PSI\textsuperscript{imple}: 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 PSI\textsuperscript{imple}, 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 PSI\textsuperscript{imple} protocol, we implemented the protocol and ran experiments with up to 32 parties and 2\textsuperscript{20} inputs. We show that PSI\textsuperscript{imple} 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 PSI\textsuperscript{imple}.

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 our 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, occurring 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 \(D\) 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 \(x \in D\) is encoded into the BF by setting all indices \(h_l(x), \ldots, h_k(x)\) 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 \(D\) 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 \(P_0, P_1\) 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 bitwise 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 \(\sigma\) 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 \(x^1\) 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 \((P_0, P_1, \ldots, P_t)\), by using add- tive 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 \(\cap_{i \in \{0, \ldots, t\}} S_i\), 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 \(P_i, t \geq 1\) 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 $y_0$, as in [14, 39], $y_0$ 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 $y_0$ 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 $y_0$ 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 $y_0$; 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 $y_{0,i}$, 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., $y_{0,i}$). 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 $P_1, \ldots, P_t$ 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 $y_{0,i}'$ of their shares (i.e., $y_{0,i}'$ is the sum of the indices $h(x_i) + \ldots + h(x_t)$ (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 $y_{0,i}'$ received by $P_0$ from the other parties using the OT in the first part, is equal to the sum of the codewords $y_{0,i}'$ 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 $y_{0,i}'$, one from each party, such that their sum is equal to $y_0$; recall that for security reasons, these $y_{0,i}'$s 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 $y_0$ 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,
P₀ can recover the intersection by checking, for each element x in its input set, if yₓ = 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 P₁ that are equal to the codewords computed by P₀, 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 yₓ' (one from each party) that sum to yₓ, the codeword computed by P₀. To the best of our knowledge, the best solution to this problem is still exponential in the number of parties, i.e., nΟ(t).

Instead, 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 Pᵢ a restriction on the size of the data set it uses when interacting with Pⱼ, we let Pᵢ act as the receiver in a (approximated, random) K-out-of-N OT execution with Pⱼ. 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 P₀ 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 P₀ perform two approximated, random K-out-of-N OT execution with each party Pᵢ. Then, P₀ sums all 2r GBFs it obtained in these executions. Let C₀ be the resulting (cumulative) GBF of P₀. Similarly, let Gᵢ be the GBF obtained by party Pᵢ as the sum of two GBFs it saw in the interaction with P₀. As before, each Pᵢ needs to rerandomize Gᵢ before sending it to P₀. Let Gᵢ′ be the rerandomized version of Gᵢ.

By generalizing our observation in the 2-party case, it follows that G₀ ⊕ ∞∈[t] G₀' is a GBF for the intersection of all the parties, where yₓ = 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 P₀ with party Pⱼ), appears twice in the above summation, once for P₀ and once for Pⱼ. So the idea is to allow P₀ to learn the above summation. However, if each Pⱼ simply sends Gⱼ' to P₀, then security is breached as P₀ can compute the intersection with each party separately. To overcome this, we let the parties first share their Gⱼ' 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 P₀, they allow P₀ to reconstruct the GBF of the intersection G₀ ⊕ ∞∈[t] G₀', 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 2⁸ to 2³⁰, 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, with existing implementations of state-of-the-art multi-party 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 N₀T executions of 1-out-of-2 OTs to allow the receiver to learn approximately K indices from the sender’s GBF of length l. 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.

<table>
<thead>
<tr>
<th>n</th>
<th>Analysis</th>
<th>k</th>
<th>N₀B</th>
<th>N₀T</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>This work</td>
<td>147</td>
<td>64,733</td>
<td>74,379</td>
</tr>
<tr>
<td>2¹²</td>
<td>[39]</td>
<td>94</td>
<td>1,121,959</td>
<td>1,187,141</td>
</tr>
<tr>
<td></td>
<td>This work</td>
<td>134</td>
<td>851,085</td>
<td>901,106</td>
</tr>
<tr>
<td>2¹⁶</td>
<td>[39]</td>
<td>91</td>
<td>10,379,297</td>
<td>10,992,857</td>
</tr>
<tr>
<td></td>
<td>This work</td>
<td>131</td>
<td>12,660,342</td>
<td>12,948,963</td>
</tr>
<tr>
<td>2²⁰</td>
<td>[39]</td>
<td>90</td>
<td>257,635,123</td>
<td>260,252,093</td>
</tr>
<tr>
<td></td>
<td>This work</td>
<td>129</td>
<td>197,052,485</td>
<td>198,793,103</td>
</tr>
</tbody>
</table>

### 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.

<table>
<thead>
<tr>
<th># parties</th>
<th>protocol</th>
<th>sec. model</th>
<th>n = 2^8</th>
<th>n = 2^{12}</th>
<th>n = 2^{16}</th>
<th>n = 2^{18}</th>
<th>n = 2^{20}</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>[21] augmented semi-honest</td>
<td>Direct</td>
<td>1.626</td>
<td>2.113</td>
<td>9.909</td>
<td>36.702</td>
<td>**</td>
</tr>
<tr>
<td></td>
<td>[28] semi-honest</td>
<td>malicious</td>
<td>0.075</td>
<td>0.277</td>
<td>3.642</td>
<td>12.095</td>
<td>47.687</td>
</tr>
<tr>
<td></td>
<td>[42] non-standard</td>
<td>Direct</td>
<td>0.76</td>
<td>2.95</td>
<td>40.91</td>
<td>–</td>
<td>513.27</td>
</tr>
<tr>
<td></td>
<td>PSImple</td>
<td>malicious</td>
<td>0.194</td>
<td>5.474</td>
<td>**</td>
<td>**</td>
<td>**</td>
</tr>
<tr>
<td>6</td>
<td>[21] augmented semi-honest</td>
<td>Direct</td>
<td>1.116</td>
<td>0.383</td>
<td>6.057</td>
<td>21.804</td>
<td>82.643</td>
</tr>
<tr>
<td></td>
<td>[28] semi-honest</td>
<td>Direct</td>
<td>5.135</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td></td>
<td>PSImple</td>
<td>malicious</td>
<td>0.237</td>
<td>0.580</td>
<td>6.906</td>
<td>30.117</td>
<td>141.277</td>
</tr>
<tr>
<td>10</td>
<td>[21] augmented semi-honest</td>
<td>Direct</td>
<td>1.711</td>
<td>2.230</td>
<td>14.259</td>
<td>56.842</td>
<td></td>
</tr>
<tr>
<td></td>
<td>[28] semi-honest</td>
<td>PSImple</td>
<td>0.161</td>
<td>0.748</td>
<td>11.718</td>
<td>46.345</td>
<td>174.908</td>
</tr>
<tr>
<td></td>
<td>PSImple</td>
<td>malicious</td>
<td>0.302</td>
<td>0.868</td>
<td>7.963</td>
<td>31.378</td>
<td>147.292</td>
</tr>
<tr>
<td></td>
<td>[28] semi-honest</td>
<td>PSImple</td>
<td>0.252</td>
<td>1.122</td>
<td>16.626</td>
<td>61.019</td>
<td>241.864</td>
</tr>
<tr>
<td></td>
<td>PSImple</td>
<td>malicious</td>
<td>0.367</td>
<td>0.911</td>
<td>13.177</td>
<td>47.334</td>
<td>*</td>
</tr>
</tbody>
</table>

* the protocol crashed
** Direct crashed on 4 parties and 2^{16} inputs; on 4 parties and 2^{16} inputs it took 109.055 seconds, while PSImple finished in 1.723 seconds

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 \( \sigma \), and the statistical security parameter by \( \lambda \). In our implementation and experiments, \( \sigma = 128 \) and \( \lambda = 40 \). For \( l \in \mathbb{N} \), \([l] \) denotes the set \( \{1, \ldots, l\} \). We use the notation \( \mathcal{P} = \{P_0, \ldots, P_l\} \) for the set of parties, where \( P_0 \) is the evaluating party, and the remaining \( l \) 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 \( P = \{ P_0, ..., P_r \} \), each having up to \( n \) items from some domain \( D \) 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 \( \mathcal{T}_{\text{MPSI}} \) is given in Figure 1.

We point out a couple of properties that are less standard, which we inherit from previous work on PSL. 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.

\[
\mathcal{T}_{\text{MPSI}}
\]
\[
P_0, P_1, ..., P_r - \text{parties; } P_i \text{ holds } X_i = \{ x_{i1}, x_{i2}, ..., x_{in_i} \}, n_i \leq n \quad - \text{the private input of the party;}
\]
\[
n \quad - \text{the maximal size of the input set of any honest party;}
\]
\[
n' \geq n \quad - \text{the maximal allowed size of the input set of any malicious party;}
\]
\[
\text{Inputs: } X_i \text{ from each party } P_i, n, \sigma.
\]
\[
\text{Computation: if size of the corrupt party input set is bigger than } n', \text{then the functionality aborts. Else it computes the intersection of all the data sets: } X = \cap_{i=1}^r X_i.
\]
\[
\text{Outputs: Party } P_0 \text{ receives } X \text{ from the functionality, all other parties receive no output.
\]

Figure 1: \( \mathcal{T}_{\text{MPSI}} \) – 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 \( S = S_1 \oplus S_2 \) 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 \( h_1, ..., h_k : \{0,1\}^* \rightarrow \{0,1\} \). 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 \( p_{FP} \) 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 \( \sigma \)-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.

\[
y_x = \bigoplus_{i \in h_i(x)} \text{GBF}[i],
\]

where \( h_i(x) \)
\[
def \{ h_j(x) | j \in \{k\} \}, \]

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 \( BF_1 \land BF_2 \) (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 \( GBF_1 \land GBF_2 \) 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 \( GBF_1 \land GBF_2 \).

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.

8The 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 \( \Pi_{\text{APPROT}} \). We recall that Rindal and Rosulek [39] require a maliciously secure K-out-of-N OT protocol to ensure that \( \Pi_0 \) 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 \( (P_{\text{false}}) \) 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 \( \Pi_{\text{APPROT}} \), is formally described in Figure 2. We next give a rough overview of the \( \Pi_{\text{APPROT}} \) protocol. In the first phase, the two parties invoke \( N_{\text{OT}} \) 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 \( N_{\text{BF}} \) 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 \( \Pi_{\text{APPROT}} \)

**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:**

1. \( [1\text{-out-of-2 OTs}] \) The sender and the receiver call \( \Pi_{\text{OT}} \) (performing \( N_{\text{OT}} \) 1-out-of-2 OTs). The receiver receives bits \( c_1, ..., c_{N_{\text{OT}}} \) with \( n \) ones among them, and \( N_{\text{OT}} - n \) 0’s (randomly permuted), and the sender chooses uniformly random \( m_0 = \{ m_{i_1}, ..., m_{i_{N_{\text{OT}}}^0} \} \) and \( M_J = \{ m_{j_1}, ..., m_{j_{N_{\text{OT}}}^1} \} \). As a result, in the \( j^{th} \) OT, the receiver uses its choice bit \( c_j \) and learns \( m_{j,c_j} \) (of length \( \sigma \)).

2. \( \text{[cut-and-choose challenge]} \) The sender randomly chooses the set \( C \subset [N_{\text{OT}}] \) of size \( N_{\text{CE}} \) and sends \( C \) to the receiver.

3. \( \text{[cut-and-choose response]} \) The receiver checks that \( |C| = N_{\text{CE}} \), 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 \( r^* = \bigoplus_{j \in R} m_{j,c_j} \). The sender aborts if \( r^* \neq \bigoplus_{j \in R} m_{j,c_j} \) or if \( |C| - |R| > N_{\text{maxones}} \), where \( N_{\text{maxones}} \) is the maximal number of ‘1’s allowed in the cut-and-choose OTs.

**Online phase:**

4. \( \text{[permute unopened OTs]} \) The receiver chooses a random injective function \( \pi : \{ 1 \} \rightarrow \{ N_{\text{OT}} \} \setminus C \) such that \( B[j] = c_\pi(j) \), and sends \( \pi \) to the sender.

The receiver permutes its random values \( m_{j,\pi} \) according to \( \pi \), and the sender permutes \( m_{\pi,\pi} \) according to \( \pi \).

**Outputs:** The receiver outputs \( M_c = \{ m_{j,\pi} \}_{j \in [B]} \); the sender outputs \( M = \{ m_{j,\pi} \}_{j \in [B]} \).

---

**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. 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 \( \Pi_{\text{APPROT}} \) with each other party \( P_i \) independently and then have \( P_0 \) XOR all the GBFs received from the \( \Pi_{\text{APPROT}} \) 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 \( h_1, ..., h_k \) is the XOR over the strings from the GBF at indices \( h_1(x), ..., h_k(x) \). 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

---

\(^{a}\)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 $G_{j}^{\mathbf{i}}$ as the XOR of all the shares it holds. After that, each party $P_2$ sends to $P_0$ the codewords attributed to its set $X_2$ with respect to its cumulative GBF $G_{j}^{\mathbf{i}}$. 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 $y_x = y_1 \oplus \cdots \oplus y_t$.

The above is indeed correct and secure. However, an exhaustive search for a combination of codewords that sum to $y_x$ grows exponentially with the 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 $\Pi_{App\text{ROT}}$ 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 $\Pi_{App\text{ROT}}$, with the parties playing reversed roles. In this way, the binding of $P_1$ is achieved similarly to its binding to $P_0$.

As a result, each party $P_i$ receives two garbled Bloom filters, one from each execution of $\Pi_{App\text{ROT}}$. Put differently, $P_i$ holds its own full GBF, and the $k$ coordinates it chose from the GBF of $P_{i-1}$ (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 GBF$^S \overset{def}{=} \text{GBF}^0 \oplus \text{GBF}^1$ (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$^S$ 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$^S$. 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 GBF$^S[s] = 0$.

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

Next, $P_1$ sends GBF$^1\text{S}$ to $P_0$. Since rerandomization does not affect the codewords, it follows that if $x$ is in $X_0 \cap X_2$, then it has the same codeword in both GBF$^0$ and GBF$^1\text{S}$. In other words, GBF$^* \overset{def}{=} \text{GBF}^0 \oplus \text{GBF}^1\text{S}$ 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 $P_0$’s input, if $x_{0j}$ is in the intersection, by testing $\left(\bigoplus_{x \in h_i(x_{0j})} \text{GBF}^*[s] = 0\right)$.

### 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 $\Pi_{App\text{ROT}}$. Initially, it would seem that each party needs to perform two instances of $\Pi_{App\text{ROT}}$, 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 $\Pi_{App\text{ROT}}$ with $P_0$.

After the executions of $\Pi_{App\text{ROT}}$, the protocol proceeds as in the 2-party case, with each party XORing the garbled Bloom filters it received from its executions of $\Pi_{App\text{ROT}}$, and rerandomizing them. Recall that the rerandomization operation is done to hide coinciding requested coordinates in the executions of $\Pi_{App\text{ROT}}$, 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 $\Pi_{App\text{ROT}}$. Let GBF$^1\text{S}$ be the rerandomized version of the cumulative GBF obtained by $P_1$ in the two interactions of $\Pi_{App\text{ROT}}$ it had with $P_0$. The idea is to let $P_0$ learn GBF$^* \overset{def}{=} \text{GBF}^0 \oplus \bigoplus_{i \in [t]} \text{GBF}^i\text{S}$, 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 $\left(\bigoplus_{x \in h_i(x_{0j})} \text{GBF}^*[s] = 0\right)$.

However, we cannot simply let each party $P_i$ send GBF$^i\text{S}$ 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 MPSImple protocol $\Pi_{PSImple}$ into two phases: an offline-phase $\Pi_{Online PSImple}$, 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 $\mathcal{BF}_i$ from its input set $X_i$ and $n - n_i$ additional random dummy items, to perform $\Pi_{\text{AppROT}}$’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 $\mathcal{F}_{\text{OT}}^{\sigma,N}$ functionality constitutes $N$ parallel ideal instances of 1-out-of-$\sigma$-long string oblivious transfer. The $\mathcal{F}_{\text{RO}}$ 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 $\Pi_{\text{PSImple}}$ protocol of Figure 3 securely realizes the functionality $\mathcal{F}_{\text{PSI}}$ with statistical UC-security with abort in the presence of a static, malicious adversary corrupting any number of parties in the $\mathcal{F}_{\text{OT}}^{\sigma,N}$, $\mathcal{F}_{\text{RO}}$-hybrid model, where the adversary makes a polynomially-bounded number of queries to $\mathcal{F}_{\text{RO}}$ (where the $\mathcal{F}_{\text{RO}}$ 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 $\Pi_{\text{PSImple}}$ 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 GBF’s 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 $p_{\text{False}}$), 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 $\sigma$-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 $\Pi_{\text{PSImple}}$**. 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 $\mathcal{F}_{\text{PSI}}$ 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 $\mathcal{F}_{\text{RO}}$ 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 $\mathcal{F}_{\text{OT}}^{\sigma,N}$ 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 $\mathcal{F}_{\text{RO}}^{\sigma,N}$. 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 re-randomized GBF’s 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 $\Pi_{\text{AppROT}}$. We show that the size of the garbled Bloom...
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 \( \Pi_{\text{AppROT}} \), 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.

The table below presents a comparison of the overall communication complexity between various protocols.

<table>
<thead>
<tr>
<th>Protocol</th>
<th>Security</th>
<th>offline</th>
<th>online</th>
</tr>
</thead>
<tbody>
<tr>
<td>IOPT18 [21]</td>
<td>semi-honest</td>
<td>-</td>
<td>( O(t^n + k) )</td>
</tr>
<tr>
<td>KMPRT17 [28]</td>
<td>semi-honest</td>
<td>-</td>
<td>( O(tn + k) )</td>
</tr>
<tr>
<td>IOPT18 [21]</td>
<td>augmented semi-honest</td>
<td>( O(tk^n) )</td>
<td>( O(tk^n) )</td>
</tr>
<tr>
<td>KMPRT17 [28]</td>
<td>augmented semi-honest</td>
<td>( O(t^2k) )</td>
<td>( O(t^n + k) )</td>
</tr>
<tr>
<td>CSJ12 [9]</td>
<td>malicious (honest majority)</td>
<td>( O(t^2k) )</td>
<td>( O(t^2k) )</td>
</tr>
<tr>
<td>KS05 [26]</td>
<td>malicious</td>
<td>( O(t^2k) )</td>
<td>( O(t^2k) )</td>
</tr>
<tr>
<td>HV17 [19]</td>
<td>malicious</td>
<td>( O(t^2k) )</td>
<td>( O(t^2k + t + n\log(n)) )</td>
</tr>
<tr>
<td>GN19 [15]</td>
<td>malicious</td>
<td>( O(t^2k) )</td>
<td>( O(t^2k + t\log(n)) )</td>
</tr>
<tr>
<td>PSimple</td>
<td>malicious</td>
<td>( O(t\log(n) + n\log(n) + \phi) )</td>
<td>( O(t\log(n) + n\log(n) + \phi) )</td>
</tr>
</tbody>
</table>
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 \( \Pi_{AppROT} \);
- \( N_{BF} \) – size of the Bloom filter of the receiver;
- \( N_{OT}^1 \) – 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, \( \sigma \) is the computational security parameter and \( \lambda \) 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 \( n, \sigma \) and \( \lambda \), we have to set \( k, N_{BF}, N_{cc}, N_{OT}, N_{maxones} \) and \( N_{OT}^1 \). 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 \( n, \sigma \), and \( \lambda \), there exist \( k, N_{BF}, N_{cc}, N_{OT}^1 \), 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 \( k_{min} = \frac{\sigma}{2}, k_{max} = 2\sigma, N_{BF}_{min} = nk \) and \( N_{BF}_{max} = 3nk \). 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 5.

### 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].

<table>
<thead>
<tr>
<th>Parameter</th>
<th>( n = 2^8 )</th>
<th>( n = 2^{12} )</th>
<th>( n = 2^{14} )</th>
<th>( n = 2^{16} )</th>
<th>( n = 2^{18} )</th>
<th>( n = 2^{20} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( k )</td>
<td>147</td>
<td>134</td>
<td>132</td>
<td>131</td>
<td>130</td>
<td>129</td>
</tr>
<tr>
<td>( N_{BF} )</td>
<td>64,733</td>
<td>851,085</td>
<td>3,253,782</td>
<td>12,660,342</td>
<td>49,786,942</td>
<td>197,052,485</td>
</tr>
<tr>
<td>( N_{OT} )</td>
<td>74,379</td>
<td>901,106</td>
<td>3,373,092</td>
<td>12,948,963</td>
<td>50,491,817</td>
<td>198,793,103</td>
</tr>
<tr>
<td>( N_{OT}^1 )</td>
<td>32,885</td>
<td>428,425</td>
<td>1,637,989</td>
<td>6,376,614</td>
<td>25,026,621</td>
<td>98,728,744</td>
</tr>
<tr>
<td>( N_{cc} )</td>
<td>7,473</td>
<td>47,422,882</td>
<td>105,863</td>
<td>262,924</td>
<td>655,322</td>
<td>1,644,397</td>
</tr>
<tr>
<td>( N_{maxones} )</td>
<td>3,627</td>
<td>21,160</td>
<td>52,620</td>
<td>131,385</td>
<td>327,830</td>
<td>821,450</td>
</tr>
</tbody>
</table>

We separate the protocol into two phases: an online 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 \( \Pi_{AppROT} \), which implies that the maximal number of threads in Steps 2 and 5 (\( \Pi_{AppROT} \)) are 8 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 \( 2^{38}-2^{200} \) 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.

---

\( ^8 \)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.

\( ^9 \)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.
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 particular 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 $\geq 6$.

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. We see from Table 2 that 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 multiplication 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

A BASIC FUNCTIONALITIES

\[ \mathcal{F}_{OT}^{\sigma} \]

**Parameters:** \( \sigma \) 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:** \( \mathcal{F}_{OT}^{\sigma} \) – Ideal 1-out-of-2 OT functionality

\[ \mathcal{F}_{OT}^{\sigma,N} \]

**Customers:** the sender, the receiver.

**Parameters:** \( N \) is the number of resulting parallel OT’s for \( \sigma \)-bit length of the OT strings.

**Inputs:** \( M_S = \{m_{S1}, ..., m_{SN}\} \) and \( M_S = \{m_{S1}, ..., m_{SN}\} \) 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:** \( \mathcal{F}_{OT}^{\sigma,N} \) – Ideal \( N \) parallel 1-out-of-2 OT’s for \( \sigma \)-bit strings functionality

\[ \mathcal{F}_{AppRO} \]

**Parameters:** \( \sigma \) – computational security parameter; \( \lambda \) – statistical security parameter;

\( N \) – number of items to create; \( K' \) – maximal allowed size of the request.

**Inputs:** From the receiver: \( I = \{i_1\} \in [K], K \leq K' \); from the sender: no input.

**Outputs:** Upon receiving \( I \) from the sender, samples the uniformly random \( M = \{m_1, m_2, ..., m_N\} \), where \( m_i \)’s are \( \sigma \)-bit strings, and computes \( M_s = \{m_{1s}, m_{2s}, ..., m_{ls}\} \). Gives \( M \) to the sender, gives \( M_s \) to the receiver.

**If the adversary corrupts the receiver**

The functionality works in two steps:

1. The corrupt receiver chooses \( K'' \leq K' \) and sends it to the ideal functionality.
2. Upon receiving \( K'' \) from the functionality, it samples and gives the receiver the uniformly random \( M' = \{m'_1, m'_2, ..., m'_{K''}\} \), where \( m'_i \)’s are \( \sigma \)-bit strings.

**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 \( m_i \)’s are \( \sigma \)-bit strings. Gives \( M_s = \{m_{1s}, m_{2s}, ..., m_{ls}\} \) to the receiver.

---

**Figure 8:** \( \mathcal{F}_{AppRO} \) – Ideal approximate \( K \)-out-of-\( N \) Random OT functionality

\[ \mathcal{F}_{gRO} \]

**Parameters:**

\( k \) – number of hash-functions; \( N \) – size of input domain; \( t(N) \) – size of output domain of the hash function;

\( \mathcal{T} \) – 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 \( P = \{PID, SID\} \) or the adversary \( S \) do:

1. If there is a pair in a form \((x, o)\) in \( Q \), then return \( o \) to \( P \) (or \( S \)).
2. else, sample \( a \) a uniformly random \( o \in [t(N)]^k \), add \((x, o)\) to the list \( Q \) and return \( o \) to \( P \) (or \( S \)).

**Parse:**

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

---

**Figure 9:** \( \mathcal{F}_{gRO} \) – Global Random Oracle functionality
B ALGORITHMS FOR THE GARBLED BLOOM FILTER

B.1 Re-randomization Algorithm for a Garbled Bloom Filter

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

Input:
- The set of items \(X = (x_1, \ldots, x_n)\); the set of codewords \(Y = (y_1, \ldots, y_n): |y_i| = \sigma, (i \in [n])\);
- the family of hash-indices \(H^* = (h_1(x_1), \ldots, h_k(x_k)): h_i(x_i) = \{s|h_j(x_j) = s, j \in [k]\}, (i \in [n])\).

Algorithm:
1: GBF = empty \(N_{GBF}\)-size array of \(\sigma\)-long strings
2: for \(i=1\) to \(n\) do
3: \(\text{finalInd}=1\)
4: \(\text{finalShare}=y_i\)
5: for each \(j \in h_s(x_i)\) do
6: if GBF\([j]\) is empty then
7: if finalInd==1 then
8: finalInd= \(j\)
9: else
10: GBF\([j]\) \(\leftarrow\) \(\{0, 1\}^\sigma\)
11: finalShare=finalShare\@GBF\([j]\)
12: else
13: finalShare=finalShare\@GBF\([j]\)
14: GBF[finalInd] =finalShare \(\uparrow\)
15: for \(i = 0\) to \(N_{GBF} - 1\) do
16: if GBF\([i]\) is empty then
17: GBF\([i]\) \(\leftarrow\) \(\{0, 1\}^\sigma\)
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 \(h_s(x)\)

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

Input:
- Item \(x\);
- \(N_{BF}\) - length of GBF;
- family of hash-functions \(H = (h_1, \ldots, h_k): \{0, 1\}^n \rightarrow \{0, 1\}^{N_{BF}}, (i \in [k])\).

Algorithm:
1: \(h_s(x) = \text{empty } 0\)-size array
2: for \(i=1\) to \(k\) do
3: if \(h_i(x) \notin h_s(x)\) then
4: add \(h_i(x)\) to \(h_s(x)\)
5: return \(h_s(x)\)

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

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

Algorithm CodewordGBF\((GBF, x, h_s(x), N_{BF}, \sigma)\)

Input:
- The item \(x\); the garbled Bloom filter GBF; the length of GBF \(N_{BF}\); the bitlength of string in GBF – \(\sigma\);
- \(h_s(x)\) – set of hash-indices of \(x\); \(\forall i \in h_s(x), i \in [N_{BF}]\).

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

Output: \(y\) – The codeword for \(x\) in the garbled Bloom filter GBF indexed by \(h_s(x)\).

\(\uparrow\) 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 \((7)\), \(p_{\text{false}} < 2^{-\sigma}\), so the union bound over all \(x \in X\) is \(n2^{-\sigma}\), which is still negligible in \(\sigma\).
C PARAMETERS OF $\Pi_{\text{AppROT}}$

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 $\Pi^{\text{Offline}}_{\text{AppROT}}$, 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 $\lambda$.

**Definition C.1.** A function $\mu : \mathbb{N} \rightarrow \mathbb{N}$ 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 in the analysis of our work 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 one’s to construct the GBF in the protocol, but this happens only with negligible probability.

The constraints on $p_{\text{false}}$ come from the second and the third requirements. Namely, the sender rejects the cut-and-choose response, if $Pr[p_{\text{false}} \geq 2^{-\sigma}] > 2^{-\lambda}$. Therefore, the choice of parameters should ensure that the false positive probability of the Bloom filters, denoted $p_{\text{false}}$, is negligible. Until 2008, it was believed that $p_{\text{false}} = p^k$ [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 $p_{\text{false}}$. They further presented the precise formula for $p_{\text{false}}$, 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 $p_{\text{false}}$. However, finding the maximal number of 1’s in the Bloom filter from $p_{\text{false}}$ 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 $p_{\text{false}}$.

We next give the formal details:

**GBF parameters:**
- $N_{OT}$: number of OTs in $\Pi_{\text{AppROT}}$;
- $N_{BF}$: size of the Bloom filter of the receiver;
- $N_{OT}^i$: number of 1’s among $N_{OT}$ choice bits of the receiver;
- $k$: number of Bloom filter hash functions;
- $N_{ce}$: number of bits to choose for the cut-and-choose check;
- $N_{\text{maxones}}$: the maximal $k$’s among the $N_{ce}$ choice bits allowed in order to pass the cut-and-choose check.

**Theorem C.2.** By choosing the parameters of $\Pi_{\text{AppROT}}$ under the following constraints

\[ N_0 = \left[ N_{BF} \left( 1 - e^{-\frac{nk}{N_{BF}}} \right) + \sqrt{\frac{nk\ln 2}{2}} \right], \quad N_1 = \left[ N_{BF} \left( 1 - e^{-\frac{nk}{N_{BF}}} \right) + \sqrt{\frac{nk\ln 2}{2}} \right], \quad N = N_0 + N_1, \alpha = N_1/N; \tag{2} \]

\[ N_{ce} > 0 : \quad N_{\text{maxones}} = N_{\text{maxhonest}} > 0, \quad \text{where} \]

\[ N_{\text{maxones}} = \left[ N_{BF} N_{ce} 2^{-\frac{\alpha}{2}} \left( 1 + \frac{k(k - 1)}{2N_{BF}} \left( \frac{1}{\alpha} - 1 \right) \right)^{\frac{1}{2}} - \sqrt{\frac{\alpha N_{ce} \ln 2}{2}} \right], \quad N_{\text{maxhonest}} = \left[ \alpha N_{ce} + \sqrt{\frac{\alpha N_{ce} \ln 2}{2}} \right], \tag{3} \]

\[ N_{\Delta} = \left[ \min(\alpha, 1 - \alpha) \right] \sqrt{\frac{\lambda N_{ce} \ln 2}{2}}, \quad N_{OT} = N + N_{\Delta} + N_{ce}, \quad N_{OT}^i = \left[ \alpha N_{OT} \right]. \tag{4} \]

\[ \text{the following requirements hold:} \]

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 $1 - 2^{-\frac{\lambda + 1}{2}}$;
2. $Pr[p_{\text{false}} \geq 2^{-\sigma}] \leq 2^{-\lambda}$ with either honest or malicious receiver in $\Pi_{\text{AppROT}}$;
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.

\[ \Pr \left[ \sum_{i=1}^{n} X_i > \lambda \right] \leq \frac{1}{\sqrt{2\pi \lambda}} \int_0^{\lambda} e^{-\frac{x^2}{2}} dx, \tag{5} \]
• 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 \( \Pr |F - E(F)| \geq \varepsilon \leq 2e^{-\frac{\varepsilon^2}{2}} \). 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

\[
\Pr[X \geq (p + t)n] \leq \left( \frac{p}{p+t} \right)^{p+t} \left( \frac{1-p}{1-p-t} \right)^{1-p-t} n^p \leq e^{-2\varepsilon^2 n}.
\]

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 \( q = (1 - 1/N_{BF})^{nk} = e^{-nk} \), and to 1 is \( \rho = 1 - q \) [29, p 116].

Using the Azuma-Hoeffding inequality [29, p 355], we get \( \Pr|N_0 - E(N_0)| \geq \varepsilon \leq e^{-\frac{\varepsilon^2}{2 \rho}} \). We require that the number of zeroes is not enough to build the Bloom filter with the negligible in \( \lambda \) probability. Hence \( e^{-\frac{\varepsilon^2}{2 \rho}} \leq 2^{-\lambda} \); solving this inequality, we get \( \varepsilon \geq \sqrt{\frac{nk \ln 2}{2 \rho}} \).

Therefore we should have at least \( N_0 \geq N_{BF}q + \sqrt{\frac{nk \ln 2}{2 \rho}} \) zeroes after cut-and-choose.

By implementation of the Azuma-Hoeffding inequality to the number of ones in the Bloom filter, we get

\[
N_{BF} = \left| N_{BF} - N_1 \right| \leq \sqrt{\frac{nk \ln 2}{2 \rho}} \text{ extra bits in random OT.}
\]

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, \( N_0 \geq \frac{1}{\rho - 1} \sqrt{\frac{N_{BF} \ln 2}{\rho}} \). Consequently, \( N_1 = \frac{1}{\min(n, 1-\alpha)} \sqrt{\frac{N_{BF} \ln 2}{\rho}} \), which is (5).

2. \( \Pr[p_{false} \geq 2^{-\sigma}] \leq 2^{-\lambda} \) with either honest or malicious receiver in \( \Pi_{AppROT} \).

Denote by \( N_{rest} \) 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 \( N_{rest} \) bits set to 1 and k hashfunctions, that we denote as \( p_{False} \). The upper bound for it (in our conditions of the experiment) is obtained in Appendix D from [17]:

\[
p_{False} \leq \frac{k}{2N_{BF}} \left( 1 + \frac{k-1}{2N_{BF}} \left( 1 - \frac{1}{\rho} \right) \right).
\]

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

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

We require that \( \Pr[p_{False} \geq 2^{-\sigma}] \leq 2^{-\lambda} \). Using (7), we can re-write the expression in parentheses as

\[
N_{rest} \geq N_{BF}2^{-\sigma} \left[ 1 + \frac{k(k-1)}{2N_{BF}} \left( 1 - \frac{1}{\alpha} \right) \right]^{-k/2}.
\]

Suppose that the malicious party chooses more ones than required, and the proportion of ones now is \( \alpha > \hat{\alpha} \), and the sender observes \( \hat{\alpha} \) proportion of ones among \( N_{cc} \) opened bits. Latter is the unbiased estimator of \( \hat{\alpha} \) with

\[
\Pr[\hat{\alpha}N_{cc} \geq (\hat{\alpha} + \delta)N_{cc}] \leq e^{-2\delta^2N_{cc}} \leq 2^{-\lambda}.
\]

Hence \( \hat{\alpha} \geq \sqrt{\frac{\ln 2}{2N_{cc}}} \).
Then, using (7), the number of ones in the remained set with the overwhelming probability is

\[ N_{\text{rest}} < (\hat{\alpha} + \tilde{\sigma})(N + N_{\Delta}) \leq (\hat{\alpha} + \sqrt{\frac{\ln 2}{2N_{\Delta}}} + N_{\Delta}), \] 

\[ N_{BF} 2^{-\frac{\alpha}{2}} \left[ 1 + \frac{k(k-1)}{2N_{BF}} \left( \frac{1}{\alpha} - 1 \right) \right]^{\frac{1}{2}}. \]

As the number of observed ones is \( \hat{\alpha} N_{cc} \), we have the inequality for the maximal number of opened ones as \( N_{\text{maxones}} \leq \hat{\alpha} N_{cc} \leq \frac{N_{\Delta} N_{cc}}{N_{BF} + N_{cc}} \leq \frac{k(k-1)}{2N_{BF}} \left( \frac{1}{\alpha} - 1 \right). \)

Hence, \( N_{\text{maxones}} = \left[ \alpha N_{cc} + \frac{1}{2N_{BF}} \ln 2 \right] \). If \( N_{\text{maxones}} > N_{\text{maxones}} > 0 \), then the honest receiver passes the cut-and-choose check.

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

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

(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.

Due 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 \( N_{\text{maxones}} - N_{\text{maxones}} > 0 \) by this particular \( N_{cc} \). (3) follows.

We next show that there exist suitable parameters \( \sigma, \lambda \) for any \( n \).

**Theorem C.3.** *For any choice of positive \( n, \sigma, \lambda \) there exist positive \( k, N_{BF}, N_{cc}, N_{OT}, N_{\text{maxones}}, N_{\text{OT}}^1 \) such that (2)-(5) hold.*

**Proof.** Considering the asymptotic when \( N_{BF} \to \infty \), we compute the following limits:

\[ \lim_{N_{BF} \to \infty} N_{BF} \left( 1 - e^{-\frac{N_{BF}}{2}} \right) = \lim_{N_{BF} \to \infty} N_{BF} \left( 1 - \left( 1 + \frac{1}{N_{BF}} \right)^{-nk} \right) = \lim_{N_{BF} \to \infty} N_{BF} \left( 1 - \left( 1 + \frac{nk}{N_{BF}} \right)^{-1} \right) = \lim_{N_{BF} \to \infty} \frac{N_{BF}^{nk}}{N_{BF} + nk} = nk; \]

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

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 \( \min(\alpha, 1-\alpha) = \alpha, \alpha = \sqrt{\frac{\ln 2}{2}}, k = \frac{N_{BF}}{2\sigma/k} \left[ 1 + \frac{k(k-1)}{2N_{BF}} \left( \frac{1}{\alpha} - 1 \right) \right]^{-\frac{1}{2}} - \alpha N - \frac{\lambda \ln 2}{\alpha}, c = \sqrt{2\ln 2} N \). For the existence of two positive roots, it is sufficient to have \( b > 0, D = b^2 - 4ac > 0 \). Let fix some value of \( k > \sigma \) and prove that there exists some \( N_{BF} \) such that those conditions hold. From (9),

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

The asymptotic of \( b \) is linear in \( N_{BF} \). The value in outer square parentheses is constant by fixed \( n, \lambda, k, \sigma \) and \( k > \sigma \), then \( 2^{\sigma} \left[ 1 + \frac{N_{BF} k(k-1)}{2nk \ln 2 \ln 2} \right]^{-\frac{1}{2}} \)

tends to 1 when \( k \) grows, while \( \frac{\ln 2}{n \ln 2 + 2nk \ln 2} \) tends to 0. Thus, for sufficiently large \( k = k_1 \), \( \lim_{N_{BF} \to \infty} b = \lim_{N_{BF} \to \infty} C_1 N_{BF} \), where \( C_1 > 0 \). Hence there exists a sufficiently large \( N_{BF} \) such that \( b > 0 \).
Now consider the asymptotic of the discriminant when \( k \geq k_1 \):

\[
\lim_{N_{BF} \to \infty} \left( b^2 - 4ac \right) = \lim_{N_{BF} \to \infty} \left( (C_1 N_{BF})^2 - 4N \lambda \ln 2 \right) = \lim_{N_{BF} \to \infty} \left( (C_1 N_{BF})^2 - 4N \lambda \ln 2 - 4 \sqrt{2n} k (\lambda \ln 2)^{3/2} \right) = \lim_{N_{BF} \to \infty} (C_1 N_{BF})^2.
\]

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). \( \square \)

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 \( \Pi_{AppROT} \)](image)

**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 \( n = 2^6 \), 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 \( n = 2^6 \) to 46.7\% for \( n = 2^{20} \)) and increases the size of the cut-and-choose set (from 287\% for \( n = 2^6 \) to 5793\% for \( n = 2^{20} \)), which results in a large increase in the overall cost.

**Table 6:** Optimal (in \( N_{BF} \)) \( \Pi_{AppROT} \) parameters for set size \( n \), statistical security \( \lambda = 40 \), and computational security \( \sigma = 128 \).
\section{False-Positive Probability of a Bloom Filter}

In this section we compute an upper bound for \( P_{\text{False}} \) 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:

\[
P_{\text{False}} = \left( \frac{E[X]}{m} \right)^k + \frac{\sigma_X^2}{2} \frac{k(k-1)}{m^2} \left( \frac{E[X]}{m} \right)^{k-2},
\]

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 \( \sigma_X^2 \) as

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

From (10) and (11), we get

\[
P_{\text{False}} < p^k + \frac{p(1-p) \cdot k(k-1)}{2m} p^{k-2} = p^k \left[ 1 + \frac{k(k-1)}{2m} \left( 1 - \frac{1}{p} \right) \right].
\]

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 \( \Pi_{\text{AppROT}} \), as a sender and as a receiver, while every other \( P_i \) performs only two instances, one as a sender and one as a receiver.

The communication of \( \Pi_{\text{Offline}} \) is dominated by OTs with communication complexity \( O(n^2) \) – in order to compute \( N_{\text{OT}} \) OTs of length \( \sigma = 128 \) with statistical security \( \lambda = 40 \), the OT-extension of Keller, Orsini, and Scholl [23] requires sending \( \sigma N_{\text{OT}} \) bits + 10KB.

The communication of \( \Pi_{\text{Online}} \) consists mainly of sending/receiving the permutations of the unopened OTs; again, here \( P_0 \) performs \( 2t \) instances of \( \Pi_{\text{Online}} \), while every other \( P_i \) only \( 2 \) instances of \( \Pi_{\text{AppROT}} \). Apart from \( \Pi_{\text{AppROT}} \), 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 \( N_{\text{cc}} \ll N_{\text{BF}} \).

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

<table>
<thead>
<tr>
<th>Table 7: Number of sent and received bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>( P_0 )</td>
</tr>
<tr>
<td><strong>Offline-phase</strong></td>
</tr>
<tr>
<td>OT-extension</td>
</tr>
<tr>
<td>Cut-and-choose challenge</td>
</tr>
<tr>
<td>Cut-and-choose response</td>
</tr>
<tr>
<td><strong>Online-phase</strong></td>
</tr>
<tr>
<td>Permutation</td>
</tr>
<tr>
<td>Sending/receiving GBFs</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table 8: Number of performed operations</th>
</tr>
</thead>
<tbody>
<tr>
<td>( P_0 )</td>
</tr>
<tr>
<td><strong>Offline-phase</strong></td>
</tr>
<tr>
<td>Hashes for OT-extension</td>
</tr>
<tr>
<td>PRG of ( \sigma )-bit strings to compute secret shares</td>
</tr>
<tr>
<td><strong>Online-phase</strong></td>
</tr>
<tr>
<td>Hashes for the Bloom filter</td>
</tr>
<tr>
<td>Performed permutations</td>
</tr>
<tr>
<td>XORs of ( \sigma )-bit strings for codewords and GBFs</td>
</tr>
<tr>
<td>PRG of ( \sigma )-bit strings for rerandomization of GBF</td>
</tr>
<tr>
<td>XORs of ( \sigma )-bit strings for the intersection</td>
</tr>
</tbody>
</table>

| Table 9: Comparison of communication complexity; \( t \) is the number of parties, \( n \) is the size of the input set, \( \lambda \) and \( \sigma \) are the statistical and computational security parameters respectively. |

| Protocol | Security | Communication complexity |
|---------------------------|---------------------------|
| \( \text{offline} \) | \( \text{online} \) | \( \text{offline} \) | \( \text{online} \) |
| RKP18 [21] | semi-honest | - | \( O(t^3n^2 \lambda) \) | - | \( O(n^2 \lambda) \) |
| KMPRT17 [28] | semi-honest | - | \( O(t^3 \lambda) \) | - | \( O(n^2 \lambda) \) |
| RKP18 [21] | augmented semi-honest | \( O(t^3 \lambda) \) | \( O(n^2 \lambda) \) | \( O(t^2 \lambda) \) | \( O(n^2 \lambda) \) |
| KMPRT17 [28] | augmented semi-honest | \( O(t^3 \lambda) \) | \( O(n^2 \lambda) \) | \( O(t^2 \lambda) \) | \( O(n^2 \lambda) \) |
| KS05 [26] | malicious | \( O(t^3 \lambda) \) | \( O(t^3 \lambda) \) | \( O(t^2 \lambda) + t n_\sigma \log(n) \) | \( O(t^3 \lambda) \) | \( O(t^2 \lambda) + t n_\sigma \log(n) \) |
| HV17 [19] | malicious | \( O(t^3 \lambda) \) | \( O(t^2 \lambda) + t n_\sigma \log(n) \) | \( O(t^3 \lambda) \) | \( O(t^2 \lambda) + t n_\sigma \log(n) \) |
| GN19 [15] | malicious | \( O(t^3 \lambda) \) | \( O(t^2 \lambda) + t n_\sigma \log(n) \) | \( O(t^3 \lambda) \) | \( O(t^2 \lambda) + t n_\sigma \log(n) \) |
| PSimple | malicious | \( O(t^3 \lambda) \) | \( O(t^2 \lambda) + t n_\sigma \log(n) \) | \( O(t^3 \lambda) \) | \( O(t^2 \lambda) + t n_\sigma \log(n) \) |
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 $\Pi_{\text{PSimple}}$ 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 $\Pi_{\text{PSimple}}$ 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 through 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 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 $\Pi_{\text{AppRO}}$ 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 $\Pi_{\text{PSimple}}$ gRO-secure (and in fact, even UC-secure at all, as $\Pi_{\text{PSimple}}$ uses the same implementation of the the RO for all instances of $\Pi_{\text{AppRO}}$, which in our implementation indeed make calls to the RO).

F.1 Consistency of PSI Simple

Consistency follows from next: consider $\text{GBF}^*$ that $P_0$ learns on the step 8.

\[
\text{GBF}^* = \bigoplus_{i \in [r]} \text{GBF}^* \bigoplus \text{GBF}^0 = \bigoplus_{i \in [r]} \bigoplus_{x \in \{0,1\}} \text{GBF}^i \bigoplus_{i \in [r], i \not\in l} \left( S^i \oplus S^i \right) \bigoplus_{i \in [r]} \left( M^i \oplus \tilde{M}^i \right) = \bigoplus_{i \in [r]} \left( \text{GBF}^i \bigoplus_{i \in [r]} \left( M^i \oplus \tilde{M}^i \right) \right).
\]

$\text{GBF}^i$ is the re-randomised $M^i \oplus \tilde{M}^i$, and the re-randomization performed by $P_0$ doesn’t affect codewords of its items, therefore for any $x_{ij} \in X_i$ the codeword $y_{ij} = y_{i \in h, (x)}$ $\text{GBF}^i[s] = y_{i \in h, (x)} \left( M^i[s] \oplus \tilde{M}^i[s] \right)$. Hence, for every item $x \in \cap_{i \in [r]} X_i$ with codewords $y_{is}$, we have

\[
\bigoplus_{s \in h, (x)} \left( y_{i \in h, (x)} \bigoplus_{s \in h, (x)} \left( \text{GBF}^i[s] \bigoplus_{s \in h, (x)} \left( \text{GBF}^i[s] \oplus \tilde{M}^i[s] \right) \right) \right) = \bigoplus_{i \in [r]} (y_i \oplus y_i) = 0.
\]

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 $\xi$, 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 $l$ of $B$, we denote by $J = \xi(l)$ 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 $\{X(k, a)\}_{k \in \mathbb{N}, a \in [0,1]}$ and $\{Y(k, a)\}_{k \in \mathbb{N}, a \in [0,1]}$ are statistically close, denoted $\{X(k, a)\} \approx \{Y(k, a)\}$, if for every non-uniform distinguisher $D$ there exists a negligible function $\mu$ such that for all $a$ and $k$, $\Pr[D(X(k, a)) = 1] - \Pr[D(Y(k, a)) = 1] \leq \mu(k)$.

**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 $t^k$ 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 $t^k$ 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 $\Pi$ (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 (SIDR$_{i}$, $x$) the string $x$ as its intended input in a protocol $\Pi$ (identified by session id SID$_{i}$). 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 (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 $\text{Real}_{\Pi_{1}}\mathcal{Z}(k)$ (instead of $\text{Real}_{\Pi_{1}}\mathcal{Z}_{-\mathcal{A}}(k)$), omitting the specification of $\mathcal{A}$. Corruptions are modeled by having an adversary send a special ‘corrupt’ message to the newly corrupted party. By $\text{Ideal}_{\Pi_{1}}\mathcal{Z},\mathcal{S}(k)$ 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 $t^k$. By $\text{Ideal}_{\Pi_{1}}\mathcal{Z},\mathcal{S}$ we denote the ensemble $\{\text{Ideal}_{\Pi_{1}}\mathcal{Z}(t^k)\}_{k\in\mathbb{N}}$ by $\text{Real}_{\Pi_{1}}\mathcal{Z}$ – the ensemble $\{\text{Real}_{\Pi_{1}}\mathcal{Z}(t^k)\}_{k\in\mathbb{N}}$.

**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 $\Pi$ 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 $\mathcal{A}$),

$$\text{Ideal}_{\Pi_{1}}\mathcal{Z},\mathcal{S} \approx_{c} \text{Real}_{\Pi_{1}}\mathcal{Z}.$$  

Similarly, for statistical and perfect security, the indistinguishability notion above is $\approx_{s}$ and $\approx_{p}$, respectively.13 14

**Definition F.5.** A protocol $\Pi$ is said to realize $\mathcal{F}$ 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 $\bot$ after receiving the prescribed output of the corrupted parties, and before sending outputs to honest parties. If $\bot$ was sent, the functionality sends $\bot$ 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 $\Pi_{\text{AppROT}}$ - 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 $\text{F}_{\text{OT}}^{\sigma,N}$-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 $\bot$- and "continue"-replies, which are omitted in the main text. The oblivious transfer functionality we use in our security setting is $\text{F}_{\text{OT}}^{\sigma,N}$ (Fig. 7) with $N = N_{\text{OT}}$ parallel instances of 1-out-of-2 OT.

---

12Note that a given ITM may participate in several sessions concurrently. To distinguish which message belongs to which protocol, every message delivered in $\Pi$ is labeled by SID$_{i}$.

13Note 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.

14The 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 $\Pi_{AppROT}$ in $\mathcal{F}_{OT}^{\sigma,N}$-hybrid model

**Parties:** Sender, Receiver.

**Parameters:** $\sigma$ - length of the OT strings (computational security parameter); $\lambda$ - statistical security parameter; $N = N_{BG}$ is the number of OTs required; $N_{OT} > N$ is the number of OTs to generate; $N_{OT}^1, N_{Ccc}, N_{maxones}$ are parameters of cut-and-choose described in Section 4.4.

**Inputs:** $B$ is the choice vector of the receiver ($B[i] \in \{0,1\}, i \in [N_{BG}]$) of size $N = N_{BG}$ consisting $K$’s.

1. [1-out-of-2 OTs] The sender and the receiver call $\mathcal{F}_{OT}^{\sigma,N}$ performing $N_{OT}$ OTs in parallel. The receiver chooses requests $c_1, ..., c_{N_{OT}}$ with $N_{OT}^1$ ones among them, and $N_{OT} - N_{OT}^1$ zeroes (randomly permuted). The senders chooses uniformly at random $M_0 = \{m_{01}, ..., m_{N_{OT}1}\}$. $M_1 = \{m_{11}, ..., m_{N_{OT}1}\}$ $m_{ij}$ are of length $\sigma$.

2. [cut-and-choose challenge] The sender chooses set $C \subseteq [N_{OT}]$ of size $N_{Ccc}$ uniformly random and sends $C$ to the receiver.

3. [cut-and-choose response] The receiver checks if $|C| = N_{Ccc}$; if $|C| > N_{Ccc}$, then truncates it, if $|C| < N_{Ccc}$, 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 $r' = \bigoplus_{j \in R} m_{ij}$. The sender replies with $\perp$ if $|C| - |R| > N_{maxones}$ or if $r' = \bigoplus_{j \in R} m_{ij}$, and with “continue” otherwise.

4. [permute unopened OTs] The receiver chooses random injective function $\pi : [N] \rightarrow ([N_{OT}] \setminus C)$ such that $B[j] = c_{\pi(j)}$, and sends $\pi$ to Sender.

The receiver permutes its random values $m_{ij}$, according the $\pi$, and the sender permutes $m_{ij}$ according to $\pi$. If $\pi$ is formed incorrectly (not from the domain $[N_{OT}] \setminus C$ or not to $[N]$), then use a default value (N consecutive values from $[N_{OT}] \setminus C$).

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

**Figure 11:** Protocol $\Pi_{AppROT}$ in $\mathcal{F}_{OT}^{\sigma,N}$-hybrid model

**Lemma F.6.** The protocol $\Pi_{AppROT}$ realizes the functionality $F_{AppROT}$ with statistical UC-security with abort in the presence against static (unbounded) malicious adversaries in the $\mathcal{F}_{OT}^{\sigma,N}$-hybrid model, and $K'$ is an upper bound for number of 1’s in Bloom filter of length $N = N_{BG}$ 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 $Z$. 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 $S$ 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 $Z$ 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 $F_{AppROT}$, which requires participation of both parties (but the honest one is waiting).

**Security in face of a corrupted receiver.** The simulator $S$, once activated by $Z$, emulates the protocol towards $Z$.

1. In the 1st step of the protocol, when emulating $\mathcal{F}_{OT}^{\sigma,N}$ and obtaining the input $c_1, c_2, ..., c_{N_{OT}}$ from $Z$, $S$ randomly chooses a set $C \subseteq [N_{OT}]$, where $|C| = N_{Ccc}$, and computes $K'' = \sum_{i \in [N_{OT}] \setminus C} c_i$.

2. If $K'' \leq K'$, $S$ passes $K''$ as an input of the receiver to $\mathcal{F}_{AppROT}$ and receives $M' = \{m'_1, m'_2, ..., m'_{K''}\}$. For any $j \in [N_{OT}] \setminus C$ with $c_j = 1$ it computes $\psi(j) = \sum_{i \in [N_{OT}] \setminus C, i \leq j} c_i$ and the function $\psi : [K''] \rightarrow [N_{OT}] \setminus C$, as the inverse of $\psi$. I.e., $\psi(i) = j$, whenever $\psi(j) = i$ for $i \in [K'']$, $j \in [N_{OT}] \setminus C$ such that $c_j = 1$. For any $j \in [N_{OT}]$ it constructs $m_{ij} = m'_j$, if $j \in [N_{OT}] \setminus C$ and $c_j = 1$, or $m_{ij} = \perp$, otherwise. The simulator generates $\{m_{ij}\}_{j \in [N_{OT}]}$ to $Z$ as the output of $\mathcal{F}_{OT}^{\sigma,N}$ in 1st step, and $C$ as the message of 2nd step of the protocol.

3. If $K'' > K'$, the simulator passes to $Z$ the vector $\{m_{ij}\}_{j \in N_{OT}}$ of uniformly random $\sigma$-bit strings as the output of $\mathcal{F}_{OT}^{\sigma,N}$ 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 $Z$ as sender’s reply, appends $\{m_{ij}\}_{j \in N_{OT}}$, and halts.

4. $S$ waits for a message from the receiver in 3rd step of the protocol. Upon receiving the response $(r,r^\ast)$ it sends back Continue and waits for the permutation.

Upon receiving permutation $\pi$ from $Z$ in 4th step, $S$ checks that $\pi : [N_{OT}] \setminus C \rightarrow [N]$. If not, then uses a default value for $\pi (N$ consecutive values from $[N_{OT}] \setminus C$).
If yes, then computes \( I' = \left( \frac{i_1}{\pi(\phi(i_1))}, \frac{i_2}{\pi(\phi(i_2))}, \ldots, \frac{i_k}{\pi(\phi(i_k))} \right) \) where \( i_k \in [K''] \) such that \( \exists \pi(\phi(i_k)) \) 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 \( 1^\sigma, 1^k, k, N \) as input.

Let \( x \) denote the input vector to all parties given at the outset by \( Z \) to all parties. In step 1, \( Z \) asks \( \mathcal{S} \) to send \( F^{\sigma,N}_{OT} \) the set \( Q \) of the requests to \( F^{\sigma,N}_{OT} \) where \( Q' \subseteq Q \) is the set of 1-requests, and the rest are 0-requests. It receives a sequence \( \{m_{j}\}_{j \in [N_{OT}]} \) 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 \( Z \) 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 \( K'' = [Q' \cap ([N_{OT}] \setminus C)] \).

The emulated \( m_{j} \)'s at locations \( j \in Q' \cap ([N_{OT}] \setminus C) \) are taken from the functionality’s step-1 output to receiver if \( K'' \leq K' \), and random independent values picked by \( \mathcal{S} \) otherwise. In both cases, the other values \( S \) sends emulating replies of \( F^{\sigma,N}_{OT} \) are random values independent of all others. Now, in the real world, the sender chooses \( C \), and either

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

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 \( Z \) 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 \( m_{j} \) 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 \( r \)). \( S \) also appends \( \bot \) 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 \( [N_{OT}] \setminus C \) 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 \( P_{\text{False}} \) computed for the Bloom filter of length \( N \) with \( K'' \)’s in it, is larger than \( 2^{-\sigma} \) except with negligible (in \( \lambda \)) probability, since (13) holds. Therefore, the ideal functionality sends \( \bot \) 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 \( 2^{-\sigma} + \text{neg}(\lambda) \).

\[
\text{Ideal}_{F, Z, S} \approx \text{Real}_{l, Z} \approx (D, \bot). \tag{14}
\]

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 \( Z \) sends \( R' \) (which differs from \( R \) induced by its \( F^{\sigma,N}_{OT} \)’s inputs) so that (13) is satisfied for \( C', R' \), or \( r^* \neq \bigoplus_{j \in R} m_{j} \), the sender outputs \( \bot \) and halts immediately. By construction of \( S \), it sends \( \bot \) in step 2 as receiver’s input, and replies with \( \bot \) to both parties. \( S \) again appends \( \bot \) as sender’s message to the simulated view. Thus, if (14) holds with 0-error (in particular, over the entire support of \( \text{Real}_{l, Z} \), the sender’s output is \( \bot \)). Otherwise, in the real world, \( Z \) proceeds by picking \( \pi \) (based on its entire view so far) and sends it to the sender, who permutes the values \( m_{j} \) it picked previously according to \( \pi \). In particular, the \( m_{j} \)'s for the \( K' \)'s for which \( \pi(j) \in Q' \) are also output to the sender at positions \( \pi(j) \), and all other \( m_{j} \)'s’s output to sender are random values, independent of receiver’s view so far (as it never received these values). In the ideal world, \( S \) receives (the same) \( \pi \) from \( Z \), and sets \( I' \) sent to the ideal functionality in step 2 in a way that preserves the sender’s output at positions \( \pi(j) \) for \( j \) with \( \pi(j) \in Q' \), equal the \( m_{j} \) 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 \( \text{Ideal}_{F, Z, S} \approx \text{Real}_{l, Z} \) with \( \delta \)-error.

Overall, we get a statistical distance at most \( \text{neg}(\lambda) + 2^{-\sigma} \) between \( \text{Ideal}_{F, Z, S} \) and \( \text{Real}_{l, Z} \).

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

In the first step of the protocol, \( S \) obtains the inputs of the sender \( M_{\emptyset} = \{m_{j}\}_{j \in [N_{OT}]} \). \( M_{1} = \{m_{j|1}\}_{j \in [N_{OT}]} \) from \( Z \).

In the 2nd step of the protocol, \( S \) waits for the message \( C \) from \( Z \). If \( |C| > N_{CC} \), then truncates it, if \( |C| < N_{CC} \), then adds indices by default (1, 2, ...). It samples the number \( |C| \) hypergeometrically \( HG(N_{OT}, N_{CC}^{1}) \) (if 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 \( S \) computes \( r^* = \bigoplus_{j \in R} m_{j} \). It gives \( R \) and \( r^* \) to \( Z \) as the message in 3rd step of the protocol.

Then it samples the uniformly random \( N \)-permutation \( \pi : [N_{OT}] \setminus C \rightarrow [N] \) and gives it to \( Z \) as a message in 4th step of the protocol, and gives \( M = \pi(M_{1}) \) as an input of corrupt sender to \( F_{\text{AppROT}} \).

**Proof of security in face of a corrupted receiver.** Consider an environment \( Z \) running on some fixed public parameters \( 1^\sigma, 1^k, k, N \). Let \( x \) denote the input vector to all parties given at the outset by \( Z \) to all parties, as in the previous case. The environment \( Z \) (via \( S \)) sends to \( F^{\sigma,N}_{OT} \) two based on \( x \) sets \( M_{\emptyset} = \{m_{j}\}_{j \in [N_{OT}]} \) and \( M_{1} = \{m_{j|1}\}_{j \in [N_{OT}]} \), whose distributions are identical in both real and ideal worlds by the construction of the simulator.
Then \( Z \) sends the set \( C \subseteq \{N_{OT}\} \) such that \( |C| = N_{C} \) based on \( x \) and receives \((R,r^*)\) in response. \( R \) is distributed identical in the real and ideal world by \( S \) 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 \( Z \) responses with either \( \perp \) or Continue, which it chooses basing on its 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 \( Z \) sends Continue, it receives the \( N \)-permutation \( \pi : \{N_{OT}\} \setminus C \rightarrow \{N\} \). This permutation in ideal world is computed randomly and in the real directly from the receiver with the same input and randomness, 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. \( \Box \)

**Consistency of \( \Pi_{AppROTr} \)** Now we show that for the honest sender and honest receiver, \( \Pi_{AppROTr} \) protocol realizes \( F_{AppROTr} \) ideal functionality. The honest receiver sends to the protocol the choice bits \( c_{1},...,c_{N_{OT}} \), recovers the subset \( C \subseteq \{N_{OT}\} \) received from the sender and sends the permutation \( \pi : \{N_{OT}\} \setminus C \rightarrow \{N\} \) such that \( \pi(c_{1},...,c_{N_{OT}}) = B \).

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 \( \pi \), as there is enough 1’s among the choice bits remaining after cut-and-choose.

The sender inputs to \( F_{OT}^{\pi_{N_{OT}}} \) sequences \((m_{i1})_{i \in \{N_{OT}\}} \) and \((m_{i1})_{i \in \{N_{OT}\}} \). The receiver has \((m_{j1})_{j \in \{N_{OT}\}} \) as an output. Applying the permutation \( \pi \) to \((m_{i1})_{i \in \{N_{OT}\}} \) and \((m_{j1})_{j \in \{N_{OT}\}} \) accordingly, the sender gets \( M = \left\{ m_{\pi(i1):i \in \{N_{OT}\}} \right\} \), and the receiver gets \( M_{e} = \left\{ \begin{array}{ll} m_{\pi(i2)} & i_{1} = \min(\{N_{OT}\} \setminus C), i_{2} = \max(\{N_{OT}\} \setminus C). \end{array} \right\} \). Thus, the elements of \( M \) and \( M_{e} \) 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 \( F_{AppROTr} \), the receiver has the values from the sender’s output set at indices from \( I \), as described by the functionality \( F_{AppROTr} \).

### F.4 PSI protocol in the hybrid model

Figure 12 describes the \( PSI_{simple} \) protocol in \( F_{gRO}, F_{AppROTr} \)-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 \( F_{AppROTr} \) is not separated into offline and online phases, so that the \( PSI_{simple} \) itself. Furthermore, we need to add a padding after \( F_{AppROTr} \), 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 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 \(|Q|p_{false} < |Q|2^{-o} \). To keep it negligible, \(|Q| = \text{poly}(|\sigma|) \). In the case of polynomially-bounded (in \( \sigma \)) domain \( D \), this requirement is fulfilled automatically, otherwise (for example, it the typical case of an exponential-size domain) we require a computationally bounded (in \( \sigma \)) adversary in Theorem 3.1.

**Lemma 7**. The protocol \( \Pi_{SI_{simple}} \) securely realizes the functionality \( F_{MPSI} \) with statistical UC-security with abort in presence of static malicious adversary corrupting up to \( t \) parties in the \( F_{AppROTr}, F_{gRO} \)-hybrid model, which makes a polynomially bounded number of queries to \( F_{gRO} \), 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, \( p_{false} \leq 2^{-o} \), and for the Bloom filter with \( n' + 1 \) and more elements, \( p_{false} > 2^{-o} \). It means, that the malicious receiver in \( F_{AppROTr} \) receives \( \perp \) from the first step of \( F_{AppROTr} \) 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 \( I \subseteq \{P_{1},...,P_{t}\} \) are corrupt.**

**Simulator description.** The simulator \( S \), once activated by \( Z \), emulates \( F_{AppROTr} \) towards \( Z \). We stress, that all the corrupt parties are emulated asynchronously, according to the message scheduling decided by \( Z \) - 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), \( S \) sends to \( Z \) uniformly random shares \( S^{U} \), as honest \( P_{i} \)’s would do according the protocol, to any corrupt party \( P_{i} \in I \), and learns \( S^{E}_i \) from \( Z \).

In step 2, \( S \) make queries \((q,PID,SID)\) to the global Random Oracle (to compute Bloom filter’s hash-indices) on behalf of corrupt parties \( P_{i} \in I \) as requested by \( Z \) and writes them to sets \( Q_{i} = \{q_{ji} \mid j \in \{N_{OT}\}\} \). Here index \( i \) corresponds to PID of corrupt \( P_{i} \), \( n_{i} \geq 0 \) is polynomially bounded, as \( Z \) is. Denote \( Q = \cup_{j \in [N]} Q_{j} \) - the joint query set of corrupt parties. The environment might continue making such a queries up to the end of the protocol, and \( S \) 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 $\Pi_{PS\text{Simple}}$ in the $\mathcal{F}_{\text{AppROT}}$-hybrid model**

**Parameters:**
- $n$ - the maximal size of the input set of the party;
- $\sigma$ - computational security parameter;
- $\lambda$ - statistical security parameter;
- $N_{BGF}$ - 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. \textbf{[R0] random shares} Each $P_i$, $i \in \{t\}$, sends $S^H = (s_1^H, ..., s_{n_i}^H)$ to any $P_j, j \in \{t\} \setminus \{i\}$, where $s^H = R^H \cdot (0,1)^{\sigma}$, $r \in [N_{BGF}]$.
2. \textbf{[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 $X_j$ with $(n - n_i)$ random dummy items.

3. \textbf{[R1] symmetric approximate ROTs} Parties perform in parallel:
   - (a) Using $BF_i$’s 1’s indices set $J$ as input, $P_0$ calls $[J]$-out-of-$N_{BGF}$ $\mathcal{F}_{\text{AppROT}}$ 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 $M_i [j] = 0$ for $j \notin [N_{BGF}] \setminus J_i$.
   - (b) Using $BF_i$ 1’s indices set $J_i$ as input, each $P_i (i \in \{t\})$ calls $[J]$-out-of-$N_{BGF}$ $\mathcal{F}_{\text{AppROT}}$ as the receiver with $P_0$. As a result, $P_i$ learns $M_i [j]$ for each $j \in J_i$ and $P_0$ receives $M_i [j] = 0$ for $j \notin [N_{BGF}] \setminus J_i$.

4. \textbf{[R2] compute and re-randomize GBFs} If $P_0$ did not receive $\bot$ from $\mathcal{F}_{\text{AppROT}}$, it computes $GBF^0 = \bigoplus_{t \in [t]} (M^i \oplus M^j)$. If $P_0$ did not receive $\bot$ from $\mathcal{F}_{\text{AppROT}}$, it computes $GBF^0 = \bigoplus_{t \in [t]} GBF^i \oplus GBF^j$.
   - (b) $P_0$ re-randomizes $GBF^0$ from $X_i$ and codewords $y_{ij} (j \in [n_i])$ according to algorithm BuildGBF from B.1.

5. \textbf{[R2] cumulative GBFs of $P_i$’s} If $P_i (i \in \{t\})$ did not receive $\bot$ from $\mathcal{F}_{\text{AppROT}}$, it computes and sends to $P_0$ the cumulative garbled Bloom filter:
   \[
   GBF^i = GBF^i \bigoplus_{t \in [t]} \left[ S^H \oplus S^H \right].
   \]
   Else it sends $\bot$.

6. \textbf{[R2] cumulative GBF of $P_0$} If $P_0$ did not receive $\bot$ from $\mathcal{F}_{\text{AppROT}}$ or from $P_i$ in the previous step, it computes $GBF^0 = \bigoplus_{t \in [t]} GBF^i \oplus GBF^j$.

7. \textbf{[R2] output} If $P_0$ did not receive $\bot$ from $\mathcal{F}_{\text{AppROT}}$ or from $P_i$ in the 6th step, it outputs $x_{ij}$ as a member of the intersection, if $\bigoplus_{r \in h_i (q_j)} GBF^i [r] = 0, j \in [n_i]$.
   Else it outputs $\bot$.

---

**Figure 12:** The PS\text{Simple} multiparty protocol in the $\mathcal{F}_{\text{AppROT}}$-hybrid model

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

- For $P_i$ acting as a corrupt sender, the simulator receives $M^i$ of length $N_{BGF}$ or $\bot$ from $P_i$.
- For any $P_i$ acting as a corrupt receiver, upon receiving $K''$ the simulator samples uniformly at random and gives $\tilde{M}^i$ of length $K''$, or gives $\bot$, then the simulator receives set of indices $J_i$ or $\bot$.

The simulator remembers each of the $M^i$’s and computes $\tilde{M}^i$’s (from $M^i$ and $J_i$ as $\mathcal{F}_{\text{AppROT}}$ would compute $M$ in the case of corrupt receiver) for all $P_i \in I$, if all the calls to emulate $\mathcal{F}_{\text{AppROT}}$ are completed successfully (without $\bot$’s).

At the 5th step, once all round-1 and round-2 executions have completed, $S$ observes $GBF^i$’s or $\bot$’s sent by $Z$ on behalf of corrupt $P_i \in I$.

Besides, it asks $\mathcal{F}_{\text{RO}}$ 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 $A$ is).

- If there were no $\bot$’s as an outputs of the simulated $\mathcal{F}_{\text{AppROTS}}$ or as the messages of the 5th step, $S$ computes the sum $GBF^0 = \bigoplus_{t \in [t]} GBF^j$. Now $S$ can subtract all the secret shares sent and received to corrupt parties on behalf of honest and vice versa:
   \[
   GBF^0 = GBF^0 \bigoplus_{P_i \in \mathcal{P}_i \setminus (I \cup P_0)} \left( S^H \oplus S^H \right).
   \]

$GBF^0$ is the effective value of $\bigoplus_{P_i \in [t]} GBF^i$. Now the simulator extracts the effective input of corrupt parties as
\[
\hat{X}_I = \left\{ q \in Q \bigg| \bigoplus_{r \in h_i (q)} GBF^i [r] = \bigoplus_{P_i \in \mathcal{P}_i \setminus (I \cup P_0)} \left( M^i [r] \oplus \tilde{M}^i [r] \right) \right\},
\]

sends it to the ideal functionality, and receives either $\hat{X}$ or $\bot$ as the output of $\mathcal{F}_{\text{PSI}}$.

- Else, the simulator sends the effective input of the adversary $\bot$ to the ideal functionality and receives $\bot$ as its output.
Simulator Analysis. Consider an environment $\mathcal{Z}$ running on some fixed public parameters $1^n, 1^\lambda, t, n, k, N_{BF}$. 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 $\mathcal{X} = \{X_i\}_{i \in P \setminus I}$ inputs of honest parties. We prove indistinguishability by induction on the message graph of $\mathcal{Z}$ - sent messages to the various parties, and to $\mathcal{F}_{\text{AppROT}}$ 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 $\mathcal{P}_0$ is honest, we have to also prove the indistinguishability of the joint view of $\mathcal{Z}$ with the output of the honest $\mathcal{P}_0$ 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.

Let us compare the output distribution of $\mathcal{P}_0$. In the ideal world, $H = X_i \cap \{\exists i \in P \setminus I \mid X_i\}$ is the output of $\mathcal{P}_0$, 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 $\mathcal{P}_0$, 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 $X_i$ that $\mathcal{P}_0$ 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 $h_j$’s, and thus does not affect the condition in step 7). Now, malicious parties may have chosen 1-items in their $\mathcal{F}_{\text{AppROT}}$ executions, at locations outside of the query set $Q$. However, then they either query too many $1$’s in that $\mathcal{F}_{\text{AppROT}}$ execution, in which case, in the ideal world as well, $\mathcal{S}$ notices it, and sends $\perp$ on behalf of $\mathcal{P}_1 \in I$ 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 $H_j = \bigcap_{i \in P \setminus I} X_i$ with the probability at most $P_{\text{false}}$ (for instance, by using the received $M_t \oplus \tilde{M}_t$ at all 1-positions in GBF (without re-randomizing) for each given party. Since elements not known to all of them are complemented by $\mathcal{P}_0$ by a random string, the probability of adding an element is upper bounded by the probability of a fixed corrupted party $\mathcal{P}_i$ adding it, and the result follow. By union bound, adding an element in $H_j$ by $\mathcal{P}_i$ occurs with probability $\leq n \cdot P_{\text{false}}$. Assuming no extra elements not covered by $\mathcal{Q}$ were added by all malicious parties, for each $x \in X_0 \setminus H$, let $j$ denote some index for which some $\mathcal{P}_i \in I$ did not learn $M_t[j]$. The probability of $\mathcal{P}_0$ adding $x_0$ to the output due to passing verification in the step 7 $\leq 2^{-\sigma}$, which is the probability of $\mathcal{P}_i$ guessing $M_t[j]$ 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 $\mathcal{F}_{\text{AppROT}}$).

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

Finally, we assume 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 $\mathcal{P}_i$ are random independent values: either shares picked in step 1, or $\mathcal{F}_{\text{AppROT}}$ 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 $\mathcal{P}_0$, and some number of other parties (including zero) $I \subset \{P_1, ..., P_t\}$ 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 $\mathcal{F}_{\text{AppROT}}$ 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 $\mathcal{P}_i \in I$ uniformly random values $S^I$, as honest $\mathcal{P}_i$’s would do, and gets $S^I$’s to be delivered by corrupted parties $\mathcal{P}_i$ from $\mathcal{Z}$.

In step 2, $\mathcal{S}$ makes queries (q, PID, SID) to the global Random Oracle (to compute Bloom filter’s hash-indices) on behalf of corrupt parties $\mathcal{P}_i \in I \cup \mathcal{P}_0$ as requested by $\mathcal{Z}$ and writes them to sets $Q_i = \{q_{ij} \mid j \in [n_i]\}$ Here index $i$ corresponds to PID of corrupt $\mathcal{P}_i$, $n_i \geq n$ is polynomially bounded, as $\mathcal{Z}$ is. Denote $Q = \bigcup_{i \in P \setminus I \cup \mathcal{P}_0} Q_i$ - 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 $\mathcal{F}_{\text{AppROT}}$ functionality for $\mathcal{P}_0$ in its interaction with any honest $\mathcal{P}_i$. Simulating its outputs as follows.

- for $\mathcal{P}_0$ acting as a corrupt receiver in the simulated interaction with any honest $\mathcal{P}_i$, upon receiving $K''$ the simulator samples $M''$ uniformly random of length $K''$ as $\mathcal{F}_{\text{AppROT}}$ would, or $\perp$ if the Bloom filter with $K''$’s has more than $n'$ items. Then the simulator receives set of indices $j$ (and then can extract the effective Bloom filter $\tilde{\mathcal{F}}_0$ or $\perp$.

- for $\mathcal{P}_0$ acting as a corrupt sender in the simulated interaction with any honest $\mathcal{P}_i$, the simulator receives $M''$. 

28
We stress, that $S$ delivers messages asynchronously, by the order $Z$ sends messages to parties of $\text{AppRO}T$, 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 $\text{AppRO}T$’s completed successfully (without $\perp$’s, and in particular did complete at all), the simulator computes a set of Bloom filters $\mathcal{B}^0_i$ from the input’s set of indexes $j_i$ in any instance between corrupt receiver $P_0$ and honest sender $P_i$, and computes $M_i’ (P_i \in \mathcal{P} \setminus \{I \cup P_0\})$ as the functionality $\text{AppRO}T$ would compute the output for the honest sender (from $M_i’$ and $j_i$). If and once all $\text{AppRO}T$ executions are completed, $S$ asks $\text{AppRO}T$ 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 $\tilde{X}_i = \{ q \in Q \mid \forall s \in h_i(q), \forall j \notin I, \mathcal{B}^0_i[s] = 1 \}$ - 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, $S$ sends either it or $\perp$ to the ideal functionality as the input of each of the corrupted parties $\text{MPSI}$, 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 $\text{AppRO}T$’s ended with $\perp$, the simulator gives $\perp$, which is the effective input of the adversary, to $\text{MPSI}$ and receives $\perp$ from there.

To simulate a step-5 message by an honest party $P_i \notin I$ replied to $Z$, right after all the $\text{AppRO}T$’s of $P_i$ are completed, even if there are another running $\text{AppRO}T$’s for other parties:

- If $P_i$, received $\perp$ from $\text{AppRO}T$, $S$ sends $\perp$ to $Z$ as the message for $P_0$.
- If $P_i$’s $\text{AppRO}T$’s are completed successfully, and $P_i$ is not the last honest party whose $\text{AppRO}T$’s done, $S$ sends a uniformly random GBF$^{ix}$ to $Z$ as the message for $P_0$.
- If all the $\text{AppRO}T$’s are completed, and were at least one $\perp$, and $P_i$ is the last honest party whose $\text{AppRO}T$’s done (without $\perp$), $S$ sends a uniformly random GBF$^{ix}$ to $Z$ as the message for $P_0$.
- If all the $\text{AppRO}T$’s are completed successfully, and $P_i$ is the last honest party whose $\text{AppRO}T$’s done, then $S$ performs the following:
  - computes Bloom filters $\mathcal{B}^f$ for the set $\tilde{X}$ for all such that $P_j \notin (I \cup P_0)$;
  - computes GBF$^f = M_i’ \bigoplus M_i^f$ for all such that $P_j \notin (I \cup P_0)$;
  - computes codewords $y_j$s from GBF$^f$ as in the protocol, but only for items $x_{j\delta} \in \tilde{X}$ for all such that $P_j \notin (I \cup P_0)$;
  - computes re-randomized GBF$^f$ for items $x_{j\delta} \in \tilde{X}$ and their codewords $y_j$s as in B.1; note, that positions at indices $r$ such that $\mathcal{B}^f_r[r] = 0$ are entirely and uniformly random.
  - computes GBF$^{ix}_{\text{temp}}$ honestly as in step 5 of the protocol for all such that $P_j \notin (I \cup P_0)$, and
  - $\text{GBF}^{ix} = \bigoplus_P, P \notin (I \cup P_0) \text{GBF}^{ix}_{\text{temp}} \bigoplus_{j \notin (I \cup P_0)} \text{GBF}^{ix} (\text{here $\text{GBF}^{ix}$ are messages sent by $S$ on behalf of other honest parties in 5th step})$.
  - $S$ sends GBF$^{ix}$ to $Z$ as the message for $P_0$.

**Simulator Analysis.** Fix a certain $Z$, running on the public parameters $i^0, i^1, k, N$. $S$ proceeds as follows. We prove indistinguishability by induction on the message graph of $Z$ - sent messages to the various parties, and to $\text{AppRO}T$ throughout the execution, starting with the inputs provided, and messages received from honest parties (emulated by $S$ in the ideal world) are statistically indistinguishable. The induction is on the message number according to the order of message delivery by $Z$ in the real world (which $S$ follows). At the start, (the partial) view of $Z$ is clearly the same in both worlds (as $Z$ 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 $Z$ 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, $\{S^\delta\}_{P_i \notin (I \cup P_0)}$ sent from honest parties are random i.i.d strings (sampled by $S$ in ideal world), and $P_i \notin I$ have the same (uniform) distribution in both ideal and real worlds. In step 2, $\{M^\delta_1 / \perp\}_{P_i \notin (I \cup P_0)}$, $\{M^\delta_0 / \perp\}_{P_i \notin (I \cup P_0)}$; these are identically distributed to the values received by the corrupted sender in the real world protocol, by definition of $\text{AppRO}T$, and the fact that at any step of interaction of the $\text{AppRO}T$ 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 $\text{AppRO}T$ does not depend on the input of the honest party, and is properly emulated by $S$ above. As the input to the $\text{AppRO}T$’s are distributed identically (resulting from the same $Z$), 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 $P_l$ sends its step-5 message and other honest parties $P_l$ have not contributed their step-5 shares $\oplus M_1 \cup \{P_l \cup P_i\} S^\delta_1$ 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 send is a random independent string (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 $\text{AppRO}T$ 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 $S^\delta$ contributed by the malicious parties $P_i$, which are known to $Z$), $\oplus GBF^{ix}$ sent in the real world by honest parties, along with 0-shares $\oplus M_1 S^\delta_l$ for $l \in I$ are random additive shares of a randomized GBF, $G$, containing the intersection of honest parties’ input with $\tilde{X}_i$, $H = \tilde{X}_i \cap \{ \bigcap_{P_i \notin (I \cup P_0)} X_i \}$, encoded via the
corresponding $\bigoplus_j \{M^i \oplus \tilde{M}^i\}$, at entries \(\{h_i(q) | q \in H\}\) with overwhelming probability. Such a GBF, $G$, has fixed sums at the locations corresponding to elements of $H$ (determined by $M, \tilde{M}$), and is random otherwise.\textsuperscript{18} The overwhelming probability is due to two observations. (1) $G[j]$ is random for every $j$ not in $\bigcup_{x_i} (h_i(x_i))$ for some $X_i$ where $P_i$ is honest, as $Z$ does not know the corresponding $\tilde{M}^i[j]$ 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 $h_i(x)$ 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 $Z$. (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 $\bot$ occurred in any $\mathcal{F}_{\text{AppROT}}$ call, the probability of this is $\leq p_{\text{False}} < 2^{-\sigma}$.

Now let us now consider the case when at least one of the $\mathcal{F}_{\text{AppROT}}$’s resulted in $\bot$ in the real world. In this case $S$ sends $\bot$ to $\mathcal{F}_{\text{MPSI}}$, and thus receives $\bot$. 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 $Z$ 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 $Z$ 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 $\mathcal{F}_{\text{AppROT}}$ are freshly random in each execution, and only the locations of the values selected can be influenced by receiver’s input). $S$ 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.

\textsuperscript{18}In other words, this is a solution to a certain linear equation system over the field $F_2^\sigma$, since the coefficients of the system are in $F_2^\sigma$.}

\[\]
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 \( F_{MPSI} \) with statistical UC security [5] in global Random Oracle model [7]. We will need the \( F_{\sigma,N}^{\sigma,N} \) defined in fig. 7 and the \( F_{RO} \), 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. Therefore, following instantiation of \( F_{\sigma,N}^{\sigma,N} \), we may move the corresponding parts to an offline phase, as in \( \Pi_{PSimple} \).

We also note that as \( \Pi_{APPROT} \) 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 \( F_{\sigma,N}^{\sigma,N} \) being UC – gRO secure. In Section 5, we discuss implementations of the idealized functionality \( F_{\sigma,N}^{\sigma,N} \) 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 random oracle for all instances of \( \Pi_{APPROT} \), which in our implementation indeed makes call 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 \( \Pi_{APPROT} \) – approximate K-out-of-N random OT protocol to obtain garbled Bloom filters for the Bloom filter of length \( N = N_{gRO} \) consisting of \( K \)’s. We divide this protocol in two phases: offline \( \Pi_{OFFLINE}^{\sigma,N} \) in \( F_{\sigma,N}^{\sigma,N} \)-hybrid model in Figure 13, which is performed before parties receive their inputs, and online \( \Pi_{ONLINE}^{\sigma,N} \) in Figure 14. The functionality of \( \Pi_{ONLINE}^{\sigma,N} \) is somewhat truncated in sense that we do not make parties use their Bloom filters inside \( \Pi_{APPROT} \) 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 \( F_{\sigma,N}^{\sigma,N} \) should go from the outputs of \( \Pi_{APPROT}^{\sigma,N} \) which requires the integrity (i.e. using MACs from both parties). Thus, we give to \( PSimple \) the burden of the intermediate processing the outputs from \( \Pi_{OFFLINE}^{\sigma,N} \) to form the inputs for \( \Pi_{APPROT}^{\sigma,N} \).

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 \( F_{\sigma,N}^{\sigma,N} \) (Fig. 7) with \( N = N_{OT} \) parallel instances of 1-out-of-2 OT.

### Protocol \( \Pi_{OFFLINE}^{\sigma,N} \) in \( F_{\sigma,N}^{\sigma,N} \)-hybrid model

**Parameters:**
- \( \sigma \) – length of the OT strings (computational security parameter);
- \( \lambda \) – statistical security parameter;
- \( N_{OT} \), \( N_{CC} \), \( N_{maxones} \) are parameters of cut-and-choose described in Section 4.

**Inputs:** no inputs. **Offline phase** \( \Pi_{OFFLINE}^{\sigma,N} \)

1. **[1-out-of-2 OTs]** The sender and the receiver call \( F_{\sigma,N}^{\sigma,N} \) performing \( N_{OT} \) OTs in parallel. The receiver chooses requests \( c_1, ..., c_{N_{OT}} \) with \( N_{ODS} \) ones among them, and \( N_{OT} - N_{ODS} \) zeros (randomly permuted). The sender chooses uniformly at random \( M_0 = \{ m_{01}, ..., m_{0N_{OT}} \}, M_1 = \{ m_{11}, ..., m_{1N_{OT}} \} \) \( \{ m_0, m_1 \} \) are of length \( \sigma \). In the \( j \)th OT, the receiver uses choice bit \( c_j \) and learns \( m_j = m_{jc_j} \).

2. **[cut-and-choose challenge]** The sender chooses set \( C \subseteq \{ N_{OT} \} \) 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 \( r^* = \bigoplus_{j \in R} m_{jc_j} \). The sender replies with \( \perp \) if \(| C | - | R | > N_{maxones} \) or if \( r^* \neq \bigoplus_{j \in R} m_{jc_j} \)

**Outputs:** \( M_2 \) to the sender; \( M_3 = \{ m_{jc_j} | c_j = 1, j \in \{ N_{OT} \} \setminus C \} \) to the receiver.

---

**Figure 13:** Protocol \( \Pi_{OFFLINE}^{\sigma,N} \) in \( F_{\sigma,N}^{\sigma,N} \)-hybrid model

**Lemma G.1.** The protocol \( \Pi_{OFFLINE}^{\sigma,N} \) realizes the functionality \( \Pi_{OFFLINE}^{\sigma,N} \) with statistical UC-security with abort in the presence against static (unbounded) malicious adversaries in the \( F_{\sigma,N}^{\sigma,N} \)-hybrid model.\(^9\)

**Proof:** In the following analysis, we do not need to explicitly deal with delaying or deleting messages by \( Z \). 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 \( S \)

\(^9\)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.
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 $Z$ 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 $F_{\text{Offline}}^{\text{AppROT}}$ which requires participation of both parties (but the honest one is waiting).

**Security in face of a corrupted receiver.** The simulator $S$, once activated by $Z$, emulates the protocol towards $Z$.\textsuperscript{20} In the 1st step of the protocol, when emulating $F_{\text{OT}}^{\sigma,N}$ and obtaining the input $c_1, c_2, \ldots, c_{\text{N\text{OT}}}$ from $Z$, $S$ randomly chooses a set $C \subseteq [\text{N\text{OT}}]$, where $|C| = N_{cc}$, computes

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

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

If $p_{\text{False}} < 2^{-\sigma}$, then passes $I$ as an input of the receiver to $F_{\text{Offline}}^{\sigma,N}$ and receives $M_s = \{m_1, m_2, \ldots, m_{K'}\}$. Then, $S$ computes the receiver’s output $\{m_j | j = 1, 2, \ldots, \text{N\text{OT}}\}$ of $F_{\text{OT}}^{\sigma,N}$ as follows:

$$m_j = \begin{cases} \text{fresh random,} & \text{if } j \in C \text{ or } c_j = 0; \\ \text{next item from } M_s, & \text{else,} \end{cases} \quad (15)$$  

gives $C$ to $Z$ 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 $p_{\text{False}} \geq 2^{-\sigma}$, it passes to $Z$ $\{m_j | j \in \text{N\text{OT}}\}$ of uniformly random $\sigma$-bit strings as the output of $F_{\text{OT}}^{\sigma,N}$ of the 2nd step of the protocol, sends to it $C$ as the cut-and-choose request and, upon getting the answer, passes $Z \perp$ as sender’s reply, appends $\{m_j | j \in \text{N\text{OT}}\}$, and halts.

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

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

$$|C \setminus R| > N_{\text{maxones}} \quad (16)$$  

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 $Z$ 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 $|C \setminus R| > N_{\text{maxones}}$ no longer holds. In the latter case, there is at least one value $j \in R$, for which $m_j$ 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 $r$). $S$ 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 $[\text{N\text{OT}}] \setminus C$ 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 $P_{\text{False}}$ computed for the Bloom filter of length $N$ with $K' = |I|$ is in it, is larger than $2^{-\sigma}$ except with negligible (in $\lambda$) 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 $2^{-\sigma} + \text{neg}(\lambda)$.

$$\text{Ideal}_{F, Z, S} \approx \text{Real}_{R, Z} \approx (D, \perp). \quad (17)$$  

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

\textsuperscript{20}As mentioned above, $A$ is fixed to just relays messages from $Z$ to the parties and back. Intuitively, $S$ attempts to do the same.
Now, consider the case when (16) is not satisfied in the real world. If Z sends R' (which differs from R induced by its $F_{OT}^\sigma N$’s inputs) so that (16) is satisfied for C, R', or $r' \neq \bigoplus_{j \in R} m_j$ the sender outputs $\perp$ and halts immediately. By construction of S, it sends the uniformly random set in 1st step, and $\perp$ in 3rd. 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 $Real_{ILZ}$, the sender’s output is $\perp$).

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

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

In the first step of the protocol, S plays $F_{OT}^\sigma N$ towards 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, S waits for the message C from Z. If $|C| > N_{CC}$, then truncates it, and if $|C| < N_{CC}$, then adds indices by default (1, 2, ...). It computes the set $S_k = \{m_i \in M_1|i \notin C\}$ and gives it to the ideal functionality.

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

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

Then $Z$ sends the set $C \subset [N_{OT}]$ 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 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 $Z$ 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 $\Pi_{Toproc}$, we do not consider the functionality of the entire protocol in $F_{AppOT}$-hybrid model, as it requires either rewriting or construction of the reactive functionality as in fig. 8 (because $F_{Online}$ is the random functionality). It is possible to prove either UC-security of the reactive $\Pi_{AppOT}$, or the security of $\Pi_{Offline}$ separately, when $F_{Online}$ is a simple functionality of the secure channel for transmitting the permutation from the receiver to the sender.

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

As the only message in an online-phase of $\Pi_{AppOT}$ is sent, then the following lemma is obviously holds.

**Lemma G.2.** The protocol $\Pi_{Online}$ UC-secure realizes the functionality $\mathcal{F}_{Online}$ in the presence against static (unbounded) malicious adversaries.
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 $\Pi^{\text{Offline}}_{\text{PSImple}}$ is shown in Fig. 17. Furthermore, we need to add a padding after $\mathcal{F}^{\text{Offline}}_{\text{APPROT}}$, 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. Our PSImple protocol is divided in offline and online phases, which requires the separate functionalities for these phases. Although the offline-phase of PSImple just uses the number $\mathcal{F}^{\text{Offline}}_{\text{APPROT}}$’s and the point-to-point secure channels to transmit secret shares $\mathcal{S}^{ij}$’s from $P_i$ and $P_j$, and thus implies a rather simple functionality, we formally describe this $\mathcal{F}^{\text{Offline}}_{\text{APPROT}}$ functionality in Fig. 18.

Theorem G.3. The protocol $\Pi^{\text{Offline}}_{\text{PSImple}}$ securely realizes the functionality $\mathcal{F}^{\text{Offline}}_{\text{PSImple}}$ with statistical UC-security with abort in presence of static malicious adversary corrupting up to $t$ parties in the $\mathcal{F}^{\text{Offline}}_{\text{APPROT}}$-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 $\mathcal{F}^{\text{Offline}}_{\text{PSImple}}$ and $\mathcal{F}^{\text{Offline}}_{\text{APPROT}}$ 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 $\mathcal{F}^{\text{Offline}}_{\text{PSImple}}$ 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, $\mathcal{F}^{\text{Offline}}_{\text{PSImple}} \circ \mathcal{F}^{\text{Online}}_{\text{APPROT}}$-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).

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{fig16.png}
\caption{The offline phase of PSImple protocol in the $\mathcal{F}^{\text{Offline}}_{\text{APPROT}}$-hybrid model.}
\end{figure}
Figure 18: $\mathcal{F}_{\text{Offline}}^{\text{PSImple}}$ – Ideal offline PSImple functionality

Online phase of Malicious-secure Multiparty PSI protocol $\Pi_{\text{PSImple}}$ in the $\mathcal{F}_{\text{Offline}}^{\text{PSImple}}, \mathcal{F}_{\text{Online}}^{\text{AppROT}}, \mathcal{F}_{\text{AppROT}}$-hybrid model

Parameters:
- $n$ - the maximal size of the input set of the party; $\sigma$ - computational security parameter; $\lambda$ - 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 $\mathcal{F}_{\text{Offline}}^{\text{PSImple}}$:
- Parties call $\mathcal{F}_{\text{Offline}}^{\text{PSImple}}$, the offline PSImple functionality.

Online Phase $\mathcal{F}_{\text{Online}}^{\text{PSImple}}$:
1. $[\text{(R1)} \text{ compute Bloom filters}]$ $P_i (i \in \{t \cup \{0\}\})$ makes queries to $\mathcal{F}_{\text{BRO}}$ 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. $[\text{(R1)} \text{ symmetric approximate ROTs-online}]$ Parties perform in parallel:
   a. Using $BF_i$’s $1$’s indices set, $P_0$ computes the random permutation $\pi'_0 : \{N_{OT} - N_{CC}\} \rightarrow [N_{BF}]$ of $M'_i$, and calls $\mathcal{F}_{\text{AppROT}}^{\text{Online}}$ as the receiver with each of the other parties $P_i (i \in \{t\})$. Each $P_i (i \neq 0)$ computes $M'_i = \pi'_0(M'_i)$ and $P_0$ computes $M'_i = \pi'_0(M'_i)$. 
   b. Using $BF_i$’s $1$’s indices set, each $P_i (i \in \{t\})$ computes the random permutation $\pi_i : \{N_{OT} - N_{CC}\} \rightarrow [N_{BF}]$ of $M'_i$, and calls $\mathcal{F}_{\text{AppROT}}^{\text{Online}}$ as the receiver with $P_0$. Each $P_i (i \neq 0)$ computes $M'_i = \pi_i(M'_i)$ and $P_0$ computes $M'_i = \pi_i(M'_i)$.
3. $[\text{(R2) compute and re-randomize GBFs}]$ If $P_0$ did not receive $\perp$ from $\mathcal{F}_{\text{Offline}}^{\text{PSImple}}$ or $\mathcal{F}_{\text{Online}}^{\text{AppROT}}$, it computes $\text{GBF}^0 = \bigoplus_{i \in \{t\}} [M'_i \oplus \tilde{M}'_i]$. If $P_i$ did not receive $\perp$ from $\mathcal{F}_{\text{Offline}}^{\text{PSImple}}$ or $\mathcal{F}_{\text{Online}}^{\text{AppROT}}$, it computes $\text{GBF}^i = M'_i \oplus \tilde{M}'_i$, codewords $y_{ij} = \bigoplus_{\text{rech.}(x_{ij})} \text{GBF}^i (j \in \{n_i\})$ and re-randomizes $\text{GBF}^i$ from $X_i$ and codewords $y_{ij} (j \in \{n_i\})$ according to algorithm BuildGBF from B.1.
4. $[\text{(R2) cumulative GBFs of } P_i s]$ If $P_i (i \in \{t\})$ did not receive $\perp$ from $\mathcal{F}_{\text{Offline}}^{\text{PSImple}}$ or $\mathcal{F}_{\text{Online}}^{\text{AppROT}}$ it computes and sends to $P_0$ the cumulative garbled Bloom filter:

\[
\text{GBF}^* = \text{GBF}^i \bigoplus_{t \in \{t\} \cup \{0\}} [S_{t1} \oplus S_{t2}].
\]

Else it sends $\perp$.
5. $[\text{(R2) cumulative GBF of } P_0]$ 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. $[\text{(R2) output}]$ If $P_0$ did not receive $\perp$’s before, it outputs $x_{0j}$ as a member of the intersection, if

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

Else it outputs $\perp$.

Figure 19: The online phase of PSImple protocol in the $\mathcal{F}_{\text{Offline}}^{\text{PSImple}}, \mathcal{F}_{\text{Online}}^{\text{AppROT}}$-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 \(|Q|p_{\text{false}} < |Q|2^{-\sigma}\). To keep it negligible, \(|Q| = \text{poly}(\sigma)\). In the case of polynomially-bounded (in \(\sigma\)) domain \(\mathcal{Z}\), this requirement is fulfilled automatically, otherwise (for example, it the typical case of an exponential-size domain) we require a computationally bounded (in \(\sigma\)) adversary in Theorem 3.1.

**Theorem G.4.** The protocol \(\Pi_{\text{PSimple}}\) 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 \(\mathcal{F}_{\text{PSimple}}\), \(\mathcal{F}_{\text{Online}}\), \(\mathcal{F}_{\text{RO}}\)-hybrid model, which makes a polynomially bounded number of queries to \(\mathcal{F}_{\text{RO}}\), 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, \(p_{\text{false}} \leq 2^{-\sigma}\), and for the Bloom filter with \(n' + 1\) and more elements, \(p_{\text{false}} > 2^{-\sigma}\). It means, that the corrupt party in \(\mathcal{F}_{\text{Offline}}\) receives \(\bot\) if and only if its input set of indexes \(j_i\) or \(\tilde{j}_i\) (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 \(I \subseteq \{P_1, ..., P_t\}\) are corrupt.

**Simulator description.** The simulator \(S\), once activated by \(Z\), emulates \(\mathcal{F}_{\text{Offline}}\) towards \(Z\). We stress, that all the corrupt parties are emulated asynchronously, according to the message scheduling decided by \(Z\) - one message at a time. We only describe the simulation by order of steps in the protocol for convenience. \(S\) sends to \(Z\) uniformly random shares \(S^{\Pi}\), as honest \(P_i\)’s would do according the protocol, to any corrupt party \(P_i \in I\).

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

- Once \(P_i\) sends \(M_i^{s}\), the simulator answers to \(P_i\) nothing;
- Once \(P_i\) sends \(\tilde{j}_i\), the simulator checks the size of it and answers with either uniformly random set \(M_i^{s}\) (\(S\) samples uniformly at random the \(N_{\text{OT}} - N_{\text{cc}}\)-element set \(M_i^{s}\) and chooses from there items with indexes from \(j_i\)) or \(\bot\).

Finally, \(S\) learns \(S^{\Pi}\) from \(Z\), emulating \(\mathcal{F}_{\text{Offline}}\).

In step 1, \(S\) makes queries \((q, \text{PID}, \text{SID})\) to the global Random Oracle (to compute Bloom filter’s hash-indices) on behalf of corrupt parties \(P_i \in I\) as requested by \(Z\) 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 \(Z\) is. Denote \(Q = \cup_{i \in \{P_i \in I\}} Q_i\) - the joint query set of corrupt parties. The environment might continue making such a queries up to the end of the protocol, and \(S\) adds them in \(Q\) up to the end of Step 4 (when the effective intersection of corrupt parties inputs is extracted).

In step 2, \(S\) plays \(\mathcal{F}_{\text{Online}}\) functionality, obtaining the permutations \(\pi_i\) from corrupt parties and sending them uniformly random permutations \(\pi_i : [N_{\text{OT}} - N_{\text{cc}}] \rightarrow [N_{\text{cc}}]\). Then the simulator computes:

\[
M_i^{s} = \pi_i(M_i^{s}), \quad M_i = M_i^{s} \oplus (M_i^{s}), \quad \hat{M}_i = \pi_i(\hat{M}_i), \quad M_i = \pi_i(\hat{M}_i).
\]

At the 4th step, \(S\) observes \(GBF^{\Pi}\)’s or \(\bot\)’s sent by \(Z\) on behalf of corrupt \(P_i \in I\). Besides, it asks \(\mathcal{F}_{\text{RO}}\) 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 \(A\) is).

- If there were no \(\bot\)’s as an outputs of the simulated \(\mathcal{F}_{\text{Offline}}\) or as the messages of the 6th step, \(S\) computes the sum \(GBF_I^{\Pi} = \bigoplus_{P_i \in I} GBF^{\Pi}\). Now \(S\) can subtract all the secret shares sent and received to corrupt parties on behalf of honest and vice versa:

\[
GBF_I = GBF_I^{\Pi} \bigoplus_{P_i \in I} \bigoplus_{P_i \in \mathcal{P}\backslash\{I \cup P_0\}} \left(\psi^{I}_{i} \oplus S^{\Pi}_{i}\right).
\]

\(GBF_I\) is the effective value of \(\bigoplus_{P_i \in I} GBF^{\Pi}\). Now the simulator extracts the effective input of corrupt parties as

\[
\tilde{X}_i = \left\{ q \in Q \bigoplus_{r \in h_i(q)} GBF_I[r] = \bigoplus_{P_i \in I} \bigoplus_{r \in h_i(q)} \left(M_i^{s}[r] \oplus \hat{M}_i^{s}[r]\right)\right\},
\]

sends it to the ideal functionality, and receives either \(\tilde{X}\) or \(\bot\) as the output of \(\mathcal{F}_{\text{PSimple}}\).

- Else, the simulator sends the effective input of the adversary \(\bot\) to the ideal functionality and receives \(\bot\) as its output.
**Simulator Analysis.** Consider an environment $Z$ running on some fixed public parameters $1^n, 1^λ, t, n, k, N_{BF}$. We assume first that all parties receive inputs from $Z$ (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 $X = \{X_i\}_{P_i \in P^f \cup \{t\}}$ inputs of honest parties. We prove indistinguishability by induction on the message graph of $Z$ - sent messages to the various parties, and to $F_{\text{Offline}}^{\text{PSImple}}$ throughout the execution, starting with the inputs provided, and messages received from honest parties (emulated by $S$ in the ideal world) are statistically indistinguishable. The induction is on the message number according to the order of message delivery by $Z$ in the real world (which $S$ follows). As $P_0$ is honest, we have to also prove the indistinguishability of the joint view of $Z$ with the output of the honest $P_0$ in the simulation and in the real-world execution of the protocol (conditioned in $Z$’s view, for an overwhelming fraction of the views, as we shall show).

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

$\{M^I_i \mid P_i \in I\}, \{M^I_i \mid \bot\}P_i \in I, \{S^I_i\}_{P_i \in P, P_i \notin I, j \neq 0}$: these messages to $Z$ are identically distributed to the values received by the corrupted sender/receiver in the real world protocol, by definition of $F_{\text{Offline}}^{\text{PSImple}}$, and the fact that at any step of interaction of the $F_{\text{Offline}}^{\text{PSImple}}$, the view of each emulated $P_i$ is distributed identically to the real world. Note that in particular, these values do not depend on $X$.

Only one transaction in the protocol from the honest party to the corrupt ones are permutations $x^0_i$ which $S$ sends to $Z$ on behalf of honest $P_i$. 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, $H = X_I \cap \{P_{i \in P} \mid X_i \}$ is the output of $P_0$, or $\bot$ if the simulator sent $\bot$ to the ideal functionality.

In case $S$ did not send $\bot$ 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 $F_{\text{Offline}}^{\text{PSImple}}$ execution, at locations outside of the query set $Q$. However, then they either query too many 1’s in $F_{\text{Offline}}^{\text{PSImple}}$ execution, in which case, in the ideal world as well, $S$ notices it, and sends $\bot$ on behalf of $P_i \in I$ 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 $H_1 = \{p_{e \in P^f \cup \{t\}} X_i \}$ with the probability at most $p_{\text{false}}$ (for instance, by using the received $M^I_i \oplus M^I_{i'}$ at all 1-positions in GBF, 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 $P_j$ adding it, and the result follow. By union bound, adding an element in $H_1$ by $P_j$ occurs with probability $\leq n \cdot p_{\text{false}}$. 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 $P_j \in I$ and $M^I_{j'}$ did not learn $\hat{X}^I_j$. The probability of $P_0$ adding $x_0$s to the output due to passing verification in the step 6 is at most $2^{-\sigma}$, which is the probability of $P_j$ guessing $M^I_{j'}$ by the adversary.

In the simulation, the ideal functionality receives $\bot$ 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 $n^\kappa$ (which is equivalent because of $F_{\text{Offline}}^{\text{PSImple}}$).

Summarizing the above, with statistical distance between real and ideal worlds is at most $\delta$ (2). Finally, consider a situation when the inputs are not provided by $Z$ at the onset of the protocol, but rather at some intermediate point. We are still able to preserve our indistinguishability invariant, since all $Z$ sees before deciding to provide an input to some honest $P_i$ are random independent values received from $F_{\text{Offline}}^{\text{PSImple}}$. Correlating honest parties’ inputs with these values does not break the indistinguishability invariant for our protocol and $S$ above in any way.

Now consider the case of corrupt $P_0$, and some number of other parties (including zero) $I \subseteq \{P_1, \ldots, P_l\}$ are corrupt. Note that at least one party is honest.

**Simulator description.** As before, the simulator interacts with $Z$ through the dummy adversary. It simulates the replies of $F_{\text{Offline}}^{\text{PSImple}}$ and $F_{\text{Online}}^{\text{AppRot}}$ and of the honest parties towards $Z$, by order of its requests. Recall that $Z$ is responsible for giving corrupted parties their input, and these do not go through $S$.

In the preprocessing step, emulating $F_{\text{Offline}}^{\text{PSImple}}$, $S$ sends to corrupt parties $P_i \in I$ uniformly random values $S^I_{t_i}$, as honest $P_i$s would do.

Then, $S$ continues emulate $F_{\text{Offline}}^{\text{PSImple}}$ functionality towards $Z$ for corrupt $P_0$. Once inputs of $P_0$ to $F_{\text{Offline}}^{\text{PSImple}}$ have been sent by $Z$:

- Once $P_0$ sends $j_i$, the simulator checks the size of it and answers with either uniformly random set $M^I_i$ ($S$ samples uniformly at random the $(N_{MT} - N_{j_i})$-element set $M^I_i$ and chooses from there items with indexes from $j_i$) or $\bot$.
- Once $P_0$ sends $M^I_i$, the simulator answers to $P_0$ nothing.
Finally, $S$ gets $s_i^{th}$ to be delivered by corrupted parties $P_i$ from $Z$. We stress, that $S$ delivers messages asynchronously, by the order $Z$ sends messages to parties of $\mathcal{F}_{\text{Offline}}^{\text{PSimple}}$ and the above presentation is written (reporting messages from $\mathcal{F}_{\mathcal{F}_{\text{Offline}}^{\text{PSimple}}}$ is done once an honest party requests to deliver a given message).

In step 1, $S$ makes queries $(q, \text{PID}, \text{SID})$ to the global Random Oracle (to compute Bloom filter’s hash-indices) on behalf of corrupt parties $P_i \in I \cup P_0$ as requested by $Z$ and writes them to sets $Q_i = \{q_{ij}\} \in \{\text{sets}\}$. Here index $i$ corresponds to PID of corrupt $P_i$, $n_i \geq n$ is polynomially bounded, as $Z$ is. Denote $Q = \cup_i q_{ij}$ as the joint query set of corrupt parties. The environment might continue making such a queries up to the end of the protocol, and $S$ 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 $\mathcal{F}_{\text{Online}}^{\text{AppROT}}$ towards corrupt $P_0$, sends it to the uniformly random permutations $\pi_i : [N_{OT} - N_{CC}] \to [N_{BF}]$ on behalf of each honest $P_i$ and receives from $P_0$ permutations $\pi_i^0$. The simulator uses $j_i$ and $\pi_i^0$ to construct the effective Bloom filter of $P_0$ $\tilde{\mathcal{B}}_{\mathcal{F}_{\text{BF}}}$ in the interaction with each honest $P_i$. Then the simulator computes:

$$M^i = \pi_i^0(M^i), \quad \hat{M}^i = \pi_i(M^i).$$

If and once all $\mathcal{F}_{\text{Online}}^{\text{AppROT}}$ executions are completed, $S$ asks $\mathcal{F}_{\text{PSimple}}^{\text{BF}}$ 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}_l = \{q \in Q | \forall s \in h_s(q), \forall j \notin I, \tilde{\mathcal{B}}_{\mathcal{F}_{\text{BF}}}[s] = 1\} \sim \text{queries which are presented in all the extracted Bloom filters of } P_0.

After all $\mathcal{F}_{\text{Online}}^{\text{AppROT}}$ executions are emulated, when the effective malicious input $\tilde{X}_l$ is extracted, $S$ sends either it or $\perp$ to the ideal functionality $\mathcal{F}_{\text{PSimple}}^{\text{BF}}$ 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 $\mathcal{F}_{\text{Offline}}^{\text{AppROT}}$ or $\mathcal{F}_{\text{Online}}^{\text{AppROT}}$’s ended with $\perp$, which is the effective input of the adversary, to $\mathcal{F}_{\text{PSimple}}^{\text{BF}}$ and receives $\perp$ from there.

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

- If $P_i$ received $\perp$ from $\mathcal{F}_{\text{Offline}}^{\text{AppROT}}$ or $\mathcal{F}_{\text{Offline}}^{\text{AppROT}}$, $S$ sends $\perp$ to $Z$ as the message for $P_0$.
- If $P_i$’s $\mathcal{F}_{\text{Online}}^{\text{AppROT}}$’s are completed successfully, and $P_i$ is not the last honest party whose $\mathcal{F}_{\text{Online}}^{\text{AppROT}}$’s done, $S$ sends a uniformly random $\tilde{\mathcal{B}}_{\mathcal{F}_{\text{BF}}}$ to $Z$ as the message for $P_0$.
- If all the $\mathcal{F}_{\text{AppROT}}$’s are completed, but there were at least one $\perp$, and $P_i$ is the last honest party whose $\mathcal{F}_{\text{Online}}^{\text{AppROT}}$’s done (without $\perp$), $S$ sends a uniformly random $\tilde{\mathcal{B}}_{\mathcal{F}_{\text{BF}}}$ to $Z$ as the message for $P_0$.
- If $\mathcal{F}_{\text{Offline}}^{\text{AppROT}}$ and all the $\mathcal{F}_{\text{Online}}^{\text{AppROT}}$’s are completed successfully, and $P_i$ is the last honest party whose $\mathcal{F}_{\text{Online}}^{\text{AppROT}}$’s done, then $S$ performs the following:

- Computes Bloom filters $\tilde{B}_i^j$ for the set $\tilde{X}$ for all $j$ such that $P_j \notin (I \cup P_0)$;
- Computes $\tilde{B}_{\mathcal{F}_{\text{BF}}} = \bigoplus_i \tilde{B}_i^j$ for all $j$ such that $P_j \notin (I \cup P_0)$;
- Computes codewords $y_j$ from $\tilde{B}_{\mathcal{F}_{\text{BF}}}$ as in the protocol, but only for items $x_j \in \tilde{X}$ for all $j$ such that $P_j \notin (I \cup P_0)$;
- Computes re-randomized $\tilde{B}_{\mathcal{F}_{\text{BF}}}$ for items $x_j \in \tilde{X}$ and their codewords $y_j$ as in B.1; note, that positions at indices $r$ such that $\tilde{B}_{\mathcal{F}_{\text{BF}}}[r] = 0$ are entirely and uniformly random;
- Computes $\tilde{B}_{\mathcal{F}_{\text{BF}}}^{\text{temp}}$, honestly as in 6th step of the protocol for all $j$ such that $P_j \notin (I \cup P_0)$, and $\tilde{B}_{\mathcal{F}_{\text{BF}}}^{\text{temp}} = \bigoplus_{j \notin I, P_0} \tilde{B}_{\mathcal{F}_{\text{BF}}}^{\text{temp}}$ (here $\tilde{B}_{\mathcal{F}_{\text{BF}}}^{\text{temp}}$ are messages sent by $S$ on behalf of other honest parties in 4th step);
- $S$ sends $\tilde{B}_{\mathcal{F}_{\text{BF}}}^{\text{temp}}$ to $Z$ as the message for $P_0$.

**Simulator Analysis.** Fix a certain $Z$, running on the public parameters $1^\ast, 1^\lambda, k, N$. $S$ proceeds as follows. We prove indistinguishability by induction on the message graph of $Z$ sent messages to the various parties, and to functionalities $\mathcal{F}_{\text{Offline}}^{\text{AppROT}}$ and $\mathcal{F}_{\text{Online}}^{\text{AppROT}}$ throughout the execution, starting with the inputs provided, and messages received from honest parties (emulated by $S$ in the ideal world) are statistically indistinguishable. The induction is on the message number according to the order of message delivery by $Z$ in the real world (which $S$ follows).

At the start, the (partial) view of $Z$ is clearly the same in both worlds (as $Z$ 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 $Z$ hands all inputs to honest parties at the onset of the protocol (we later explain how to get rid of this assumption).
In \( T_{\text{Offline}} \) 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 \( T_{\text{Offline}} \) \footnote{In other words, this is a solution to a certain linear equation system over the field \( \mathbb{F}_2 \), since the coefficients of the system are in \( \mathbb{F}_2 \).} and by the construction of \( S \). In particular, \( \{S^1\}_{P_i \in P \setminus \{P_0\}} \) are random i.i.d strings, messages \( \{\tilde{M}^i / \bot\} \) \footnote{Noted that when a step-4 message from one or more honest party was not yet sent when another honest party \( P_j \) sends its step-4 message and other honest parties \( P_l \) have not contributed their step-4 shares \( \oplus_{j \in P \setminus \{P_0\}}(S^1_{P_j}) \) yet, they send an additive share of the final sum (the distribution of which we analyze below). In particular, if an honest \( P_j \) 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 send is a random independent string share.} are also uniformly random, and \( \{\tilde{M}^i / \bot\} \) \footnote{Noted that when a step-4 message from one or more honest party was not yet sent when another honest party \( P_j \) sends its step-4 message and other honest parties \( P_l \) have not contributed their step-4 shares \( \oplus_{j \in P \setminus \{P_0\}}(S^1_{P_j}) \) yet, they send an additive share of the final sum (the distribution of which we analyze below). In particular, if an honest \( P_j \) 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 send is a random independent string share.} are defined by the adversary based on the previous transcript of outputs from \( T_{\text{Offline}} \).

Calls to the \( T_{\text{RO}} \) 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 \( T_{\text{Online}} \) \footnote{In other words, this is a solution to a certain linear equation system over the field \( \mathbb{F}_2 \), since the coefficients of the system are in \( \mathbb{F}_2 \).} are also distributed identically in both worlds due to the randomness. The order in which the simulator sends answers to \( P_0 \) or to \( P_1 \) 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 \( P_j \) sends its step-4 message and other honest parties \( P_l \) have not contributed their step-4 shares \( \oplus_{j \in P \setminus \{P_0\}}(S^1_{P_j}) \) yet, they send an additive share of the final sum (the distribution of which we analyze below). In particular, if an honest \( P_j \) 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 send is a random independent string share.

We now compare the effect of step 4, assuming \( T_{\text{Offline}} \) and all executions of \( T_{\text{Online}} \) with honest parties have completed. Assume first they have completed without any \( \bot \)’s in the real world. In this case, (after canceling the shares \( S^1 \) contributed by the malicious parties \( P_i \), which are known to \( Z \)), \( \oplus_{\tilde{M} \in I} \) sent in the real world by honest parties, along with 0-shares \( \oplus_{l \in P \setminus \{P_0\}}(S^1_{P_l}) \) for \( l \in I \) are random additive shares of a randomized GBF, \( G \), containing the intersection of honest parties’ input with \( \tilde{X} \), \( H = \tilde{X} \cap (\bigcap_{P_i \in P \setminus \{P_0\}} X_i) \), encoded via the corresponding \( \oplus_{i \in I} \{M^i \oplus \tilde{M}^i\} \), at entries in \( \{h_i(q)|q \in H\} \) with overwhelming probability. Such a GBF, \( G \), has fixed sums at the locations corresponding to elements of \( H \) (determined by \( M, \tilde{M} \), and is random otherwise.\footnote{In other words, this is a solution to a certain linear equation system over the field \( \mathbb{F}_2 \), since the coefficients of the system are in \( \mathbb{F}_2 \).} The overwhelming probability is due to two observations.

(1) \( G[j] \) is random for every \( j \) not in \( \cup_{x_i} (h_i(x_i)) \) for some \( x_i \), where \( P_i \) is honest, as \( Z \) does not know the corresponding \( M^i[j] \) 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 \( h_i(x) \) we queried in 2(a) as \( P_0 \) for some element \( x \), \( G[j] \) is distributed uniformly at random, due to the fact that \( M^i[j] \) is not learned by \( Z \). (3) Words \( x \) used by \( P_0 \) in step 2(a) (from all parties) on which the \( T_{\text{RO}} \) was not queried (resulting in no indices from the first or second kind).

As no \( \bot \) occurred in \( T_{\text{Offline}} \) or any \( T_{\text{Online}} \) call, the probability of this is \( p_{\text{False}} < 2^{-\sigma} \).

Now let us now consider the case when \( T_{\text{Offline}} \) \footnote{In other words, this is a solution to a certain linear equation system over the field \( \mathbb{F}_2 \), since the coefficients of the system are in \( \mathbb{F}_2 \).} or at least one of the \( T_{\text{Online}} \)’s resulted in \( \bot \) in the real world. In this case \( S \) sends \( \bot \) to \( T_{\text{MPI}} \), and thus receives \( \bot \). 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 \( Z \) 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 \( Z \) 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 \( T_{\text{RO}} \) are freshly random in each execution, and only the locations of the values selected can be influenced by receiver’s input). \( S \) 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.\footnote{In other words, this is a solution to a certain linear equation system over the field \( \mathbb{F}_2 \), since the coefficients of the system are in \( \mathbb{F}_2 \).}