Protocol Π_{AppROT}

Parties: Sender, Receiver.
Parameters:
κ – length of the OT strings (computational security parameter);
λ – statistical security parameter;
N_{bf} is the required Bloom filter size;
$N_{ot} > N_{bf}$ is the number of random OTs to generate;
N_{ot}^1, N_{cc}, $N_{maxones}$ are parameters of cut-and-choose described in Section **??**.
Inputs: $b_1, ..., b_{N_{bf}}$ is Bloom filter of Receiver ($b_i \in \{0,1\}$, $i = 1, ..., N_{bf}$);
Offline phase $\Pi_{AppROT}^{Offline}$**:**

1. **Random OT:** Sender performs N_{ot} random OTs with Receiver. Receiver chooses requests $c_1, ..., c_{N_{ot}}$ with N_{ot}^1s ones among them, and $N_{ot} - N_{ot}^1$ zeroes (randomly permuted). As a result, in the jth ROT, Sender learns random strings m_{j0}, m_{j1} (length of κ) chosen by the functionality $\mathcal{F}_{ROT}$. Receiver uses choice bit c_j and learns $m_{j*} = m_{jc_j}$.
2. **Cut-and-choose challenge:** Sender chooses sets $C \subseteq [N_{ot}]$ of size N_{cc} and sends C to Receiver, who aborts if $|C| \neq N_{cc}$.
3. **Cut-and-choose response:** Receiver computes and sends to Sender the set $R = \{j \in C | c_j = 0\}$. To prove that he used choice bit 0 in the OTs indexed by R, it also sends $r^* = \bigoplus_{j \in R} m_{j*}$. Sender aborts if $|C| - |R| > N_{maxones}$ or if $r^* \neq \bigoplus_{j \in R} m_{j0}$.

Online phase Π_{AppROT}^{Online}**:**

4. **Permute unopened OTs:** Receiver chooses random injective function $\pi : [N_{bf}] \to ([N_{ot}] \setminus C)$ such that $b_j = c_{\pi(j)}$, and sends π to Sender.

   Receiver permutes its random values m_{j*} according the π, and Sender permutes all the m_{j0}, m_{j1} according to π.

Outputs: Receiver has output m_{j*} ($j = 1, ..., N_{bf}$) - Garbled Bloom filter that corresponds to Bloom filter of Receiver;
Sender has m_{j0}, m_{j1}, ($j = 1, ..., N_{bf}$) – random strings corresponding to "zeroes" and "ones" respectively in Garbled Bloom filter of Receiver.

Figure 5: protocol Π_{AppROT}.

Two-party Malicious PSI (offline-phase) $\Pi_{2PSI}^{Offline}$

Parameters:
P_0, P_1 – parties; n – size of input sets;
κ - computational security parameter;
λ - statistical security parameter.
Offline-phase of the protocol:

1. **Compute parameters:** Parties compute parameters k, N_{bf}, N_{ot}, N_{bf}^1, N_{cc}, $N_{maxones}$ from n, κ, λ as described in Section **??**.

2. **Hash seeds agreement:** Parties agree about hash-functions $h_1, h_2, ..., h_k : \{0,1\} \to [N_{bf}]$.

3. **Approximate ROTs with P_0:** P_0 as a Receiver performs $\Pi_{AppROT}^{Offline}$ with P_1 with parameters κ, λ, N_{ot}, N_{bf}, N_{ot}^1, N_{cc}, $N_{maxones}$.
As a result P_1 learns random strings m_{j0}, m_{j1} (of length κ). P_0 uses choice bits c_j and learns $m_{j*} = m_{jc_j}$ ($j \in [N_{ot} - N_{cc}]$).

4. **Approximate ROTs for secret-sharing:** P_1 as a Receiver performs $\Pi_{AppROT}^{Offline}$ with P_0 with parameters κ, λ, N_{ot}, N_{bf}, N_{ot}^1, N_{cc}, $N_{maxones}$. As a result, P_0 has set of strings m_{j*}^*, and P_1 have 2 sets m_{j0}^*, m_{j1}^* ($j \in [N_{ot} - N_{cc}]$).

Figure 6.1: offline-phase of our malicious-secure two-party protocol Π_{2PSI}.

Two-party Malicious PSI (offline-phase) Π_{2PSI}^{Online}

P_0, P_1 - parties; P_i holds $X_i = \{x_{i1}, x_{i2}, ..., x_{in}\}$;
$h_1, h_2, ..., h_k : \{0,1\}^* \rightarrow [N_{bf}]$ - Bloom filter hash-functions; N_{bf} - size of Bloom filter
κ - computational security parameter; λ - statistical security parameter.

Online-phase Π_{2PSI}^{Online}:

5. **Compute Bloom filters:** P_i ($i \in \{0,1\}$) computes sets of indexes $h_*(x_{ij}) = \{h_s(x_{ij}) : s \in [k]\}$ ($j \in [n]$) using Algorithm 3.2, and Bloom filter $b^i = (b_1^i$, b_2^i, ..., $b_{N_{bf}}^i)$ of items from X_i.

6. **Compute Garbled Bloom filter:** Using b^0 as an input, P_0 performs Π_{AppROT}^{Online} with P_1 to finish Π_{AppROT} started on Step 3. As a result, it receives GBF^0.

7. **Compute garbled share for P_0:** Using b^1 as an input, P_1 performs Π_{AppROT}^{Online} with P_0 to finish Π_{AppROT} started on Step 4. P_0 computes the garbled Bloom filter of his input set $GBF_0^{1*} = (m^*_{2b_2^0}, ..., m^*_{N_{bf}b^0_{N_{bf}}})$, P_1 receives share $(m^*_{1*}, m^*_{2*}, ..., m^*_{N_{bf}*})$.

8. **Compute codewords:** P_0 computes codewords $y_{0j} = \bigoplus_{s \in h_*(x_{0j})} GBF^0[s]$.
P_1 computes $y_{1j} = \bigoplus_{s \in h_*(x_{1j})} m_{s1}$

9. **Re-randomize GBF:** P_1 computes GBF^{1*}, performing Algorithm B.1 with items x_{1j} and codewords y_{1j} ($j \in [N_{bf}]$).

10. **Compute garbled share for P_1:** P_1 computes garbled share $GBF_1^{1*} = GBF^{1*} \oplus (m^*_{1*}, m^*_{2*}, ..., m^*_{N_{bf}*})$ and sends it to P_0.

11. **Output:** P_0 computes $GBF^* = GBF_0^{1*} \oplus GBF_1^{1*}$ and outputs x_{0j} as a member of the intersection, if

$$y_{0j} = \bigoplus_{s \in h_*(x_{0j})} GBF^*[s], j \in [n].$$

Figure 6.2: online-phase of our malicious-secure two-party protocol Π_{2PSI}.

B.1. Algorithm of re-randomization of Garbled Bloom filter

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

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

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

B.2. Algorithm for computation of hash-indexes set $h_*(x)$

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

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

[1] 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 (**??**), $p_{False} \leq 2^{-\kappa}$, so the union bound over all $x \in X$ is $n2^{-\kappa}$ is negligible in κ.

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

B.3. Algorithm for computation of codeword from Garbled Bloom filter

Algorithm CodewordGBF$(GBF, x, h_*(x), N_{bf}, \kappa)$
Input:
x – item;
GBF – random garbled Bloom filter;
N_{bf} – length of GBF;
κ – bitlength of string in GBF;
$h_*(x)$ – set of hash-indexes of x; $\forall i \in h_*(x)$, $i \in [N_{bf}]$.

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