PSImple: Practical Multiparty Maliciously-Secure Private Set Intersection

Aner Ben Efraim
Ariel University
Department of Computer Science
Ariel, Israel
anermosh@post.bgu.ac.il

Olga Nissenbaum
Ariel University
Department of Computer Science
Ariel, Israel
olga@nissenbaum.ru

Eran Omri
Ariel University
Department of Computer Science
Ariel, Israel
omrier@ariel.ac.il

Anat Paskin-Cherniavsky
Ariel University
Department of Computer Science
Ariel, Israel
anatpc@ariel.ac.il

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. Private set intersection 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, despite the high interest in this problem, prior to 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 construction is based on oblivious transfer and garbled Bloom filters. To demonstrate the practicality of the PSImple protocol, we implemented the protocol and ran experiments with up to 32 parties and 2^{20} inputs. We show that PSImple is competitive even with the state-of-the-art concretely efficient *semi-honest* multiparty PSI protocols.

Additionally, we revisit the garbled Bloom filter parameters used in the 2-party PSI protocol of Rindal and Rosulek (Eurocrypt 2017). Using a more careful analysis, we show that the size of the garbled Bloom filters and the number of oblivious transfers required for malicious security can be significantly reduced, often by more than 20%. These improved parameters can be used both in the 2-party PSI protocol of Rindal and Rosulek and in PSImple.

KEYWORDS

private set intersection, secure multiparty computation, concrete efficiency, malicious security, UC-security, 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: As pointed out by Kolesnikov et al. [28], MPSI is useful for targeted advertising, where several organizations wish to combine their data to find a target audience for an ad campaign and test the advertisement efficacy, and for network monitoring, where a set of enterprises which have private audit logs of connections to their corporate networks, and wish to identify similar activities in all networks.

PSI is also useful for collaborative botnet attack detection and collaborative intrusion detection, where agencies aim to find suspicious IP addresses by looking at the intersection of their user IP sets or their suspected IPs, without disclosing their entire user list or the users they suspect. Additional possible uses of PSI include computing proximity testing that informs only if necessary (e.g., for COVID-19 contact tracing), and many more.

Despite the many uses and high interest in this problem, prior to this work there has been no concretely efficient maliciously secure multiparty PSI protocol. Note that while threshold PSI might be more appropriate for some applications, concretely-efficient maliciously secure threshold PSI is still an open problem, so a maliciously secure multiparty PSI can be used instead.

Indeed, PSI is a special case of secure multiparty computation (MPC), allowing a set of parties to perform some computational task 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, which we focus on, 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 fast 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 exponentiations, which result in large constants. For example, Rindal and Rosulek [39] presented

a 2-party maliciously-secure PSI protocol, based on oblivious transfer[1] and garbled Bloom filters, and showed that it is more than an order of magnitude faster than the protocol of [13], which is based on Diffie-Hellman, even though the latter requires significantly less communication.

Concretely efficient *generic* MPC protocols (e.g., [2, 12, 18, 24, 41]) are less suitable for the PSI problem than PSI-tailored protocols, as their complexity highly depends on the circuit size, which is large for PSI (typically, incurring a slowdown of two orders of magnitude, see [28]). Over the last decade, substantial research has been dedicated to the construction of *concretely efficient* PSI protocols. However, these protocols were either restricted to the two-party setting (e.g., [13, 14, 25, 30, 34, 35, 37, 39, 40]) or were restricted to deal with semi-honest adversaries [21, 28]. 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, which is secure against malicious adversaries and scales well with the number of parties and the data set size.

1.1 Review of Prior Works Based on GBFs

In this work, we present the PSImple protocol. This construction relies on two main cryptographic primitives, oblivious transfer (OT) [36] and garbled Bloom filters (GBF) [14]. A K-out-of-N oblivious transfer allows a receiver to interact with a sender, holding N strings $s_1, \ldots, s_N$, 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) [3] 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 $h_1, \ldots, h_k$. An element $x \in \mathcal{D}$ is encoded into the BF by setting all indices $h_1(x), \ldots, h_k(x)$ in the BF to be 1. We next review the existing ideas for PSI protocols based on Bloom filters, but first note that the domain $\mathcal{D}$ is assumed to be very large (e.g. 2^{128}), so using K-out-of-N OT directly on the domain size is infeasible.

The first naïve solution is the following: Say that two parties $\mathsf{P}_0, \mathsf{P}_1$ wish to compute the intersection between their respective sets S_0 and S_1. First, each party constructs a BF, according to its private set. Then, they engage in an OT protocol so that P_0 (as the receiver) learns the values of P_1's BF, but only in the indices that are 1 in P_0's BF. Finally, by computing the bit-wise AND over the two BFs, P_0 can learn the BF of the intersection. While this protocol is correct, it is not secure even in the semi-honest setting, as P_0 may learn about 1 value indices in P_0's BF, even if they were set to one on account of elements that are not in the intersection (but are in P_1's set). Thus, this naïve solution leaks additional information.

Two-party semi-honest PSI of [14]. Dong et al. [14] overcame the leakage in the above naïve solution, by introducing a new variant of Bloom filters, called *garbled Bloom filters* (GBF). A GBF is attributed with same k hash functions as its respective BF, but in each coordinate of the GBF there is a σ long random string (instead of a single bit as in BF). The strings of a GBF are chosen independently and uniformly, with the only requirement being that for any element x in the underlying set, the sum (XOR) of the strings in the GBF corresponding to the indices $h_1(x), \ldots, h_k(x)$ equals to some value y_x, called the codeword of x. In [14], y_x was predetermined to be $y_x = x$.

The protocol of [14] follows roughly as the above naïve solution, with the main 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. Then, P_0 can test whether an element x is in the intersection by checking if the sum of all the strings in indices $h_1(x), \ldots, h_k(x)$ (which it got from P_1's GBF) equals $y_x = x$. 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.

Note that this construction is secure only for semi-honest adversaries, as a malicious P_0 can also ask for indices that are 0 in its Bloom filter, and a malicious P_1 can create a GBF corresponding to significantly more than n elements.[2]

Multiparty semi-honest PSI of [21]. Inbar et al. [21] extended the work of [14] to the multiparty setting for *augmented* semi-honest security,[3] with $t+1$ parties $\{\mathsf{P}_0, \mathsf{P}_1, \ldots, \mathsf{P}_t\}$, by using additive secret sharing. Specifically, P_0 computes the Bloom filter of its input set, and each other party P_i computes a GBF for its set, but using a variant of GBF where $y_x = 0$ for every element x. Then, each P_i shares its GBF among all the parties and sums all the shares it received from the other parties. It follows by the linearity of the secret-sharing scheme and the GBF variant used, that the parties now hold shares for the GBF of the intersection, i.e., $y_x = 0$ if $x \in \cap_{i\in\{1,\ldots,t\}} S_i$. Next, each party engages in an OT protocol with P_0, to allow P_0 to learn the coordinates from the share of P_i that correspond to 1 in P_0's BF. Thus, for every x in its input set S_0, it receives the indices $h_1(x), \ldots, h_k(x)$ of the share of each party. Then, by reconstruction of the shares and checking whether $y_x = 0$, P_0 learns if x also belongs to $\cap_{i\in\{1,\ldots,t\}} S_i$, in which case $x \in \cap_{i\in\{0,\ldots,t\}} S_i$.

We again observe that this construction is secure only for *augmented semi-honest* adversaries, as a malicious P_0 can also ask for indices that are 0 in its Bloom filter, and a malicious $\mathsf{P}_i, i \geq 1$ can create a GBF corresponding to significantly more than n elements.[4]

Two-party malicious PSI of [39]. Rindal and Rosulek [39] presented an efficient translation of Dong et al.'s 2-party protocol to the malicious setting:

In the first part, as in [14], for each $x \in S_0$, P_0 computes y_x as the sum of the indices $h_1(x), \ldots, h_k(x)$, which it receives using OT. However, to ensure that P_0 does not ask for significantly more indices than the number of 1s in its BF, they used a *maliciously secure, approximate* K-out-of-N OT protocol. Rindal and Rosulek [39] show that this ensures that a malicious P_0 can only

[1] While OT is based on public key operations, modern MPC protocols use OT *extension* [22], in which only a small amount of OTs require public key operations, and the rest are generated using symmetric-key primitives.

[2] In MPC, malicious parties may choose their inputs. In the context of PSI, however, security should still ensure some bound on the size of their input sets.

[3] An augmented semi-honest adversary is an adversary that may choose to change its input, but then follows the prescribed protocol honestly.

[4] This protocol is also not semi-honestly secure because a corrupt P_0 may cancel out the effect of the inputs of other corrupt parties ([21] also have a semi-honestly secure protocol, but it is less efficient). To prove augmented semi-honest security, the simulator sets the inputs of all corrupt parties to be the same as P_0's input.

use a slightly larger input set than the bound for honest parties. For more details, see Section 2.

In the second part, to prevent a malicious P_1 from using a larger set than the bound for the honest parties, they use the following idea: In P_1's GBF, rather than having a predetermined value y_x as in [14, 39], y_x is chosen randomly by P_1. After P_0 has learned (approx.) K of the strings in P_1's GBF from the first part above, P_1 sends to P_0 the codewords y_x for every element x in its input set. This puts a firm limit on the number of elements P_1 can use.

Note that for security reasons, the codewords sent in the second part cannot be labelled (i.e., P_1 cannot reveal that y_x corresponds to x, as this would reveal that $x \in S_1$); in fact, they must be sent in a random order. Therefore, in order for P_0 to find the intersection, it needs to check for each $x \in S_0$ if there exists a codeword sent in the second part that is equal to y_x it computed in the first part. The task of checking for all n inputs if their codeword was sent can be done in $O(n \log n)$ (using sorting) or expected $O(n)$ (using hash tables). However, as we shall see, this task becomes hard when moving to the multiparty case.

1.2 Contributions

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, incorporated several code optimizations, and ran experiments with up to 32 parties and 2^{20} inputs to show the practicality of PSImple. The results show that PSImple is competitive even with the state-of-the-art concretely efficient multiparty PSI protocols [21, 28, 42], which achieve weaker security guarantees, and orders of magnitude faster than using a direct extension of the protocol of [39] to the multiparty setting.
(3) We revisit the parameter analysis of previous works on efficient PSI, based on garbled Bloom filters (GBF). Performing a careful analysis, we were able to reduce the number of required oblivious transfer (OT) calls by up to 25%.

We next elaborate on each of these contributions. However, before this, we explain why directly extending [39] using the ideas from [21] does not work. Note that although this idea is "natural", it is already non-trivial. We next show that while such a direct solution is possible, it results in an inefficient protocol.

Direct combination of [39] and [21]. The direct protocol works as follows: Extending [39], each party $P_i, i \geq 1$ creates a GBF with random codewords y_x, and P_0 performs an (approximate, random) K-out-of-N OT with each of the other parties separately (i.e., the first part of the [39] 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, even unlabelled in a random order, as P_0 could learn the intersection of its set with the set of each other party separately, which is not allowed.

Therefore, to complete the protocol, the parties $\mathsf{P}_1, \ldots \mathsf{P}_t$ secret share their GBFs using additive secret-sharing, and sum the received shares from all the parties. By linearity, the parties now hold shares of the sum of their GBFs. Then, the parties send the codewords y'_x of their shares (i.e., y'_x is the sum of the indices $h_1(x), \ldots, h_k(x)$ in their share) to P_0.

The crux of the idea, is that for any element $x \in \cap_{i \in \{0,\ldots,t\}} S_i$, the sum of all the codewords y_x received by P_0 from the other parties using the OT in the first part, is equal to the sum of the codewords y'_x of all the parties sent to P_0 in the second part; this follows from the linearity of the secret-sharing scheme.

However, the problem is that in order for P_0 to check if x is in the intersection, it needs to find codewords y'_x, one from each party, such that their sum is equal to y_x; recall that for security reasons, these y'_xs need to be sent unlabelled and in a random order. Unfortunately, this problem is hard – all known algorithms for finding these codewords grow exponentially in the number of parties, i.e., $n^{O(t)}$. Hence, this protocol is inefficient.

The *PSImple* protocol – A multiparty PSI protocol in the malicious model. As we saw, trying to directly combine the ideas from [39] and [21] results in an inefficient protocol. Therefore, in order to construct PSImple we take a different path, first revisiting the 2-party construction of [39].

A new two-party malicious PSI. We observe that a key idea in the two-party malicious PSI protocol of [39] 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 P_1, the binding effect comes from the fact that P_1 can only send a fixed number of codewords to P_0, but this results in an exponential blowup when moving to the multiparty setting.

In our two-party malicious construction, the parties start in the same way as in [39], 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 [39], this binds P_0 to choose a bounded subset of coordinates from P_1's GBF.

Next we need to similarly bind P_1, but without sending the codewords as in the direct construction. Therefore, in a second phase, the parties perform the first part again, with switched roles. I.e., P_0 constructs a GBF and the parties then perform an (approx., 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 of P_0.

The important observation is that by summing their GBF with the strings received from the OT (in the corresponding indices), they now both hold GBFs that agree on the codewords of elements in the intersection. In other words, they now hold additive shares of a GBF in which for every element x in the intersection, the codeword y_x is 0. This, however, leads to a new difficulty – P_1 cannot just send his GBF to P_0, as this 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 *rerandomizing a GBF*, and present a simple and efficient rerandomization algorithm. That is, given a GBF G for a set S, the rerandomization algorithm selects a uniformly random GBF G' that agrees with G on all the codewords for elements in S. We can thus complete the protocol by P_1 sending to P_0 a rerandomized version of the GBF it obtained. Then P_0 sums the received rerandomized GBF with its own GBF. Since rerandomization does not affect the codewords of the elements in the set, the above observation remains valid. Hence,

P_0 can recover the intersection by checking, for each element x in its input set, if $y_x = 0$ in the sum of the GBFs.

We note that this alternative protocol more than doubles the communication compared with [39], and hence we do not recommend to use it in the 2-party case. However, this protocol avoids the task of finding codewords sent from P_1 that are equal to the codewords computed by P_0, and replaces it with the tasks of rerandomizing the GBF and of summing the garbled Bloom filters. While this has little effect in the two-party case, in the case of more than two parties, the saving is drastic – the above direct extension of [39] to the multiparty setting requires finding, for each input x, codewords y'_x (one from each party) that sum to y_x, the codeword computed by P_0. To the best of our knowledge, the best solution to this problem is still exponential in the number of parties, i.e., $n^{O(t)}$. In contrast, rerandomizing the GBF is independent of the number of parties and summing the garbled Bloom filters only grows linearly with 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 sums 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 sum 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.

By generalizing our observation in the 2-party case, it follows that $G_0 \oplus \bigoplus_{i\in[t]} G_i^*$ is a GBF for the intersection of all the parties, where $y_x = 0$ for every element x in the intersection. 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_0 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 4 to 32 parties and input size of 2^8 to 2^{20}, in order to demonstrate the practicality of PSImple, and analyzed the runtime to understand the asymptotics and the cost of the various steps. In table 2 we compare the runtimes, on the same platform[5], with existing implementations of state-of-the-art multiparty PSI protocols [21, 28, 42], which give a significantly weaker security guarantee. As PSImple achieves security against a malicious adversary, 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. Additionally, we compare PSImple with our implementation of the direct extension of [39] to the multiparty setting (denoted by *Direct* in Table 2), and show that PSImple is orders of magnitude faster.

Improved analysis and choice of parameters. The approximate K-out-of-N oblivious transfer (OT) protocol of [39] uses the cut-and-choose technique over N_{OT} executions of 1-out-of-2 OTs to allow the receiver to learn approximately K indices from the sender's GBF of length . Specifically, some of the strings selected in the OTs are revealed to prove honest behavior. Performing a more careful analysis of the parameters of the garbled Bloom filter and the cut-and-choose process, we reduced the number of required 1-out-of-2 OTs in the protocol by up to 25% compared to [39]. Table 1 demonstrates some examples for the actual parameter choices compared to [39]. Since OT is often the bottleneck in PSI protocols based on GBFs, this improvement has a great effect on the overall efficiency of these protocols. In particular, our analysis directly improves PSImple, as well as the protocols of [39, 42].

Table 1: Comparison of Π_{AppROT} parameters for different input set sizes n.
Parameters: k – number of Bloom filter hash-functions
N_{BF} – Bloom filter length
N_{OT} – number of OTs

n	Analysis	k	N_{BF}	N_{OT}
2^8	[39]	94	88,627	99,372
	This work	147	**64,733**	**74,379**
2^{12}	[39]	94	1,121,959	1,187,141
	This work	134	**851,085**	**901,106**
2^{16}	[39]	91	16,579,297	16,992,857
	This work	131	**12,660,342**	**12,948,963**
2^{20}	[39]	90	257,635,123	260,252,093
	This work	129	**197,052,485**	**198,793,103**

1.3 Additional Related Work

Currently, the state-of-the-art in two-party maliciously secure PSI are the protocols of [40] and [30], both concretely-efficient, have quasi-linear and linear communication complexities, respectively, and are almost as efficient as the fastest semi-honest PSI protocol [27]. The benchmarks made in [30] suggest that it is currently the fastest two-party, maliciously secure, PSI protocol. We remark that [30] 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 [30] can also be extended to the multiparty setting.

Apart from [21], an additional PSI protocol in the *semi-honest multiparty* setting is the protocol of [28], which is based on symmetric-key techniques. The protocol of [28] is significantly faster than [21]

[5] Zhang et al. [42] did not publish their implementation. Therefore, we used their reported results instead of measuring their runtime.

Table 2: Total runtime comparison of PSImple with other multiparty PSI protocols.
Key: – an unreported result[5]
*** the protocol crashed**
**** Direct crashed on 4 parties and 2^{16} inputs; on 4 parties and 2^{14} inputs it took 109.055 seconds, while PSImple finished in 1.723 seconds**

# parties	protocol	sec. model	n = 2^8	n = 2^{12}	n = 2^{16}	n = 2^{18}	n = 2^{20}
4	[21]	augmented semi-honest	1.626	2.113	9.909	36.702	*
	[28]	semi-honest	**0.075**	**0.277**	**3.642**	**12.095**	**47.687**
	[42][5]	non-standard	0.76	2.95	40.91	–	513.27
	Direct	**malicious**	0.194	5.474	**	*	*
	PSImple	**malicious**	0.201	0.550	6.623	27.086	128.248
6	[21]	augmented semi-honest	1.611	2.257	10.213	39.473	*
	[28]	semi-honest	**0.116**	**0.383**	**6.057**	**21.804**	**82.643**
	Direct	**malicious**	5.135	*	*	*	*
	PSImple	**malicious**	0.237	0.580	6.906	30.117	141.277
10	[21]	augmented semi-honest	1.711	2.230	14.259	56.842	*
	[28]	semi-honest	**0.161**	**0.748**	11.718	46.345	174.908
	PSImple	**malicious**	0.302	0.868	**7.963**	**31.378**	**147.292**
16	[21]	augmented semi-honest	1.746	2.714	19.296	*	*
	[28]	semi-honest	**0.252**	1.122	16.626	61.019	241.864
	PSImple	**malicious**	0.367	**0.911**	**13.177**	**47.334**	*

for a small number 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 early works [9, 26] have high asymptotic communication complexity. Additionally, they use expensive homomorphic encryptions, polynomial interpolations and evaluations, and zero-knowledge proofs, which are known to take a very long time in practice. The recent works of [19] and [15] both have very good asymptotic communication complexity, but were not implemented and are most likely not concretely efficient as well: The solution of [19] requires $O(n^2)$ exponentiations, rendering it impractical for real-world scenarios.

The solution of [15] requires expensive interpolation and evaluation, which were shown (e.g., in [30]) to often be even more dominant in runtime than communication. Additionally, party P_0 in [15] performs $8tn$ OLEs, and the OLE protocol itself requires some additional interpolation and evaluation. Furthermore, [15] requires significantly more communication rounds (overall and in the online phase) than PSImple, several of which are broadcast rounds. It should also be noted that despite having seemingly slightly better asymptotic communication than PSImple, it has large hidden constants. Thus, for the standard security parameter $\sigma = 128$, [15] does not concretely require less communication than PSImple, and unlike PSImple, most of the communication and computation of [15] is performed in the online phase.

Since [15] did not attempt to implement their protocol, we cannot determine whether it is concretely efficient or not, but it seems safe to conclude that [15] will at least be significantly slower than PSImple. However, we note that [15] achieve a stronger security guarantee than PSImple and [19], because in the protocol of [15] all the parties output the intersection.

Zhang et al. [42] recently made an interesting attempt to build a concretely efficient maliciously secure protocol extending the protocol of [39]. 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 sets of the honest parties. In particular, in the three party setting, this implies leaking the Bloom filter of the set 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 [20, 31–33]. In [32], 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 [20]. However, these works are mainly 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.

Notation. We denote the computational security parameter by σ, and the statistical security parameter by λ. In our implementation and experiments, $\sigma = 128$ and $\lambda = 40$. For $l \in \mathbb{N}$, $[l]$ denotes the set $\{1, ..., l\}$. We use the notation $\mathcal{P} = (\mathsf{P}_0, ..., \mathsf{P}_t)$ 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.

The security model. The security of our protocol is proved in the Universal Composability framework of Canetti [6]. We assume a static, 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. We assume all parties are connected via secure point-to-point channels. We also assume that all parties have access to (the same) *global* random oracle (that is, the same oracle may also be used in the executions of other protocols).

Private Set Intersection. In a private set intersection protocol, a set of parties $\mathcal{P} = (P_0, ..., P_t)$, each having up to n items from some domain $\mathcal{D}$ as their private inputs, compute the intersection over 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.

Following the real vs. ideal paradigm, security is defined with respect to an ideal functionality, by proving that a real-world adversary cannot do more harm than a very limited (ideal-world) adversary interacting with the honest parties via the ideal functionality. The ideal functionality $\mathcal{F}_{\text{MPSI}}$ is given in Figure 1.

We point out a couple of properties that are less standard, which we inherit from previous work on PSI. First, as mentioned above, only a designated party P_0 receives the output. Second, as in the work of [39], the size n' of the input sets of corrupted parties 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 $3.46n$, which significantly improves over the bound of $6n$ using the parameters given by [39]; see Remark 1 for more details.

$\mathcal{F}_{\text{MPSI}}$

$P_0, P_1, ..., P_t$ - 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;
Inputs: X_i from each party P_i, n, σ.
Computation: If size of the corrupt party input set is bigger than n', then the functionality aborts. Else it computes the intersection of all the 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 1: $\mathcal{F}_{\text{MPSI}}$ – Ideal multiparty private set intersection functionality

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.

In the case of binary fields (which can also be used for bitstrings by considering them as elements of a binary field), each party P_i receives a value S_i of length $|S|$, called P_i's *share* of S, such that $S = S_1 \oplus ... \oplus S_t$. 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. 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 shares of S_1 and S_2.

Bloom Filters and Garbled Bloom Filters. A Bloom filter is a compact data structure [3] to store a set of items that allows efficient probabilistic membership testing. It consists of N_{BF} bits and is 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 in the underlying set), if other elements of the set turn the Bloom filter bits at indices $h_1(x),...,h_k(x)$ to 1's.

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 on the length of the Bloom filter N_{BF}, the number of hash-functions k, and 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 initially seems [4, 13, 17, 29]. In this paper we used the refined formula from [17].

Garbled Bloom filters (GBF) were introduced by Dong et al. [14] 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 $h_1(x), ..., h_k(x)$. 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 an element x in the GBF, we compute the *codeword* y_x as follows.

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

where $h_*(x) \stackrel{def}{=} \{h_j(x) | j \in [k]\}$. [6] 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 filters for any given set, even if the codewords are fixed as well. Based on the GBF construction algorithm of Dong et al. [14], 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 B.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 $\text{BF}_1 \wedge \text{BF}_2$ (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 $\text{GBF}_1 \oplus \text{GBF}_2$ is the GBF of $X_1 \cap X_2$, where the codeword for any element is the XOR of its codewords in Y_{X_1} and Y_{X_2}. In particular, if the items that are in both X_1 and X_2 have the same codewords in GBF_1 and GBF_2, then all the items of $X_1 \cap X_2$ have all-zero codewords in $\text{GBF}_1 \oplus \text{GBF}_2$.

Oblivious Transfer. A 1-out-of-2 oblivious transfer (OT) involves two parties: a sender and a receiver. The input of the receiver is its choice bit b, the input of the sender are two values: m_0 and m_1. The output of the receiver is m_b, while the sender has no output.

[6] The summation in (1) is performed over the set of indices without repetitions. Pinkas et al. [34] 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.

In a K-out-of-N random OT, the input of the receiver is its set of choice indices of size K, denoted by $J = \{j_1, ..., j_K\}$, where $j_i \in [N]$, $i \in [K]$. The sender has no input. The output of the sender are N random values: $M = \{m_1, ..., m_N\}$, and the output of the receiver values indexed by J, namely $M_J = \{m_{j_1}, ..., m_{j_K}\}$.

Cut-and-Choose. 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.

Approximate K-out-of-N Random OT Π_{AppROT}. We recall that Rindal and Rosulek [39] require a *maliciously secure* K-out-of-N OT protocol to ensure that P_0 does not ask for significantly more indices than the number of 1s in its BF. To the best of our knowledge, no concretely-efficient maliciously-secure K-out-of-N OT protocol exists. To circumvent this, [39] implement a maliciously secure *approximate* K-out-of-N Random OT, allowing the receiver to request slightly more than K strings. Rindal and Rosulek [39] 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}, is formally described in Figure 2. 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 OT (where the inputs of the sender are random strings m_0, m_1). In a known fraction of these executions the receiver P_0 requests to learn the string m_1, and in all others it requests 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.

Remark 1. *In the PSI protocol of [39], 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. In [39] they showed that $n' < 2N_{\text{BF}}/\sigma$ (the authors stress that this is a very rough bound given for the worst case); we refer the reader to [39] for the detailed analysis.*

We note that with the choice of parameters in [39], $N_{\text{BF}} < 3n\sigma$, so $n' < 6n$. With our choice of parameters in Section 4, since N_{BF} is comparatively lower (see Tables 1 & 4), $N_{\text{BF}} < 1.73n\sigma$ so $n' < 3.46n$. Thus, our improved parameters also give a better security guarantee.

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) **[1-out-of-2 OTs]** The sender and the receiver call $\mathcal{F}_{\text{OT}}^{\sigma,N_{\text{OT}}}$ (performing N_{OT} 1-out-of-2 OTs). The receiver chooses bits $c_1, ..., c_{N_{\text{OT}}}$ with N^1_{OT} '1's among them, and $N_{\text{OT}} - N^1_{\text{OT}}$ '0's (randomly permuted), and the sender chooses uniformly random $M_0 = \{m_{10}, ..., m_{N_{\text{OT}}0}\}$ and $M_1 = \{m_{11}, ..., m_{N_{\text{OT}}1}\}$. As a result, in the jth OT, the receiver uses its choice bit c_j and learns $m_{j*} = m_{jc_j}$ (of length σ).

(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 2: Approximate K-out-of-N Random 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. We first give some motivation, by showing that the "natural" way to combine the techniques from [39] and [21] results in an inefficient protocol. Then, in Section 3.1, we describe the PSImple protocol for the two-party case.[7] Finally, the full fledged multiparty PSImple protocol is described in Section 3.2. The formal description of the PSImple protocol appears in Figure 3.

Before moving on to describe PSImple, let us consider what may be seen as the direct way to extend the protocol of [39] to the multiparty case, using the ideas of [21], and why it is inefficient. 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 $h_1, \ldots, h_k$ is the XOR over the strings from the GBF at indices $h_1(x), \ldots, h_k(x)$. Now, if each party P_i sends to P_0 the codewords attributed to its set X_i with respect to its GBF, then P_0 can compute the intersection of all parties, however, it can

[7] We do not suggest to use PSImple in the 2-party case, as it is less efficient than the state-of-the-art 2-party PSI protocols. For more than two parties, however, PSImple is the only concretely efficient PSI protocol secure against malicious adversaries.

also compute the intersection with party separately, which is not allowed.

To avoid this leakage, each P_i can additively share 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 with respect to 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 $P_1, \ldots, P_t$, 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, i.e., $n^{O(t)}$, 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 [39] 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 [39], 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 codewords, one from each party, 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 $\text{GBF}^{\text{IS}} \stackrel{def}{=} \text{GBF}^0 \oplus \text{GBF}^1$ (i.e., the XOR of the two cumulative GBFs) is a GBF that has the zero string on all indices 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 GBF^{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 GBF^{IS}. On the negative side, however, this method is insecure, as it allows P_0 to identify indices that were queried by P_1, even if they are not indices of an element in the intersection. This occurs if for some index s there is a 1 in the BF of both parties, as in this case $\text{GBF}^{\text{IS}}[s] = 0$.

To avoid this, P_1 rerandomizes its cumulative GBF. 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 $\text{GBF}^0[s]$ and $\text{GBF}^{1*}[s]$, 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, $\text{GBF}^* \stackrel{def}{=} \text{GBF}^0 \oplus \text{GBF}^{1*}$ 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 $\bigoplus_{s \in h_*(x_{0j})} \text{GBF}^*[s] = 0$.

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 $\text{GBF}^* \stackrel{def}{=} \text{GBF}^0 \oplus_{i \in [t]} \text{GBF}^{i*}$, 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 $\bigoplus_{s \in h_*(x_{0j})} \text{GBF}^*[s] = 0$.

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 3. Following the offline/online paradigm, we divide our MPSI protocol $\Pi_{PSImple}$ into two phases: an offline-phase $\Pi^{Offline}_{PSImple}$, which can be executed by the parties before they know their inputs, and an online-phase $\Pi^{Online}_{PSImple}$, 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 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. However, these dummy items are not treated as input items in the rerandomization procedure. See Appendix F for more details.

3.3 Security and Correctness

We prove the security of the protocol via the real vs. ideal paradigm (specifically, in the universal composability framework of [6]). In Appendix F, we provide a complete proof for the following theorem, stating the security of the PSImple protocol. The theorem refers to a hybrid model with two functionalities. The $\mathcal{F}_{\mathrm{OT}}^{\sigma,N}$ functionality constitutes N parallel ideal instances of 1-out-of-2 σ-long string oblivious transfer. The $\mathcal{F}_{\mathrm{gRO}}$ functionality is an ideal formulation of a global random oracle that is used by many protocols in parallel – this functionality better models the situation in the real-world, where the same random oracle realization (e.g., SHA 256) is used by many protocols, concurrently.

Theorem 3.1. *The $\Pi_{PSImple}$ protocol of Figure 3 securely realizes the functionality $\mathcal{F}_{\mathrm{MPSI}}$ with statistical UC-security with abort in the presence of a static, malicious adversary corrupting any number of parties in the $\mathcal{F}_{\mathrm{OT}}^{\sigma,N}$, $\mathcal{F}_{\mathrm{gRO}}$-hybrid model, where the adversary makes a polynomially-bounded number of queries to $\mathcal{F}_{\mathrm{gRO}}$ (where the $\mathcal{F}_{\mathrm{gRO}}$ models the Bloom filter's hash functions), assuming the protocol parameters are chosen as described in Section 4.*

For simplicity, the above theorem refers to $\Pi_{PSImple}$ as a single shot protocol. In Section G, we prove the security of our construction for the offline-online setting. We next sketch the ideas behind the proof, addressing its 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 the sets. Then, for every $i \in [t]$ in the interactions between P_0 and P_i, each of the two parties is 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, all these codewords will cancel out, and hence, P_0 will indeed output x as part of the intersection.

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 $\Pi_{PSImple}$. Proving the security of MPC protocols is a delicate task, requiring a rigorous analysis. We next present very high level ideas behind 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 $\mathcal{F}_{\mathrm{MPSI}}$ functionality and simulates a view that is closely distributed to the view real-world adversary in an execution of the protocol.

The main 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. Thus, for any element x on which the adversary made no query to the oracle, it cannot know the Bloom filter coordinates that are attributed with x. Using the global oracle formulation via the $\mathcal{F}_{\mathrm{gRO}}$ functionality, the simulator is informed of all queries ever made to oracle. This constitutes a list of possible elements that the adversary may use as inputs.

Once this is done, the simulator can extract the effective inputs for the corrupt parties, from the OT interactions of malicious parties, together with the secret shares they send to honest parties (recall that when interacting with the adversary, the simulator emulates for both the honest parties and the $\mathcal{F}_{\mathrm{OT}}^{\sigma,N}$ functionality). This is done as follows.

If P_0 is corrupt, then it suffices to send to the functionality the effective set of P_0. To obtain that we do the following. The simulator obtains the Bloom filter of P_0 from it choices in the OT interactions via $\mathcal{F}_{\mathrm{OT}}^{\sigma,N}$. Finally, for each random oracle query (i.e., possible item) the simulator tests whether this item is in the Bloom filter of P_0.

If P_0 is honest, then the simulator needs to obtain the intersection over the effective sets of malicious parties. The simulator obtains the Bloom filter of each corrupt P_i from its OT interactions with P_0. In addition, it learns the sum of shares of the rerandomized GBFs of corrupt parties from the messages sent to the P_0 (omitting the shares of honest parties). Finally, for each random oracle query (i.e., possible item) the simulator tests whether the effective GBF of the intersection over corrupt parties' sets.

Once this is done, the simulator can go to the ideal functionality with the intersection of all sets of malicious parties. Finally, if P_0 is corrupt, then upon receiving the output from the functionality, the simulator can send the missing GBF shares to P_0 that would result in the reconstruction of the correct output.

3.4 Asymptotic Complexity

In Table 3, we compare the communication complexity of PSImple with that of the multiparty PSI protocols of [9, 15, 19, 21, 26, 28] in the offline and online phases.

We observe that the overall communication complexity (total complexity of all the parties) is asymptotically approximately the same as the protocol of [21], which is only semi-honestly secure. The communication complexity is significantly better than the maliciously secure protocols of [9, 26], but slightly worse than the protocols of [19] and [15]. 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. A detailed analysis of the asymptotic communication and computation complexity of PSImple is given in Appendix E.

4 PROTOCOL PARAMETERS

In this section, we revisit the parameter analysis of [39] for the parameters of Π_{AppROT}. We show that the size of the garbled Bloom

Protocol of Malicious-secure Multiparty PSI $\Pi_{PSImple}$

Parameters: σ - computational security parameter; λ - statistical security parameter; N_{BF} - size of the Bloom filter; $N_{\text{OT}} > N_{\text{BF}}$ - 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 $\Pi^{Offline}_{PSImple}$**:**

(1) **[hash seeds agreement]** Parties run a coin-tossing protocol to agree on random hash-functions $h_1, h_2, \ldots, h_k$: $\{0,1\} \rightarrow [N_{\text{BF}}]$.
(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 party P_i, $i \in [t]$
   (a) Chooses a random $\mathbf{seed}_{il} \xleftarrow{R} \{0,1\}^\sigma$, sends $\mathbf{seed}_{il}$ to party P_l, and receives $\mathbf{seed}_{li}$ from P_l, for every $l \in [t] \setminus \{i\}$.
   (b) Locally computes $S^{il} = PRG(\mathbf{seed}_{li})$ and $S^{li} = PRG(\mathbf{seed}_{li})$ for every $l \in [t] \setminus \{i\}$, where $PRG : \{0,1\}^\sigma \rightarrow \{0,1\}^{N_{\text{BF}}\sigma}$ is a pseudorandom generator.

Online-phase $\Pi^{Online}_{PSImple}$**:**

(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 $\text{GBF}^0 = \bigoplus_{i\in[t]} \left(M^i_* \oplus \hat{M}^i\right)$. Each P_i, $i \in [t]$, computes $\text{GBF}^i = M^i \oplus \hat{M}^i_*$.
(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** B.1) .
(7) **[secret-sharing of GBFs]** Each P_i, $i \in [t]$, locally computes

$$\text{GBF}^{i*} = \text{GBF}^i \bigoplus_{l\in[t]\setminus\{i\}} \left[S^{li} \oplus S^{il}\right]$$

   and sends GBF^{i*} to P_0.
(8) **[reconstructing the GBF of the intersection]** P_0 computes $\text{GBF}^* = \bigoplus_{i\in[t]} \text{GBF}^{i*} \bigoplus \text{GBF}^0$. 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

$$\bigoplus_{r\in h_*(x_{0j})} \text{GBF}^*[r] = 0.$$

Figure 3: The PSImple Multiparty protocol

Table 3: Comparison of overall communication complexity
Parameters: t **– number of parties**
n **– size of the input set**
λ **,** σ **– statistical and computational security parameters, respectively.**

Protocol	Security	offline	online
IOP18 [21]	semi-honest	-	$O(t^2n\lambda^2)$
KMPRT17 [28]	semi-honest	-	$O(t^2n\lambda)$
IOP18 [21]	augmented semi-honest	$O(t^2\sigma)$	$O(tn\lambda^2)$
KMPRT17 [28]	augmented semi-honest	$O(t^2\sigma)$	$O(tn\lambda)$
CJS12 [9]	malicious (honest maj.)	$O(t^5\sigma)$	$O(t^3n\sigma)$
KS05 [26]	malicious	$O(t^3n\sigma)$	$O(t^3n^2\sigma)$
HV17 [19]	malicious	$O(t^2\sigma)$	$O(t^3\sigma + tn\sigma\log(n))$
GN19 [15]	malicious	$O(t^2\sigma)$	$O(t^3\sigma + tn\sigma)$
PSImple	malicious	$O(tn\sigma^2)$	$O(tn\sigma(\log(n\sigma) + \sigma))$

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 [39, 42].

The first difference from the analysis of [39] 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 [39], 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 [39] is technical: in [39], the sender chooses bits to check with 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 calculating the parameters. Additionally, we give the optimal parameters for several input sizes, and compare them with the parameters used in [39].

GBF parameters:

N_{OT} – number of OTs in Π_{AppROT};
N_{BF} – size of the Bloom filter of the receiver;
N^1_{OT} – number of 1's 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 $k, N_{\text{BF}}, N_{cc}, N_{\text{OT}}, N_{maxones}$ and N^1_{OT}. As we have three conditions and six variables, three of the 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 OT's is the heaviest part of the protocol.[8]

For any positive n, σ, and λ, there exist $k, N_{\text{BF}}, N_{cc}, N_{\text{OT}}, N^1_{\text{OT}}$, and $N_{maxones}$ that meet the above requirements, and we construct an algorithm to find the optimal parameter values; the proof and the full algorithm appear in Appendix C. We next briefly explain the algorithm: Based on the 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. We heuristically adopted the search boundaries $k_{min} = \frac{\sigma}{2}, k_{max} = 2\sigma, N_{\text{BF,min}} = nk$ and $N_{\text{BF,max}} = 3nk$. 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 appear in Appendix C.

Running the constructed algorithm with $\lambda = 40$, $\sigma = 128$, we obtained the parameters presented in Table 4. A comparison with the parameters used in [39] is given in Table 1.

5 IMPLEMENTATION, CODE OPTIMIZATIONS, AND EXPERIMENTAL RESULTS

We wrote our code in C++, using the LibOTe library [38] for the cryptographic primitives and the maliciously secure OT extension of Keller et al. [23].

[8] 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 C.

Table 4: Optimized N_{OT} parameter in Π_{AppROT} for different input set sizes n, with $\lambda = 40$, $\sigma = 128$.

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

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 scenarios where the parties can communicate much before they need to find the intersection the online phase should be as short as possible, while the offline phase can be significantly longer.

We have made the following code optimizations: The BF hash functions are computed using fixed-key AES and taking modulus.[9] As suggested by Araki et al. [1], we have performed the additive secret-sharing in the offline phase, and using seeds. 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. We used parallel computation, and in particular used 36 threads for computing Steps 4 (Bloom filter) and 9 (output). Additionally, we used up to 4 threads in each instance of Π_{AppROT}, which implies that the maximal number of threads in Steps 2 and 5 (Π_{AppROT}) are $8t$ in P_0 and 8 for every other party P_i. Our code will be made available on Github.

To benchmark PSImple, we ran experiments on Amazon Web Server using a c5.18xlarge machine (36 cores and 144 GB RAM) for P_0 and c5.4xlarge machines (8 cores and 32 GB RAM) for each other P_i, all with Unix OS, running on a LAN network with 1ms latency and 10Gb bandwidth. We tested the protocol with 4-32 parties and input size of 2^8-2^{20} per party. The results are given in Table 5. The running time of PSImple grows approximately linearly both in the number of parties and in the number of inputs,[10] as illustrated in Figures 4 and 5, respectively.

Table 5: Total runtime and online time (in parenthesis) of PSImple, in seconds.

*** signifies that the protocol crashed due to insufficient memory.**

parties	n = 2^8	n = 2^{12}	n = 2^{14}	n = 2^{16}	n = 2^{18}	n = 2^{20}
4	.20(.01)	.55(.22)	1.72 (.97)	6.62(4.18)	27.09(17.31)	128.25(76.81)
6	.24(.02)	.58(.23)	1.83 (1.02)	6.91(4.29)	30.12(19.47)	141.28(79.19)
8	.25(.02)	.66(.24)	1.94 (1.02)	7.62(4.77)	30.82(18.75)	143.20(78.1)
12	.31(.02)	.74(.26)	2.22 (1.08)	8.78(4.51)	35.5(20.43)	*
16	.37(.02)	.91(.31)	2.44 (1.21)	13.18(6.65)	47.33(28)	*
32	.8(.25)	1.60(.6)	5.12 (2.53)	21.54(11.92)	85.37(48.87)	*

We next consider the cost of the various steps of PSImple. An interesting aspect of the runtime is that, for a small number of parties (e.g., 4, 6), 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 number

[9] This restricts the input items' domain to 120 bits, as it requires 8 bits for the hash function selection, and also makes some assumptions on the randomness of AES.

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

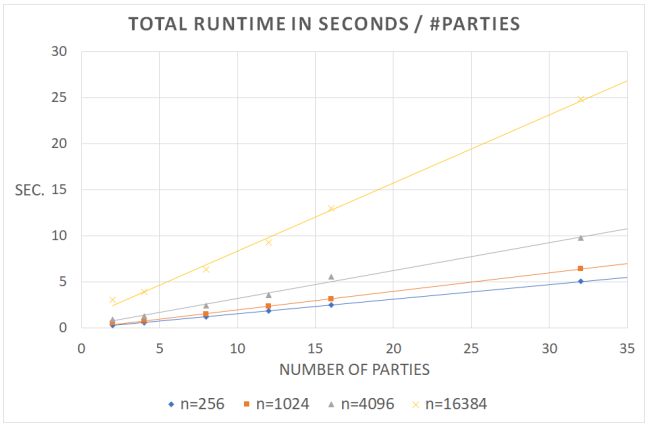

Figure 4: Total time for different number of parties

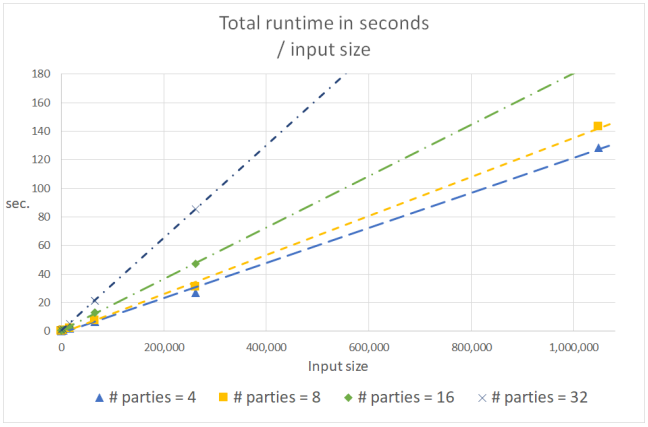

Figure 5: Total time for different size inputs.

of parties, the runtime is dominated by the rerandomization (in the online phase), while for a large number of parties the runtime is dominated by the OTs (in the offline phase). The computation complexity of the rerandomization algorithm **ReRand** is $O(nk)$. Possibly, for a small number of parties, PSImple would run faster using a different set of parameters, i.e., reducing the number of hash functions, k, at the cost of increasing the number of OTs. An interesting question is whether rerandomization can be computed efficiently using parallel computation, as this would drastically improve the online time. We plan to look into this question further.

In Table 2 we compare PSImple's performance with that of existing concretely efficient multiparty PSI protocols in the literature, in particutlar with [21, 28, 42]. A more appropriate comparison is with a maliciously secure multiparty PSI protocol, but previous maliciously secure multiparty PSI protocols were not concretely efficient and, therefore, were not implemented. Hence, in order to compare with a maliciously secure multiparty PSI protocol, we implemented the direct extension of the Rindal and Rosulek protocol [39] (two-party malicious) to the multiparty setting. This protocol, which is described in the beginning of Section 3, appears in Table 2 as *Direct*.

Remark 2. *We ran the codes of all the protocols, with the exception of [42], on the same testing platform described above. Unfortunately, [42] did not publish their code, so we used their reported times.*

Comparison with the protocols of Inbar et al. [21] and Zhang et al. [42]. The protocols of Inbar et al. [21] and Zhang et al. [42] are similarly based on GBFs. Recall that [21] 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 [42] 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 and semi-honest models.

Surprisingly, despite the fact that PSImple achieves significantly stronger security guarantees, the runtime of PSImple in our experiments is not significantly slower, and in many cases even faster, than the runtimes of [21] and [42]. We attribute much of this to the implementation itself (e.g., [21] wrote their code in Java using older OT extension protocols), and that [42] ran their experiments on a slightly weaker testing platform. Nevertheless, this already suggests that moving to malicious security using PSImple does not incur a very high penalty.

Comparison with the semi-honest protocol of Kolsenikov et al. [28]. The protocol of Kolsenikov et al. [28] uses different techniques in order to achieve a *semi-honestly* secure multiparty PSI protocol. Although we see in Table 2 that for a small number of parties their protocol is faster than PSImple, which achieves malicious security, it is also evident that PSImple scales better with the number of parties. Thus, for a large number of parties, we already see that PSImple is faster, despite achieving stronger security.

Additionally, most of the runtime of [28] is in the online phase, whereas in PSImple it is split more evenly, with the offline proportion increasing with the number of parties. The experiments suggest that the online phase of PSImple is generally faster than that of [28] when the number of parties is ≥ 6.

We note that [28] mention that their *augmented* semi-honest protocol is faster by a significant constant factor than their semi-honest protocol. We therefore conjecture that this *augmented* semi-honest protocol will be slightly faster than PSImple, but so far we have not managed to run it; we hope to remedy this situation soon. However, recall that this protocol offers a significantly weaker security guarantee than PSImple.

Comparison with the direct extension of RR [39]. As explained at the beginning of Section 3, directly extending [39] to the multiparty scenario makes the computation complexity grow exponentially with the number of parties, i.e., $n^{O(t)}$, whereas PSImple only grows approx. linearly with the number of parties. This makes the direct protocol impractical except for a very small number of parties and inputs, as is evident from Table 2 – even for modest parameters such as 4 parties and input size $n = 2^{14}$, the direct multiparty extension of [39] already takes almost 2 minutes, while this takes less than 2 seconds using PSImple. For a larger number of parties,

the direct multiparty extension of [39] becomes completely impractical, while PSImple's runtime grows only linearly in the number of parties.

ACKNOWLEDGMENTS

This work was supported by ISF grant 152/17 and by the Ariel Cyber Innovation Center in conjunction with the Israel National Cyber directorate in the Prime Minister's Office.

REFERENCES

[1] T. Araki, J. Furukawa, Y. Lindell, A. Nof, and K. Ohara. 2016. High-Throughput Semi-Honest Secure Three-Party Computation with an Honest Majority. *ACM CCS* (2016), 805–817. https://doi.org/10.1145/2976749.2978331

[2] A. Ben-Efraim, M. Nielsen, and E. Omri. 2019. Turbospeedz: Double Your Online SPDZ! Improving SPDZ Using Function Dependent Preprocessing. *ACNS* (2019), 530–549.

[3] B. H. Bloom. 1970. Space/Time Trade-offs in Hash Coding with Allowable Errors. *Commun. ACM* 13, 7 (1970), 422–426.

[4] P. Bose, H. Guo, E. Kranakis, A. Maheshwari, P. Morin, J. Morrison, M. Smid, and Y. Tang. 2008. On the false-positive rate of Bloom filters. *Inform. Process. Lett.* 108, 4 (2008), 210–213.

[5] R. Canetti. 2000. Security and composition of multiparty cryptographic protocols. *J. of CRYPTOLOGY* 13, 1 (2000), 143–202.

[6] R. Canetti. 2020. Universally Composable Security. *J. ACM* 67, 5 (2020), 28:1–94.

[7] R. Canetti, A. Jain, and A. Scafuro. 2014. Practical UC security with a Global Random Oracle. *ACM CCS* (2014), 597–608.

[8] C. A. Charalambides. 2002. *Enumerative combinatorics.* CRC Press.

[9] J. H. Cheon, S. Jarecki, and J. H. Seo. 2012. Multi-Party Privacy-Preserving Set Intersection with Quasi-Linear Complexity. *IEICE Trans. Fundam. Electron. Commun. Comput. Sci.* 95-A(8) (2012), 1366–1378.

[10] K. Christensen, A. Roginsky, and M. Jimeno. 2010. A new analysis of the false positive rate of a bloom filter. *Inform. Process. Lett.* 110, 21 (2010), 944–949.

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

[12] I. Damgård, M. Keller, E. Larraia, V. Pastro, P. Scholl, and N. P. Smart. 2013. Practical Covertly Secure MPC for Dishonest Majority - Or: Breaking the SPDZ Limits. *ESORICS* 8134 (2013), 1–18.

[13] E. De Cristofaro, J. Kim, and G. Tsudik. 2010. Linear-complexity private set intersection protocols secure in malicious model. *ASIACRYPT* (2010), 213–231.

[14] C. Dong, L. Chen, and Z. Wen. 2013. When private set intersection meets big data: An efficient and scalable protocol. *ACM CCS* (2013), 789–800.

[15] S. Ghosh and T. Nilges. 2019. An algebraic approach to maliciously secure private set intersection. *EUROCRYPT* (2019), 154–185.

[16] O. Goldreich. 2004. *The Foundations of Cryptography, vol. 2.* Cambridge University Press.

[17] F. Grandi. 2018. On the analysis of Bloom filters. *Inform. Process. Lett.* 129 (2018), 35–39.

[18] C. Hazay, P. Scholl, and E. Soria-Vazquez. 2017. Low Cost Constant Round MPC Combining BMR and Oblivious Transfer. *ASIACRYPT* 10624 (2017), 598–628.

[19] C. Hazay and M. Venkitasubramaniam. 2017. Scalable Multi-Party Private Set-Intersection. *PKC* (2017), 175–203.

[20] Y. Huang, D. Evans, and J. Katz. 2012. Private Set Intersection: Are Garbled Circuits Better than Custom Protocols? *NDSS* (2012).

[21] R. Inbar, E. Omri, and B. Pinkas. 2018. Efficient scalable multiparty private set-intersection via garbled bloom filters. *ICSCN* (2018), 235–252.

[22] Y. Ishai, J. Kilian, K. Nissim, and E. Petrank. 2003. Extending Oblivious Transfers Efficiently. *CRYPTO* 2729 (2003), 145 – 161.

[23] M. Keller, E. Orsini, and P. Scholl. 2015. Actively secure OT extension with optimal overhead. *CRYPTO* (2015), 724–741.

[24] M. Keller, V. Pastro, and D. Rotaru. 2018. Overdrive: Making SPDZ Great Again. *EUROCRYPT* 10822 (2018), 158–189.

[25] M. Kim, H. T. Lee, and J. H. Cheon. 2011. Mutual private set intersection with linear complexity. *WISA* (2011), 219–231.

[26] L. Kissner and D. Song. 2005. Privacy-preserving set operations. *CRYPTO* (2005), 241–257.

[27] V. Kolesnikov, R. Kumaresan, M. Rosulek, and N. Trieu. 2016. Efficient batched oblivious PRF with applications to private set intersection. *ACM CCS* (2016), 818–829.

[28] V. Kolesnikov, N. Matania, B. Pinkas, M. Rosulek, and N. Trieu. 2017. Practical Multi-party Private Set Intersection from Symmetric-Key Techniques. *ACM CCS* (2017), 1257–1272.

[29] M. Mitzenmacher and E. Upfal. 2017. *Probability and computing: Randomization and probabilistic techniques in algorithms and data analysis.* Cambridge university press.

[30] B. Pinkas, M. Rosulek, N. Trieu, and A. Yanai. 2020. PSI from PaXoS: Fast, malicious private set intersection. *EUROCRYPT* (2020), 739–767.

[31] B. Pinkas, T. Schneider, G. Segev, and M. Zohner. 2015. Phasing: Private Set Intersection Using Permutation-based Hashing. *USENIX Security* (2015), 515–530.

[32] B. Pinkas, T. Schneider, O. Tkachenko, and A. Yanai. 2019. Efficient circuit-based PSI with linear communication. *EUROCRYPT* (2019), 122–153.

[33] B. Pinkas, T. Schneider, C. Weinert, and U. Wieder. 2018. Efficient Circuit-Based PSI via Cuckoo Hashing. *EUROCRYPT* 10822 (2018), 125–157.

[34] B. Pinkas, T. Schneider, and M. Zohner. 2014. Faster Private Set Intersection Based on OT Extension. *USENIX Security* (2014), 797–812.

[35] B. Pinkas, T. Schneider, and M. Zohner. 2018. Scalable private set intersection based on OT extension. *ACM TOPS* 21, 2 (2018), 1–35.

[36] M. O. Rabin. 1981. How to Exchange Secrets by Oblivious Transfer. Tech. Memo TR-81, Aiken Computation Laboratory, Harvard University.

[37] A. C. D. Resende and D. F. Aranha. 2018. Faster unbalanced private set intersection. In *FC*. 203–221.

[38] P. Rindal. [n.d.]. libOTe: an efficient, portable, and easy to use Oblivious Transfer Library. https://github.com/osu-crypto/libOTe.

[39] P. Rindal and M. Rosulek. 2017. Improved Private Set Intersection Against Malicious Adversaries. *EUROCRYPT* (2017), 235–259.

[40] P. Rindal and M. Rosulek. 2017. Malicious-secure private set intersection via dual execution. *ACM CCS* (2017), 1229–1242.

[41] X. Wang, S. Ranellucci, and J. Katz. 2017. Global-Scale Secure Multiparty Computation. *ACM CCS* (2017), 39–56.

[42] E. Zhang, F. Liu, Q. Lai, G. Jin, and Y. Li. 2019. Efficient Multi-Party Private Set Intersection Against Malicious Adversaries. *ACM CCSW* (2019), 93–104.

A BASIC FUNCTIONALITIES

$\mathcal{F}_{\text{OT}}^{\sigma}$

Parties: the sender, the receiver.
Parameters: σ is the length of the OT strings.
Inputs: $b \in \{0,1\}$ from the receiver; $m_0, m_1 \leftarrow \{0,1\}^{\sigma}$ from the sender.
Outputs: Give output m_b to the receiver.

Figure 6: $\mathcal{F}_{\text{OT}}^{\sigma}$ – Ideal 1-out-of-2 OT functionality

Functionality $\mathcal{F}_{\text{OT}}^{\sigma,N}$

Parties: the sender, the receiver.
Parameters: N is the number of resulting parallel OT's for σ-bit length of the OT strings.
Inputs: $M_0 = \{m_{10}, ..., m_{N0}\}$ and $M_1 = \{m_{11}, ..., m_{N1}\}$ where $m_{ij} \leftarrow \{0,1\}^{\sigma}$ ($i \in [N]$, $j \in \{0,1\}$) from the sender; $\{c_1, ..., c_N\}$, where $c_i \in \{0,1\}$ ($i \in [N]$) from the receiver.
Outputs: Give output $M_c = \{m_{1c_1}, ..., m_{Nc_N}\}$ to the receiver.

Figure 7: $\mathcal{F}_{\text{OT}}^{\sigma,N}$ – Ideal N parallel 1-out-of-2 OT's for σ-bit strings functionality

$\mathcal{F}_{AppROT}$

Parties: a sender, a receiver.
Parameters: σ – computational security parameter; λ – statistical security parameter;
N – number of items to create; K' – the maximal allowed size of the request.
Inputs: From the receiver: $I = \{i_j\}_{j\in[K]}$, $K \le K'$; 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 works in two steps:

(1) The corrupt receiver chooses $K'' \le K'$ and sends it to the ideal functionality.
Upon receiving K'' from the receiver, it 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 a partial injective mapping
$I' = \begin{pmatrix} i_1 & i_2 & ... & i_{K'} \\ j_1 & j_2 & ... & j_{K'} \end{pmatrix}$, where $i_s \in [K'']$, $j_s \in [N]$, $s \in [K']$, $K'' \ge K' \ge K$ and $m_j = \begin{cases} m'_{i_s}, & \text{if } j = j_s, s \in [K']; \\ \text{fresh random}, & \text{else.} \end{cases}$
The functionality gives $M = \{m_i\}_{i\in[N]}$ to the sender and $M_* = \{m_i\}_{i\in[N]:i=i_s,s\in[K']}$ to the receiver.

If the adversary corrupts the sender.
Upon receiving I from the receiver, waits for $M = \{m_1, m_2, ..., m_N\}$ from the corrupt sender, where m_is are σ-bit strings. Gives $M_* = \{m_{i_1}, m_{i_2}, ..., m_{i_K}\}$ to the receiver.

Figure 8: $\mathcal{F}_{AppROT}$ – Ideal approximate K-out-of-N Random OT functionality

$\mathcal{F}_{\text{gRO}}$

Parameters:
k – number of hash-functions; N – size of input domain; $\ell(N)$ – size of output domain of the hash function;
$\overline{\mathcal{F}}$ – a list of ideal functionality programs.
Initialization: Initialize Q to be an empty list of prior queries.
Output: Upon receiving a query $x \in [N]$ from party $\mathsf{P} = (\mathsf{PID}, \mathsf{SID})$ or the adversary $\mathcal{S}$ do:

(1) • If there is a pair in a form (x, v) in Q, then return v to P (or $\mathcal{S}$);
• else, sample a uniformly random $v \in [\ell(N)]^k$, add (x, v) to the list Q and return v to P (or $\mathcal{S}$).

(2) Parse x as (s, x'). If $\mathsf{SID} \neq s$ then add (s, x', v) to the (initially empty) list Q_s of illegitimate queries for SID s.

Upon receiving a request from an instance of an ideal functionality in the list $\overline{\mathcal{F}}$, with SID s, return to this instance the list Q_s of illegitimate queries for SID s.

Figure 9: $\mathcal{F}_{\text{gRO}}$ – Global Random Oracle functionality

B ALGORITHMS FOR THE GARBLED BLOOM FILTER

B.1 Re-randomization Algorithm for a 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 $Y = (y_1, ..., y_n)$: $|y_i| = \sigma$, $(i \in [n])$;
the family of hash-indices $H^* = (h_*(x_1), ..., h_*(x_k))$: $h_*(x_i) = \{s | h_j(x_i) = s, j \in [k]\}$, $(i \in [n])$.

Algorithm:

1: GBF = *empty N_{BF}-size array of σ-long strings*
2: ***for*** i=1 ***to*** n ***do***
3: finalInd=−1
4: finalShare=y_i
5: ***for each*** $j \in h_*(x_i)$ ***do***
6: ***if*** GBF$[j]$ ***is*** *empty* ***then***
7: ***if*** finalInd==−1 ***then***
8: finalInd= j
9: ***else***
10: GBF$[j] \xleftarrow{R} \{0,1\}^\sigma$
11: finalShare=finalShare⊕GBF$[j]$
12: ***else***
13: finalShare=finalShare⊕GBF$[j]$
14: GBF[finalInd] =finalShare [11]
15: ***for*** $i = 0$ ***to*** $N_{\text{BF}} - 1$ ***do***
16: ***if*** GBF$[i]$ ***is*** *empty* ***then***
17: GBF$[i] \xleftarrow{R} \{0,1\}^\sigma$
18: ***return*** GBF

Output: GBF – A garbled Bloom filter of the set X with the codewords from Y.

B.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:

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

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

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

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

Input:

The item x; the garbled Bloom filter GBF; the length of GBF N_{BF}; the bitlength of string in GBF – σ;
$h_*(x)$ – set of hash-indices of x; $\forall i \in h_*(x)$, $i \in [N_{\text{BF}}]$.

Algorithm:

1: y=0
2: ***for each*** $i \in h_*(x)$ ***do***
3: y=$y \oplus$ GBF$[i]$
4: ***return*** y

Output: y – The codeword for x in the garbled Bloom filter GBF indexed by $h_*(x)$.

[11] 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.

C PARAMETERS OF Π_{AppROT}

In this section we explain in more detail our parameter choices for the number of required OTs and the Bloom filter size. Experimental results, given in Table 1, show that our parameter choice results in a 23-25% reduction in the number of required ROTs in comparison with [39], as well as smaller Bloom filter sizes.

The informal requirements from the parameter choice should ensure that:

- *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 the Bloom filter of its inputs. The probability of fail should be negligible in λ.

Definition C.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 $\mu(x) < 1/p(x)$. In our case, we require that the value of the functions is less than $2^{-\lambda}$ for statistical security and $2^{-\sigma}$ for computational security.

The main difference of our analysis from the analysis of [39] is that instead of requiring a sufficient number of 1's after the cut-and-choose to build any Bloom filter, as done in [39], we require this number to be sufficient to build almost all Bloom filters. This change significantly reduces the total number of required 1's from the OT, and consequently, the total number of required OTs. Note that this change implies that there might not be enough 1's to construct the GBF in the protocol, but this happens only with negligible probability.

The constraints on p_{False} come from the second and the third requirements. Namely, the sender rejects the cut-and-choose response, if $Pr[p_{False} \geq 2^{-\sigma}] > 2^{-\lambda}$. Therefore, the choice of parameters should ensure that the false positive probability of the Bloom filters, denoted p_{False}, is negligible. Until 2008, it was believed that $p_{False} = p^k$ [29], where p is the proportion of ones, and k is the number of the Bloom filter hash-functions. However, in 2008 Bose et al. [4] showed that this formula is only a lower bound for p_{False}. They further presented the precise formula for p_{False}, as well as a non-trivial upper bound. However, they did not provide any efficient algorithm for computing these formulas. In 2010, Christensen et al. [10] presented an algorithm for computing p_{False}. However, finding the maximal number of 1's in the Bloom filter from p_{False} remained hard. Therefore, we use the second-order Taylor's approximation of the false-positive probability presented by Grandi [17] in 2018, as it allows to more easily compute the maximal number of 1's from p_{False}.

We next give the formal details:

GBF parameters:

N_{OT} : number of OTs 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.

Theorem C.2. *By choosing the parameters of Π_{AppROT} under the following constraints*

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

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

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

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

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 $1 - 2^{-\lambda}$.*

In the following proof, we use the following two tail inequalities.

- The Azuma-Hoeffding inequality [29, 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 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 tail of distribution.
- The tail inequality, obtained by V. Chvatal in [11] for hypergeometric distribution. Namely, for $HG(M, N, n)$ by $0 < t < pn$, we have $Pr[X \geq (p+t)n] \leq \left(\left(\frac{p}{p+t}\right)^{p+t}\left(\frac{1-p}{1-p-t}\right)^{1-p-t}\right)^N \leq e^{-2t^2 n}$.

Proof. Consider every constraint one by one.

(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 $1 - 2^{-\lambda+1}$.*
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 $p = 1 - q$ [29, p 116]. Using the Azuma-Hoeffding inequality [29, 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 [11], 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) $Pr[p_{False} \geq 2^{-\sigma}] \leq 2^{-\lambda}$ *with either honest or malicious receiver in* Π_{AppROT}.
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 Appendix D from [17]:

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

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 theorem), 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 the cut-and-choose check successfully with probability at least* $1-2^{-\lambda}$.
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:

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

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.

□

We next show that there exist suitable parameters σ, λ for any n.

Theorem C.3. *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:

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

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

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),

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

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 $k \geq k_1$:

$$\lim_{N_{\mathrm{BF}}\to\infty}(b^2-4ac)=\lim_{N_{\mathrm{BF}}\to\infty}\left((C_1N_{\mathrm{BF}})^2-4N\lambda ln2\right)=\lim_{N_{\mathrm{BF}}\to\infty}\left((C_1N_{\mathrm{BF}})^2-4N_{\mathrm{BF}}\lambda\ln 2-4\sqrt{2nk}(\lambda\ln 2)^{3/2}\right)=\lim_{N_{\mathrm{BF}}\to\infty}(C_1N_{\mathrm{BF}})^2.$$

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). □

Using the relations proved in Theorem C.2, we construct the algorithm in Figure 10 and found the parameters for several values of n which optimize N_{OT}, 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
$$N_1=\left\lceil N_{\mathrm{BF}}\left(1-e^{-\frac{nk}{N_{\mathrm{BF}}}}\right)+\sqrt{\frac{nk\lambda\ln 2}{2}}\right\rceil,\ N=\left\lceil N_{\mathrm{BF}}+\sqrt{2nk\lambda\ln 2}\right\rceil,\ \alpha=\frac{N_1}{N}.$$
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
$$N_{maxones}=\left\lfloor\frac{b+\frac{\lambda\ln 2}{\min(\alpha,1-\alpha)}+\alpha N}{N+N_\Delta}N_{cc}-\sqrt{\frac{\lambda\ln 2N_{cc}}{2}}\right\rfloor,\ N_{maxhonest}=\left\lceil\alpha N_{cc}+\sqrt{\frac{\lambda\ln 2N_{cc}}{2}}\right\rceil,\ N_\Delta=\left\lceil\frac{a}{\alpha}\sqrt{N_{cc}}\right\rceil.$$
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 $(k,\ N_{\mathrm{BF}},\ N_{cc},\ N_{\mathrm{OT}},\ N^1_{\mathrm{OT}},\ N_{maxones})$, 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 10: Numerical algorithm for computing parameters for Π_{AppROT}

Remark 3. *For the online-phase, N_{BF} is more critical. In Table 6 we give the optimal by N_{BF} parameters, where the improvement 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, achieving this improvement for N_{BF} requires significantly more OTs (from 20,6% for $n=2^8$ to 46,7% for $n=2^{20}$) and increases the size of the cut-and-choose set (from 287% for $n=2^8$ to 5793,9% for $n=2^{20}$), which results in a large increase in the overall cost.*

Table 6: Optimal (in N_{BF}) Π_{AppROT} parameters for set size n, statistical security $\lambda=40$, and computational security $\sigma=128$.

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

D FALSE-POSITIVE PROBABILITY OF A BLOOM FILTER

In this section we compute an upper bound for p_{False} relative to the proportion of 1's in the Bloom filter. This is because the sender who evaluates this probability knows p, the proportion of 1's, but does not know n, the number of items.

The second-order Taylor's approximation of the false-positive probability in the Bloom filter, derived in [17], is:

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

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. Recall that if the receiver is malicious then the number of items can be higher than n (which is the event that the sender is trying to prevent in cut-and-choose).

From [29], it follows that $p = \left[1-(1-1/m)^{kn}\right]$ and $E[X] = mp$. Also notice that $1-\frac{1}{m-1} < 1-\frac{1}{m}$. Thus, 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).

E COMPLEXITY ANALYSIS

In this section we give further details on the complexity analysis of PSImple. In Tables 7 and 8 we present the communication and computational cost of the main operations of our protocol. The tables are split into the evaluating party P_0, and each other party P_i (i.e., P_1, ..., P_t). Recall that the workload of P_0 is significantly higher, as P_0 performs $2t$ instances of Π_{AppROT}, t as a sender and t as a receiver, while every other P_i performs only two instances, one as a sender and one as a receiver.

The communication of $\Pi^{Offline}_{AppROT}$ is dominated by OTs with communication complexity $O(n\sigma^2)$ – in order to compute N_{OT} OTs of length $\sigma = 128$ with statistical security $\lambda = 40$, the OT-extension of Keller, Orsini, and Scholl [23] requires sending σN_{OT} bits+10KB. The communication of Π^{Online}_{AppROT} consists mainly of sending/receiving the permutations of the unopened OTs; again, here P_0 performs $2t$ instances of Π^{Online}_{AppROT}, while every other P_i only 2 instances of Π_{AppROT}. Apart from Π^{Online}_{AppROT}, in the online phase of PSImple, each P_i is required to send its GBF to P_0. Table 7 summarizes the number of bits that are sent or received by the parties in the different steps of the protocol. Based on this table, we computed the communication complexity of PSImple in Table 9,with respect to $N_{\text{OT}} \approx N_{\text{BF}} = O(nk) = O(n\sigma)$ and $N_{cc} << N_{\text{BF}}$.

The main computational costs are summarized in Table 8. To compute N_{OT} OTs, the OT-extension of Keller, Orsini, and Scholl [23] requires $2N_{\text{OT}} + 336$ hashes.

Table 7: Number of sent and received bits

	P_0	P_i
	Offline-phase	
OT-extension	$2tN_{\text{OT}}\sigma$	$2N_{\text{OT}}\sigma$
Cut-and-choose challenge	$tN_{cc}\log_2 N_{\text{OT}}$	$N_{cc}\log_2 N_{\text{OT}}$
Cut-and-choose response	$t(N_{cc}\log_2 N_{\text{OT}} + \sigma)$	$N_{cc}\log_2 N_{\text{OT}} + \sigma$
	Online-phase	
Permutation	$tN_{\text{BF}}\log_2 N_{\text{OT}}$	$N_{\text{BF}}\log_2 N_{\text{OT}}$
Sending/receiving GBF^{i*}	$tN_{\text{BF}}\sigma$	$N_{\text{BF}}\sigma$

Table 8: Number of performed operations

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

Table 9: Comparison of communication complexity; t is the number of parties, n is the size of the input set, λ and σ are the statistical and computational security parameters respectively.

Protocol	**Security**	**Communication complexity**			
		overall		P_0	
		offline	**online**	**offline**	**online**
IOP18 [21]	semi-honest	-	$O(t^2n\lambda^2)$	-	$O(tn\lambda^2)$
KMPRT17 [28]	semi-honest	-	$O(t^2n\lambda)$	-	$O(tn\lambda)$
IOP18 [21]	augmented semi-honest	$O(t^2\sigma)$	$O(tn\lambda^2)$	$O(t\sigma)$	$O(tn\lambda^2)$
KMPRT17 [28]	augmented semi-honest	$O(t^2\sigma)$	$O(tn\lambda)$	$O(t\sigma)$	$O(tn\lambda)$
CJS12 [9]	malicious (honest maj.)	$O(t^4\sigma)$	$O(t^3n\sigma)$	$O(t^5\sigma)$	$O(t^2n\sigma)$
KS05 [26]	malicious	$O(t^3n\sigma)$	$O(t^3n^2\sigma)$	$O(t^2n\sigma)$	$O(t^2n^2\sigma)$
HV17 [19]	malicious	$O(t^2\sigma)$	$O(t^3\sigma + tn\sigma\log(n))$	$O(t^2\sigma)$	$O(t^2\sigma + tn\sigma\log(n))$
GN19 [15]	malicious	$O(t^2\sigma)$	$O(t^3\sigma + tn\sigma)$	$O(t\sigma)$	$O(t^2\sigma + tn\sigma)$
PSImple	malicious	$O(tn\sigma^2)$	$O(tn\sigma(\log(n\sigma) + \sigma))$	$O(tn\sigma^2)$	$O(tn\sigma(\log(n\sigma) + \sigma))$

F PROVING THEOREM 3.1

In this section, we prove the main theorem of this work in the setting when there is no division in offline- and online-phases. We start by restating the theorem.

Theorem F.1 (Restating Theorem 3.1). *The $\Pi_{PSImple}$ protocol of Figure 3 securely realizes the functionality $\mathcal{F}_{\text{MPSI}}$ with statistical UC-security with abort in the presence of a static, malicious adversary corrupting any number of parties in the $\mathcal{F}_{\text{OT}}^{\sigma,N}, \mathcal{F}_{\text{gRO}}$-hybrid model, where the adversary makes a polynomially-bounded number of queries to $\mathcal{F}_{\text{gRO}}$ (where the $\mathcal{F}_{\text{gRO}}$ models the Bloom filter's hash functions), assuming the protocol parameters are chosen as described in Section 4.*

We prove $\Pi_{PSImple}$ implements $\mathcal{F}_{\text{MPSI}}$ with statistical UC security [5] with global random oracle (gRO) [7]. We will need the $\mathcal{F}_{\text{OT}}^{\sigma,N}$ defined above and the $\mathcal{F}_{\text{gRO}}$, given in Figure 9. Theorem 3.1 follows from the consistency proof (appearing in Section F.1), together with an application of the UC-composition theorem of [6] to Lemma F.6 (see Section F.3) and Lemma F.7 (see Section F.4).

UC-gRO security. The security proof for stand-alone protocols allows using random oracle model, where there is separate RO for each protocol and each session, and the adversary can make queries only trough the simulator.In particular, this framework assumes that different protocols do not share state (or else security is not guaranteed), and a separate RO is assumed to exist for every subprotocol (marked by SID). This assumption is often not consistent with the common practice of instantiating the RO with a global hash function, used throughout the distributed system. To capture the need for proving UC security security in such a setting, a useful model of a "global" RO (UC-gRO) has been proposed in [7]. In a nutshell, the main difficulty in proving security in this model is that the environment can make queries to the gRO via a different sub-protocol, without the simulator being aware of them, which often hinders simulation. To handle this, the RO is augmented as follows, resulting in a gRO (global RO) functionality. Every RO query space is of the form (SID, x). If a party queries gRO with an sid that does not match its own, that query is marked as 'illegitimate' by gRO (but is nevertheless answered by it). Now, in addition to making regular queries to the RO, a functionality with ID SID can ask gRO for all of the illegitimate queries of the form $(SID, *)$. We also note that as Π_{AppROT} itself does not use the gRO at all, but the underlying OT's do. In Section 5, we discuss implementations of the idealized functionality $\mathcal{F}_{\text{OT}}^{\sigma,N}$ that are *gRO*-secure, which is essential for making our concrete implementation $\Pi_{PSImple}$ *gRO*-secure (and in fact, even UC-secure at all, as $\Pi_{PSImple}$ uses the same implementation of the the RO for all instances of Π_{AppROT}, which in our implementation indeed make calls to the RO).

F.1 Consistency of PSImple

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

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

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 $y_{ij} = \oplus_{s\in h_*(x)}\text{GBF}^i[s] = \oplus_{s\in h_*(x)} \left(M^i[s] \oplus \hat{M}_*^i[s]\right) = \oplus_{s\in h_*(x)} \left(M_*^i[s] \oplus \hat{M}^i[s]\right)$.

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

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

F.2 Security Model and Notation

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 [8, 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 F.2. 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 F.3. 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 [5] with global random oracle [7], 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 [5] for more details - the following is a brief recap of the setting, which is identical to the standard setting of [5], 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. [12] 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 $\mathcal{F}$ 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 $\mathcal{F}$, 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 [16]. 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 F.4. 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.[13] [14]

Definition F.5. 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 $\mathcal{F}'$, defined exactly as $\mathcal{F}$, 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 [16, Chapter 7] and [5].[15]

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 $N = N_{\text{BF}}$ consisting of K 1's. We give this protocol in $\mathcal{F}_{\text{OT}}^{\sigma,N}$-hybrid model in Figure 11. 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 oblivious transfer functionality we use in our security setting is $\mathcal{F}_{\text{OT}}^{\sigma,N}$ (Fig. 7) with $N = N_{\text{OT}}$ parallel instances of 1-out-of-2 OT.

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

[13] 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.

[14] 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.

[15] $\mathcal{F}'$ is a specific kind of a generalized $\mathcal{F}$ functionalities, allowing for extra 'abort after seeing output' capabilities for the adversary.

Protocol Π_{AppROT} in $\mathcal{F}_{\text{OT}}^{\sigma,N}$-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 OTs to generate;
$N^1_{\text{OT}}, N_{cc}, N_{maxones}$ 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) **[1-out-of-2 OTs]** The sender and the receiver call $\mathcal{F}_{\text{OT}}^{\sigma,N_{\text{OT}}}$ performing N_{OT} OTs 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). The senders chooses uniformly at random $M_0 = \{m_{10}, ..., m_{N_{\text{OT}}0}\}$, $M_1 = \{m_{11}, ..., m_{N_{\text{OT}}1}\}$ (m_{j0}, m_{j1} are of length σ). In the jth OT, the 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 the 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 the 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]** The 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 11: Protocol Π_{AppROT} in $\mathcal{F}_{\text{OT}}^{\sigma,N}$-hybrid model

Lemma F.6. *The protocol Π_{AppROT} realizes the functionality $\mathcal{F}_{AppROT}$ with statistical UC-security with abort in the presence against static (unbounded) malicious adversaries in the $\mathcal{F}_{\text{OT}}^{\sigma,N}$-hybrid model, and K' is an upper bound for number of 1's in Bloom filter of length $N = N_{\text{BF}}$ such that $p_{False} < 2^{-\sigma}$.*[16]

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, 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 $\mathcal{Z}$.[17]

(1) In the 1st step of the protocol, when emulating $\mathcal{F}_{\text{OT}}^{\sigma,N}$ 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
$$K'' = \sum_{i \in [N_{\text{OT}}] \setminus C} c_i.$$
- If $K'' \leq K'$, $\mathcal{S}$ passes K'' as an input of the receiver to $\mathcal{F}_{AppROT}$ and receives $M' = \{m'_1, m'_2, ..., m'_{K''}\}$. 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 $c_j = 1$. 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. The simulator gives $\{m_{j*}\}_{j \in [N_{\text{OT}}]}$ to $\mathcal{Z}$ as the output of $\mathcal{F}_{\text{OT}}^{\sigma,N}$ in 1st step, and C as the message of 2nd step of the protocol.
- If $K'' > K'$, the simulator passes to $\mathcal{Z}$ the vector $\{m_{j*}\}_{j \in N_{\text{OT}}}$ of uniformly random σ-bit strings as the output of $\mathcal{F}_{\text{OT}}^{\sigma,N}$ of the 1st step of the protocol, sends to it C as the cut-and-choose request and, upon getting the answer, passes $\perp$ to $\mathcal{Z}$ as sender's reply, appends $\{m_{j*}\}_{j \in N_{\text{OT}}}$, and halts.

(2) $\mathcal{S}$ waits for a message from the receiver in 3rd step of the protocol. Upon receiving the response $(R, r*)$ it sends back Continue and waits for the permutation.
Upon receiving permutation π from $\mathcal{Z}$ in 4th step, $\mathcal{S}$ checks, that $\pi : [N_{\text{OT}}] \setminus C \to [N]$. If not, then uses a default value for π (N consecutive values from $[N_{\text{OT}}] \setminus C$).

[16] 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.
[17] 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.

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 $\mathcal{F}_{\text{OT}}^{\sigma,N}$ the set Q of the requests to $\mathcal{F}_{\text{OT}}^{\sigma,N}$, where $Q' \subseteq Q$ is the set of 1-requests, and the rest are 0-requests. It receives a sequence $\{m_{j*}\}_{j\in[N_{\text{OT}}]}$ of random strings in response. 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 $K'' = |Q' \cap ([N_{\text{OT}}] \setminus C)|$. The emulated m_{j*}'s at locations $j \in Q' \cap ([N_{\text{OT}}] \setminus C)$ are taken from the functionality's step-1 output to receiver if $K'' \le K'$, and random independent values picked by $\mathcal{S}$ otherwise. In both cases, the other values $\mathcal{S}$ sends emulating replies of $\mathcal{F}_{\text{OT}}^{\sigma,N}$ 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 $[N_{\text{OT}}] \setminus C$ on behalf of the receiver in step 1 (of the adversary's interaction with the functionality). With our choice of parameters, according Claim C.2, 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 $2^{-\sigma} + neg(\lambda)$.

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

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 $\mathcal{F}_{\text{OT}}^{\sigma,N}$'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 $Ideal_{\mathcal{F},\mathcal{Z},\mathcal{S}} \overset{s}{\approx} Real_{\Pi,\mathcal{Z}}$ with 0-error.

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

In the first step of the protocol, $\mathcal{S}$ obtains the inputs of the sender $M_0 = \{m_{j0}\}_{j\in[N_{OT}]}$, $M_1 = \{m_{j1}\}_{j\in[N_{OT}]}$ from $\mathcal{Z}$.

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 $(1, 2, \dots)$. It samples the number of 1's $|C \setminus R|$ hypergeometrically $HG(N_{\text{OT}}, N_{\text{OT}}^1, N_{cc})$ (N_{OT}^1 is determined in Section 4), computes $|R| = N_{cc} - |C \setminus R|$ 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.

Then it samples the uniformly random N-permutation $\pi : [N_{\text{OT}}] \setminus C \to [N]$ and gives it to $\mathcal{Z}$ as a message in 4th step of the protocol, and gives $M = \pi(M_1)$ as an input of corrupt sender to $\mathcal{F}_{AppROT}$.

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 $\mathcal{S}$) sends to $\mathcal{F}_{\text{OT}}^{\sigma,N}$ two based on x sets $M_0 = \{m_{j0}\}_{j\in[N_{\text{OT}}]}$ and $M_1 = \{m_{j1}\}_{j\in[N_{\text{OT}}]}$, whose distributions are identical in both real and ideal worlds by the construction of the simulator.

Then $\mathcal{Z}$ sends the set $C \subset [N_{\text{OT}}]$ such that $|C| = N_{cc}$ based on x 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 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 Continue, it receives the N-permutation $\pi : [N_{\text{OT}}] \setminus C \to [N]$. This permutation in ideal world is computed randomly and in the real directly from the receiver with the same input and randoness, 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 $c_1, ..., c_{N_{\text{OT}}}$, recovers the subset $C \in [N_{\text{OT}}]$ received from the sender and sends the permutation $\pi : [N_{\text{OT}}] \setminus C \to [N]$ such that $\pi(c_1, ..., c_{N_{\text{OT}}}) = B$.

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.

The sender inputs to $\mathcal{F}_{\text{OT}}^{\sigma,N_{\text{OT}}}$ sequences $(m_{j0})_{j\in[N_{\text{OT}}]}$ and $(m_{j1})_{j\in[N_{\text{OT}}]}$. The receiver has $(m_{jc_j})_{j\in[N_{\text{OT}}]}$ as an output. Applying the permutation π to $(m_{j1})_{j\in[N_{\text{OT}}]}$ and $(m_{jc_j})_{j\in[N_{\text{OT}}]}$ 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 $i_1 = \min([N_{\text{OT}}] \setminus C)$, $i_N = \max([N_{\text{OT}}] \setminus C)$. 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 12 describes the *PSImple* protocol in $\mathcal{F}_{\text{gRO}}, \mathcal{F}_{AppROT}$-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 3) is omitted in Figure 12. Additionally, the ideal functionality $\mathcal{F}_{AppROT}$ is not separated into offline and online phases, so as the $\Pi_{PSImple}$ itself. 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 12 we explicitly describe all the $\perp$-replies that parties can send (as we consider security with abort and asynchronous execution).

In Lemma F.7 and consequently in Theorem 3.1 we require a non-uniform polynomial-time adversary in sense of polynomially-bounded requests to the global 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 3.1.

Lemma F.7. *The protocol $\Pi_{PSImple}$ securely realizes the functionality $\mathcal{F}_{\text{MPSI}}$ with statistical UC-security with abort in presence of static malicious adversary corrupting up to t parties in the $\mathcal{F}_{AppROT}, \mathcal{F}_{\text{gRO}}$-hybrid model, which makes a polynomially bounded number of queries to $\mathcal{F}_{\text{gRO}}$, where the Bloom filter hash functions modelled as non-programmable global random oracle, 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 contains 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 (which we consider as a preprocessing step at Round 0, though it also could be considered as Round-1 transaction performed in parallel with step 2), $\mathcal{S}$ 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 $(q, \mathsf{PID}, \mathsf{SID})$ to the global Random Oracle (to compute Bloom filter's hash-indices) on behalf of corrupt parties $\mathsf{P}_i \in I$ as requested by $\mathcal{Z}$ and writes them to sets $Q_i = \{q_{ij} | j \in [n_i]\}$. Here index i corresponds to PID of corrupt P_i, $n_i \geq n$ is polynomially bounded, as $\mathcal{Z}$ is. Denote $Q = \cup_{i|\mathsf{P}_i\in I} Q_i$ – the joint query set of corrupt parties. The environment might continue making such a queries up to the end of the protocol, and $\mathcal{S}$ adds them in Q up to the end of Step 5 (when the effective intersection of corrupt parties inputs is extracted).

Protocol of Malicious-secure Multiparty PSI $\Pi_{PSImple}$ 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 $\mathcal{D}$ ($i \in \{0, ..., t\}$).

(1) **[(R0) random shares]** Each P_i, $i \in [t]$, sends $S^{il} = (s_1^{il}, ..., s_{N_{\text{BF}}}^{il})$ to any P_l, $l \in [t] \setminus \{i\}$, where $s_r^{il} \xleftarrow{R} \{0,1\}^\sigma$, $r \in [N_{\text{BF}}]$.
(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 sets $M_*^i[j] = 0$ for $j \in [N_{\text{BF}}] \setminus J$.
   (b) Using BF_i 1's indices set J_i as input, each P_i ($i \in [t]$) calls $|J_i|$-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 $j \in J_i$, and P_0 receives $\hat{M}^i$. P_i sets $\hat{M}_*^i[j] = 0$ for $j \in [N_{\text{BF}}] \setminus J_i$.
(4) **[(R2) compute and re-randomize GBFs]** If P_0 did not receive $\perp$ from $\mathcal{F}_{AppROT}$, it computes $\text{GBF}^0 = \bigoplus_{i\in[t]} \left(M_*^i \oplus \hat{M}^i\right)$. If P_i did not receive $\perp$ from $\mathcal{F}_{AppROT}$, it computes $\text{GBF}^i = M^i \oplus \hat{M}_*^i$, codewords $y_{ij} = \bigoplus_{r\in h_*(x_{ij})} \text{GBF}^i[r]$ ($j \in [n_i]$) and re-randomizes GBF^i from X_i and codewords y_{ij} ($j \in [n_i]$) according to algorithm **BuildGBF** from B.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:
$$\text{GBF}^{i*} = \text{GBF}^i \bigoplus_{l\in[t]\setminus\{i\}} \left[S^{li} \oplus S^{il}\right].$$
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 $\text{GBF}^* = \bigoplus_{i\in[t]} \text{GBF}^{i*} \bigoplus \text{GBF}^0$.
(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
$$\bigoplus_{r\in h_*(x_{0j})} \text{GBF}^*[r] = 0, j \in [n_0].$$
Else it outputs $\perp$.

Figure 12: The PSImple multiparty protocol in the $\mathcal{F}_{AppROT}$-hybrid model

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 receives M^i of length N_{BF} or $\perp$ from P_i.
- 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 or $\perp$.

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 $\mathsf{P}_i \in I$, if all the calls to emulated $\mathcal{F}_{AppROT}$ are completed successfully (without $\perp$'s).

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 $\mathsf{P}_i \in I$. Besides, it asks $\mathcal{F}_{\text{gRO}}$ for all the illegitimate queries made with the current's execution SID and append those queries to the set Q (which remains polynomially bounded as $\mathcal{A}$ is).

- 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 $\text{GBF}_I^* = \bigoplus_{i\in I} \text{GBF}^{i*}$. Now $\mathcal{S}$ can subtract all the secret shares sent and received to corrupt parties on behalf of honest and vice versa:
$$\text{GBF}_I = \text{GBF}_I^* \bigoplus_{\mathsf{P}_i\in I} \bigoplus_{\mathsf{P}_l\in\mathcal{P}\setminus(I\cup\mathsf{P}_0)} \left(S^{il} \oplus S^{li}\right).$$
GBF_I is the effective value of $\bigoplus_{\mathsf{P}_i\in I} \text{GBF}^i$. Now the simulator extracts the effective input of corrupt parties as
$$\tilde{X}_I = \left\{ q \in Q \;\middle|\; \bigoplus_{r\in h_*(q)} \text{GBF}_I[r] = \bigoplus_{\substack{\mathsf{P}_i\in I \\ r\in h_*(q)}} \left(M^i[r] \oplus \hat{M}_*^i[r]\right) \right\},$$
sends it to the ideal functionality, and receives either $\tilde{X}$ or $\perp$ as the output of $\mathcal{F}_{\text{MPSI}}$.
- Else, the simulator sends the effective input of the adversary $\perp$ to the ideal functionality and receives $\perp$ as its output.

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 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 $\mathcal{X} = \{X_i\}_{\mathsf{P}_i \in \mathcal{P} \setminus I}$ – 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.

$\{\hat{M'}^i/\perp\}_{\mathsf{P}_i \in I}$, $\{M^i/\emptyset\}_{\mathsf{P}_i \in I}$: 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, $H = \tilde{X}_I \bigcap \left(\bigcap_{\mathsf{P}_i \in \mathcal{P} \setminus I} X_i\right)$ 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 $H_1 = \bigcap_{\mathsf{P}_i \in \mathcal{P} \setminus I} X_i$ 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, $\mathcal{S}$ 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, $\mathcal{S}$ make queries $(q, \mathsf{PID}, \mathsf{SID})$ to the global Random Oracle (to compute Bloom filter's hash-indices) on behalf of corrupt parties $\mathsf{P}_i \in I \cup \mathsf{P}_0$ as requested by $\mathcal{Z}$ and writes them to sets $Q_i = \{q_{ij} | j \in [n_i]\}$. Here index i corresponds to PID of corrupt P_i, $n_i \geq n$ is polynomially bounded, as $\mathcal{Z}$ is. Denote $Q = \cup_{i | \mathsf{P}_i \in I \cup \mathsf{P}_0} Q_i$ – the joint query set of corrupt parties. The environment might continue making such a queries up to the end of the protocol, and $\mathcal{S}$ adds them in Q up to the end of Step 5 (when the effective intersection of corrupt parties inputs is extracted).

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'' as $\mathcal{F}_{AppROT}$ would, or $\perp$ if the Bloom filter with K'' 1's has more than n' items. 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 receives $\hat{M}^i$.

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 computes a set of Bloom filters $\tilde{\text{BF}}_{0i}$ from the input's set of indexes j_i in any instance between corrupt receiver P_0 and honest sender 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, $\mathcal{S}$ asks $\mathcal{F}_{\text{gRO}}$ for all the illegitimate queries with the session ID of current execution, append them to Q and extracts an effective input of the adversary as $\tilde{X}_I = \left\{q \in Q \mid \forall s \in h_*(q), \forall j \notin I, \tilde{\text{BF}}_{0j}[s] = 1\right\}$ – 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 $\mathcal{F}_{\text{MPSI}}$, 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 $\mathcal{F}_{\text{MPSI}}$ 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 $\text{GBF}^j = M^j \bigoplus \hat{M}^j$ 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 B.1; note, that positions at indices r such that $\text{BF}_j[r] = 0$ 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 $\text{GBF}^{i*} = \bigoplus_{\mathsf{P}_j \notin (I \cup \mathsf{P}_0)} \text{GBF}^{j*}_{temp} \bigoplus_{j \neq i, \mathsf{P}_j \notin (I \cup \mathsf{P}_0)} \text{GBF}^{j*}$ (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, $\{M'^i/\perp\}_{\mathsf{P}_i \in \mathcal{P} \setminus (I \cup \mathsf{P}_0)}$, $\{\hat{M}^i/\emptyset\}_{\mathsf{P}_i \in \mathcal{P} \setminus (I \cup \mathsf{P}_0)}$: 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 *gRO* also don't break indistinguishability, because they actually refer to the same *gRO* 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 $\tilde{X}_I$, $H = \tilde{X}_I \bigcap \left(\bigcap_{\mathsf{P}_i \in \mathcal{P} \setminus I \cup \{\mathsf{P}_0\}} X_i\right)$, 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.[18] 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 $\mathcal{F}_{\text{MPSI}}$, 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.

□

[18] 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 PROVING THE SECURITY OF $\Pi_{PSIMPLE}$ FOR THE OFFLINE-ONLINE SETTING

In this section, we prove the security of $\Pi_{PSImple}$ in the offline-online setting. The proof follows from the combination of theorems G.3 and G.4 (see Section G.2), with respect to lemmas G.1, G.2 (see Section G.1). The consistency proof for $\Pi_{PSImple}$ is given in Section F.1.

We prove $\Pi_{PSImple}$ implements $\mathcal{F}_{\text{MPSI}}$ with statistical UC security [5] in global Random Oracle model [7]. We will need the $\mathcal{F}_{\text{OT}}^{\sigma,N}$ defined in fig. 7 and the $\mathcal{F}_{\text{gRO}}$, which is a variant of the standard random oracle, in Figure 9.

Note that in the resulting protocol, the "offline" part appearing in the protocol in Figure 3 indeed line up at the beginning of our protocol, and all computation performed there does not depend on the inputs. Therefor, following instantiation of $\mathcal{F}_{\text{OT}}^{\sigma,N}$, we may move the corresponding parts to an offline phase, as in $\Pi_{PSImple}$.

We also note that as Π_{AppROT} does not use the gRO, the lemmas G.1, G.2 and G.3 do not mention gRO at all, although we imply the underlying $\mathcal{F}_{\text{OT}}^{\sigma,N}$ being $UC-gRO$ secure. In Section 5, we discuss implementations of the idealized functionality $\mathcal{F}_{\text{OT}}^{\sigma,N}$ that are gRO-secure, which is essential for making our concrete implementation $\Pi_{PSImple}$ gRO-secure (and in fact, even UC-secure at all, as $\Pi_{PSImple}$ uses the same implementation of the the random oracle for all instances of Π_{AppROT}, which in our implementation indeed make calls to the gRO).

Most of the definitions such as $UC-gRO$-security, statistical/computational indistinguishability, Ideal- vs Real-world model etc. are given in Appendix F.

G.1 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 $N = N_{\text{BF}}$ consisting of K 1's. We divide this protocol in two phases: offline $\Pi_{AppROT}^{Offline}$ in $\mathcal{F}_{\text{OT}}^{\sigma,N}$-hybrid model in Figure 13, which is performed before parties receive their inputs, and online Π_{AppROT}^{Online} in Figure 14. The functionality of Π_{AppROT}^{Online} is somewhat truncated in sense that we do not make parties use their Bloom filters *inside* Π_{AppROT}^{Online}, but we consider the calculation of the permutation as the part of external, $\Pi_{PSImple}$ protocol. This is done for the sake of simplicity of the proof in offline-online model, since otherwise the inputs to $\mathcal{F}_{AppROT}^{Online}$ should go from the outputs of $\mathcal{F}_{AppROT}^{Offline}$, which requires the integrity (i.e. using *MAC*s from both parties). Thus, we give to PSImple the burden of the intermediate processing the outputs from $\Pi_{AppROT}^{Offline}$ to form the inputs for $\Pi_{AppROT}^{Offline}$.

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 oblivious transfer functionality we use in our security setting is $\mathcal{F}_{\text{OT}}^{\sigma,N}$ (Fig. 7) with $N = N_{\text{OT}}$ parallel instances of 1-out-of-2 OT.

Protocol $\Pi_{AppROT}^{Offline}$ in $\mathcal{F}_{\text{OT}}^{\sigma,N}$-hybrid model

Parties: Sender, Receiver.
Parameters: σ – length of the OT strings (computational security parameter); λ – statistical security parameter;
$N_{\text{OT}} > N$ is the number of OTs to generate;
N_{OT}^1, N_{cc}, $N_{maxones}$ are parameters of cut-and-choose described in Section 4.
Inputs: no inputs. **Offline phase** $\Pi_{AppROT}^{Offline}$

(1) **[1-out-of-2 OTs]** The sender and the receiver call $\mathcal{F}_{\text{OT}}^{\sigma,N_{\text{OT}}}$ performing N_{OT} OTs in parallel. The receiver chooses requests $c_1, ..., c_{N_{\text{OT}}}$ with N_{OT}^1s ones among them, and $N_{\text{OT}} - N_{\text{OT}}^1$ zeroes (randomly permuted). The senders chooses uniformly at random $M_0 = \{m_{10}, ..., m_{N_{\text{OT}}0}\}$, $M_1 = \{m_{11}, ..., m_{N_{\text{OT}}1}\}$ (m_{j0}, m_{j1} are of length σ). In the jth OT, the 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 the 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 the 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.

Outputs: M_1 to the sender; $M_* = \{m_{jc_j} | c_j = 1,\ j \in [N_{\text{OT}}] \setminus C\}$ to the receiver.

Figure 13: Protocol $\Pi_{AppROT}^{Offline}$ in $\mathcal{F}_{\text{OT}}^{\sigma,N}$-hybrid model

Lemma G.1. *The protocol $\Pi_{AppROT}^{Offline}$ realizes the functionality $\mathcal{F}_{AppROT}^{Offline}$ with statistical UC-security with abort in the presence against static (unbounded) malicious adversaries in the $\mathcal{F}_{\text{OT}}^{\sigma,N}$-hybrid model.*[19]

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 $\mathcal{S}$

[19] 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^{Online}_{AppROT}

Parties: Sender, Receiver.
Parameters: N_{OT} is the number of OTs required; $N = N_{\text{BF}}$ is the length of Bloom filter.
Inputs: An injective function (permutation) $\pi : ([N_{\text{OT}}] \setminus C) \to [N]$ from the receiver; no input from the sender.
[sending permutation] The receiver sends π to the sender.
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 sender has output π.

Figure 14: Protocol Π^{Online}_{AppROT}

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, and the first message received by the corrupted party comes from $\mathcal{F}^{Offline}_{AppROT}$, 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 $\mathcal{Z}$.[20]

In the 1st step of the protocol, when emulating $\mathcal{F}^{\sigma,N}_{\text{OT}}$ 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 $|C| = N_{cc}$, computes

$$I = \{i \in [N_{\text{OT}}] \setminus C | c_i = 1\}.$$

Then, $\mathcal{S}$ computes $K' = |I|$, p_{False} as the false-positive probability of the Bloom filter of length N with k hash-functions and K' ones.

If $p_{False} < 2^{-\sigma}$, then passes I as an input of the receiver to $\mathcal{F}^{Offline}_{AppROT}$ and receives $M_* = \{m_1, m_2, ..., m_{K'}\}$. Then, $\mathcal{S}$ computes the receiver's output $\{m_{j*} | j = 1, 2, ..., N_{\text{OT}}\}$ of $\mathcal{F}^{\sigma,N}_{\text{OT}}$ as follows:

$$m_{j*} = \begin{cases} \text{fresh random}, & \text{if } j \in C \text{ or } c_j = 0; \\ \text{next item from } M_*, & \text{else}, \end{cases} \tag{15}$$

gives C to $\mathcal{Z}$ as the message of 2nd step of the protocol, gets from it R and r^* as messages of 3rd step, sends Continue and halts.

If $p_{False} \geq 2^{-\sigma}$, it passes to $\mathcal{Z}$ $\{m_{j*}\}_{j \in N_{\text{OT}}}$ of uniformly random σ-bit strings as the output of $\mathcal{F}^{\sigma,N}_{\text{OT}}$ 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.

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 $\mathcal{F}^{\sigma,N}_{\text{OT}}$ the set Q of the requests to $\mathcal{F}^{\sigma,N}_{\text{OT}}$, where $Q' \subseteq Q$ is the set of 1-requests, and the rest are 0-requests. It receives a sequence $\{m_{j*}\}_{j \in [N_{\text{OT}}]}$ of random strings in response. 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 uniformly at random. The emulated m_{j*}'s at locations $j \in Q' \cap ([N_{\text{OT}}] \setminus C)$ are taken from the functionality's 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 $\mathcal{F}^{\sigma,N}_{\text{OT}}$ are random values independent of all others.

Now, in the real world, the sender chooses C, and either

$$|C \setminus R| > N_{maxones} \tag{16}$$

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 (16) is satisfied is identical in both worlds. Then $\mathcal{Z}$ responds with the same R (identical in both worlds). Consider the case when the inequality (16) 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 I as the set of indices of 1-OT requests in $[N_{\text{OT}}] \setminus C$ on behalf of the receiver in step 1 (of the adversary's interaction with the functionality). With our choice of parameters, according Claim C.2, the evaluation of p_{False} computed for the Bloom filter of length N with $K' = |I|$ 1's in it, is larger than $2^{-\sigma}$ except with negligible (in λ) probability, since (16) 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 $2^{-\sigma} + neg(\lambda)$.

$$Ideal_{\mathcal{F},\mathcal{Z},\mathcal{S}} \overset{s}{\approx} Real_{\Pi,\mathcal{Z}} \overset{s}{\approx} (D, \perp). \tag{17}$$

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

[20] 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.

Now, consider the case when (16) is not satisfied in the real world. If $\mathcal{Z}$ sends R^* (which differs from R induced by its $\mathcal{F}_{\text{OT}}^{\sigma,N}$'s inputs) so that (16) 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 the uniformly random set in 1st step, and $\perp$ in 3rd. $\mathcal{S}$ again appends $\perp$ as sender's message to the simulated view. Thus, if (17) holds with 0-error (in particular, over the entire support of $Real_{\Pi,Z}$, the sender's output is $\perp$).

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

In the first step of the protocol, $\mathcal{S}$ plays $\mathcal{F}_{\text{OT}}^{\sigma,N}$ towards $\mathcal{Z}$, obtaining inputs of the sender as $M_0 = \{m_{j0}\}_{j\in[N_{OT}]}$, $M_1 = \{m_{j1}\}_{j\in[N_{OT}]}$.

In 2nd step of the protocol, $\mathcal{S}$ waits for the message C from $\mathcal{Z}$. If $|C| > N_{cc}$, then truncates it, and if $|C| < N_{cc}$, then adds indices by default $(1, 2, ...)$. It computes the set $M_* = \{m_i \in M_1 | i \notin C\}$ and gives it to the ideal functionality.

Then it samples choice bits $c_1, ..., c_{N_{\text{OT}}}$ as the honest receiver would do according the protocol, computes (R, r^*) from M_0, C and $c_1, ..., c_{N_{\text{OT}}}$ honestly and gives (R, r^*) to the sender as the message of 3rd step of the protocol. Then $\mathcal{S}$ gets back either $\perp$ or Continue, sends M_1 over the indexes $[N_{\text{OT}}] \setminus C$ to the ideal functionality, and halts.

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 $\mathcal{S}$) sends to $\mathcal{F}_{\text{OT}}^{\sigma,N}$ two based on x sets $M_0 = \{m_{j0}\}_{j\in[N_{\text{OT}}]}$ and $M_1 = \{m_{j1}\}_{j\in[N_{\text{OT}}]}$, whose distributions are identical in both real and ideal worlds by the construction of the simulator.

Then $\mathcal{Z}$ sends the set $C \subset [N_{\text{OT}}]$ such that $|C| = N_{cc}$ based on x 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 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.

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

As in our protocols - both the approximate K-out-of-N OT and *PSImle*, we separate them in two phases (online and offline), is is secure to consider separate functionalities for both phases. In case of Π_{AppROT}, we do not consider the functionality of the entire protocol in $\mathcal{F}_{AppROT}^{Offline}$-hybrid model, as it requires either rewinding or construction of the reactive functionality as in fig. 8 (because $\mathcal{F}_{AppROT}^{Online}$ is the random functionality). It is possible to prove either UC-security of the reactive Π_{AppROT}, or the security of $\Pi_{AppROT}^{Offline}$ and Π_{AppROT}^{Online} separately, when $\mathcal{F}_{AppROT}^{Online}$ is a simple functionality of the secure channel for transmitting the permutation from the receiver to the sender.

$\mathcal{F}_{AppROT}^{Offline}$

Parties: a sender, a receiver.
Parameters:
σ – computational security parameter; N – number of items to create;
$N_{\text{OT}} - N_{cc} > N$ – number of items to create in the offline phase.
Inputs:
From the receiver: $I = \{i_j\}_{j\in[K']}$, $K' \geq K$; from the sender: no input.
Outputs:
Upon receiving I from the receiver, samples the uniformly random $M = \{m_1, m_2, ..., m_{N_{\text{OT}}-N_{cc}}\}$, 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 sender.
Upon receiving I from the receiver, waits for $M'' = \{m''_1, m''_2, ..., m''_{N_{\text{OT}}-N_{cc}}\}$ from the corrupt sender, where m''_is are σ-bit strings. Then it gives $M_* = \{m_{i_1}, m_{i_2}, ..., m_{i_{K'}}\}$ to the receiver.

Figure 15: $\mathcal{F}_{AppROT}^{Offline}$ – Ideal offline approximate K-out-of-N Random OT functionality

As the only message in an online-phase of Π_{AppROT} is sent, then the following lemma is obviously holds.

Lemma G.2. *The protocol Π_{AppROT}^{Online} UC-secure realizes the functionality $\mathcal{F}_{AppROT}^{Online}$ in the presence against static (unbounded) malicious adversaries.*

$\mathcal{F}_{AppROT}^{Online}$

Parties: a sender, a receiver.
Parameters: N – the length of the permutation.
Inputs: from the receiver: the N-length permutation π; from the sender: no input.
Outputs: gives π to the sender.

Figure 16: $\mathcal{F}_{AppROT}^{Online}$ – Ideal online approximate K-out-of-N Random OT functionality

G.2 PSI protocol in the hybrid model

In this section, we prove the security of *PSImple* in offline-online setting. Therefore we describe those phases separately as two protocols. The offline phase $\Pi_{PSImple}^{Offline}$ id shown in fig. 17. Furthermore, we need to add a padding after $\mathcal{F}_{AppROT}^{Offline}$, 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. Our PSImple protocol is divided in offline and online phases, which requires the separate

Offline phase of Malicious-secure Multiparty PSI protocol $\Pi_{PSImple}^{Offline}$ in the $\mathcal{F}_{AppROT}^{Offline}$-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: no inputs.

Offline Phase $\Pi_{PSImple}^{Offline}$:

(1) **[(R0) symmetric approximate ROT-offline]** Parties call in parallel:
   (a) P_0 as a receiver calls $\mathcal{F}_{AppROT}^{Offline}$ with each P_i, $i \in [t]$ inputting the set of indices J_i as K distinct random values from $[N_{\text{OT}} - N_{cc}]$ (K is a hypergeometrically distributed value from $HG(N_{\text{OT}}, N_{\text{OT}} - N_{cc}, N_{\text{OT}}^1)$). As a result, it receives t sets of string $M^{i\prime}_*[j]$ for each $j \in J_i$, P_i learns $M^{i\prime}$. P_0 sets $M^{i\prime}_*[j] = 0$ for $j \in [N_{\text{OT}} - N_{cc}] \setminus J_i$.
   (b) Each P_i, $i \in [t]$, as a receiver calls $\mathcal{F}_{AppROT}^{Offline}$ with P_0 inputting the set of indices $\hat{J}_i$ as K distinct random values from $[N_{\text{OT}} - N_{cc}]$ (K is a hypergeometrically distributed value from $HG(N_{\text{OT}}, N_{\text{OT}} - N_{cc}, N_{\text{OT}}^1)$). As a result, P_i learns $\hat{M}^{i\prime}_*[j]$ for each $j \in \hat{J}_i$, and P_0 receives $\hat{M}^i$. P_i sets $\hat{M}^{i\prime}_*[j] = 0$ for $j \in [N_{\text{OT}} - N_{cc}] \setminus \hat{J}_i$.
(2) **[(R0) random shares]** Each P_i, $i \in [t]$, sends $S^{il} = (s_1^{il}, ..., s_{N_{\text{BF}}}^{il})$ to any P_l, $l \in [t] \setminus \{i\}$, where $s_r^{il} \xleftarrow{R} \{0,1\}^\sigma$, $r \in [N_{\text{BF}}]$.

Outputs: P_0 gets $M^{i\prime}_*$ and $\hat{M}^{i\prime}$ ($i \in [t]$); P_i gets $M^{i\prime}$, $\hat{M}^{i}{}_*{}'$ and S^{ij} for each $j \in [t]$, $j \neq i$.

Figure 17: The offline phase of PSImple protocol in the $\mathcal{F}_{AppROT}^{Offline}$-hybrid model

functionalities for these phases. Although the offline-phase of *PSImple* just uses the number $\mathcal{F}_{AppROT}^{Offline}$'s and the point-to point secure channels to transmit secret shares S^{ij}s from P_i and P_j, and thus implies a rather simple functionality, we formally describe this $\mathcal{F}_{\text{PSImple}}^{Offline}$ functionality in fig. 18.

Theorem G.3. *The protocol $\Pi_{PSImple}^{Offline}$ securely realizes the functionality $\mathcal{F}_{\text{PSImple}}^{Offline}$ with statistical UC-security with abort in presence of static malicious adversary corrupting up to t parties in the $\mathcal{F}_{AppROT}^{Offline}$-hybrid model, if the protocol parameters are chosen as described in subsection 4.*

Proof. The soundness of the statement of this theorem follows from lemma G.1 and is quite obvious: all that the simulator should do is to take inputs from the corrupt parties, give them directly to the ideal functionality and send the functionality's output directly to the environment on the order of taking the inputs of corrupt parties asynchronously. As the upper bound for the number of indexes on the inputs of $\mathcal{F}_{AppROT}^{Offline}$ and $\mathcal{F}_{\text{PSImple}}^{Offline}$ are the same, it holds that the probability of having $\perp$ as an output of the party is the same in both worlds (thanks to our parameters choice, the statistical distance is at most $2^{-\lambda}$), and the statistical UC-security is achieved. □

Remark 4. *We stress, that the ideal functionality $\mathcal{F}_{\text{PSImple}}^{Offline}$ allows the adversary the reactive behaviour, namely choosing the inputs based on partial outputs of the same functionality. For instance, the adversary may choose it's secret shares after it receives outputs for OT's, or to use outputs from one OT to compute inputs to another. Later, in the proof of security of PSImple protocol, we'll show that this doesn't affect security.*

Figure 19 describes the PSImple protocol in gRO, $\mathcal{F}_{\text{PSImple}}^{Offline}$, $\mathcal{F}_{AppROT}^{Online}$-hybrid model. Note that as the hash functions are modeled by the global random oracle, the coin-tossing step for hash-seed agreement (Step 1 in Figure 3) is omitted in Figure 19.

For clarity, in Figure 19 we explicitly describe all the $\perp$-replies that parties can send (as we consider security with abort and asynchronous execution).

$\mathcal{F}_{\text{PSImple}}^{Offline}$

Parties: $\mathsf{P}_0, \mathsf{P}_1, ..., \mathsf{P}_t$.
Parameters: K – the maximal number of 1's in Bloom filter of size N_{BF} with k hash-function such that $p_{False} < 2^{-\sigma}$.
Inputs: from P_0: J_i – the set of indexes from $[N_{\text{OT}} - N_{cc}]$ ($i \in [t]$); from P_i: $\hat{J}_i$ – the set of indexes from $[N_{\text{OT}} - N_{cc}]$, S_{ij} – the $N_{\text{BF}} \times \sigma$-bit value, ($i, j \in [t], j \neq i$).
Outputs:
The functionality first takes all the inputs from all the honest parties and gives S^{ji} ($j \in [t], j \neq i$) from any honest P_j to any P_i.
If P_0 is honest, the functionality samples $\hat{M}^{i\prime}$ uniformly at random, and if P_i is honest, it samples $M^{i\prime}$ uniformly at random.
Then functionality waits for inputs from corrupt parties. As well as it receives (asynchronously):

- S^{ij} from corrupt P_i: it passes it to P_j;
- J_i from corrupt P_0: if $|J_i| \leq K$ it computes $M_*^{i\prime}$ as the subset of $M^{i\prime}$ on indexes from J_i and gives it to P_0; else it gives $\perp$ to both P_0 and P_i;
- $\hat{J}_i$ from corrupt P_i: if $|\hat{J}_i| \leq K$ it computes $\hat{M}_*^{i\prime}$ as the subset of $\hat{M}^{i\prime}$ on indexes from $\hat{J}_i$ and gives it to P_i; else it gives $\perp$ to both P_0 and P_i;
- $M^{i\prime}$ from corrupt P_i: it computes $M_*^{i\prime}$ as the subset of $M^{i\prime}$ on indexes from J_i and gives it to honest P_0;
- $\hat{M}^{i\prime}$ from corrupt P_0: it computes $\hat{M}_*^{i\prime}$ as the subset of $\hat{M}^{i\prime}$ on indexes from $\hat{J}_i$ and gives it to honest P_i.

Figure 18: $\mathcal{F}_{\text{PSImple}}^{Offline}$ – Ideal offline PSImple functionality

Online phase of Malicious-secure Multiparty PSI protocol $\Pi_{PSImple}$ in the $\mathcal{F}_{\text{PSImple}}^{Offline}$, $\mathcal{F}_{AppROT}^{Online}$-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 $\Pi_{PSImple}^{Offline}$:
Parties call $\mathcal{F}_{\text{PSImple}}^{Offline}$ - the offline PSImple functionality.
Online Phase $\Pi_{PSImple}^{Online}$:

(1) **[(R1) compute Bloom filters]** P_i ($i \in [t] \cup \{0\}$) makes queries to $\mathcal{F}_{\text{gRO}}$ and 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.
(2) **[(R1) symmetric approximate ROTs-online]** Parties perform in parallel:
  (a) Using BF_0's 1's indices set, P_0 computes the random permutation $\pi_0^i : [N_{\text{OT}} - N_{cc}] \to [N_{\text{BF}}]$ of $M_*^{i\prime}$ and calls $\mathcal{F}_{AppROT}^{Online}$ as the receiver with each of the other parties P_i ($i \in [t]$). Each P_i ($i \neq 0$) computes $M^i = \pi_0^i(M^{i\prime})$ and P_0 computes $M_*^i = \pi_0^i(M_*^{i\prime})$.
  (b) Using BF_i 1's indices set, each P_i ($i \in [t]$) computes the random permutation $\pi_i : [N_{\text{OT}} - N_{cc}] \to [N_{\text{BF}}]$ of $\hat{M}^{i\prime}{}_*$ and calls $\mathcal{F}_{AppROT}^{Online}$ as the receiver with P_0. Each P_i ($i \neq 0$) computes $\hat{M}_*^i = \pi_i(\hat{M}^{i\prime}{}_*)$ and P_0 computes $\hat{M}^i = \pi_i(\hat{M}^{i\prime})$.
(3) **[(R2) compute and re-randomize GBFs]** If P_0 did not receive $\perp$ from $\mathcal{F}_{\text{PSImple}}^{Offline}$ or $\mathcal{F}_{AppROT}^{Online}$, it computes $\text{GBF}^0 = \bigoplus_{i \in [t]} \left(M_*^i \oplus \hat{M}^i\right)$. If P_i did not receive $\perp$ from $\mathcal{F}_{\text{PSImple}}^{Offline}$ or $\mathcal{F}_{AppROT}^{Online}$, it computes $\text{GBF}^i = M^i \oplus \hat{M}_*^i$, codewords $y_{ij} = \bigoplus_{r \in h_*(x_{ij})} \text{GBF}^i[r]$ ($j \in [n_i]$) and re-randomizes GBF^i from X_i and codewords y_{ij} ($j \in [n_i]$) according to algorithm **BuildGBF** from B.1.
(4) **[(R2) cumulative GBFs of P_is]** If P_i ($i \in [t]$) did not receive $\perp$ from $\mathcal{F}_{\text{PSImple}}^{Offline}$ or $\mathcal{F}_{AppROT}^{Online}$, it computes and sends to P_0 the cumulative garbled Bloom filter:

$$\text{GBF}^{i*} = \text{GBF}^i \bigoplus_{l \in [t] \setminus \{i\}} \left[S^{li} \oplus S^{il}\right].$$

  Else it sends $\perp$.
(5) **[(R2) cumulative GBF of P_0]** If P_0 did not receive $\perp$s before, it computes $\text{GBF}^* = \bigoplus_{i \in [t]} \text{GBF}^{i*} \bigoplus \text{GBF}^0$.
(6) **[(R2) output]** If P_0 did not receive $\perp$s before, it outputs x_{0j} as a member of the intersection, if

$$\bigoplus_{r \in h_*(x_{0j})} \text{GBF}^*[r] = 0, j \in [n_0].$$

  Else it outputs $\perp$.

Figure 19: The online phase of PSImple protocol in the $\mathcal{F}_{\text{PSImple}}^{Offline}$, $\mathcal{F}_{AppROT}^{Online}$-hybrid model

In Theorem G.4 and consequently in Theorem 3.1 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 3.1.

Theorem G.4. *The protocol $\Pi_{PSImple}$ securely realizes the functionality $\mathcal{F}_{\text{MPSI}}$ with statistical UC-security with abort in presence of static malicious adversary corrupting up to t parties in the $\mathcal{F}_{\text{PSImple}}^{Offline}$, $\mathcal{F}_{AppROT}^{Online}$, $\mathcal{F}_{\text{gRO}}$-hybrid model, which makes a polynomially bounded number of queries to $\mathcal{F}_{\text{gRO}}$, where the Bloom filter hash functions are modelled as non-programmable global random oracle, 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 containing 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 corrupt party in $\mathcal{F}_{\text{PSImple}}^{Offline}$ receives $\perp$ if and only if its input set of indexes J_i or $\hat{J}_i$ (and hence it's effective Bloom filter) contains 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 $\mathcal{F}_{\text{PSImple}}^{Offline}$ 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. $\mathcal{S}$ sends to $\mathcal{Z}$ uniformly random shares S^{il}, as honest P_i's would do according the protocol, to any corrupt party $\mathsf{P}_l \in I$.

Then, $\mathcal{S}$ continues playing $\mathcal{F}_{\text{PSImple}}^{Offline}$ functionality towards $\mathcal{Z}$ for any corrupt party. Once inputs of $\mathcal{F}_{\text{PSImple}}^{Offline}$ have been sent by $\mathcal{Z}$:

- Once P_i sends $M^{i\prime}$, the simulator answers to P_i nothing;
- Once P_i sends $\hat{J}_i$, the simulator checks the size of it and answers with either uniformly random set $\hat{M}_*^{i\prime}$ ($\mathcal{S}$ samples uniformly at random the $N_{\text{OT}} - N_{cc}$-element set $\hat{M}^{i\prime}$ and chooses from there items with indexes from $\hat{J}_i$) or $\perp$.

Finally, $\mathcal{S}$ learns S^{li}s from $\mathcal{Z}$, emulating $\mathcal{F}_{\text{PSImple}}^{Offline}$.

In step 1, $\mathcal{S}$ make queries $(q, \mathsf{PID}, \mathsf{SID})$ to the global Random Oracle (to compute Bloom filter's hash-indices) on behalf of corrupt parties $\mathsf{P}_i \in I$ as requested by $\mathcal{Z}$ and writes them to sets $Q_i = \{q_{ij} | j \in [n_i]\}$. Here index i corresponds to PID of corrupt P_i, $n_i \geq n$ is polynomially bounded, as $\mathcal{Z}$ is. Denote $Q = \cup_{i|\mathsf{P}_i \in I} Q_i$ – the joint query set of corrupt parties. The environment might continue making such a queries up to the end of the protocol, and $\mathcal{S}$ adds them in Q up to the end of Step 4 (when the effective intersection of corrupt parties inputs is extracted).

In step 2, $\mathcal{S}$ plays $\mathcal{F}_{AppROT}^{Online}$ functionality, obtaining the permutations π_i from corrupt parties and sending them uniformly random permutations $\pi_0^i : [N_{\text{OT}} - N_{cc}] \to [N_{\text{BF}}]$. Then the simulator computes:

$$M_*^i = \pi_0^i(M_*^{i\prime}), \quad M^i = \pi_0^i(M^{i\prime}), \quad \hat{M}_*^i = \pi_i(\hat{M}_*^{i\prime}), \quad \hat{M}^i = \pi_i(\hat{M}^{i\prime}).$$

At the 4th step, $\mathcal{S}$ observes GBF^{i*}s or $\perp$s sent by $\mathcal{Z}$ on behalf of corrupt $\mathsf{P}_i \in I$. Besides, it asks $\mathcal{F}_{\text{gRO}}$ for all the illegitimate queries made with the current's execution SID and append those queries to the set Q (which remains polynomially bounded as $\mathcal{A}$ is).

- If there were no $\perp$'s as an outputs of the simulated $\mathcal{F}_{\text{PSImple}}^{Offline}$ or as the messages of the 6th step, $\mathcal{S}$ computes the sum $\text{GBF}_I^* = \bigoplus_{i \in I} \text{GBF}^{i*}$. Now $\mathcal{S}$ can subtract all the secret shares sent and received to corrupt parties on behalf of honest and vice versa:

$$\text{GBF}_I = \text{GBF}_I^* \bigoplus_{\mathsf{P}_i \in I} \bigoplus_{\mathsf{P}_l \in \mathcal{P} \setminus (I \cup \mathsf{P}_0)} \left(S^{il} \oplus S^{li} \right).$$

  GBF_I is the effective value of $\bigoplus_{\mathsf{P}_i \in I} \text{GBF}^i$. Now the simulator extracts the effective input of corrupt parties as

$$\tilde{X}_I = \left\{ q \in Q \;\middle|\; \bigoplus_{r \in h_*(q)} \text{GBF}_I[r] = \bigoplus_{\substack{\mathsf{P}_i \in I \\ r \in h_*(q)}} \left(M^i[r] \oplus \hat{M}_*^i[r] \right) \right\},$$

  sends it to the ideal functionality, and receives either $\tilde{X}$ or $\perp$ as the output of $\mathcal{F}_{\text{MPSI}}$.
- Else, the simulator sends the effective input of the adversary $\perp$ to the ideal functionality and receives $\perp$ as its output.

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 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 $\mathcal{X} = \{X_i\}_{\mathsf{P}_i \in \mathcal{P} \setminus I}$ – inputs of honest parties. We prove indistinguishability by induction on the message graph of $\mathcal{Z}$ - sent messages to the various parties, and to $\mathcal{F}_{\text{PSImple}}^{Offline}$ 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, for $\mathcal{F}_{\text{PSImple}}^{Offline}$ any value sent and received by $\mathcal{Z}$, or sent on behalf of an honest party to $\mathcal{F}_{\text{PSImple}}^{Offline}$ (a random share of 0) are identically distributed.

$\{M^{i\prime}/\emptyset\}_{\mathsf{P}_i \in I}$, $\{\hat{M}_*^{i\prime}/\perp\}_{\mathsf{P}_i \in I}$, $\{S^{ji}\}_{\mathsf{P}_i \in I, \mathsf{P}_j \notin I, j \neq 0}$: 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 $\mathcal{F}_{\text{PSImple}}^{Offline}$, and the fact that at any step of interaction of the $\mathcal{F}_{\text{PSImple}}^{Offline}$, the view of each emulated P_i is distributed identically to the real world. Note that in particular, these values do not depend on $\mathcal{X}$.

Only one transaction in the protocol from the honest party to the corrupt ones are permutations π_0^i which $\mathcal{S}$ sends to $\mathcal{Z}$ on behalf of honest P_0. Those permutations are identically distributed both in real and ideal world. In the simulation they do not depend on the inputs.

Let us compare the output distribution of P_0. In the ideal world, $H = \tilde{X}_I \bigcap \left(\bigcap_{\mathsf{P}_i \in \mathcal{P} \setminus I} X_i \right)$ 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 8 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 6). Now, malicious parties may have chosen 1-items in $\mathcal{F}_{\text{PSImple}}^{Offline}$ execution, at locations outside of the query set Q. However, then they either query too many 1's in $\mathcal{F}_{\text{PSImple}}^{Offline}$ 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 $H_1 = \bigcap_{\mathsf{P}_i \in \mathcal{P} \setminus I} X_i$ 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 6 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 $\mathcal{F}_{\text{PSImple}}^{Offline}$).

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 received from $\mathcal{F}_{\text{PSImple}}^{Offline}$. 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 $\mathcal{F}_{\text{PSImple}}^{Offline}$ and $\mathcal{F}_{AppROT}^{Online}$ 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 the preprocessing step, emulating $\mathcal{F}_{\text{PSImple}}^{Offline}$, $\mathcal{S}$ sends to corrupt parties $\mathsf{P}_l \in I$ uniformly random values S^{il}, as honest P_is would do. Then, $\mathcal{S}$ continues emulate $\mathcal{F}_{\text{PSImple}}^{Offline}$ functionality towards $\mathcal{Z}$ for corrupt P_0. Once inputs of P_0 to $\mathcal{F}_{\text{PSImple}}^{Offline}$ have been sent by $\mathcal{Z}$:

- Once P_0 sends J_i, the simulator checks the size of it and answers with either uniformly random set $M_*^{i\prime}$ ($\mathcal{S}$ samples uniformly at random the $(N_{\text{OT}} - N_{cc})$-element set $M^{i\prime}$ and chooses from there items with indexes from J_i) or $\perp$.
- Once P_0 sends $\hat{M}^{i\prime}$, the simulator answers to P_0 nothing.

Finally, $\mathcal{S}$ gets S^{li}s to be delivered by corrupted parties P_l from $\mathcal{Z}$. We stress, that $\mathcal{S}$ delivers messages asynchronously, by the order $\mathcal{Z}$ sends messages to parties of $\mathcal{F}_{\text{PSImple}}^{Offline}$, and the above presentation is written (reporting messages from $\mathcal{F}_{\text{PSImple}}^{Offline}$ is done once an honest party requests to deliver a given message).

In step 1, $\mathcal{S}$ make queries $(q, \mathsf{PID}, \mathsf{SID})$ to the global Random Oracle (to compute Bloom filter's hash-indices) on behalf of corrupt parties $\mathsf{P}_i \in I \cup \mathsf{P}_0$ as requested by $\mathcal{Z}$ and writes them to sets $Q_i = \{q_{ij} | j \in [n_i]\}$. Here index i corresponds to PID of corrupt P_i, $n_i \geq n$ is polynomially bounded, as $\mathcal{Z}$ is. Denote $Q = \cup_{i|\mathsf{P}_i \in I \cup \mathsf{P}_0} Q_i$ – the joint query set of corrupt parties. The environment might continue making such a queries up to the end of the protocol, and $\mathcal{S}$ adds them in Q up to the end of Step 5 (when the effective intersection of corrupt parties inputs is extracted).

In step 2, the simulator emulates $\mathcal{F}_{AppROT}^{Online}$ towards corrupt P_0, sends to it the uniformly random permutations $\pi_i : [N_{\text{OT}} - N_{cc}] \to [N_{\text{BF}}]$ on behalf of each honest P_i and receives from P_0 permutations π_0^i. The simulator uses J_i and π_0^i to construct the effective Bloom filter of P_0 $\tilde{\text{BF}}_{0i}$ in the interaction with each honest P_i. Then the simulator computes:

$$M^i = \pi_0^i(M^{i\prime}), \quad \hat{M}^i = \pi_i(\hat{M}^{i\prime}).$$

If and once all $\mathcal{F}_{AppROT}^{Online}$ executions are completed, $\mathcal{S}$ asks $\mathcal{F}_{\text{gRO}}$ for all the illegitimate queries with the session ID of current execution and append them to Q. Then it extracts an effective input of the adversary as $\tilde{X}_I = \left\{q \in Q | \forall s \in h_*(q), \forall j \notin I, \tilde{\text{BF}}_{0j}[s] = 1\right\}$ – queries which are presented in all the extracted Bloom filters of P_0.

After all $\mathcal{F}_{AppROT}^{Online}$ executions are emulated, when the effective malicious input $\tilde{X}_I$ is extracted, $\mathcal{S}$ sends either it or $\perp$ to the ideal functionality $\mathcal{F}_{\text{MPSI}}$ as the input of each of the corrupted parties, 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 $\mathcal{F}_{\text{PSImple}}^{Offline}$ or $\mathcal{F}_{AppROT}^{Online}$'s ended with $\perp$, the simulator gives $\perp$, which is the effective input of the adversary, to $\mathcal{F}_{\text{MPSI}}$ and receives $\perp$ from there.

To simulate a step-4 message by an honest party $\mathsf{P}_i \notin I$ replied to $\mathcal{Z}$, right after all the $\mathcal{F}_{AppROT}^{Online}$'s of P_i are completed, even if there are another running $\mathcal{F}_{AppROT}^{Online}$'s for other parties:

- If P_i received $\perp$ from $\mathcal{F}_{\text{PSImple}}^{Offline}$ or $\mathcal{F}_{AppROT}^{Online}$, $\mathcal{S}$ sends $\perp$ to $\mathcal{Z}$ as the message for P_0.
- If P_i's $\mathcal{F}_{AppROT}^{Online}$'s are completed successfully, and P_i is not the last honest party whose $\mathcal{F}_{AppROT}^{Online}$'s done, $\mathcal{S}$ sends a uniformly random GBF^{i*} to $\mathcal{Z}$ as the message for P_0.
- If all the $\mathcal{F}_{AppROT}^{Online}$'s are completed, but there were at least one $\perp$, and P_i is the last honest party whose $\mathcal{F}_{AppROT}^{Online}$'s done (without $\perp$), $\mathcal{S}$ sends a uniformly random GBF^{i*} to $\mathcal{Z}$ as the message for P_0.
- If $\mathcal{F}_{\text{PSImple}}^{Offline}$ and all the $\mathcal{F}_{AppROT}^{Online}$'s are completed successfully, and P_i is the last honest party whose $\mathcal{F}_{AppROT}^{Online}$'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 $\text{GBF}^j = M^j \bigoplus \hat{M}^j$ 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 B.1; note, that positions at indices r such that $\text{BF}_j[r] = 0$ are entirely and uniformly random.
  - computes GBF_{temp}^{j*} honestly as in 6th step of the protocol for all j such that $\mathsf{P}_j \notin (I \cup \mathsf{P}_0)$, and $\text{GBF}^{i*} = \bigoplus_{\mathsf{P}_j \notin (I \cup \mathsf{P}_0)} \text{GBF}_{temp}^{j*} \bigoplus_{j \neq i, \mathsf{P}_j \notin (I \cup \mathsf{P}_0)} \text{GBF}^{j*}$ (here GBF^{j*} are messages sent by $\mathcal{S}$ on behalf of other honest parties in 4th 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 functionalities $\mathcal{F}_{\text{PSImple}}^{Offline}$ and $\mathcal{F}_{AppROT}^{Online}$ 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 up to 3 for some given party, and step 4, *before* the last honest party sends its step-4-message. In the second part we analyze the last message delivered in step 6. 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 $\mathcal{F}_{\text{PSImple}}^{Offline}$ call, as in the previous case, all the inputs and outputs are distributed identically in both worlds and do not depend on honest parties inputs by definition of $\mathcal{F}_{\text{PSImple}}^{Offline}$ and by the construction of $\mathcal{S}$. In particular, $\{S^{il}\}_{\substack{\mathsf{P}_i \in \mathcal{P} \setminus (I \cup \mathsf{P}_0) \\ \mathsf{P}_l \in I}}$ are random i.i.d strings, messages $\{M_*^{i\prime}/\perp\}_{\mathsf{P}_i \in \mathcal{P} \setminus (I \cup \mathsf{P}_0)}$ are also uniformly random, and $\{\hat{M}^{i\prime}/\emptyset\}_{\mathsf{P}_i \in \mathcal{P} \setminus (I \cup \mathsf{P}_0)}$ are defined by the adversary based on the previous transcript of outputs from $\mathcal{F}_{\text{PSImple}}^{Offline}$.

Calls to the $\mathcal{F}_{\text{gRO}}$ also don't break indistinguishability, because they actually refer to the same global Random Oracle in both worlds.

The permutations received by the adversary from $\mathcal{F}_{AppROT}^{Online}$ are also distributed identically in both worlds due to the randomness. The order in which the simulator send answers to P_0 or to $\mathsf{P}_i \in I$ is the same as the order in which honest parties answer in the real-world execution.

Note that when a step-4 message from one or more honest party was not yet sent when another honest party $\mathsf{P}_{l'}$ sends its step-4 message and other honest parties P_l have not contributed their step-4 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-2 output share, it in particular hasn't sent its step-4 message yet, and every other party is not the last, and the value the latter would sent is a random independent sting (share).

Let us now compare the effect of step 4, assuming $\mathcal{F}_{\text{PSImple}}^{Offline}$ and all executions of $\mathcal{F}_{AppROT}^{Online}$ 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 $\tilde{X}_I$, $H = \tilde{X}_I \bigcap \left(\bigcap_{\mathsf{P}_i \in \mathcal{P} \setminus I \cup \{\mathsf{P}_0\}} X_i \right)$, 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.[21] 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 4). (2) If (1) does not happen, for j outside of a set $h_*(x)$ we queried in 2(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 2(a) (from all parties) on which the $\mathcal{F}_{\text{gRO}}$ was not queried (resulting in no indices from the first or second kind). As no $\perp$ occurred in $\mathcal{F}_{\text{PSImple}}^{Offline}$ or any $\mathcal{F}_{AppROT}^{Online}$ call, the probability of this is $\leq p_{False} < 2^{-\sigma}$.

Now let us now consider the case when $\mathcal{F}_{\text{PSImple}}^{Offline}$ or at least one of the $\mathcal{F}_{AppROT}^{Online}$'s resulted in $\perp$ in the real world. In this case $\mathcal{S}$ sends $\perp$ to $\mathcal{F}_{\text{MPSI}}$, and thus receives $\perp$. The simulation of step 4'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 4, 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 4 due to part of the shares contributed by the stalled party missing. □

[21] 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 $\mathbb{F}_2$.