the $\mathcal{F}_{AppROT}$'s are completed successfully, and P_i is the last honest party whose $\mathcal{F}_{AppROT}$'s done, then $\mathcal{S}$ performs the following:
  - computes Bloom filters BF_j for the set $\tilde{X}$ for all j such that $\mathsf{P}_j \notin (I \cup \mathsf{P}_0)$;
  - computes $\mathrm{GBF}^j = M^j \bigoplus \hat{M}^j$ for all j such that $\mathsf{P}_j \notin (I \cup \mathsf{P}_0)$;
  - computes codewords y_{js} from GBF^j as in the protocol, but only for items $x_{js} \in \tilde{X}$ for all j such that $\mathsf{P}_j \notin (I \cup \mathsf{P}_0)$;
  - computes re-randomized GBF^j for items $x_{js} \in \tilde{X}$ and their codewords y_{js} as in C.1; note, that positions at indices r such that $\mathrm{BF}_j[r] = 0$ are entirely and uniformly random.
  - computes GBF^{j*}_{temp} honestly as in 5th step of the protocol for all j such that $\mathsf{P}_j \notin (I \cup \mathsf{P}_0)$, and $\mathrm{GBF}^{i*} = \bigoplus_{\mathsf{P}_j \notin (I \cup \mathsf{P}_0)} \mathrm{GBF}^{j*}_{temp} \bigoplus_{j \neq i, \mathsf{P}_j \notin (I \cup \mathsf{P}_0)} \mathrm{GBF}^{j*}$ (here GBF^{j*} are messages sent by $\mathcal{S}$ on behalf of other honest parties in 5th step).
  - $\mathcal{S}$ sends GBF^{i*} to $\mathcal{Z}$ as the message for P_0.

Simulator Analysis. Fix a certain $\mathcal{Z}$, running on the public parameters 1^σ, 1^λ, k, N. $\mathcal{S}$ proceeds as follows. We prove indistinguishability by induction on the message graph of $\mathcal{Z}$ - sent messages to the various parties, and to $\mathcal{F}_{AppROT}$ throughout the execution, starting with the inputs provided, and messages received from honest parties (emulated by $\mathcal{S}$ in the ideal world) are statistically indistinguishable. The induction is on the message number according to the order of message delivery by $\mathcal{Z}$ in the real world (which $\mathcal{S}$ follows). At the start, the (partial) view of $\mathcal{Z}$ is clearly the same in both worlds (as $\mathcal{Z}$ and other parties receive the same public parameters) at the onset of the execution. We prove the claim in two steps. First, we consider input distribution of the call graph, with messages corresponding to steps 1-4 for some given party, and step 5, *before* the last honest party sends its step-5-message. In the second part we analyze the last message delivered in step 5. Let us first assume $\mathcal{Z}$ hands all inputs to honest parties at the onset of the protocol (we later explain how to get rid of this assumption).

In step 1, as in the previous case, $\{S^{il}\}_{\substack{\mathsf{P}_i\in\mathcal{P}\setminus(I\cup\mathsf{P}_0)\\ \mathsf{P}_l\in I}}$ sent from honest parties are random i.i.d strings (sampled by $\mathcal{S}$ in ideal world), and have the same (uniform) distribution in both ideal and real worlds. In step 2, $\{M'^i/\perp\}_{\mathsf{P}_i\in\mathcal{P}\setminus(I\cup\mathsf{P}_0)}$, $\{\hat{M}''^i\}_{\mathsf{P}_i\in\mathcal{P}\setminus(I\cup\mathsf{P}_0)}$, $\{\hat{M}^i/\emptyset\}_{\mathsf{P}_i\in\mathcal{P}\setminus(I\cup\mathsf{P}_0)}$: these are identically distributed to the values received by the corrupted sender in the real world protocol, by definition of $\mathcal{F}_{AppROT}$, and the fact that at any step of interaction of the $\mathcal{F}_{AppROT}$ instances, the view of each emulated P_i is distributed identically to the real world. In both cases (by inspection) when one of the parties is corrupted, the output of $\mathcal{F}_{AppROT}$ does not depend on the input of the honest party, and is properly emulated by $\mathcal{S}$ above. As the input to the $\mathcal{F}_{AppROT}$'s are distributed identically (resulting from the same $\mathcal{Z}$), so are the outputs. To see this, note that when a round-2 (step 5) message from one or more honest party was not yet sent when another honest party $\mathsf{P}_{l'}$ sends its step-5 message and other honest parties P_l have not contributed their step-5 shares $\oplus_{j\notin I\cup\{\mathsf{P}_0\}\cup\{\mathsf{P}_l\}}S^{jl}$ yet, they send an additive share of the final sum (the distribution of which we analyze below). In particular, if an honest P_l has not received its step-3 output share, it in particular hasn't sent its step-5 message yet, and every other party is not the last, and the value the latter would sent is a random independent sting (share). Calls to the *RO* also don't break indistinguishability, because they actually refer to the same RO in both worlds.

Let us now compare the effect of step 5, assuming all executions of $\mathcal{F}_{AppROT}$ with honest parties have completed. Assume first they have completed without any $\perp$'s in the real world. In this case, (after canceling the shares S^{il} contributed by the malicious parties P_i, which are known to $\mathcal{Z}$), $\oplus_i GBF^{i^*}$ sent in the real world by honest parties, along with 0-shares $\bigoplus_{i\notin I}S^{il}$ for $l\in I$ are random additive shares of a randomized GBF, G, containing the intersection of honest parties' input with $\tilde{X}$, $H=\tilde{X}\bigcap\left(\bigcap_{\mathsf{P}_i\in\mathcal{P}\setminus I\cup\{\mathsf{P}_0\}}X_i\right)$, encoded via the corresponding $\bigoplus_i\left(M^i\oplus\hat{M}^i\right)$, at entries in $\{h_*(q)|q\in H\}$ with overwhelming probability. Such a GBF, G, has fixed sums at the locations corresponding to elements of H (determined by M, $\hat{M}$), and is random otherwise.[20] The overwhelming probability is due to two observations. (1) $G[j]$ is random for every j not in $\cup_{x_i}(h_*(x_i))$ for some X_i where P_i is honest, as $\mathcal{Z}$ does not know the corresponding $\hat{M}^i[j]$ used by it (randomly complemented by P_i upon rerandomizing this 0-entry in step 5). (2) If (1) does not happen, for j outside of a set $h_*(x)$ we queried in 3(a) as 1's by P_0 for some element x, $G[j]$ is distributed uniformly at random, due to the fact that $M^i[j]$ is not learned by $\mathcal{Z}$. (3) Words x used by P_0 in step 3a (from all parties) on which the RO was not queried (resulting in no indices from the first or second kind). As no $\perp$ occurred in any $\mathcal{F}_{AppROT}$ call, the probability of this is $\leq p_{False}<2^{-\sigma}$.

Now let us now consider the case when at least one of the $\mathcal{F}_{AppROT}$'s resulted in $\perp$ in the real world. In this case $\mathcal{S}$ sends $\perp$ to $\mathcal{F}_{MPSI}$, and thus receives $\perp$. The simulation of step 5's messages is perfect in this case, since at least one of the shares is not delivered by at least one honest parties for each GBF entry, resulting in random i.i.d entries.

Finally, let us address a situation where $\mathcal{Z}$ does not give input to (at least) one of the parties until a certain point in the protocol. In this case, by analysis similar to the above, all $\mathcal{Z}$ sees in the real execution of our protocol until the last party receives its input and advances through the protocol to complete step 5, are random values (crucially, as the values provided by $\mathcal{F}_{AppROT}$ are freshly random in each execution, and only the locations of the values selected can be influenced by receiver's input). $\mathcal{S}$ emulates this distribution perfectly. In particular, all of this happens during the first phase of our call graph construction. Random independent values are again obtained in step 5 due to part of the shares contributed by the stalled party missing.

□

F.5 Proving Theorem 1

Using the UC framework of [4], to prove Theorem 1, it suffices to prove the following.

Lemma 3. *The protocol Π_{MPSI} securely realizes the functionality $\mathcal{F}_{MPSI}$ with statistical UC-security with abort in presence of static (non-uniform polynomial-time) malicious adversary corrupting up to t parties in the random OT, $\mathcal{F}_{ROT}$-hybrid model, where the Bloom filter hash functions are non-programmable random oracles, and the other protocol parameters are chosen as described in subsection 4.*

Proof. Follows from Lemmas 1 and 2. □

Finally, Theorem 1 follows directly from Lemma 3, in instantiating $\mathcal{F}_{ROT}$ via a UC-secure OT protocol, and applying the UC composition theorem. Note that in the resulting protocol, the "offline" part appearing in the protocol in Figure 2 indeed line up at the beginning of our protocol, and all computation performed there does not depend on the inputs. Therefor, following instantiation of $\mathcal{F}_{ROT}$, we may move the corresponding parts to an offline phase. To formally allow for receiving inputs only at the onset of the online phase, we may reformulate the protocol as a sequence of a par of protocols, where the offline phase has no inputs and a distinct *SID*, while the online part has inputs, and a different *SID* (both executed by the same set of parties).

[20] In other words, this is a solution to a certain linear equation system over the *field* $\mathbb{F}_{2^\sigma}$, since the coefficients of the system are in $\mathbb{F}_2$.

G Complexity Analysis

In Table 2, we compare the communication complexity for the evaluating party with other multiparty PSI protocols. We use the relations $k = \Omega(\sigma)$, $N_{\text{OT}} = O(nk)$, $N_{\text{BF}} = O(nk)$, as follows from our parameter settings. The communication complexity is as good as in [38], which is not fully malicious-secure, and just slightly lower than [17], which is semi-honest.

The communication load of the evaluating party is the highest. For other parties, it is t times lower at the online-phase, and it is $O(tn\sigma^2)$ at the offline-phase due to the necessity of sending and receiving the random secret shares. With the optimization random seeds rather than random shares are sending, the communication complexity in offline-phase for P_0 remains the same (as it's not involved in the secret-sharing step), and for P_i it is $O(n\sigma^2)$, since dominated by random OTs rather than secret-sharing. Thus, with such an optimization, the asymptotic communication complexity of P_i is t times lower that P_0.

In Tables 10 and 11 we present the communication and computational complexity by operations and transactions of our protocol. Tables are split into P_0, who is an evaluating party, and P_i, who are other parties $(\mathsf{P}_1, ..., \mathsf{P}_t)$. P_0 performs $2t$ instances of Π_{AppROT}: t as a sender, and t as a receiver, and P_i performs two instances: one as a sender, and one as a receiver. As Π_{AppROT} instances dominate the communication and computation complexity at the online-phase, we give Π_{AppROT} complexity below.

The offline phase of Π_{AppROT} is dominated by random OTs with communication complexity $O(n\sigma^2)$. The ROT-extension of Keller, Orsini, and Scholl [19] has the communication complexity $\sigma N_{\text{OT}} + 10KB$ for $\lambda = 40$. The overwhelming amount of data on the online phase consists of sent and received permutations over unopened OTs. P_0 performs $2t$, and P_i – 2 instances of Π_{AppROT}. At the offline phase, each P_i sends and receives $t-1$ additive secret shares of GBF^i, each of size $N_{\text{BF}}\sigma$ bits.

Table 10 "Communication complexity" consists of the number of bits that are sent or received by the party on the particular step of the protocol. Based on this table, we give an asymptotic evaluation of the communication complexity with respect $N_{\text{OT}} \approx N_{\text{BF}} = O(nk) = O(n\sigma)$ and $N_{cc} << N_{\text{BF}}$.

For P_0, the offline phase is dominated by random OTs with communication complexity $O(tn\sigma^2)$. The overwhelming amount of data $O(tn\sigma \log_2(n\sigma))$ on the online phase is sent and received permutations over unopened OTs. For P_i, the offline phase is dominated by $2(t-1)$ sent and received random secret shares of GBFs with the communication complexity $O(tn\sigma^2)$. In the online phase, the overwhelming amount of data $O(n\sigma^2)$ is the transmission of GBF^{i*} to P_0, and $O(n\sigma \log(n\sigma))$ is the permutations of Π_{AppROT}s that P_i sends and receives to/from P_0.

The main components of a computational cost are summed in Table 11. The number of Random OT's for P_0 is $2t$: t ROT as a sender, and t as a receiver, for P_is it's 2. The ROT-extension, proposed by Keller, Orsini, and Scholl [19] requires $2N_{\text{OT}} + 336$ hashes that constitutes its main computational cost. So, in general, the computational cost for P_0 is dominated by $O(tnk)$ hash computations, and $O(tnk)$ XORs of σ-bit strings, and – for P_i – $O(nk)$ hash computations, $O(tnk)$ PRG accesses for σ-bit strings. Thus, for P_0 in offline-phase, we got $O(tnk)$ hashes, and in online phase nk hashes, $O(tnk)$ XORs and t permutations. P_i in the offline phase has to perform $O(nk)$ hashes, and $O(tnk)$ PRG accesses; in online phase – nk hashes, $O(nk)$ PRG accesses and $O(tnk)$ XORs of σ-bit strings. Note that finding of intersection is linear in the number of parties both in communications and computations.

	Party	
Step	P_0	P_i
Offline-phase		
Random OTs	$4tn\sigma^2$	$4n\sigma^2$
Cut-and-choose challenge	$2tN_{cc}\log_2 N_{\text{OT}}$	$2N_{cc}\log_2 N_{\text{OT}}$
Cut-and-choose response	$2t(N_{cc}\log_2 N_{\text{OT}} + \sigma)$	$2(N_{cc}\log_2 N_{\text{OT}} + \sigma)$
Secret shares	–	$2(t-1)N_{\text{BF}}\sigma$
Online-phase		
Permutation	$2tN_{\text{BF}}\log_2 N_{\text{OT}}$	$2N_{\text{BF}}\log_2 N_{\text{OT}}$
Sending/receiving GBF^{i*}	$tN_{\text{BF}}\sigma$	$N_{\text{BF}}\sigma$

Table 10: Number of sent and received bits

	Party	
Computation	P_0	P_i
Offline-phase		
Hashes for random OTs	$2tN_{\mathrm{OT}}$	$2N_{\mathrm{OT}}$
PRG of σ-bit strings to compute secret shares	-	$(t-1)N_{\mathrm{BF}}-n$
Online-phase		
Hashes for Bloom filter	nk	
Performed permutations	$2t$	2
XORs of σ-bit strings for codewords and cumulative GBFs	$nk+2tnk$	
PRG of σ-bit strings for re-randomization of GBF	-	$N_{\mathrm{BF}}-n$
XORs of σ-bit strings for the intersection	$2tN_{\mathrm{BF}}+nk$	-

Table 11: Number of performed operations