PSIImples: Practical Multiparty Maliciously-Secure Private Set Intersection

Aner Ben Efraim
Ariel University
anermosh@post.bgu.ac.il

Olga Nissenbaum
Ariel University
olga@nissenbaum.ru

Eran Omri
Ariel University
omrier@ariel.ac.il

Anat Paskin-Cherniavsky
Ariel University
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. PSI is a rapidly developing area and there exist many highly efficient protocols. However, almost all of these protocols are for the case of two parties or for semi-honest security. In particular, prior to our work, there has been no concretely efficient, maliciously secure multiparty PSI protocol.

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

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

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

1 Introduction

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

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

Our focus in this work is on the construction of concretely efficient PSI-tailored protocols (where by “concrete efficiency” we mean faster run-time in practice). It is instructive to note that a protocol may have very good asymptotic efficiency, but perform poorly in practical scenarios. This is usually due to extensive use of public-key operations, typically requiring exponentiation, which result in large constants. In particular, for 2-party malicious PSI protocols, Rindal and Rosulek [36] showed that the protocol of [10], which is based on Diffie-Helman, despite requiring significantly less communication, is more than an order of magnitude slower than their protocol, which is based on oblivious transfer\(^1\) and garbled Bloom filters. Concretely efficient generic MPC protocols (e.g., [2, 9, 15,

\(^{1}\)While OT is based on public key operations, modern MPC protocols use OT extension [19], in which only a small amount of OTs require public key operations, and the rest are generated using symmetric-key primitives.
21, 38]) are less suitable for the PSI problem, as their complexity highly depends on the circuit size, which is large for PSI (typically, incurring a slowdown of two orders of magnitude, see [25]). 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., [10,11,22,27,31,32,34,36,37]) or were restricted to deal with semi-honest adversaries [18, 25].

1.1 Review of Related Previous Works

The PSImple protocol relies on two main primitives, oblivious transfer (OT) [33] and garbled Bloom filters (GBF) [11]. $K$-out-of-$N$ oblivious transfer is a cryptographic primitive, allowing a receiver to interact with a sender, holding $N$ strings $s_0,\ldots,s_N$, 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 $h_1(x),\ldots,h_K(x)$ in the BF to be 1.

We next review the existing ideas for PSI protocols based on GBFs, starting with the two-party semi-honest construction of [11].

Two-party semi-honest PSI of [11]. Say that two parties $P_0,P_1$ wish to compute the intersection between their respective sets $S_0$ and $S_1$. Using the above primitive it is natural to consider the following idea. First, each party constructs the BF, according to its private set. Then, they engage in an OT protocol so that $P_0$ (as the receiver) learns the values from $P_1$’s BF – only in the indices holding a 1 value in $P_0$’s BF. Finally, by taking the bit-wise AND from both BFs, $P_0$ learned the BF of the intersection.

While the above protocol is correct, it is not secure, as $P_0$ may learn about 1 value indices in $P_0$’s BF, even if they where set to one on account of elements that are not in the intersection (but are in $P_1$’s set). To overcome this leakage, Dong et al. [11] introduced a variant of Bloom filters, called garbled Bloom filters (GBF). A GBF is attributed with same $K$ hash functions as its respective BF. In each coordinate of the GBF there is a a $\sigma$ long random string. The strings are chosen independently and uniformly, with the only requirement, that for any element $x \in D$ in the underlying set, the XOR over all strings in indices $h_1(x),\ldots,h_k(x)$ in the GBF equals some value $y_x$.

The protocol of [11] follows as before with the only difference that $P_0$ learns the desired coordinates (with value 1 in the Bloom filter of $P_0$) from the garbled Bloom filter of $P_1$. In the GBF variant proposed by [11], it is predetermined that $y_x = x$ for any $x$. Thus, given the appropriate strings, $P_0$ can test whether an element $x$ is in the intersection by checking if the XOR over all strings in indices $h_1(x),\ldots,h_k(x)$ (which it got from $P_1$’s GBF) equals $x$. On the other hand, for any $x'$ that does not belong to $P_0$’s set, $P_0$ learns nothing but random and independent strings.

Two-party malicious PSI of [36]. The construction of [11] works only for semi-honest adversaries, as malicious parties can use an input that is much larger than the allowed set size. Rindal and Rosulek [36] suggested an efficient translation to the malicious setting: First, to prevent $P_0$ from cheating, they needed to use a maliciously secure $K$-out-of-$N$ OT protocol. Second, to prevent $P_1$ from cheating (pretending to have a larger set), they use the following idea. Rather than having a predetermined value $y_x$ for every possible element $x$, the strings of the GBF are chosen uniformly and independently, and $y_x$ is taken to be the result of these choices, i.e., $y_x$ is simply the XOR over all strings in indices $h_1(x),\ldots,h_k(x)$. Finally, after $P_0$ has learned $t$ of the strings, $P_1$ needs to send to $P_0$ the values of $y_x$ for every element $x$ in its set. This, indeed, puts a limit on the number of elements $P_1$ can use. We note that in order for $P_0$ to find the intersection, given the (encoded) items of $P_1$, it is required to perform $\Theta(N \log N)$ comparisons in the number of items in a data set (reduced from a quadratic number by sorting).

Unfortunately, there is no known concretely efficient $K$-out-of-$N$ OT protocol with malicious security. Rindal and Rosulek [36] overcome this problem by constructing a concretely efficient, maliciously secure, approximate random oblivious transfer protocol. They show that this primitive suffices for the security of the above protocol, with a caveat that a malicious $P_0$ can use a slightly larger set than the bound for honest parties. For more details on this protocol, see Section 2.1.

Multiparty semi-honest PSI of [18]. Inbar et al. [18] extended the work of [11] to the multiparty setting for augmented semi-honest security. To compute the intersection over a set of $t+1$ parties $\{P_0,P_1,\ldots,P_t\}$, they used the XOR secret sharing scheme. Specifically, each party $P_i$ computes a GBF $G_i$ for its set – using another variant of GBF, where $y_x = 0$ for every element $x$. Then, $P_i$ shares its GBF $G_i$ among all parties (making the share of each party $P_j$ an independent uniform string $s_j$ and making its own share the XOR of all other $s_j$’s with the GBF $G_i$, i.e., according to the XOR secret sharing scheme). Next each party locally XORs all the shares it got from all parties. It follows, by the GBF variant used, that the XOR all these shares is a valid GBF of the intersection. However, $P_0$ cannot learn this GBF, as it may extract information from it by removing the randomness incorporated by corrupt parties. Hence, $P_0$ uses a (semi-honest secure) OT to learn only the coordinates attributed to its subset. To compute the intersection, $P_0$ computes a cumulative GBF $G^*$ by XORing all the GBFs it obtained from all other parties. Finally, $P_0$ outputs all elements $x$ in its set, for which the XOR of the coordinates in $G^*$ that are attributed with $x$ is zero.
1.2 Contributions

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

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

Our main contributions can be summarized as follows.

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

2. We implemented PSImple and incorporated several code optimizations. We ran experiments to show the practicality of PSImple, showing that PSImple is competitive even against similar protocols that are limited to 2-parties [36] or give a weaker security guarantee [18, 39].

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

We next elaborate on each of these contributions.

The PSImple protocol – A multiparty PSI protocol in the malicious model. A key idea in the two-party malicious PSI protocol of [36] is to somehow “bind” each party to a restricted subset of the coordinates (of the computed GBF for the intersection) that will be correlated with the other party’s GBF. For party $P_0$, this is obtained by the $K$-out-of-$N$ OT it performs (as a receiver) with $P_1$. For $P_1$, the binding effect comes from the fact that $P_1$ can only send a fixed number of codewords to $P_0$.

Trying to combine the ideas from [36] and [18], we note that it is possible to let $P_0$ perform an (approximated, random) $K$-out-of-$N$ OT with each of the other parties separately (i.e., the first part of the [36] protocol). After this phase, $P_0$ holds a garbled Bloom filter for every party, however, it does not know the appropriate codewords (i.e., $y_s$). Obviously, the parties cannot just send the codewords to $P_0$, as $P_0$ is not allowed to learn the intersection of its set with the set any proper subset of the honest parties.

A possible idea for completing the protocol is to let the parties secret share their GBFs (using the XOR scheme) and then reveal the codewords of these XORed shares to $P_0$. This is indeed secure, and it allows $P_0$ to find the intersection by finding codewords, one from each party, that sum to its own codeword. However, all known algorithms for finding these codewords grow exponentially in the number of parties. Therefore, we take a different path, revisiting the construction of [36].

A new two-party malicious PSI. In our two-party malicious construction, the parties start in the same way as in [36], 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 [36], this binds $P_0$ to choose a bounded subset of coordinates from $P_1$’s that may affect the output GBF. To similarly bind $P_1$, in a second phase, the parties switch roles and perform an (approximated, random) $K$-out-of-$N$ OT with $P_1$ as the receiver and $P_0$ as the sender, letting $P_1$ learn the appropriate parts in the GBF $G_0$ of $P_0$.

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

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

We note that this alternative protocol more than doubles the communication complexity compared with [36], but reduces the number of comparisons to be linear in the size of each set. While this has little effect in the two-party case, in the case of more than two parties, the saving is drastic – in PSImple the number of comparisons remains linear in the size of $P_0$’s set. In contrast, the above direct extension of [36] to the multiparty setting would require finding codewords (one from each party) that XOR to 0. To the best of our knowledge, the best solution to this problem is still exponential in the number of parties.

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

Indeed, to generalize our two-party protocol to the multiparty case, we first let $P_0$ perform two approximated, random $K$-out-of-$N$ OT execution with each party $P_i$. Then, $P_0$ XORs all 2r GBFs it obtained in these executions. Let $G_0$ be the re-
We implemented PSImple and incorporated several code optimizations that significantly reduced the communication and the required memory, and also allowed us to move much of the computation to the offline phase (i.e., can be done before the inputs are known to the parties). We ran experiments with 2 to 32 parties and input size of $2^8$ to $2^{18}$, in order to demonstrate the practicality of PSImple, and analyzed the runtime to understand the asymptotics and the cost of the various steps. We compare our results with existing protocols that are based on GBFs, in particular the 2-party maliciously secure PSI protocol of [36] and the multiparty PSI protocols of [18, 39] that give a significantly weaker security guarantee. As PSImple is specifically designed as a multiparty PSI protocol with malicious security, we expected PSImple to be significantly slower than these protocols. However, somewhat surprisingly, our experimental results show that PSImple is quite competitive and in some cases even faster.

1.3 Additional Related Work

Currently, the state-of-the-art in two-party maliciously secure PSI are the protocols of [37] and [27], both concretely-efficient, have quasi-linear and linear communication complexities, respectively, and are almost as efficient as the fastest semi-honest PSI protocol [24]. The benchmarks made in [27] suggest that it is currently the fastest two-party, maliciously secure, PSI protocol. We remark that [27] 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 [27] can also be extended to the multiparty setting.

Apart from [18], an additional PSI protocol in the semi-honest multiparty setting is the protocol of [25], which is based on symmetric-key techniques. The protocol of [25] is significantly faster than [18] for a small amount of parties. However, it does not scale as well with the number of parties, and we do not know if it can be efficiently extended to the malicious setting.

Regarding maliciously-secure multiparty PSI protocols, the works of [16] and [12] both have very good asymptotic communication complexity. However, both of these protocols are not concretely efficient and, therefore, have not been implemented. We remark that [12] achieve a stronger security guarantee than PSImple and [16], because in the protocol of [12] all the parties output the intersection.

Zhang et al. [39] recently made an interesting attempt to build a concretely efficient maliciously secure protocol extending the protocol of [36]. 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 $P_t$. If these parties do collude, then the corrupt parties may learn the intersection of the Bloom filters of the honest parties. In particular, in the three party setting, this implies leaking the BF of the honest party. Furthermore, this leakage occurs even in the semi-honest setting. Hence, the security model they dealt with is significantly more relaxed than the standard malicious security model we assume.

Additionally, there is a line of work that is based on circuits [17, 28–30]. In [29], 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 [17]. 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. Additionally, we provide here a brief description of the PSI protocols of Rindal and Rosulek [36] and Inbar, Omri and Pinkas [18].
Notations. We denote the computational security parameter by σ, and the statistical security parameter by λ. In our implementation, σ = 128 and λ = 40. For l ∈ ℕ, [l] denotes the set {1,...,l}.

We use the notation ℙ = (P₀,...,Pₜ) for the set of parties, where P₀ is the evaluating party, and the remaining t parties are non-evaluating parties. The size of the input set of any honest party is bounded by n, and D is the domain of the input items.

Private Set Intersection. In a private set intersection protocol, a set of parties ℙ = (P₀,...,Pₜ), each having up to n items from domain D as their private inputs, compute the intersection of their input sets. As a result of the protocol, the evaluating party P₀ learns (only) the intersection of those sets, and all other parties learn nothing.

We assume a malicious adversary that may corrupt up to t parties (i.e., all parties but one). The adversary has full control over these parties, and may instruct them to arbitrarily deviate from the prescribed protocol. Following the real vs. ideal paradigm for proving security, our goal is to prove that such an adversary cannot do more harm than a very limited (ideal-world) adversary. In particular, the ideal-world adversary may only choose the inputs of the corrupted parties. We note that the size of these adversarial input sets n’ could be slightly larger than the prescribed bound, i.e., n’ > n (it is possible to show that using our parameters n’ must be smaller than 4n).

Indeed, this is less standard in secure computation, however, we inherit this assumption from the work of [36]. The ideal functionality Π_{MPSI} is given in Figure 1.

Additive Secret Sharing. An additive secret sharing scheme enables a set of t parties to share a secret S such that no proper subset of them can learn any information about S (apart from its length). However, all t parties together are able to reconstruct the secret. Each party Pᵢ receives a value Sᵢ of length |S|, called Pᵢ’s share of S, such that S = S₁ ⊕ ... ⊕ Sₜ. To obtain such a sharing, t − 1 of the shares can be selected uniformly at random, and the last share is set to be the XOR of these t − 1 shares and the secret S.

Additive secret-sharing is linear, so in particular, given two additive sharings of secrets S₁ and S₂, parties can locally compute shares of the sum S₁ ⊕ S₂ by XORing their own shares of S₁ and S₂.

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 NBF bits and associated with k independent random hash functions h₁,...,hₖ: {0,1}ˢ → [NBF]. 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 h₁(x),...,hₖ(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 h₁(x),...,hₖ(x) equal 1, then this is interpreted as if x is a member of the set. Note that this might be a false positive result (i.e., x is misidentified as being represented in the Bloom filter), if other elements of the set turn the Bloom filter bits at indices h₁(x),...,hₖ(x) to 1’s.

We denote by pₚ₉₉ the false-positive probability for a given Bloom filter, i.e., the probability of a positive result for some randomly chosen item. This probability depends (apart from the length of the Bloom filter NBF and the number of hash-functions k) on the number of items currently stored in the Bloom filter (more precisely, on the number on 1’s in Bloom filter). The analysis of the false positive probability of a Bloom filter is less trivial than it may initially seem [4, 10, 14, 26]. In this paper we used the refined formula from [14].

Garbled Bloom filters (GBF) were introduced by Dong et al. [11] as the garbled version of a Bloom filter, obtained by expanding each bit in the original BF to a σ-long bit string. The compactness of the original Bloom filter is somewhat compromised in a GBF for the sake of obtaining an obliviousness property. As before, any element x is attributed with k coordinates in the GBF, i.e., the hash-values h₁(x),...,hₖ(x). Intuitively, this obliviousness property means that for a given element x, it is impossible to learn anything on whether x is in the data set without querying the GBF on all k coordinates attributed to x. On the other hand, given the strings in all coordinates attributed with x, we compute the codeword yₓ as follows.

\[ yₓ = \bigoplus_{i \in hₓ} \text{GBF}[i], \]  

(1)

where \( hₓ = \{h_j(x) : j \in [k]\} \).

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).

<table>
<thead>
<tr>
<th>n</th>
<th>2ⁿ</th>
<th>2ⁿ⁺¹</th>
<th>2ⁿ⁺²</th>
<th>2ⁿ⁺³</th>
<th>2ⁿ⁺⁴</th>
</tr>
</thead>
<tbody>
<tr>
<td>k</td>
<td>147</td>
<td>94</td>
<td>134</td>
<td>94</td>
<td>131</td>
</tr>
<tr>
<td>NBF</td>
<td>64,733</td>
<td>88,627</td>
<td>851,085</td>
<td>1,211,959</td>
<td>12,660,342</td>
</tr>
<tr>
<td>NₒT</td>
<td>74,379</td>
<td>99,372</td>
<td>901,006</td>
<td>1,187,141</td>
<td>12,948,963</td>
</tr>
</tbody>
</table>

Table 1: Comparison of our Π_{MPSI} parameters k, NBF and NₒT with [36] for set size n, statistical security λ = 40, computational security σ = 128.
Appendix B.1. we construct an efficient algorithm to rerandomize

In a 1-out-of-2 random transfer protocol there are two parties: a sender and a receiver. The private input of the receiver is its codewords in $X$ for all data sets: $X = \cap_{x \in X}^{\prime}X_i$. Similarly, if GBF $G_1 \otimes G_2$ is the GBF of $X_1 \cap X_2$, where codewords for any element is XOR of its codewords in $Y_{X_1}$ and $Y_{X_2}$. In particular, if the equal items in $X_1$ and $X_2$ have the same codewords in GBF $G_1$ and GBF $G_2$, then all the items from $X_1 \cap X_2$ have all-zero codewords in GBF $G_1 \otimes G_2$.

Random Oblivious Transfer. In a 1-out-of-2 random oblivious transfer (ROT) protocol there are two parties: a sender and a receiver. The private input of the receiver is its choice bit $b$, while the sender has no input. As a result of the ROT, the sender receives two random values: $m_0$ and $m_1$, and the receiver receives $m_b$. The sender learns no information about $b$, and the receiver learns nothing about $m_{1-b}$. Another functionality we will need, which realizes $N$ parallel instances of 1-out-of-2 ROT for $\sigma$-bit strings appears in Figure 2.

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

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

2.1 The Two-Party Protocol of Rindal and Rosulek [36]

The starting point of our protocol is the maliciously secure two-party PSI protocol of Rindal and Rosulek [36]. We describe the main ideas of their protocol here, as they are important for understanding our protocol.

The high-level idea of [36] for finding the intersection between the data sets of two parties is to let the parties construct GBFs that agree on the codewords of their joint items. This can be achieved using a $K$-out-of-$N$ random OT protocol, by allowing $P_0$ to learn $K$ of the $N$ random strings that $P_1$ learns. $P_0$ then chooses a permutation over $[N]$ that relocates these $K$ strings in indices that are equal 1 in $P_0$’s BF. To turn these $N$ (permuted) strings into a GBF, all $P_1$ needs to do is compute the resulting codewords attributed with its elements (i.e., to obtain the codeword for an element $x$, it computes $h_1(x) \oplus \ldots \oplus h_k(x)$). Finally, $P_1$ sends these codewords to $P_0$.

Approximate $K$-out-of-$N$ Random OT $\Pi_{\text{AppROT}}$. Unfortunately, to the best of our knowledge, no concretely-efficient maliciously-secure $K$-out-of-$N$ OT protocol exists. Instead, [36] implement an approximate $K$-out-of-$N$ Random OT, allowing the receiver to request slightly more than $K$ strings. Rindal and Rosulek [36] 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
We briefly review the augmented to change its input, but then follows the prescribed protocol honestly. We note that the semi-honest secure protocol of [18] is less efficient as every pair of parties need to perform a -out-of-2 OT interaction with the receiver and give the receiver the appropriate coordinates from the GBFs of all other parties (such that the codeword of any encoded item is 0). The difficulty is that if each party now sends its GBF to P0, then P0 would not only recover the intersection of all the parties’ inputs, but also the intersection of its input set with the input set of each other party separately.

To make sure P0 learns the intersection and nothing else, each non-evaluating party begins by additively sharing its GBF among all other parties. The cumulative GBF, i.e., the sum of all the shares Pj got, is then used as the inputs the sender Pj in a -out-of- OT interaction with P0 as the receiver. Note that the XOR of all these cumulative GBFs is a GBF of the intersection of all parties but P0 (this follows by the linear secret sharing and the fact that all codewords equal 0). However, as P0 only selected the coordinates in accordance with its input set, by summing (XORing) all the GBFs it received, P0 computes the GBF of the intersection of the sets of all parties, including P0 itself. Note that this protocol is not secure in the malicious setting as there is no mechanism to prevent the parties from using all-ones Bloom filters, which represents the entire domain of the items.

Remark. We mention that in the PSI protocol of [36] one possible input set size of the adversary’s may be larger than n, and the PSImple protocol inherits this property. This happens because in the cut-and-choose check only the number of 1’s in the Bloom filter is bounded, but not n itself. It is shown in [36] that n’ < 2NBF/σ (the authors stress that this is a very tight bound given for the worst case); we refer the reader to [36] for the detailed analysis.

We note that with the choice of parameters in [36], NBF < 3nσ, so n’ < 6n. With our choice of parameters, since NBF is comparatively lower (see Tables 1 & 4), NBF < 1.73nσ so n’ < 3.46n. Thus, it seems that our improved parameters also give a slightly better security guarantee.

2.2 The Semi-Honest Multiparty PSI Protocol of Inbar et al. [18]

We briefly review the augmented semi-honest secure multiparty PSI protocol of Inbar et al. [18].3 Using a semi-honest

---

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

---

K-out-of-N OT, the evaluating party P0 might receive the appropriate coordinates from the GBFs of all other parties (such that the codeword of any encoded item is 0). The difficulty is that if each party now sends its GBF to P0, then P0 would not only recover the intersection of all the parties’ inputs, but also the intersection of its input set with the input set of each other party separately.

To make sure P0 learns the intersection and nothing else, each non-evaluating party begins by additively sharing its GBF among all other parties. The cumulative GBF, i.e., the sum of all the shares Pj got, is then used as the inputs the sender Pj in a K-out-of-N OT interaction with P0 as the receiver. Note that the XOR of all these cumulative GBFs is a GBF of the intersection of all parties but P0 (this follows by the linear secret sharing and the fact that all codewords equal 0). However, as P0 only selected the coordinates in accordance with its input set, by summing (XORing) all the GBFs it received, P0 computes the GBF of the intersection of the sets of all parties, including P0 itself. Note that this protocol is not secure in the malicious setting as there is no mechanism to prevent the parties from using all-ones Bloom filters, which represents the entire domain of the items.

3 The PSImple Protocol

In this Section we explain in detail our multiparty maliciously-secure PSI protocol, PSImple, and the underlying techniques we use. As mentioned above, we build on the techniques of [36] and [18] (both works, in turn, extend [11]).

As a warm up, we first describe in Section 3.1 the protocol for the two-party case. We do not suggest to use PSImple in the two-party scenario, because it is less efficient than the protocol of [36].4 For more than two parties, however, PSImple is the only concretely efficient PSI protocol secure against malicious adversaries. The full fledged multiparty PSImple protocol is described in Section 3.2. The formal description of the PSImple protocol appears in Figure 5.
\( \mathcal{F}_{\text{AppROT}} \)

**Parties:** a sender, a receiver.

**Parameters:**
- \( \sigma \) – computational security parameter;
- \( \lambda \) – statistical security parameter;
- \( k \) – number of hash-functions in Bloom filter;
- \( N \) – number of items to create.

**Inputs:**
- From the receiver: \( I = \{ i_j \} \), where \( i_j \in [K] \);
- From the sender: no input.

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

If the adversary corrupts the receiver
The functionality works in two steps:

1. The corrupt receiver chooses \( K'' \) and sends it to the ideal functionality.
   - Upon receiving \( K'' \) from the receiver, computes \( p_{\text{False}} \) – the false-positive probability in the Bloom filter of length \( N \), with \( k \) hash-functions and \( K'' \) cells consisting 1’s. If \( p_{\text{False}} \geq 2^{-\sigma} \), then gives \( \perp \) to the sender and to the receiver. Otherwise samples and gives to the receiver the uniformly random \( M' = \{ m'_1, m'_2, \ldots, m'_{K''} \} \), where \( m'_i \)s are \( \sigma \)-bit strings.

2. After the response from the functionality, the receiver chooses and sends either \( \perp \) or a partial injective mapping \( I' = \left( i_1 \ i_2 \ \ldots \ i_{K'} \right) \), where \( i_s \in [K''], j_s \in [N], s \in [K'], K'' \geq K' \geq K \).
   - Upon receiving \( \perp \) from the receiver, gives \( \perp \) to the sender.
   - Upon receiving \( I' \) from the receiver, computes \( m_j = \begin{cases} m_i', & \text{if } j = j_s, s \in [K']; \\ \text{fresh random}, & \text{else}. \end{cases} \)
   - The functionality gives \( M = \{ m_i \} \) to the sender.

If the adversary corrupts the sender
The functionality works in two steps:

1. Upon receiving \( I \) from the receiver, samples and gives to the sender the uniformly random \( M'' = \{ m''_1, m''_2, \ldots, m''_{N_{\text{OT}}} \} \), where \( m''_i \)s are \( \sigma \)-bit strings.

2. Receives from the corrupt sender either \( \perp \) or \( C \subseteq [N_{\text{OT}}] \); \( |C| = N_{\text{cc}} \). Upon receiving \( \perp \) from the sender, gives \( \perp \) to the receiver.
   - Upon receiving \( C \) from the sender, samples the uniformly random \( N \)-permutation \( \psi: [N_{\text{OT}}] \setminus C \rightarrow [N] \), computes \( M = \psi(M'') = \{ m_1, m_2, \ldots, m_N \} \) and \( M_\ast = \{ m_i, m_i, \ldots, m_i \} \). Gives \( M \) to the sender and \( M_\ast \) to the receiver.

Figure 3: \( \mathcal{F}_{\text{AppROT}} \) – Ideal approximate \( K \)-out-of-\( N \) Random OT functionality
Before moving on to describe the protocol, let us first consider what may seem as the direct way to extend the protocol of [36] to the multiparty case, using the ideas of [18], and why it does not work for us. The idea is to have $P_0$ to perform $\Pi_{AppROT}$ with each other party $P_i$ independently and then have $P_0$ XOR all the GBFs received from the $\Pi_{AppROT}$’s to compute its cumulative GBF, denote it by $G^*$.

Recall that the codeword for an element $x$ with respect to a GBF $G$ with hashes $h_1, \ldots, h_k$ is the XOR over the strings in coordinates $h_1(x), \ldots, h_k(x)$. Now, if each party $P_i$ sends to $P_0$ the codewords attributed to its set $X_i$ and its GBF, then $P_0$ can compute the intersection of all parties, however, it can also compute the intersection with party separately, which is not allowed. To avoid this leakage, each $P_i$ can additionally share the its GBF among all parties $P_j$ ($j \in [t]$), and then compute the cumulative GBF $G_i^*$ as the XOR of all the shares it holds. After that, each $P_i$ sends to $P_0$ the codewords attributed to its set $X_i$ and its cumulative GBF $G_i^*$. Finally, $P_0$ concludes that an item $x$ in its input set with codeword $y_x$ is in the intersection, if there exist codewords $y^1, \ldots, y^t$ received from $P_1, \ldots, P_t$, respectively, such that $y_x = y^1 \oplus \ldots \oplus y^t$.

The above is indeed correct and secure. However, an exhaustive search for a combination of codewords that sum to $y_x$ grows exponentially with number of parties, 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 [36] 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 [36], such a binding is achieved for $P_0$ by participating in $\Pi_{AppROT}$ as the receiver. The binding for $P_1$ is achieved when it sends the codewords that correspond to its elements to $P_0$.

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

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

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

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

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

### 3.2 PSImple, Multiparty Case

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

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

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

However, we cannot simply let each party $P_i$ send $GBF^{i\ast}$ to $P_0$ as in the 2-party case, since this would be identical to $i$ 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., $GBF^s$, and from the secrecy property it follows that $P_0$ learns nothing but $GBF^s$.

The PSImple multi-party protocol is described formally in Figure 5. Following the offline/online paradigm, we divide our MPSI protocol $\Pi_{\text{MPSI}}$ into two phases: an offline-phase $\Pi_{\text{Offline}}^{\text{MPSI}}$, which can be executed by the parties before they know their inputs, and an online-phase $\Pi_{\text{Online}}^{\text{MPSI}}$, which is executed after the parties learn their inputs.

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

### 3.3 Security and Correctness

We prove the security of a protocol via the real vs. ideal paradigm (specifically, in the UC model). In Appendix F, we provide a complete proof for the the following theorem, stating the security of the PSImple protocol.

**Theorem 1.** The $\Pi_{\text{MPSI}}$ protocol of Figure 5 securely realizes the functionality $f_{\text{MPSI}}$ with computational UC-security with abort in the presence of a static, non-uniform computationally bounded, malicious adversary $A$ corrupting any number of parties in the $\sigma^{\mathcal{N}}_{\text{ROT}} \cdot f_{\text{RO}}$-hybrid model, where the Bloom filter hash functions are modeled as (non-programmable) random oracles, and the other protocol parameters are chosen as described in Section 4.

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

**Correctness.** Our goal here is to prove that in an honest execution of the protocol the output of $P_0$ is indeed the intersection. Let $x$ be an item in the intersection of all sets. It follows by construction, as previously explained, that $P_0$ outputs $x$ as part of the intersection. Specifically, for every $i \in [t]$ in the interactions between $P_0$ and $P_i$, both parties are going to request the coordinates attributed with $x$ from the other party. Thus, both codewords for $x$ (for both $P_0$ and $P_i$) are going to be summed into the cumulative $GBF$ of each of
Protocol of Malicious-secure Multiparty PSI \( \Pi^{MPSI} \)

**Parameters:**
- \( \sigma \) - computational security parameter;
- \( \lambda \) - statistical security parameter;
- \( N_{BF} \) - size of the Bloom filter;
- \( N_{OT} > N_{BF} \) - number of random OTs to perform;
- \( N_{OT}, N_{cc}, N_{maxones} \) - parameters for \( \Pi_{AppROT} \) computed as in Sec. 4.

**Inputs:** Each party \( P_i, i \in \{0,...,t\} \), inputs its set of items \( X_i = \{x_{i1}, x_{i2},...,x_{in_i}\}, n_i \leq n, x_{ij} \in \mathcal{D}. \)

**Offline-phase \( \Pi^{Offline}_{MPSI} \):**

1. [hash seeds agreement]
   Parties run a coin-tossing protocol to agree on random hash-functions \( h_1, h_2, ... h_k \colon \{0,1\} \rightarrow [N_{BF}] \).

2. [symmetric approximate ROT-offline] Parties perform in parallel (with parameters \( N_{OT}, N_{OT}, N_{cc}, \text{ and } N_{maxones} \)):
   (a) P_0 as a receiver performs \( \Pi^{Offline}_{AppROT} \) with each \( P_i, i \in [t] \).
   (b) Each \( P_i, i \in [t], \) as a receiver performs \( \Pi^{Offline}_{AppROT} \) with \( P_0 \).

3. [random shares] Each \( P_i, i \in [t], \) sends \( S^i = (s^i_1, ..., s^i_{N_{maxones}}) \) to any \( P_j, j \in [t] \setminus \{i\} \), where \( s^i_r \in \{0,1\}^\sigma, r \in [N_{BF}] \).

**Online-phase \( \Pi^{Online}_{MPSI} \):**

4. [compute Bloom filters] Each party \( P_i, i \in [t] \cup \{0\}, \) locally computes the Bloom filter BF_i of its input set \( X_i \). If \( n_i < n \), then \( P_i \) computes the Bloom filter of the joint set \( X_i \) with \( (n-n_i) \) random dummy items.

5. [symmetric approximate ROT-online]
   (a) Using BF_0 as its input, \( P_0 \) performs \( \Pi^{Online}_{AppROT} \) with every other party to finish \( \Pi_{AppROT}s \) started on Step 2a. As a result, it receives \( t \) arrays \( M^i_1, P_i \) learns \( M^i \), where \( M^i_1 \) and \( M^i \) are \( N_{BF} \)-size arrays of \( \sigma \)-bit values.
   (b) Using BF_i as its input, every party \( P_i \) performs \( \Pi^{Online}_{AppROT} \) with \( P_0 \) to finish \( \Pi_{AppROT}s \) started on Step 2b. As a result, \( P_i \) learns \( M^i_1 \) and \( P_0 \) receives \( M^i \)'s, where \( M^i_1 \) and \( M^i \) are \( N_{BF} \)-size arrays of \( \sigma \)-bit values.
   (c) \( P_0 \) computes \( GBF^0 = \bigoplus_{i \in [t]} (M^i_1 \oplus M^i) \). Each \( P_i, i \in [t], \) computes \( GBF^i = M^i \oplus M^i_1 \).

6. [re-randomize GBFs] Each \( P_i \) locally re-randomizes its garbled Bloom filter GBF^i for items and corresponding codewords only from \( X_i \) (without dummy items) (Algorithm ReRandGBF B.1).

7. [secret-sharing of GBFs] Each \( P_i, i \in [t], \) locally computes
   \[
   GBF^i_s = GBF^i \bigoplus_{i \in [t] \setminus \{i\}} [S^i \oplus S^i] \]
   and sends GBF^i_s to \( P_0 \).

8. [reconstructing the GBF of the intersection] \( P_0 \) computes \( GBF^* = \bigoplus_{i \in [t]} GBF^i \bigoplus GBF^0 \). Recall that this corresponds to a GBF of the intersection with codewords 0 for all items in the intersection.

9. [output] For each \( x_{0j} \in X_0, \) \( P_0 \) outputs \( x_{0j} \) as a member of the intersection, if
   \[
   \bigoplus_{r \in \delta(x_0)} GBF^*[r] = 0. 
   \]

Figure 5: The PSImple Multiparty protocol
them. In addition, $P_i$ will still keep these codewords in the rerandomization process. Finally, as all GBFs are XORed by $P_0$, these all codeword will cancel out.

If $x$ is not in the intersection, then there exists a party $P_i$ for $i \in [t] \cup \{0\}$ whose set does not contain $x$. Thus, except for the overwhelmingly small probability of a false positive in the underlying BF (i.e., probability $p_{\text{False}}$), 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 $\ell$-long string. Hence, the probability that $x$ is in the output is $2^{-\ell}$, which is negligible.

A proof of the consistency appears in Appendix F.1.

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

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

3.4 Asymptotic Complexity

In Table 2, we compare the communication complexity of PSImple with that of the multiparty PSI protocols of [12, 16, 18, 23]. The distribution of the communication complexity in the offline and online phases is given in Table 3.

We observe that the overall communication complexity is asymptotically approximately the same as the protocol of [18], which is only semi-honestly secure. The communication complexity is slightly worse than the protocols of [16] and [12]. 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.\footnote{With a standard optimization of generating the secret shares from pre-shared random seeds (see Section 5). Without it, the communication complexity for $P_i$ is $O((n^2 + n\log(n))$, and the overall communication complexity is $O(t^2 n^2 + t n\sigma \log(n\sigma))$.} A detailed

<table>
<thead>
<tr>
<th>Protocol</th>
<th>Communication complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>overall</td>
</tr>
<tr>
<td>semi-honest security</td>
<td>$O(n\sigma)$</td>
</tr>
<tr>
<td>malicious security</td>
<td>$O(t^2 n\sigma)$</td>
</tr>
<tr>
<td>PSImple</td>
<td>$O(n\sigma + n\sigma \log(n\sigma))$</td>
</tr>
</tbody>
</table>

Table 2: Comparison of communication complexity.

<table>
<thead>
<tr>
<th>Party</th>
<th>offline</th>
<th>online</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P_0$</td>
<td>$O((n^2 + n\sigma \log(n\sigma))$</td>
<td>$O((n^2 + n\sigma \log(n\sigma))$</td>
</tr>
<tr>
<td>$P_i$</td>
<td>$O(n\sigma^2)$</td>
<td>$O(n\sigma (\log(n\sigma) + \sigma))$</td>
</tr>
</tbody>
</table>

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

4 Protocol Parameters

In this section, we revisit the parameter analysis of [36] for the parameters of $\Pi_{\text{AppRO}}$. We show that the size of the garbled Bloom filters and the number of required OTs can, in some cases, be reduced by 23-25% (see Table 1). These improved parameters can be used in our protocol, as well as in previous PSI protocols based on GBFs and cut-and-choose such as [36, 39].

The first difference from the analysis of [36] is the following: The number of OTs depends on the number of 1’s that would be necessary to build the Bloom filter of the receiver. For $n$ items in the input set of the receiver, with $k$ hash-functions, the upper bound of required 1’s is $nk$ (k indices per each of $n$ items), which is sufficient even if each item has a separate set of hash-indices. However, the probability of collisions is quite high, and the number of 1’s in a Bloom filter has a Poisson distribution with very low deviation. Thus, instead of requiring a sufficient number of 1’s after the cut-and-choose to build any Bloom filter, as done in [36], 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 [36] is technical: in [36], the sender chooses bits to check with the probability $p_{\text{chk}}$, whereas in our version of $\Pi_{\text{AppRO}}$, the size of the checked set is deterministic. Fixing the size of the checked
set simplifies the protocol instructions and the simulation in the security proof.

Below we give a brief explanation about the restrictions which allow us to build the optimization problem, and the algorithm for the calculating the parameters. Additionally, we give the optimal parameters for several input sizes, and compare them with the parameters used in [36].

**GBF parameters:**

\(N_{\text{OT}}\) – number of ROTs in \(\Pi_{\text{AppRO}}\);

\(N_{\text{BF}}\) – size of the Bloom filter of the receiver;

\(N_{\text{BF}}^1\) – number of ones that an honest receiver should have among \(N_{\text{OT}}\) choice bits;

\(k\) – number of Bloom filter hash functions;

\(N_{\text{cc}}\) – number of bits to choose for the cut-and-choose check;

\(N_{\text{maxones}}\) – the maximal number of 1’s among the \(N_{\text{cc}}\) 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 \(p_{\text{False}}\) is the probability of a false-positive in the Bloom filter of the receiver. The second condition requires that \(p_{\text{False}}\) is negligible, even if the receiver is malicious. I.e. that \(\Pr[p_{\text{False}} \geq 2^{-\sigma}] \leq 2^{-\lambda}.\)

Fixing \(n, \sigma\) and \(\lambda\), we have to set \(k, N_{\text{BF}}, N_{\text{cc}}, N_{\text{OT}}, N_{\text{maxones}}\) and \(N_{\text{BF}}^1\). As we have three conditions and six variables, three of parameters are free. It is reasonable to take \(k\) and \(N_{\text{BF}}\) free and find the values of the other parameters that minimize \(N_{\text{OT}}\), because the number of random OT is the heaviest part of the protocol.\(^6\)

We prove that for any positive \(n, \sigma\) and \(\lambda\) there exist \(k, N_{\text{BF}}, N_{\text{cc}}, N_{\text{OT}}, N_{\text{maxones}}\) and \(N_{\text{BF}}^1\) that meet the above requirements, and construct an algorithm to find the optimal parameter values. Based on the view of constrains, the feasible region of the parameters is bounded from below by \(k\) and \(N_{\text{BF}}\), and the minimum of \(N_{\text{OT}}\) is located near their minimum values. For reasons of the guaranteed existence of a solution for \(k > \sigma\) and that \(N_{\text{BF}} = O(nk)\), we heuristically adopted the search boundaries \(k_{\text{min}} = \sigma, k_{\text{max}} = 2\sigma, N_{\text{BF},\text{min}} = nk\) and \(N_{\text{BF},\text{max}} = 3nk\). The algorithm then works by going over all the possible values of \(N_{\text{BF}}\) and \(k\) in this region and taking the parameters which result in the minimal \(N_{\text{OT}}\). A formal description of the algorithm is presented in Figure 6. The full parameter analysis appears 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 [36] is given in Table 1.

![Algorithm for computing parameters for \(\Pi_{\text{AppRO}}\)](image)

We wrote our code in C++, using the LibOTe library [35] for the cryptographic primitives and the maliciously secure OT extension of Keller et al. [20]. The hash functions are computed using fixed-key AES and taking modulus.\(^7\) We

---

\(^6\)We note that for the online-phase, \(N_{\text{BF}}\) is more critical. However, trying to optimize \(N_{\text{BF}}\) results in a very poor \(N_{\text{OT}}\). More details can be found in Appendix C.

\(^7\)This restricts the input items’ domain to 120 bits, as it requires 8 bits for the hash function selection. Note also that this makes some assumptions on the randomness of AES, and that taking modulus \(N_{\text{BF}}\) already implies that this is not fully random. However, since \(N_{\text{BF}}\) is significantly smaller than the
separate the protocol into two phases: an offline phase, which can be run before the parties know their inputs, and an online phase, which is run after the parties know their inputs. In many scenarios, the parties can communicate much before they require to find the intersection. In such cases, it is often preferable that the online time is as short as possible, while the offline phase can be significantly longer. We note that currently the proof of security is not given in offline-online setting, but rather treats the protocol as a whole. We defer the adaption to the offline-online model to the full version of the paper.

We have made the following code optimizations: 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. As we executed the experiments on machines with only 2 cores (see below), we chose to use a single thread for each instance of AppROT, which implies that the total number of threads in $P$ we chose to use a single thread for each instance of AppROT, generating shares can be done locally. Additionally, we have moved memory allocation to the offline phase and reduced the required amount of memory by XORing results directly into the cumulative GBF on the fly. As we executed the experiments on machines with only 2 cores (see below), we chose to use a single thread for each instance of AppROT, which implies that the total number of threads in $P$.

Notice that the memory required by all the parties is linear in $N_{OT}$, statistical security $\lambda = 40$, and computational security $\sigma = 128$.

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

<table>
<thead>
<tr>
<th>Parameters</th>
<th>$n = 2^8$</th>
<th>$n = 2^{10}$</th>
<th>$n = 2^{14}$</th>
<th>$n = 2^{18}$</th>
<th>$n = 2^{20}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$k$</td>
<td>147</td>
<td>139</td>
<td>134</td>
<td>132</td>
<td>131</td>
</tr>
<tr>
<td>$N_{BF}$</td>
<td>64,733</td>
<td>229,055</td>
<td>851,085</td>
<td>3,253,782</td>
<td>12,660,342</td>
</tr>
<tr>
<td>$N_{OT}$</td>
<td>12,948,963</td>
<td>102,078,504</td>
<td>509,498,536</td>
<td>2,407,490,576</td>
<td>11,737,452,736</td>
</tr>
<tr>
<td>$N_{bf}$</td>
<td>3,627</td>
<td>37,282</td>
<td>324,408</td>
<td>1,297,632</td>
<td>5,390,496</td>
</tr>
</tbody>
</table>

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

As can be observed from the results, the running time of PSImple grows approximately linearly both in the number of parties and in the number of inputs. This is illustrated in Figure 7 for different number of parties.

![Figure 7: Total time for different number of parties](image)

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

<table>
<thead>
<tr>
<th>parties</th>
<th>$n = 2^8$</th>
<th>$n = 2^{10}$</th>
<th>$n = 2^{14}$</th>
<th>$n = 2^{18}$</th>
<th>$n = 2^{20}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>0.315</td>
<td>0.466</td>
<td>0.956</td>
<td>3.052</td>
<td>10.717</td>
</tr>
<tr>
<td>4</td>
<td>0.538</td>
<td>0.736</td>
<td>1.310</td>
<td>3.891</td>
<td>14.242</td>
</tr>
<tr>
<td>8</td>
<td>1.188</td>
<td>1.589</td>
<td>2.411</td>
<td>6.334</td>
<td>23.240</td>
</tr>
<tr>
<td>16</td>
<td>2.374</td>
<td>3.124</td>
<td>5.549</td>
<td>12.983</td>
<td>41.965</td>
</tr>
<tr>
<td>32</td>
<td>5.066</td>
<td>6.420</td>
<td>9.804</td>
<td>24.815</td>
<td>*</td>
</tr>
</tbody>
</table>

We next consider at the cost of the various steps of PSImple. An interesting aspect of the runtime is that, for a small number of parties (e.g., 2, 4), the main bottleneck is the rerandomization step. However, the runtime of the rerandomization step remains constant with the number of parties. As a result, for a small amount of parties, the runtime is dominated by the rerandomization (in the online phase), while for a large amount of parties the runtime is dominated by the OTs (in the offline phase). The computation complexity of the rerandomization algorithm ReRand is $O(nk)$. Thus, it is possible that for a small amount of parties, PSImple would run faster using a different set of parameters, in which the number of hash functions, $k$, is smaller at the cost of increasing the number of OTs. An interesting question is if the ReRand algorithm can be computed more efficiently using parallel computation, or that the protocol crashed due to insufficient memory.
We next compare PSImple’s runtime with the reported runtimes of IOP [18] (multiparty augmented semi-honest protocol), Zhang et al. [39] (multiparty, non-standard security model), and [36] (two-party malicious protocol).\footnote{Importantly, the runtime heavily depends on the machines and network used for the experiments. Therefore, in order to make a more fair comparison with these protocols, we plan to run them on the same machines as our protocol soon.} We chose to compare with these protocols as they are similarly based on GBFs. Note that a more appropriate comparison would be with a maliciously secure multiparty PSI protocol, but previous maliciously secure multiparty PSI protocols were not concretely efficient and, therefore, were not implemented.

**Comparison with IOP [18] and Zhang et al. [39].** In Tables 6 and 7 we compare PSImple’s runtime with the reported runtimes of the multiparty PSI protocols of [18] and [39], respectively. Recall that [18] 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 [39] is in a very non-standard security model, since it makes the assumption that two dedicated parties, $P_0$ and $P_1$, are not simultaneously corrupted. This relaxation of the security model allows them to have a significantly simpler protocol, which is insecure in the standard malicious model.

Surprisingly, despite the fact that PSImple achieves much stronger security guarantees, the runtime of PSImple in our experiments is not significantly slower, and in some cases even faster, than the reported runtimes of [18] and [39]. We attribute this to the fact that [18] wrote their code in Java, and that both [18] and [39] ran their experiments on slightly weaker testing platforms. Nevertheless, this shows that moving to maliciously secure using PSImple does not incur a very high penalty.

<table>
<thead>
<tr>
<th>parties</th>
<th>sec. model</th>
<th>n = 2^8</th>
<th>n = 2^12</th>
<th>n = 2^16</th>
</tr>
</thead>
<tbody>
<tr>
<td>[18]</td>
<td>semi-honest</td>
<td>2.08</td>
<td>25.03</td>
<td>91.48</td>
</tr>
<tr>
<td>PSImple</td>
<td>malicious</td>
<td>0.555</td>
<td>14.242</td>
<td>56.875</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[18]</td>
<td>semi-honest</td>
<td>2.29</td>
<td>38.53</td>
<td>195.51</td>
</tr>
<tr>
<td>PSImple</td>
<td>malicious</td>
<td>1.826</td>
<td>32.82</td>
<td>91.48</td>
</tr>
</tbody>
</table>

Table 6: Comparison of PSImple with [18] in total runtime.

<table>
<thead>
<tr>
<th>parties</th>
<th>sec. model</th>
<th>n = 2^8</th>
<th>n = 2^12</th>
<th>n = 2^16</th>
</tr>
</thead>
<tbody>
<tr>
<td>[39]</td>
<td>non-standard malicious</td>
<td>0.76</td>
<td>2.95</td>
<td>40.91</td>
</tr>
<tr>
<td>PSImple</td>
<td>malicious</td>
<td>0.558</td>
<td>1.310</td>
<td>14.242</td>
</tr>
</tbody>
</table>

Table 7: Comparison of PSImple with [39] in total runtime.

**Comparison with RR [36].** Recall that the maliciously secure protocol of [36] is limited to two-parties, while PSImple is intended as a multiparty protocol. As explained, directly extending [36] to the multiparty scenario would make the computation complexity grow exponentially with the number of parties, while PSImple only grows approx. linearly with the number of parties. Thus, even for modest parameters such as 4 parties and input size $n = 2^{10}$, we expect the direct multiparty extension of [36] to be barely practical, while this takes less than 10 seconds using PSImple. For a larger number of parties, we expect the direct multiparty extension of [36] to be completely impractical, while PSImple’s runtime grows linearly in the number of parties.

For the two-party scenario, PSImple requires two instances of $\Pi_{AppROT}$, while [36] requires only a single instance. Additionally, PSImple’s online phase requires both more computation, due to the additional rerandomization step, and more communication, since it requires two executions of $\Pi_{Online AppROT}$ and in the last communication round the entire GBF is sent, while [36] requires a single execution of $\Pi_{Online AppROT}$ and only the codewords are sent in the last round. Furthermore, [36] benchmarked their protocol using stronger machines.

For the above reasons, we expected PSImple to be outperformed by [36] in the 2-party scenario. Although our comparison in Table 8 confirms our expectation, we observe that the total runtime is not significantly slower than [36]. In contrast, the online time of PSImple, dominated by the rerandomization step, is significantly slower than the online time reported by [36].

<table>
<thead>
<tr>
<th>n = 2^8</th>
<th>n = 2^12</th>
<th>n = 2^16</th>
</tr>
</thead>
<tbody>
<tr>
<td>[36] (single thread)</td>
<td>0.2</td>
<td>0.9</td>
</tr>
<tr>
<td>[36] (4 threads)</td>
<td>0.17</td>
<td>0.63</td>
</tr>
<tr>
<td>PSImple (2 threads)</td>
<td>0.515</td>
<td>0.926</td>
</tr>
</tbody>
</table>

Table 8: Comparison of 2-party PSImple with [36].
References


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

Protocol [36] (Malicious-secure Two-Party PSI):

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

1. **[setup]** The parties perform a secure coin-tossing subprotocol to choose (seeds for) random Bloom filter hash functions h_1,...,h_k: \{0,1\}^* \rightarrow [N_{BF}].

2. **[random OTs]** Bob chooses a random string b = b_1,...,b_{N_{OT}} with an α fraction of 1s. Parties perform N_{OT} OTs of random messages (of length σ), with Alice as sender. In the i-th OT, Alice learns random strings m_{i,0}, m_{i,1} chosen by the functionality. Bob uses choice bit b_i and learns m_i^* = m_{i,b_i}.

3. **[cut-and-choose challenge]** Alice chooses a set C \subseteq [N_{OT}] by choosing each index with independent probability p_{chk}. She sends C to Bob. Bob aborts if |C| > N_{OT} - N_{BF}.

4. **[cut-and-choose response]** Bob computes the set R = \{i \in C | b_i = 0\} and sends R to Alice. To prove that he used choice bit 0 in the OTs indexed by R, Bob computes r^* = \bigoplus_{i \in R} m_i^* and sends it to Alice. Alice aborts if |C| - |R| > N_{maxones} or if r^* \neq \bigoplus_{i \in R} m_i^*.

5. **[permute unopened OTs]** Bob generates a Bloom filter BF containing his items Y. He chooses a random injective function π: [N_{BF}] \rightarrow ([N_{OT}] \setminus C) such that BF[i] = b_{π(i)}, and sends π to Alice.

6. **[randomized GBF]** For each item x in Alice’s input set, she computes a summary value

   \[ K_x = H \left( x \mid \bigoplus_{i \in h_x(x)} m_{π(i),1} \right), \]

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

7. **[output]** Bob outputs \{y \in Y | H(y) \mid \bigoplus_{i \in h_y(y)} m_{π(i),1}^* \in K\}.

Figure 9: Malicious-secure two-party PSI protocol of Rindal and Rosulek
B Algorithms for the Garbled Bloom Filter

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

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

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

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

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

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

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

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

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

Output: \(h_*(x)\) – set of indices of 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\((GBF, x, h_*(x), N_{BF}, \sigma)\)

Input:
\(x\) – item;
GBF – random garbled Bloom filter;
\(N_{BF}\) – length of GBF;

\(\sigma\)Note, that the probability of fail in this algorithm, that can appear in case finalInd=–1, is the probability of false-positive for one of \(n\) items. According (7),
\(p_{False} < 2^{-\sigma}\), so the union bound over all \(x \in X\) is \(n2^{-\sigma}\), which is still negligible in \(\sigma\).
σ – bitlength of string in GBF;
h_i(x) – set of hash-indices of x; \forall i \in h_i(x), i \in \mathbb{N}_{BF}.

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

Output: \(y\) – codeword for \(x\) in garbled Bloom filter GBF indexed by \(h_i(x)\).
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 [36], 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 $\Pi_{\text{AppROT}}^{\text{offline}}$, 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 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 of our analysis from the analysis of [36] is that instead of requiring a sufficient number of 1’s after the cut-and-choose to build any Bloom filter, as done in [36], we require this number to be sufficient to build almost all Bloom filters. This change significantly reduces the total number of required 1’s from the OT, and consequently, the total number of required OTs. Note that this change implies that there might not be enough 1’s to construct the GBF in the protocol, but this happens only with negligible probability.

The constraints on $p_{\text{False}}$ 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 $p_{\text{False}} = p^k$ [26], 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. [7] 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 [14] 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:**

$N_{\text{OT}}$ : number of ROTs in $\Pi_{\text{AppROT}}$;

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

$N_{1\text{OT}}$ : number of 1’s among $N_{\text{OT}}$ choice bits of the receiver;

$k$ : number of Bloom filter hash functions;

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

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

**Claim C.1.** By choosing the parameters of $\Pi_{\text{AppROT}}$ under the following constraints

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

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

We next give the formal details:
proof.
Consider every constraint one by one.

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

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

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

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

- The Azuma-Hoeffding inequality [26, 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 $Pr[|F - E(F)| \geq \varepsilon] \leq 2e^{-2\varepsilon^2/n}$. 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 [8] for hypergeometric distribution. Namely, for $HG(M, N, n)$ by $0 < t < pn$, we have $Pr[X \geq (p + t)n] \leq \left( \frac{p}{p+t} \right)^{p+t} \left( \frac{1-p}{1-p-t} \right)^{1-p-t} N^{p+t} \leq e^{-2t^2n}$.

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 $q = (1 - 1/N_{BF})^{nk} \approx e^{-\frac{n\lambda k}{2N_{BF}}}$ and to 1 is $p = 1 - q$ [26, p 116]. Using the Azuma-Hoeffding inequality [26, p 355], we get $Pr[|N_0 - E(N_0)| \geq \varepsilon] \leq e^{-\frac{\varepsilon^2}{2}}$. We require that the number of zeroes is not enough to build the Bloom filter with the negligible in $\lambda$ probability. Hence $e^{-\frac{\varepsilon^2}{2}} \leq 2^{-\lambda}$; solving this inequality, we get $\varepsilon \geq \sqrt{\frac{n\lambda k \ln 2}{2}}$. Therefore we should have at least $N_0 = \left[ N_{BF}q + \sqrt{\frac{n\lambda \ln 2}{2}} \right]$ zeroes after cut-and-choose.

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

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

   The number of ones in the cut-and-choose set is distributed hypergeometrically $HG(\alpha N_{OT}, N_{OT}, N_{cc})$. Using the V. Chvatal inequality for hypergeometric distribution [8], we get

   \[ Pr[X \geq \alpha(N_\Delta + N_{cc})] = Pr[X \geq (\alpha + \frac{\alpha N_{OT}}{N_{cc}})N_{cc}] \leq e^{-2\frac{(\alpha N_{OT})^2}{N_{cc}}} \leq 2^{-\lambda}. \]

   Hence we need to have $N_\Delta \geq \frac{1}{\alpha} \sqrt{\frac{N_{cc} \lambda \ln 2}{2}}$ extra bits in random OT.

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

22
2. \( Pr[p_{\text{False}} \geq 2^{-\sigma}] \leq 2^{-\lambda} \) with either honest or malicious receiver in \( \Pi_{\text{AppROT}} \).

Denote by \( N_{\text{rest}} \) 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 \( N_{\text{rest}} \) bits set to 1 and \( k \) hash-functions, that we denote as \( p_{\text{False}} \). The upper bound for it (in our conditions of the experiment) is obtained in Appendix D from [14]:

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

(6)

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

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

(7)

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

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

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

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

Hence \( \tilde{\delta} \geq \sqrt{\frac{\ln 2}{2N_{cc}}} \).

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

\[
N_{\text{maxones}} \leq \tilde{\alpha} N_{cc} \leq \frac{N_{BF} N_{cc}}{N + N_{\Delta}} 2^{-\frac{\sigma}{2}} \left[ 1 + \frac{k(k-1)}{2N_{BF}} \left( \frac{1}{\alpha} - 1 \right) \right]^{-1} - \frac{\sqrt{2N_{BF} \ln 2}}{2}. \]

Equality (4) follows.

3. An honest receiver passes the cut-and-choose check successfully with probability at least \( 1 - 2^{-\lambda} \).

Denote by \( N_{\text{maxhonest}} \) the maximal number of ones in the cut-and-choose set in the case of an honest receiver, and

\[
Pr[N_{\text{maxhonest}} \geq (\alpha + \tilde{\delta}) N_{cc}] \leq e^{-2\sigma N_{cc}} \leq 2^{-2\lambda}, \]

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

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

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

(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.

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

\[ \square \]

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

**Claim C.2.** For any choice of positive \( n, \sigma, \lambda \) there exist positive \( k, N_{BF}, N_{cc}, N_{OT}, N_{\text{maxones}}, N_{\text{maxhonest}} \) such that (2)-(5) hold.
Proof. Considering the asymptotic when $N_{BF} \to \infty$, we compute the following limits:

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

$$\lim_{N_{BF} \to \infty} \alpha = \lim_{N_{BF} \to \infty} \frac{N_{BF} \left(1 - e^{-\frac{nk}{N_{BF}}}\right) + \sqrt{nk \ln 2}}{N_{BF} + \sqrt{2nk \ln 2}} = \lim_{N_{BF} \to \infty} \frac{nk + \sqrt{nk \ln 2}}{N_{BF} + \sqrt{2nk \ln 2}} = \lim_{N_{BF} \to \infty} \frac{nk + \sqrt{nk \ln 2}}{N_{BF}} = 0.$$

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

After fixing $k$ and $N_{BF}$, the rest of the parameters can be computed directly, with the exception of $N_{cc}$, which is derived from Inequality (8). So, the question of the existence of suitable parameters is the question of existence of positive roots in the square equation $ax^2 - bx + c = 0$, where, taking $\min(\alpha, 1 - \alpha) = \alpha$, $a = \sqrt{\frac{k}{n^2}}$, $b = \frac{N_{BF}}{\sqrt{2n}} \left[1 + \frac{k(k - 1)}{2N_{BF}} \left(\frac{1}{\alpha} - 1\right)\right]^{-\frac{1}{2}} - \alpha N - \frac{k \ln 2}{\alpha}$, $c = \sqrt{2k \ln 2}$. 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),

$$\lim_{N_{BF} \to \infty} b = \lim_{N_{BF} \to \infty} \frac{N_{BF} \left(1 + \frac{k(k - 1)}{2N_{BF}} \left(\frac{1}{\alpha} - 1\right)\right)^{-\frac{1}{2}} - \alpha - \frac{\lambda \ln 2}{\alpha}}{\alpha} = \lim_{N_{BF} \to \infty} \frac{\lambda \ln 2}{\alpha} = \lim_{N_{BF} \to \infty} \frac{\lambda \ln 2}{\alpha} = \lim_{N_{BF} \to \infty} \frac{\lambda \ln 2}{\alpha} = \lim_{N_{BF} \to \infty} \frac{\lambda \ln 2}{\alpha} = \lim_{N_{BF} \to \infty} \frac{\lambda \ln 2}{\alpha}.$$

The asymptotic of $b$ is linear in $N_{BF}$. The value in outer square parentheses is constant by fixed $n$, $\lambda$, $k$, and $\sigma$. If $k > \sigma$, then $\left(\frac{\lambda \ln 2}{\alpha}\right)$ tends to 1 when $k$ grows, while $\frac{2\lambda \ln 2}{nk + \sqrt{2nk \ln 2}}$ 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 $k \geq k_1$:

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

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).

In the proof of Claim C.1 below, we use several tail inequalities for different distributions.

- The Azuma-Hoeffding inequality [26, 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 $Pr[|F - E(F)| \geq \varepsilon] \leq 2e^{-\frac{\varepsilon^2}{m}}$. In our case, the number of bins is $nk$, and we are interested only in the right tail of distribution.

- The tail inequality, obtained by V. Chvatal in [8] for hypergeometric distribution. Namely, for $HG(M, N, n)$ by $0 < t < pn$, we have $Pr[X \geq (p + t)n] \leq \left(\frac{p}{p + t}\right)^{p + t} \left(\frac{1 - p}{p + t}\right)^{1 - p + t} \leq e^{-2t^2n}$.

Using the relations proved in Claim C.1, we construct the algorithm in Figure 6 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.

Remark. For the online-phase, $N_{BF}$ is more critical. In Table 9 we give the optimal by $N_{BF}$ parameters, where the improvement in $N_{BF}$ is from 10.4% for $n = 2^8$, to 1.4% for $n = 2^{20}$ in comparison with the optimization of $N_{OT}$. However, achieving this improvement for $N_{BF}$ requires significantly more OTs (from 20.6% for $n = 2^8$ to 46.7% for $n = 2^{20}$) and increases the size of the cut-and-choose set (from 287% for $n = 2^8$ to 5793.9% for $n = 2^{20}$), which results in a large increase in the overall cost.
Table 9: Optimal (in $N_{BF}$) $\Pi_{AppROF}$ parameters for set size $n$, statistical security $\lambda = 40$, and computational security $\sigma = 128$.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>$n = 2^8$</th>
<th>$n = 2^{10}$</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>139</td>
<td>133</td>
<td>130</td>
<td>129</td>
<td>128</td>
<td>128</td>
<td></td>
</tr>
<tr>
<td>$N_{BF}$</td>
<td>57,993</td>
<td>210,014</td>
<td>797,706</td>
<td>3,107,680</td>
<td>12,265,989</td>
<td>48,735,894</td>
<td>194,288,832</td>
</tr>
<tr>
<td>$N_{OT}$</td>
<td>89,772</td>
<td>220,253</td>
<td>1,206,819</td>
<td>4,681,730</td>
<td>18,439,057</td>
<td>73,183,339</td>
<td>291,592,668</td>
</tr>
<tr>
<td>$N_{cc}$</td>
<td>28,994</td>
<td>104,963</td>
<td>398,851</td>
<td>1,553,817</td>
<td>6,132,901</td>
<td>24,367,378</td>
<td>97,143,998</td>
</tr>
<tr>
<td>$N_{maxones}$</td>
<td>13,960</td>
<td>51,323</td>
<td>196,635</td>
<td>771,384</td>
<td>3,063,672</td>
<td>12,145,310</td>
<td>48,495,359</td>
</tr>
</tbody>
</table>

D False-Positive Probability of a Bloom Filter

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

The second-order Taylor’s approximation of the false-positive probability in the Bloom filter, derived in [14], is:

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

(10)

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

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

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 [26], it follows that $p = \left[1 - \left(1 - \frac{1}{m}\right)^{kn}\right]$ and $E[X] = mp$. Also notice that $1 - \frac{1}{m-1} < 1 - \frac{1}{m}$. Thus, we can rewrite $\sigma^2_X$ as

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

(11)

From (10) and (11), we get

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

(12)

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

In this section we give further details on the complexity analysis of PSImple. In Tables 10 and 11 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 $\Pi_{App\text{RO}T}$, $t$ as a sender and $t$ as a receiver, while every other $P_i$ performs only two instances, one as a sender and one as a receiver.

The communication of $\Pi_{\text{Off line}}^{\text{App\text{RO}T}}$ is dominated by random OTs with communication complexity $O(n\sigma^2)$ – in order to compute $N_{\text{OT}}$ random OTs of length $\sigma = 128$ with statistical security $\lambda = 40$, the ROT-extension of Keller, Orsini, and Scholl [20] requires sending $\sigma N_{\text{OT}} + 10$KB. The communication of $\Pi_{\text{Online}}^{\text{App\text{RO}T}}$ consists mainly of sending/receiving the permutations of the unopened OTs; again, here $P_0$ performs $2t$ instances of $\Pi_{\text{Online}}^{\text{App\text{RO}T}}$, while every other $P_i$ only 2 instances of $\Pi_{\text{App\text{RO}T}}$. Apart from $\Pi_{\text{Online}}^{\text{App\text{RO}T}}$, in the online phase of PSImple, each $P_i$ is required to send its GBF to $P_0$. Table 10 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 Tables 2 and 3, with respect to $N_{\text{OT}} \approx N_{\text{BF}} = O(nk) = O(n\sigma)$ and $N_{cc} \ll N_{\text{BF}}$.

The main computational costs are summarized in Table 11. To compute $N_{\text{OT}}$ ROTs, the ROT-extension of Keller, Orsini, and Scholl [20] requires $2N_{\text{OT}} + 336$ hashes.

<table>
<thead>
<tr>
<th>Step</th>
<th>Party</th>
</tr>
</thead>
<tbody>
<tr>
<td>Offline-phase</td>
<td></td>
</tr>
<tr>
<td>Random OTs</td>
<td>$2N_{\text{OT}}\sigma$</td>
</tr>
<tr>
<td>Cut-and-choose challenge</td>
<td>$tN_c \log_2 N_{\text{OT}}$</td>
</tr>
<tr>
<td>Cut-and-choose response</td>
<td>$t(N_c \log_2 N_{\text{OT}} + \sigma)$</td>
</tr>
<tr>
<td>Online-phase</td>
<td></td>
</tr>
<tr>
<td>Permutation</td>
<td>$tN_{\text{BF}} \log_2 N_{\text{OT}}$</td>
</tr>
<tr>
<td>Sending/receiving GBF</td>
<td>$tN_{\text{BF}}\sigma$</td>
</tr>
</tbody>
</table>

Table 10: Number of sent and received bits

<table>
<thead>
<tr>
<th>Computation</th>
<th>Party</th>
</tr>
</thead>
<tbody>
<tr>
<td>Offline-phase</td>
<td></td>
</tr>
<tr>
<td>Hashes for random OTs</td>
<td>$2N_{\text{OT}}$</td>
</tr>
<tr>
<td>PRGs of $\sigma$-bit strings to compute secret shares</td>
<td>$-\quad (t-1)N_{\text{BF}} - n$</td>
</tr>
<tr>
<td>Online-phase</td>
<td></td>
</tr>
<tr>
<td>Hashes for the Bloom filter</td>
<td>$nk$</td>
</tr>
<tr>
<td>Performed permutations</td>
<td>$2t$</td>
</tr>
<tr>
<td>XORs of $\sigma$-bit strings for codewords and GBFs</td>
<td>$nk + 2nk$</td>
</tr>
<tr>
<td>PRGs of $\sigma$-bit strings for rerandomization of GBFs</td>
<td>$-\quad N_{\text{BF}} - n$</td>
</tr>
<tr>
<td>XORs of $\sigma$-bit strings for the intersection</td>
<td>$2N_{\text{BF}} + nk$</td>
</tr>
</tbody>
</table>

Table 11: Number of performed operations
F Security Proof

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

F.1 Consistency of PSImple

Consistency follows from next: consider GBF* that P0 learns on the step 8.

\[ \text{GBF}^* = \bigoplus_{i \in [t]} \text{GBF}^i \bigoplus \text{GBF}^0 = \bigoplus_{i \in [t]} \text{GBF}^i \bigoplus \left( \sum_{i \in [t]} \left( M_i^* \oplus \hat{M}_i \right) \right) = \bigoplus_{i \in [t]} \text{GBF}^i \bigoplus \left( M_i^* \oplus \hat{M}_i \right). \]

GBF* is the re-randomised \( M^* \oplus \hat{M}_i \), and the re-randomization performed by \( P_i \) doesn’t affect codewords of its items, therefore for any \( x_{ij} \in X_i \) the codeword \( y_{ij} = \oplus_{s \in h_i(a)} \text{GBF}^*[s] = \oplus_{s \in h_i(a)} (M_i^*[s] \oplus \hat{M}_i[s]) \) = \( \oplus_{s \in h_i(a)} (M_i^*[s] \oplus \hat{M}_i[s]) \).

Hence, for every item \( x \in \bigcap_{i \in [t]} X_i \) with codewords \( y_i \), we have

\[ \bigoplus_{s \in h_i(a)} \text{GBF}^*[s] = \bigoplus_{s \in h_i(a)} \left( \text{GBF}^*[s] \bigoplus \left( M_i^*[s] \oplus \hat{M}_i[s] \right) \right) = \bigoplus_{i \in [t]} (y_i \oplus y_i) = 0. \]

F.2 Security

**Notation.** Let \( W = \{w_1, \ldots, w_n\} \) be a set of \( n \) elements. A partial injective and onto mapping \( \xi : A \rightarrow B \) where \( |B| = k \), and \( |A| = n \) is called a \( k \)-permutation from \( n \) [6, 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 \( I \) of \( B \), we denote by \( J = \xi(I) \) the ordering. For indices we write \( j = \xi(i) \) if we use the permutation to define the mapping.

**Definition 2.** 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 \( \mu(x) < 1/p(x) \).

**Definition 3.** We say that two distribution ensembles \( \{X(\kappa, a)\}_{\kappa \in \mathbb{N}, a \in \{0,1\}^*} \) and \( \{Y(\kappa, a)\}_{\kappa \in \mathbb{N}, a \in \{0,1\}^*} \) are statistically close, denoted \( \{X(\kappa, a)\} \approx \{Y(\kappa, a)\} \), if for every non-uniform distinguisher \( D \) there exists a negligible function \( \mu \) such that for all \( a \) and \( \kappa \), \( |Pr[D(X(\kappa, a)) = 1] - Pr[D(Y(\kappa, a)) = 1]| \leq \mu(\kappa) \).

**Model.** We prove our results in the standard UC model [5], assuming private authenticated channels between the parties (in fact, authentication of sender’s identity is already guaranteed due to the fact that the adversary may only reorder messages, delay or delete messages sent between a pair of parties but not modify them). The latter can be modeled by assuming communication via ideal channel functionalities that allow for the suitable adversarial behavior (allowing interventions as above, but adding secrecy of message content), see [5] for more details - the following is a brief recap of the setting, which is identical to the standard setting of [5], except of assuming channels as above, instead of the ‘bare bones’ network.

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

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

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

### Ideal World - evaluating a functionality $F$. For our purposes, the setting is conceptually similar to the ideal world in the stand alone setting \cite{13}. In particular, the distinguishing entity (i.e., $Z$) cannot observe the content of messages between the ideal functionality $F$ 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 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 $S$ (a simulator), such that for every nonuniform polynomially bounded environment $Z$ running in the presence of a dummy adversary $A$, corrupting parties at the outset of the protocol (before sending or receiving any other messages via $A$),

\[
\text{Ideal}_{f, Z, S} \approx \text{Real}_{\Pi, Z}.
\]

Similarly, for statistical and perfect security, the indistinguishability notion above is $\approx_s$ and $\approx_p$ respectively.\footnote{Note that the above requirement is only for real executions which terminate, in the sense that all parties wrote some value to the output tape of $Z$. Nothing is guaranteed before the execution has terminated.}

**Definition 5.** A protocol $\Pi$ is said to realize $F$ with type security with abort (type is either computational, statistical, or perfect) if it type-securely realizes $F'$, defined exactly as $F$, but it additionally allows the adversary to send $\bot$ 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 \cite{13, Chapter 7} and \cite{5}.\footnote{The standard notion for statistical and perfect security allows the simulator run in polynomial time in the runtime of $A$, and does not require that $A$ and $Z$ 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 $A, Z$ to be polynomially bounded. In fact, we could handle somewhat super-polynomial adversaries.}

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

Parties in our $PSImple$ protocol use $\Pi_{\mathcal{A}pp\mathcal{R}OT}$ – approximate $K$-out-of-$N$ random OT protocol to obtain garbled Bloom filters for the Bloom filter of length $N = N_{BF}$ consisting of $K$ 1’s. We give this protocol in $\mathcal{F}_{\mathcal{R}OT}^N$-hybrid model in Figure 10. To make the security proof clearer, we explicitly define the default values for the cut-and-choose challenge, and the $\bot$- and "continue"-replies, which are omitted in the main text. The oblivious transfer functionality we use in our security setting is $\mathcal{F}_{\mathcal{R}OT}^N$ (Fig. 2) with $N = N_{\mathcal{R}OT}$ parallel instances of 1-out-of-2 random OT.

**Lemma 1.** The protocol $\Pi_{\mathcal{A}pp\mathcal{R}OT}$ realizes the functionality $\mathcal{F}_{\mathcal{A}pp\mathcal{R}OT}$ with statistical UC-security with abort in the presence against static malicious (non-uniform polynomial-time) adversaries in the $\mathcal{F}_{\mathcal{R}OT}^N$-hybrid model.\footnote{$F$ is a specific kind of a generalized $F$ functionalities, allowing for extra ‘abort after seeing output’ capabilities for the adversary.}

**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,
Protocol \( \Pi_{\text{AppRO}T} \) in \( \mathcal{F}_{\text{RO}T}^{\sigma,N} \)-hybrid model

**Parties:** Sender, Receiver.

**Parameters:** \( \sigma \) – length of the OT strings (computational security parameter); \( \lambda \) – statistical security parameter; 
\( N = N_{\text{BF}} \) is the number of OTs required; \( N_{\text{OT}} > N \) is the number of random OTs to generate; 
\( N_{\text{OT}}, N_{\text{cc}}, N_{\text{maxones}} \) are parameters of cut-and-choose described in Section 4.

**K** is the number of 1’s in Bloom filter of the receiver.

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

1. **[random OTs]** The sender and the receiver call \( \mathcal{F}_{\text{RO}T}^{\sigma,N} \) performing \( N_{\text{OT}} \) random OTs in parallel. The receiver chooses requests \( c_1, \ldots, c_{N_{\text{OT}}} \) with \( N_{\text{OR}} \) ones among them, and \( N_{\text{OT}} - N_{\text{OR}} \) zeros (randomly permuted). As a result, in the 1st step, the sender learns random strings \( m_{j0}, m_{j1} \) (of length \( \sigma \)). Receiver uses choice bit \( c_j \) and learns \( m_{j0} = m_{j1} \).

2. **[cut-and-choose challenge]** The simulator chooses set \( C \subseteq [N_{\text{OT}}] \) of size \( N_{\text{cc}} \) uniformly random and sends \( C \) to the receiver.

3. **[cut-and-choose response]** The receiver checks if \( |C| = N_{\text{cc}} \); if \( |C| > N_{\text{cc}} \), then truncates it, if \( |C| < N_{\text{cc}} \), then adds indices by default (for example, 1, 2, \ldots). Receiver computes and sends to the sender the set \( R = \{ j \in C \} \). To prove that he used choice bit 0 in the OTs indexed by \( R \), it also sends \( r^* = \bigoplus_{j \in R} m_{j0} \). The sender replies with \( \perp \) if \( |C| - |R| > N_{\text{maxones}} \) or if \( r^* \neq \bigoplus_{j \in R} m_{j0} \), and with "continue" otherwise.

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

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

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

Figure 10: Protocol \( \Pi_{\text{AppRO}T} \) in \( \mathcal{F}_{\text{RO}T}^{\sigma,N} \)-hybrid model

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 \( \mathcal{F}_{\text{AppRO}T} \), 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 \( \mathcal{F}_{\text{RO}T}^{\sigma,N} \) and obtaining the input \( c_1, c_2, \ldots, c_{N_{\text{OT}}} \) from \( Z \), \( S \) randomly chooses a set \( C \subseteq [N_{\text{OT}}] \), where \( |C| = N_{\text{cc}} \), and computes

\[
K'' = \sum_{i \in [N_{\text{OT}}] \setminus C} c_i.
\]

Then \( S \) passes \( K'' \) as an input of the receiver to \( \mathcal{F}_{\text{AppRO}T} \) and receives either \( M' = \{ m'_{1}, m'_{2}, \ldots, m'_{K''} \} \) or \( \perp \).

If the simulator received \( \perp \) from \( \mathcal{F}_{\text{AppRO}T} \), it passes to \( Z \) \( \{ m_{j\pi(j)} \} \) of uniformly random \( \sigma \)-bit strings as the output of \( \mathcal{F}_{\text{RO}T}^{\sigma,N} \) of the 2nd step of the protocol, sends it to \( C \) as the cut-and-choose request and, upon getting the answer, passes \( Z \) \( \perp \) as sender’s reply, appends \( \{ m_{j\pi(j)} \} \) to \( N_{\text{OT}} \), and halts.

If \( S \) receives \( M' \), for any \( j \in [N_{\text{OT}}] \setminus C \), with \( c_j = 1 \) it computes

\[
\psi(j) = \sum_{i \in [N_{\text{OT}}] \setminus C, i \leq j} c_i
\]

and the function \( \phi : [K''] \rightarrow [N_{\text{OT}}] \setminus C \) as the inverse of \( \psi \). I.e., \( \phi(i) = j \), whenever \( \psi(j) = i \) for \( i \in [K''] \) or \( j \in [N_{\text{OT}}] \setminus C \) such that \( c_j = 1 \). The simulator constructs the injective mapping for any \( j \in [N_{\text{OT}}] \) it constructs \( m_{j\pi(j)} = m'_{\phi(j)}, \) if \( j \in [N_{\text{OT}}] \setminus C \) and \( c_j = 1 \), or \( m_{j\pi(j)} \leftarrow \{0,1\}^\sigma \), otherwise. Simulator gives \( \{ m_{j\pi(j)} \} \) to \( Z \) as the output of \( \mathcal{F}_{\text{RO}T}^{\sigma,N} \) in 1st step.

The simulator sends \( C \) to \( Z \) as the message of 2nd step of the protocol.

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

\[18\]As mentioned above, \( \mathcal{R} \) is fixed to just relays messages from \( Z \) to the parties and back. Intuitively, \( S \) attempts to do the same.
receiver and sends $Z \perp$ as sender’s round-3 message. If the receiver passes the check, then sends him Continue and waits for the permutation.

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

If yes, then computes $I' = \left( \begin{array}{cccc} i_1 & i_2 & \ldots & i_{K'} \\ \pi(\phi(i_1)) & \pi(\phi(i_2)) & \ldots & \pi(\phi(i_{K'})) \end{array} \right)$ 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 $1^k$, $k$, $N$ 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 $S$ to send $\mathcal{F}_{\text{ROT}}^{\sigma,N}$ the set $Q$ of the requests to $\mathcal{F}_{\text{ROT}}^{\sigma,N}$, where $Q' \subseteq Q$ is the set of 1-requests, and the rest are 0-requests. It receives a sequence $m_j$, $(j \in [N_{\text{OT}}])$ of random strings in response (by specification of $\mathcal{F}_{\text{ROT}}^{\sigma,N}$). By definition of $S$, 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 $m_j$, $S$ picks $C$ and sends $K''$ to the ideal functionality, where $K'' = |Q' \cap ([N_{\text{OT}}] \setminus C)|$. The emulated $m_j$’s at locations $j \in Q' \cap ([N_{\text{OT}}] \setminus C)$ are taken from the functionality’s step-1 output to receiver if it does not equal $\perp$, and random independent values picked by $S$ otherwise. In both cases, the other values $S$ sends emulating replies of $\mathcal{F}_{\text{ROT}}^{\sigma,N}$ are random values independent of all others. Now, in the real world, the sender chooses $C$, and either

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

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 $|C \setminus R| > N_{\text{maxones}}$ 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 $K''$ as the number of 1-OT requests in $[N_{\text{OT}}] \setminus C$ on behalf of the receiver in step 1 (of the adversary’s interaction with the functionality). With our choice of parameters, according Claim C.1, 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 $\perp$ 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 $2^{-\sigma} + \text{neg}(\lambda)$.

$$\text{Ideal}_{\mathcal{F},Z,S} \approx \tilde{=} \text{Real}_{\text{I},Z} \approx (D, \perp). \tag{14}$$

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

Now, consider the case when (13) is not satisfied in the real world. If $Z$ sends $R^*$ (which differs from $R$ induced by its $\mathcal{F}_{\text{ROT}}^{\sigma,N}$’s inputs) so that (13) 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 $\perp$ in step 2 as receiver’s input, and replies with $\perp$ to both parties, $S$ again appends $\perp$ as sender’s message to the simulated view. Thus, if (14) holds with 0-error (in particular, over the entire support of $\text{Real}_{\text{I},Z}$, the sender’s output is $\perp$). 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 $m_{j1}$ it picked previously according to $\pi$. In particular, the $m_{j1}$’s for the $K'$ $j$’s for which $\pi(j) \in Q'$ are also output to the sender at positions $\pi(j)$, and all other $m_{j1}$’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 ensures the sender’s output at positions $\pi(j)$ for $j$ with $\pi(j) \in Q'$, equal the $m_{j1}$ at this position from the receiver’s view. The rest are random independent values, as initially generated by the functionality. We conclude that in this latter case $\text{Ideal}_{\mathcal{F},Z,S} \approx \text{Real}_{\text{I},Z}$ with 0-error.

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

30
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.

1. In the first step of the protocol, $S$ calls the ideal functionality $\mathcal{F}_{\text{AppROT}}$ (with no input on behalf of the sender) and receives $M' = \{m_1', m_2', \ldots, m_{|\mathcal{M}|}'\}$. Then samples the uniformly random $M_0 = \{m_j\}_j \in [N_{\text{OT}}]$ and computes $M_1 = \{m_j\}_j \in [N_{\text{OT}}] = M''$.

   It gives $M_0$ and $M_1$ to $Z$ as the sender’s reply from emulated $g_S^{\sigma,N}$.

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

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

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

Proof of security in face of a corrupted sender. Consider an environment $Z$ running on some fixed public parameters $1^\sigma, 1^\lambda$, $k, N$. 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 $S$) receives from $g_{\mathcal{S}}^{\sigma,N}$ two sets $M_0 = \{m_j\}_j \in [N_{\text{OT}}]$ and $M_1 = \{m_j\}_j \in [N_{\text{OT}}]$, whose distributions are identical and independent on $x$ in both real and ideal (from $g_{\mathcal{S}}^{\sigma,N}$’s emulated by $S$) worlds by the construction of the simulator.

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

   The environment $Z$ responds with either $\perp$ or Continue, which it chooses basing on his view $(x, M_0, M, R, r^s)$, 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 $\pi: [N_{\text{OT}}] \setminus C \rightarrow [N]$. This permutation is random and has the same distribution in both protocol and simulation. In both worlds it is constructed so that it places $m_j$’s over input indices of the receiver $j$ to the same positions in outputs of the sender and of the receiver.

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

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

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

After $g_{\mathcal{S}}^{\sigma,N}$, the sender has $(m_{11}, \ldots, m_{1N_{\text{OT}}})$ and $(m_{10}, \ldots, m_{N_{\text{OT}}})$. The receiver has $(m_{c1}, \ldots, m_{cN_{\text{OT}}})$. Applying the permutation $\pi$ to $(m_{11}, \ldots, m_{1N_{\text{OT}}})$ and $(m_{c1}, \ldots, m_{cN_{\text{OT}}})$ accordingly, the sender gets $M = (m_{\pi(i1)}, \ldots, m_{\pi(i1)})$, and the receiver gets $M_* = (m_{\pi(i1)c_{\pi(i1)}}, \ldots, m_{\pi(iN)c_{\pi(iN)}}) = (m_{\pi(i1)}B[1], \ldots, m_{\pi(iN)}B[N])$, where $i_1 = \min([N_{\text{OT}}] \setminus C)$, $i_N = \max([N_{\text{OT}}] \setminus C)$. Thus, the elements of $M$ and $M_*$ at the indices $i$ such that $B[i] = 1$ collude, and at the other indices differ. As the set of indices $i$ such that $B[i] = 1, i \in N$ defines $I$ – the input set of the honest receiver to $\mathcal{F}_{\text{AppROT}}$, the receiver has the values from the sender's output set at indices from $I$, as described by the functionality $\mathcal{F}_{\text{AppROT}}$.

F.4 PSI protocol in the hybrid model

Figure 11 describes the PSI simple protocol in random oracle, $\mathcal{F}_{\text{AppROT}}$-hybrid model. Note that as the hash functions are modeled by the random oracle, the coin-tossing step for hash-seed agreement (Step 1 in Figure 5) is omitted in Figure 11. Additionally, the ideal functionality $\mathcal{F}_{\text{AppROT}}$ is not separated into offline and online phases. Furthermore, we need to add a padding after $\mathcal{F}_{\text{AppROT}}$, 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. For clarity, in Figure 11 we explicitly describe all the $\perp$-replies that parties can send (as we consider security with abort and asynchronous execution).
Protocol of Malicious-secure Multiparty PSI $\Pi_{MPSI}$ in the $\mathcal{F}_{AppROT}$-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;
- $D$ - a domain of input items;

Inputs: $P_i$ inputs $X_i = \{x_{i1}, x_{i2}, \ldots, x_{in_i}\}, n_i \leq n$ – the set of items from $D (i \in \{0, \ldots, i\})$.

Offline Phase:

1. [(R0) random shares] Each $P_i, i \in [\ell]$, sends $S^0 = (s^0_1, \ldots, s^0_{N_{BF}})$ to any $P_l, l \in [\ell] \setminus \{i\}$, where $s^0_t \leftarrow \{0, 1\}^\sigma, r \in [N_{BF}]$.

Online Phase:

2. [(R1) compute Bloom filters] $P_i, (i \in [\ell] \cup \{0\})$ computes Bloom filter $BF_i$ of its items from $X_i$. If $n_i < n$, then $P_i$ computes the Bloom filter of the joint set $X_i$ with $(n - n_i)$ random dummy items.

3. [(R1) symmetric approximate ROTs] Parties perform in parallel:

   (a) Using $BF_0$’s 1’s indices set $J$ as input, $P_0$ calls $|J|$-out-of-$N_{BF} \mathcal{F}_{AppROT}$ as the receiver with each of the other parties $P_i (i \in [\ell])$. As a result, it receives $t$ sets of string $M'_i[j]$ for each $j \in J$.

   (b) Using $BF_i$ 1’s indices set $J$ as input, each $P_i (i \in [\ell])$ calls $|J|$-out-of-$N_{BF} \mathcal{F}_{AppROT}$ as the receiver with $P_0$. As a result, $P_i$ learns $M'_i[j]$ for each $j \in J$, and $P_0$ receives $M'$. $P_i$ sets $M'_i[j] = 0$ for $j \in [N_{BF}] \setminus J$.

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

5. [(R2) cumulative GBFs of $P_i$s] If $P_i (i \in [\ell])$ did not receive ⊥ from $\mathcal{F}_{AppROT}$, it computes and sends to $P_0$ the cumulative garbled Bloom filter:

   $$\text{GBF}^* = \text{GBF} \bigoplus \left[ t \in [\ell] \setminus \{i\} \right] \left[ \text{GBF}^t \oplus S^t \right].$$

   Else it sends ⊥.

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

7. [(R2) output] If $P_0$ did not receive ⊥ from $\mathcal{F}_{AppROT}$ or from $P_i$ in the previous step, it outputs $x_{0j}$ as a member of the intersection, if

   $$\bigoplus_{r \in h_i(x_0)} \text{GBF}^*[r] = 0, j \in [n_0].$$

   Else it outputs ⊥.

Figure 11: The PSiImpl multiparty protocol in the $\mathcal{F}_{AppROT}$-hybrid model
In Lemma 2 and consequently in Theorem 1 we require a non-uniform polynomial-time adversary in sense of polynomially-bounded requests to the Random Oracle. This follows from the next: the union bound of the probability of having at least one false-positive result over \(|Q|\) requests is \(|Q|p_{\text{False}} < |Q|2^{-\sigma}.\) 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 1.

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

**Proof.** In our protocol and functionality, we take \(n'\) such that for the Bloom filter consisting \(n'\) or less elements, \(p_{\text{False}} \leq 2^{-\sigma},\) and for the Bloom filter with \(n' + 1\) and more elements, \(p_{\text{False}} > 2^{-\sigma}.\) It means, that the malicious receiver in \(F_{\text{ROT}}\) receives \(\bot\) from the first step of \(F_{\text{MPSI}}\) functionality if and only if its effective Bloom filter consists more than \(n'\) items.

Consider the case when evaluating party \(P_0\) is honest, and some subset of other parties \(I \subseteq \{P_1, ..., P_t\}\) are corrupt.

**Simulator description.** The simulator \(S,\) once activated by \(Z,\) emulates \(F_{\text{MPSI}}\) towards \(F_{\text{ROT}}\). 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 (preprocessing), \(S\) sends to \(Z\) uniformly random shares \(S^{\mathrm{il}_i}\) as honest \(P_i\)'s would do according the protocol, to any corrupt party \(P_i \in I,\) and learns \(S^{\mathrm{il}_i}\) from \(Z.\)

In step 2, \(S\) make queries \(Q_i = \{q_j | j \in [n_i]\}\) on behalf of corrupt parties \(P_i \in I\) as requested by \(Z\) (where \(n_i \geq n\) is polynomially bounded, as \(Z\) is), to the random oracle to compute Bloom filter’s hash-indices.\(^\text{19}\) Denote \(Q = \bigcup_{P_i \in I} Q_i\) - the joint query set of corrupt parties.

In step 3, \(S\) plays \(F_{\text{MPSI}}\) functionality towards \(Z\) for any corrupt party. Once both inputs of \(F_{\text{MPSI}}\) have been requested by \(Z\) to be delivered:

- For \(P_i\) acting as a corrupt sender, the simulator samples uniformly at random \(M^{\mathrm{il}_i}\) of length \(N_{\text{OT}}\) and gives as the first part of the output to \(Z.\) Upon receiving from \(P_i\) either \(\bot\) or set \(C,\) gives to \(Z\) for \(P_i\) the vector \(M'\) or \(\emptyset.\)^\text{20}

- For any \(P_i\) acting as a corrupt receiver, upon receiving \(K^{\mathrm{il}_i}\) the simulator samples uniformly at random and gives \(M'\) of length \(K^{\mathrm{il}_i},\) or gives \(\bot,\) then the simulator receives set of indices \(J_i\) (and then can extract the effective Bloom filter \(B_F)\) or \(\bot.\)

First, all of them receive \(M^{\mathrm{il}_i}\)'s, and then each of them in its turn sends its inputs to \(F_{\text{MPSI}}\)’s.

The simulator remembers each of the \(M'\)'s and computes \(M'\)’s (from \(M'\)'s and \(J_i\) as \(F_{\text{MPSI}}\) would compute \(M\) in the case of corrupt receiver) for all \(P_i \in I,\) and extracts a set of Bloom filters \(B_F\) for any \(P_i\) (the second string of the mapping \(J_i\) is the sequence of \(1\)'s indices of \(B_F\)), if all the calls to emulated \(F_{\text{MPSI}}\) are completed successfully (without \(\bot\)’s).

In step 4, parties have no interaction, so \(S\) does nothing.

At the 5th step, once all round-1 and round-2 executions have completed, \(S\) observes \(B_F\)'s or \(\bot\)s sent by \(Z\) on behalf of corrupt \(P_i \in I.\)

- If there were no \(\bot\)’s as an outputs of the simulated \(F_{\text{MPSI}}\)'s or as the messages of the 5th step, \(S\) computes the sum \(\bigoplus_{P_i \in I} GBF_{P_i} = \bigoplus_{P_i \in I} (S^{\mathrm{il}_i} \oplus S^{\mathrm{il}_i'}).\) Now \(S\) can subtract all the secret shares sent and received to corrupt parties on behalf of honest and vice versa:

\[
GBF_I = GBF_I^{\ast} \bigoplus_{P_i \in I} \bigoplus_{P_i \in P \setminus \{I, P_0\}} (S^{\mathrm{il}_i} \oplus S^{\mathrm{il}_i'}). 
\]

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

\[
\hat{X}_I = \left\{ q \in Q | \bigoplus_{r \in B_F(q)} GBF_{P_i} \right\} \bigoplus_{P_i \in I} \left( M'_{P_i} \right) \bigoplus_{P_i \in I} \left( S^{\mathrm{il}_i} \oplus S^{\mathrm{il}_i'} \right),
\]

sends it to the ideal functionality, and receives either \(\hat{X}\) or \(\bot\) as the output of \(F_{\text{MPSI}}.\)

- Else, the simulator sends the effective input of the adversary \(\bot\) to the ideal functionality and receives \(\bot\) as its output.

\(^{19}\)For the simplicity, we suppose that \(Z\) makes queries to the random oracle right before the computation of the Bloom filter. After \(\text{R1},\) when all the \(F_{\text{MPSI}}\)'s done, the adversary can also make queries, but the probability of the new item is in the existing Bloom filter is \(p_{\text{False}},\) which is negligible.

\(^{20}\)Recall, that according the \(F_{\text{ROT}}\) specification, \(M = \psi_i(M^{\mathrm{il}}),\) where \(\psi_i : [N_{\text{OT}}] \rightarrow [N_{\text{BF}}]\) is the uniformly random \(N_{\text{BF}}\)-permutation.
Simulator Analysis. Consider an environment $Z$ running on some fixed public parameters $1^\sigma$, $1^\xi$, $t, n, k, N_{\text{BF}}$. We assume first that all parties 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 $X = \{X_i\}_{i \in \mathbb{P}^l \cup \{0\}}$ — inputs of honest parties. We prove indistinguishability by induction on the message graph of $Z$ - sent messages to the various parties, and to $f_{\text{AppROT}}$ 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). 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, step 1 any value sent and received by an honest party from $Z$, or sent from an honest party to $Z$ (a random share of 0) are identically distributed.

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

Let us compare the output distribution of $P_0$. In the ideal world, $H = X \cap \left( \bigcap_{P_i \in \mathbb{P}^l \cup \{0\}} X_i \right)$ is the output of $P_0$, or $\perp$ if the simulator sent $\perp$ to the ideal functionality.

In case $S$ did not send $\perp$ to the ideal functionality, $H$ is a subset of the real-world output of $P_0$, as the honest parties act honestly, and the contribution of the malicious parties does not ‘spoil’ the equality verified, for each of the items in $H$ that $P_0$ checks the condition in step 7 for (GBF re-randomization in step 5 by honest parties does not take place in the ideal world, but does not affect their codewords $y_j$’s, and thus does not affect the condition in step 7). Now, malicious parties may have chosen 1-items in their $f_{\text{AppROT}}$ executions, at locations outside of the query set $Q$. However, then they either query too many 1’s in that $f_{\text{AppROT}}$ execution, in which case, in the ideal world as well, $S$ notices it, and sends $\perp$ 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 $H_1 = \bigcap_{P_i \in \mathbb{P}^l \cup \{0\}} X_i$ with the probability at most $p_{\text{False}}$ (for instance, by using the received $M_i \oplus \hat{M}^i$ 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 $P_i$, adding it, and the result follow. By union bound, adding an element in $H_1$ by $P_i$ occurs with probability $\leq n \cdot p_{\text{False}}$. 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 $P_i \in I$ did not learn $M^i[j]$. The probability of $P_0$ adding $x_0$ to the output due to passing verification in the step 7 is at most $2^{-\sigma}$, which is the probability of $P_i$ guessing $M^i[j]$ by the adversary.

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

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

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: either shares picked in step 1, or $f_{\text{AppROT}}$ replies from step 3. 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) $I \subseteq \{P_1, \ldots, P_t\}$ 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 $f_{\text{AppROT}}$ 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 step 1 (preprocessing), $S$ sends to corrupt parties $P_i \in I$ uniformly random values $S^{\mathbb{P}^l}_i$, as honest $P_i$s would do, and gets $S^{\mathbb{P}^l}_i$ to be delivered by corrupted parties $P_i$ from $Z$.

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

\footnote{In fact, they could make them at the later round 2 as well, for that particular party.}
In step 3, the simulator plays $\mathcal{F}_{AppROT}$ functionality for $P_0$ in its interaction with any honest $P_i$. Simulating its outputs as follows.

- for $P_0$ acting as a corrupt receiver in the simulated interaction with any honest $P_i$, upon receiving $K''$ the simulator samples $M''$ uniformly random of length $K''$, or $\perp$ as $\mathcal{F}_{AppROT}$ would. then the simulator receives set of indices $I_i$ (and then can extract the effective Bloom filter $BF_0$) or $\perp$;
- for $P_0$ acting as a corrupt sender in the simulated interaction with any honest $P_i$, the simulator samples uniformly at random $M''$ of length $N_{OT}$ and gives as the first part of the output. Upon receiving from $P_0$ either $\perp$ or set $C$, gives to $P_0$ the vector $M''$ or $\emptyset$.

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

If the simulated $\mathcal{F}_{AppROT}$’s completed successfully (without $\perp$’s, and in particular did complete at all), the simulator extracts a set of Bloom filters $BF_0$ from any instance between corrupt $P_0$ and honest $P_i$ and computes $M''(P_i \in P \setminus \{I \cup P_0\})$ as the functionality $\mathcal{F}_{AppROT}$ would compute the output for the honest sender (from $M''$ and $I_i$). If and once all $\mathcal{F}_{AppROT}$ executions are completed, $S$ extracts an effective input of the adversary as $\hat{X}_I = \{q \in Q | \forall s \in h_s(q), \forall j \notin I, BF_0[S_j][q] = 1\}$ - 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 $\hat{X}_I$ is extracted, $S$ sends either it or $\perp$ to the ideal functionality as the input of each of the corrupted parties $\mathcal{F}_{MPI}$, and receives either $\hat{X}$ or $\perp$ as the output. In the above, $\perp$ is sent if and only if at least one of the emulated $\mathcal{F}_{AppROT}$’s ended with $\perp$, the simulator gives $\perp$, which is the effective input of the adversary, to $\mathcal{F}_{MPI}$ 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 $\mathcal{F}_{AppROT}$’s of $P_i$ are completed, even if there are another running $\mathcal{F}_{AppROT}$’s for other parties:

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

**Simulator Analysis.** Fix a certain $Z$, running on the public parameters $1^0$, $1^\lambda$, $k$, $N$. $S$ proceeds as follows. We prove indistinguishability by induction on the message graph of $Z$ sent messages to the various parties, and to $\mathcal{F}_{AppROT}$ 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 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, \( \{ S^l \}_{P_i \in \mathcal{P} \setminus \{ l, P_0 \}} \) sent from honest parties are random i.i.d strings (sampled by \( S \) in ideal world), and have the same (uniform) distribution in both ideal and real worlds. In step 2, \( \{ M^l \uell \}_{P_i \in \mathcal{P} \setminus \{ l, P_0 \}}, \{ \tilde{M}^l \}_{P_i \in \mathcal{P} \setminus \{ l, P_0 \}} \) are identically distributed to the values received by the corrupted sender in the real world protocol, by definition of \( \mathcal{F}_{\text{AppROT}} \), and the fact that at any step of interaction of the \( \mathcal{F}_{\text{AppROT}} \) instances, the view of each emulated \( P_i \) is distributed identically to the real world. In both cases (by inspection) when one of the parties is corrupted, the output of \( \mathcal{F}_{\text{AppROT}} \) does not depend on the input of the honest party, and is properly emulated by \( S \) above. As the input to the \( \mathcal{F}_{\text{AppROT}} \)'s are distributed identically (resulting from the same \( \mathcal{Z} \)), so are the outputs. To see this, note that when a round-2 (step 5) message from one or more honest party was not yet sent when another honest party \( P_l \) sends its step-5 message and other honest parties \( P_i \) have not contributed their step-5 shares \( \bigoplus_{j \in \mathcal{I} \cup \{ l, P_0 \} \cup \{ P_j \}} S^j \) 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 sting (share). Calls to the RO also don’t break indistinguishability, because they actually refer to the same RO in both worlds.

Let us now compare the effect of step 5, assuming all executions of \( \mathcal{F}_{\text{AppROT}} \) with honest parties have completed. Assume first they have completed without any \( \perp \)’s in the real world. In this case, (after canceling the shares \( S^l \) contributed by the malicious parties \( P_i \), which are known to \( \mathcal{Z} \), \( \bigoplus_{i \in \mathcal{I}} GBF^i \) sent in the real world by honest parties, along with \( 0 \)-shares \( \bigoplus_{i \in \mathcal{I}} S^l \) for \( l \in \mathcal{I} \) are random additive shares of a randomized GBF, \( G \), containing the intersection of honest parties’ input with \( \tilde{X} \), \( H = \tilde{X} \cap \left( \bigcap_{P_i \in \mathcal{P} \setminus \{ l, P_0 \}} X_i \right) \), encoded via the corresponding \( \bigoplus_{i} (M^i \bigoplus \hat{M}^i) \), at entries in \( \{ h_x(q) | q \in H \} \) with overwhelming probability. Such a GBF, \( G \), has fixed sums at the locations corresponding to elements of \( H \) (determined by \( M, \hat{M} \)), and is random otherwise.\(^{22}\) The overwhelming probability is due to two observations. (1) \( G[j] \) is random for every \( j \) not in \( \cup_{x} (h_x(x_i)) \) for some \( X_i \) where \( P_i \) is honest, as \( \mathcal{Z} \) does not know the corresponding \( \hat{M}[j] \) used by it (randomly complemented by \( P_l \) upon rerandomizing this 0-entry in step 5). (2) If (1) does not happen, for \( j \) outside of a set \( h_x(x) \) 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 \( M^l[j] \) is not learned by \( \mathcal{Z} \). (3) Words \( x \) used by \( P_0 \) in step 3a from all parties on which the RO was not queried (resulting in no indices from the first or second kind). As no \( \perp \) occurred in any \( \mathcal{F}_{\text{AppROT}} \) 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 \( \perp \) in the real world. In this case \( S \) sends \( \perp \) to \( \mathcal{F}_{\text{MPSI}} \), and thus receives \( \perp \). The simulation of step 5’s messages is perfect in this case, since at least one of the shares is not delivered by at least one honest party for each GBF entry, resulting in random i.i.d entries.

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

\[ \square \]

### F.5 Proving Theorem 1

We prove our protocol satisfies the standard UC security [5]. We will need the \( \mathcal{F}_{\text{RO}}^{\sigma, N} \) defined above and the \( \mathcal{F}_{\text{RO}} \), which is a variant of the standard random oracle in Figure 12.

Theorem 1 follows directly from Lemmas 1 and 2, by an application of the Universal composition theorem.

Note that in the resulting protocol, the "offline" part appearing in the protocol in Figure 5 indeed line up at the beginning of our protocol, and all computation performed there does not depend on the inputs. Therefor, following instantiation of \( \mathcal{F}_{\text{RO}}^{\sigma, N} \), we may move the corresponding parts to an offline phase, as in \( \Pi_{\text{MPSI}} \).

\(^{22}\)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 \( \mathbb{F}_2 \).
\[ \mathcal{F}_{\text{RO}} \]

**Parameters:**
- \( k \) – number of hash-functions;
- \( N \) – size of input domain;
- \( \ell(N) \) – size of output domain of each hash function.

**Initialization:** Initialize \( Q \) to be an empty list of prior queries. **Output:** upon receiving a query \( x \in [N] \) from party \((SID, PID)\) or the adversary \( S \).

1. If no item of the form \((x, \ast)\) is not in \( Q \), sample a random vector \( v \in [\ell(N)]^k \), and add \((x, v)\) to the list \( Q \).
2. Let \((x, v)\) be the (unique) item in \( Q \) of the form \((x, \ast)\). Return \( v \).

Figure 12: Random Oracle functionality