The GMW Protocol in IPDL

November 12, 2020

1 The setup

We assume $N+2$ parties labeled $n \coloneqq 0, \dots, N+1$, where an arbitrary proper subset of the parties are corrupted. We assume a finite number $I+1$ of inputs labeled $i \coloneqq 0, \dots, I$, and a well-formed Boolean circuit C with XOR and AND gates on inputs $0, \dots, I$ with $K+1$ wires labeled $k \coloneqq 0, \dots, K$, where an arbitrary nonzero subset of the wires are declared as outputs. Furthermore, we assume a function $owner(-)$ that assigns each input to a specific party; this party is the only one who knows the value of the input. Finally, for each pair of parties $n < m$ we assume a function $sender(n, m)$ that returns true if n is the sender in the OT exchange between n and m and false otherwise.

2 The ideal-world protocol

The ideal world protocol has parties $\mathcal{P}^n_{ideal}$ for $0 \leq n \leq N+1$ and one ideal functionality $\mathcal{F}$.

2.1 $\mathcal{P}^n_{ideal}$

The code for $\mathcal{P}^n_{ideal}$ is as follows:

1. For each input $0 \leq i \leq I$:
   (a) If n is not the owner of the input i, we do nothing: $\mathbb{1}_{\emptyset}$
   (b) If n is the owner of the input i, we leak the knowledge that the input has been received:
   - $in(i) \rightsquigarrow leakInputOk(i)_{ideal} \coloneqq in(i)$

   If applicable, we also leak the value of the input:
   i. If n is not corrupted, we do nothing: $\mathbb{1}_{\{in(i)\}}$
   ii. Otherwise we leak the value of the input:
   - $in(i) \rightsquigarrow leakInput(i)_{ideal} \coloneqq in(i)$

   Finally, we forward the input to the functionality:
   - $in(i) \rightsquigarrow sendInputToFunc(i) \coloneqq in(i)$
2. For each wire $0 \leq k \leq K$:
   (a) If k is not declared as an output, we do nothing: $\mathbb{1}_{\emptyset}$
   (b) If k is declared as an output, we output the value we received from the functionality:
   - $sendOutputFromFunc(k, n) \rightsquigarrow out(k, n) \coloneqq sendOutputFromFunc(k, n)$

   If applicable, we leak the value of the output:
   i. If n is not corrupted, we do nothing: $\mathbb{1}_{\{sendOutputFromFunc(k,n)\}}$
   ii. Otherwise we leak the the value we received from the functionality:
   - $sendOutputFromFunc(k, n) \rightsquigarrow leakOutput(k, n)_{ideal} \coloneqq sendOutputFromFunc(k, n)$

2.2 $\mathcal{F}$

Given a circuit C with L wires, we define a program $\mathcal{A}(C, L)$ by induction on C, computing the value on each wire $0 \leq l < L$. The functionality $\mathcal{F}$ consists of such a computation for the ambient circuit C with $K+1$ wires, plus the revealing of each output to each party. Formally, the code for $\mathcal{F}$ is as follows:

1. We compute the value on each wire $0 \leq k \leq K$: $\mathcal{A}(C, K+1)$
2. We forward each output to each party. For each wire $0 \leq k \leq K$ and party $0 \leq n \leq N+1$:
   (a) If k is not declared as an output, we do nothing: $\mathbb{1}_{\{value(k)\}}$
   (b) Otherwise we forward the value on wire k to party n:
      - $value(k) \rightsquigarrow sendOutputFromFunc(k, n) \coloneqq value(k)$
3. At last we hide the channels
   - $value(k)$ for $0 \leq k \leq K$

We now define $\mathcal{A}(C, L)$.

2.2.1 $\mathcal{A}(\cdot, 0)$

If the circuit is empty, we do nothing: $\mathbb{1}_{\emptyset}$.

2.2.2 $\mathcal{A}(C; \mathsf{input}(i), L+1)$

If the last wire draws from the input i, the result is the composition of the program $\mathcal{A}(C, L)$ that computes the value on each wire $0 \leq l < L$ and the program

- $sendInputToFunc(i) \rightsquigarrow value(L) \coloneqq sendInputToFunc(i)$

that sets the value on wire L to the value of the input i as communicated by the party that owns i.

2.2.3 $\mathcal{A}(C; l_1 \,\mathsf{XOR}\, l_2, L+1)$

If the last wire is an XOR gate of wires l_1 and l_2, the result is the composition of the program $\mathcal{A}(C, L)$ that computes the value on each wire $0 \leq l < L$ and the program

- $value(l_1), value(l_2) \rightsquigarrow value(L) \coloneqq value(l_1) \oplus value(l_2)$

that sets the value on wire L to be the sum of the values on wires l_1 and l_2.

2.2.4 $\mathcal{A}(C; l_1 \,\mathsf{AND}\, l_2, L+1)$

If the last wire is an AND gate of wires l_1 and l_2, the result is the composition of the program $\mathcal{A}(C, L)$ that computes the value on each wire $0 \leq l < L$ and the program

- $value(l_1), value(l_2) \rightsquigarrow value(L) \coloneqq value(l_1) * value(l_2)$

that sets the value on wire L to be the product of the values on wires l_1 and l_2.

2.3 The ideal-world protocol

The ideal-world protocol is the composition of the programs

1. $\mathcal{P}^n_{ideal}$ for $0 \leq n \leq N+1$
2. $\mathcal{F}$

followed by the hiding of the channels

- $sendInputToFunc(i)$ for $0 \leq i \leq I$
- $sendOutputFromFunc(k, n)$ for $0 \leq k \leq K$ and $0 \leq n \leq N+1$, if k is declared as an output

3 Cleaning up the ideal-world protocol

We can formulate the ideal-world protocol equivalently as described below, where we in particular eliminated the channels that serve to propagate information between the parties and the functionality. To compute the values on wires $0 \leq k \leq K$ directly from inputs, we define a program $\mathcal{B}(C, L)$ for an arbitrary circuit C with L wires by induction on C entirely analogously to $\mathcal{A}(C, L)$, as shown below. The ideal-world protocol thus becomes the composition of the following programs:

1. We compute the values on wires $0 \leq k \leq K$: $\mathcal{B}(C, K+1)$
2. We perform the outputs. For each wire $0 \leq k \leq K$ and party $0 \leq n \leq N+1$:
   (a) If k is not declared as an output, we do nothing: $\mathbf{1}_{\{value(k)\}}$
   (b) Otherwise we output the value on wire k:
      - $value(k) \rightsquigarrow out(k, n) := value(k)$
3. We leak the knowledge that the inputs have been received. For each input $0 \leq i \leq I$:
   (a) The owner of i informs the adversary that input i has been received:
      - $in(i) \rightsquigarrow leakInputOk(i)_{ideal} := \star$
4. If applicable, we leak the value of an input. For each input $0 \leq i \leq I$:
   (a) If the owner of i is not corrupted, nothing is leaked: $\mathbf{1}_{\{in(i)\}}$
   (b) Otherwise the owner leaks the value of i:
      - $in(i) \rightsquigarrow leakInput(i)_{ideal} := in(i)$
5. If applicable, we leak the value of an output. For each wire $0 \leq k \leq K$ and party $0 \leq n \leq N+1$:
   (a) If k is not declared as an output or if n is not corrupted, we do nothing: $\mathbf{1}_{\{value(k)\}}$
   (b) Otherwise we leak the the value on wire k:
      - $value(k) \rightsquigarrow leakOutput(k, n)_{ideal} := value(k)$
6. At last we hide the channels
   - $value(k)$ for $0 \leq k \leq K$

We now define $\mathcal{B}(C, L)$.

3.0.1 $\mathcal{B}(\cdot, 0)$

If the circuit is empty, we do nothing: $\mathbf{1}_{\emptyset}$.

3.0.2 $\mathcal{B}(C; \mathsf{input}(i), L+1)$

If the last wire draws from the input i, the result is the composition of the program $\mathcal{B}(C, L)$ that computes the value on each wire $0 \leq l < L$ and the program

- $in(i) \rightsquigarrow value(L) := in(i)$

that sets the value on wire L to the value of the input i.

3.0.3 $\mathcal{B}(C; l_1 \,\mathsf{XOR}\, l_2, L+1)$

If the last wire is an XOR gate of wires l_1 and l_2, the result is the composition of the program $\mathcal{B}(C, L)$ that computes the value on each wire $0 \leq l < L$ and the program

- $value(l_1), value(l_2) \rightsquigarrow value(L) := value(l_1) \oplus value(l_2)$

that sets the value on wire L to be the sum of the values on wires l_1 and l_2.

3.0.4 $\mathcal{B}(C;\, l_1\ \mathsf{AND}\ l_2, L+1)$

If the last wire is an AND gate of wires l_1 and l_2, the result is the composition of the program $\mathcal{B}(C, L)$ that computes the value on each wire $0 \le l < L$ and the program

- $\mathit{value}(l_1), \mathit{value}(l_2) \rightsquigarrow \mathit{value}(L) \coloneqq \mathit{value}(l_1) * \mathit{value}(l_2)$

that sets the value on wire L to be the product of the values on wires l_1 and l_2.

4 The real-world-protocol

The real world protocol has parties $\mathcal{P}^n_{\mathit{real}}$ for $0 \le n \le N+1$, and one ideal functionality $\mathcal{OT}$. The parties interact with one another at the beginning and at the end of the protocol, when they send their initial and final shares. For every AND gate encountered, the parties interact with the OT functionality by engaging in a 1-out-of-4 OT exchange, where for each pair of parties $n < m$ we have a designated sender and a designated receiver, as determined by $\mathit{sender}(n, m)$. The code for each party can be divided into three main parts:

1. the computation and revealing of input shares, $\mathit{Init}(n)$
2. the computation corresponding to each circuit body, $\mathcal{C}(n, C, K+1)$
3. the revealing of final shares and computation of outputs, $\mathit{Fin}(n)$

The OT functionality will consist of copies of a basic OT functionality, with each copy indexed by a wire $0 \le k \le K$ and the two parties n, m – where n is the sender and m is the receiver – that together serve as a session ID. The basic OT functionality has inputs $\mathit{senOT}_{00}, \mathit{senOT}_{01}, \mathit{senOT}_{10}, \mathit{senOT}_{11} : \mathsf{bool}$ for the 4 bits from the sender and $\mathit{recOT}_1, \mathit{recOT}_2 : \mathsf{bool}$ for the 2 bits from the receiver, and output $\mathit{resOT} : \mathsf{bool}$. The code is given by the following program:

- $\mathit{senOT}_{00}, \mathit{senOT}_{01}, \mathit{senOT}_{10}, \mathit{senOT}_{11}, \mathit{recOT}_1, \mathit{recOT}_2 \rightsquigarrow$
  $\mathit{resOT} \coloneqq \mathsf{if}\ \mathit{recOT}_1\ \mathsf{then\ if}\ \mathit{recOT}_2\ \mathsf{then}\ \mathit{senOT}_{11}\ \mathsf{else}\ \mathit{senOT}_{10}\ \mathsf{else\ if}\ \mathit{recOT}_2\ \mathsf{then}\ \mathit{senOT}_{01}\ \mathsf{else}\ \mathit{senOT}_{00}$

We now describe each part in turn.

4.1 $\mathit{Init}(n)$

The code for $\mathit{Init}(n)$ is the composition of the following programs:

1. For each input $0 \le i \le I$:
   (a) If n is not the owner of input i, we receive the input share from the owner of i:
      - $\mathit{sendInitShare}(i, n) \rightsquigarrow \mathit{inputShare}(i, n) \coloneqq \mathit{sendInitShare}(i, n)$

      If applicable, we leak the value of the share:
      i. If n is not corrupted, we leak nothing: $\mathsf{1}_{\{\mathit{sendInitShare}(i,n)\}}$
      ii. Otherwise we leak the share we received:
         - $\mathit{sendInitShare}(i, n) \rightsquigarrow \mathit{leakInputShare}(i, n)_{\mathit{real}} \coloneqq \mathit{sendInitShare}(i, n)$

   (b) If n is the owner of input i, we first inform the adversary that input i has been received:
      - $\mathit{in}(i) \rightsquigarrow \mathit{leakInputOk}(i)_{\mathit{real}} \coloneqq \star$

      If applicable, we also leak the value of i:
      i. If n is not corrupted, we leak nothing: $\mathsf{1}_{\{\mathit{in}(i)\}}$
      ii. Otherwise we leak the value of the input:
         - $\mathit{in}(i) \rightsquigarrow \mathit{leakInput}(i)_{\mathit{real}} \coloneqq \mathit{in}(i)$

      We next randomly generate input shares for each party $0 \le m \le N$ (for everyone except party $N+1$):
      - $\mathit{in}(i) \rightsquigarrow \mathit{genShare}(i, m) \leftarrow \mathsf{rand}(\mathsf{bool})$

If applicable, we leak these shares to the adversary.

i. If n is not corrupted, we leak nothing: $\mathbf{1}_{\emptyset}$

ii. Otherwise for each party $0 \leq m \leq N$:

- $\mathit{genShare}(i,m) \rightsquigarrow \mathit{leakGenShare}(i,m)_{\mathit{real}} \coloneqq \mathit{genShare}(i,m)$

To determine the share of party $N+1$, we compute the sum of all the shares we generated: for $0 \leq m \leq N$, the channel $\mathit{genShare}_{\Sigma}(i,m)$ will hold the sum of all shares $\mathit{genShare}(i,-)$ where the last index is less than or equal to m. We define $\mathit{genShare}_{\Sigma}(i,m)$ inductively as follows:

i. In the zero case we are summing a single share:

- $\mathit{genShare}(i,0) \rightsquigarrow \mathit{genShare}_{\Sigma}(i,0) \coloneqq \mathit{genShare}(i,0)$

ii. In the successor case we add the last share to the sum:

- $\mathit{genShare}_{\Sigma}(i,m), \mathit{genShare}(i,m+1) \rightsquigarrow$
  $\mathit{genShare}_{\Sigma}(i,m+1) \coloneqq \mathit{genShare}_{\Sigma}(i,m) \oplus \mathit{genShare}(i,m+1)$

We let the share of party $N+1$ be the sum of the input i and the shares of all the other parties:

- $\mathit{in}(i), \mathit{genShare}_{\Sigma}(i,N) \rightsquigarrow \mathit{genShare}(i,N+1) \coloneqq \mathit{in}(i) \oplus \mathit{genShare}_{\Sigma}(i,N)$

We then distribute the shares to all other parties. For $0 \leq m \leq N+1$:

i. If $m = n$, we do nothing: $\mathbf{1}_{\{\mathit{genShare}(i,n)\}}$

ii. Otherwise we send the share we generated for party m:

- $\mathit{genShare}(i,m) \rightsquigarrow \mathit{sendInitShare}(i,m) \coloneqq \mathit{genShare}(i,m)$

At last we set our own share:

- $\mathit{genShare}(i,n) \rightsquigarrow \mathit{inputShare}(i,n) \coloneqq \mathit{genShare}(i,n)$

and hide the intermediate channels

- $\mathit{genShare}(i,m)$ for $0 \leq m \leq N+1$
- $\mathit{genShare}_{\Sigma}(i,m)$ for $0 \leq m \leq N$

4.2 $\mathcal{C}(n, C, L)$

We define $\mathcal{C}(n, C, L)$ for an arbitrary circuit C with L wires by induction on C.

4.2.1 $\mathcal{C}(n, \cdot, 0)$

If the circuit is empty we do nothing: $\mathbf{1}_{\emptyset}$.

4.2.2 $\mathcal{C}(n, C; \mathsf{input}(i), L+1)$

If the last wire draws from the input i, the result is the composition of the program $\mathcal{C}(n, C, L)$ that computes the share for party n on each wire $0 \leq l < L$ and the program

- $\mathit{inputShare}(i,n) \rightsquigarrow \mathit{share}(L,n) \coloneqq \mathit{inputShare}(i,n)$

that sets the share for party n on wire L to the input share for party n of the input i.

4.2.3 $\mathcal{C}(n, C; l_1 \,\mathsf{XOR}\, l_2, L+1)$

If the last wire is an XOR gate of wires l_1 and l_2, the result is the composition of the program $\mathcal{C}(n, C, L)$ that computes the share for party n on each wire $0 \leq l < L$ and the program

- $\mathit{share}(l_1,n), \mathit{share}(l_2,n) \rightsquigarrow \mathit{share}(L,n) \coloneqq \mathit{share}(l_1,n) \oplus \mathit{share}(l_2,n)$

that sets the share for party n on wire L to be the sum of the shares for n on wires l_1 and l_2.

4.2.4 $\mathcal{C}(n, C; l_1 \,\mathsf{AND}\, l_2, L+1)$

If the last wire is an AND gate of wires l_1 and l_2, the result is the composition of the program $\mathcal{C}(n, C, L)$ that computes the share for party n on each wire $0 \leq l < L$ and the following program:

1. We generate a random bit for every OT exchange where n is the sender. For each party $0 \leq m \leq N+1$:
   (a) If $n < m$ and $\mathit{sender}(n, m)$ is true or $m < n$ and $\mathit{sender}(m, n)$ is false – so n is the sender and m is the receiver – we choose a bit at random:
      - $\mathit{share}(l_1, n), \mathit{share}(l_2, n) \rightsquigarrow \mathit{OTBitSen}(L, n, m) \leftarrow \mathsf{rand}(\mathsf{bool})$

   (b) Otherwise we do nothing: $\mathbf{1}_{\{\mathit{share}(l_1,n), \mathit{share}(l_2,n)\}}$

   The dependency on $\mathit{share}(l_1, n)$ and $\mathit{share}(l_2, n)$ is there to indicate that the computation of the n-th party's share on wire L is only initiated once the respective shares on wires l_1 and l_2 have been computed.

2. If applicable, we leak the bits we generated in the previous step.
   (a) If n not corrupted, we leak nothing: $\mathbf{1}_{\emptyset}$.
   (b) Otherwise for each party $0 \leq m \leq N+1$:
      i. If $n < m$ and $\mathit{sender}(n, m)$ is true or $m < n$ and $\mathit{sender}(m, n)$ is false – so n is the corrupted sender and m is the receiver – we leak the value of $\mathit{OTBitSen}(L, n, m)$ that we randomly generated:
         - $\mathit{OTBitSen}(L, n, m) \rightsquigarrow \mathit{leakRandOTBitSen}(L, n, m)_{\mathit{real}} := \mathit{OTBitSen}(L, n, m)$

      ii. Otherwise we leak nothing: $\mathbf{1}_{\emptyset}$.

3. We now perform the OT exchanges where n is the sender. For each party $0 \leq m \leq N+1$:
   (a) If $n < m$ and $\mathit{sender}(n, m)$ is true or $m < n$ and $\mathit{sender}(m, n)$ is false – so n is the sender and m is the receiver – we compute the 4 bits to send to the OT functionality as a sender:
      - $\mathit{OTBitSen}(L, n, m), \mathit{share}(l_1, n), \mathit{share}(l_2, n) \rightsquigarrow$ $\mathit{senOT}_{00}(L, n, m) := \mathit{OTBitSen}(L, n, m)$
      - $\mathit{OTBitSen}(L, n, m), \mathit{share}(l_1, n), \mathit{share}(l_2, n) \rightsquigarrow$ $\mathit{senOT}_{01}(L, n, m) := \mathit{share}(l_1, n) \oplus \mathit{OTBitSen}(L, n, m)$
      - $\mathit{OTBitSen}(L, n, m), \mathit{share}(l_1, n), \mathit{share}(l_2, n) \rightsquigarrow$ $\mathit{senOT}_{10}(L, n, m) := \mathit{share}(l_2, n) \oplus \mathit{OTBitSen}(L, n, m)$
      - $\mathit{OTBitSen}(L, n, m), \mathit{share}(l_1, n), \mathit{share}(l_2, n) \rightsquigarrow$ $\mathit{senOT}_{00}(L, n, m) := \big(\mathit{share}(l_1, n) \oplus \mathit{share}(l_2, n)\big) \oplus \mathit{OTBitSen}(L, n, m)$

   (b) Otherwise we do nothing: $\mathbf{1}_{\{\mathit{share}(l_1,n), \mathit{share}(l_2,n)\}}$

4. We now perform the OT exchanges where n is the receiver. For each party $0 \leq m \leq N+1$:
   (a) If $m < n$ and $\mathit{sender}(m, n)$ is true or $n < m$ and $\mathit{sender}(n, m)$ is false – so m is the sender and n is the receiver – we compute the 2 bits to send to the OT functionality as a receiver:
      - $\mathit{share}(l_1, n), \mathit{share}(l_2, n) \rightsquigarrow \mathit{recOT}_1(L, m, n) := \mathit{share}(l_1, n)$
      - $\mathit{share}(l_1, n), \mathit{share}(l_2, n) \rightsquigarrow \mathit{recOT}_2(L, m, n) := \mathit{share}(l_2, n)$

   (b) Otherwise we do nothing: $\mathbf{1}_{\{\mathit{share}(l_1,n), \mathit{share}(l_2,n)\}}$

5. Here we record the bits we received from the OT exchanges. For each party $0 \leq m \leq N+1$:
   (a) If $m < n$ and $\mathit{sender}(m, n)$ is true or $n < m$ and $\mathit{sender}(n, m)$ is false – so m is the sender and n is the receiver – we record the bit we received from the OT exchange with party m:
      - $\mathit{resOT}(L, m, n) \rightsquigarrow \mathit{OTBitRec}(L, n, m) := \mathit{resOT}(L, m, n)$

   (b) Otherwise we do nothing: $\mathbf{1}_{\emptyset}$

6. If applicable, we leak the bits we received from the OT exchanges in the previous step.

(a) If n not corrupted, we leak nothing: $1_\emptyset$.

(b) Otherwise for each party $0 \leq m \leq N+1$:

i. If $m < n$ and $sender(m,n)$ is true or $n < m$ and $sender(n,m)$ is false – so m is the sender and n is the corrupted receiver – we leak the value of $resOT(L,m,n)$ that we received as an input from the OT functionality:

- $resOT(L,m,n) \rightsquigarrow leakRandOTBitRec(L,m,n)_{real} := resOT(L,m,n)$

ii. Otherwise we leak nothing: $1_\emptyset$.

7. We now compute a bit $bit(L,n,m)$ for each party $0 \leq m \leq N+1$ as follows:

(a) If $m = n$, the resulting bit is the product of our own shares on wires l_1 and l_2:

- $share(l_1,n), share(l_2,n) \rightsquigarrow bit(L,n,m) := share(l_1,n) * share(l_2,n)$

(b) If $n < m$ and $sender(n,m)$ is true or $m < n$ and $sender(m,n)$ is false – so n is the sender and m is the receiver – the resulting bit is the random bit we generated for the OT exchange with party m:

- $OTBitSen(L,n,m), share(l_1,n), share(l_2,n) \rightsquigarrow bit(L,n,m) := OTBitSen(L,n,m)$

(c) If $m < n$ and $sender(m,n)$ is true or $n < m$ and $sender(n,m)$ is false – so m is the sender and n is the receiver – the resulting bit is the bit we received as the result of the OT exchange between n and m:

- $OTBitRec(L,n,m), share(l_1,n), share(l_2,n) \rightsquigarrow bit(L,n,m) := OTBitRec(L,n,m)$

8. We now sum up all the bits computed earlier to obtain the share of party n on wire L: for $0 \leq m \leq N+1$, the channel $bit_\Sigma(L,n,m)$ will hold the sum of all the bits $bit(L,n,-)$ where the last index is less than or equal to m. We define $bit_\Sigma(L,n,m)$ inductively as follows:

(a) In the zero case we are summing a single bit:

- $bit(L,n,0) \rightsquigarrow bit_\Sigma(L,n,0) := bit(L,n,0)$

(b) In the successor case we add the last bit to the sum:

- $bit_\Sigma(L,n,m), bit(L,n,m+1) \rightsquigarrow bit_\Sigma(L,n,m+1) := bit_\Sigma(L,n,m) \oplus bit(L,n,m+1)$

9. We compute the share as the sum of all bits:

- $bit_\Sigma(L,n,N+1) \rightsquigarrow share(L,n) := bit_\Sigma(L,n,N+1)$

10. At last we hide the intermediate channels:

- $OTBitSen(L,n,m)$ for $0 \leq m \leq N+1$ where $n < m$ and $sender(n,m)$ is true or $m < n$ and $sender(m,n)$ is false
- $OTBitRec(L,n,m)$ for $0 \leq m \leq N+1$ where $m < n$ and $sender(m,n)$ is true or $n < m$ and $sender(n,m)$ is false
- $bit(L,n,m)$ for $0 \leq m \leq N+1$
- $bit_\Sigma(L,n,m)$ for $0 \leq m \leq N+1$

4.3 $Fin(n)$

The code for $Fin(n)$ is as follows:

1. For each wire $0 \leq k \leq K$:

(a) If k is not declared as an output, we do nothing: $1_\emptyset$

(b) If k is declared as an output, we reveal our share on this wire to everybody else. For each party $0 \leq m \leq N+1$:

i. If $m = n$, we do nothing: $1_{\{share(k,n)\}}$

ii. Otherwise we send our share on wire k to party m:

- $\mathit{share}(k,n) \rightsquigarrow \mathit{sendFinShare}(k,n,m) \coloneqq \mathit{share}(k,n)$

We then record everybody's output share. For each party $0 \le m \le N+1$:

i. If $m = n$, this is our own share on wire k:
   - $\mathit{share}(k,n) \rightsquigarrow \mathit{outputShare}(k,n,n) \coloneqq \mathit{share}(k,n)$

ii. Otherwise we obtain the share from party m:
   - $\mathit{sendFinShare}(k,m,n) \rightsquigarrow \mathit{outputShare}(k,n,m) \coloneqq \mathit{sendFinShare}(k,m,n)$

If applicable, we leak each output share received:

i. If n is not corrupted, we do nothing: $\mathbf{1}_\emptyset$

ii. Otherwise we leak the final shares we received from the other parties. For each party $0 \le m \le N+1$:
   A. If $m = n$, we leak nothing: $\mathbf{1}_\emptyset$
   B. Otherwise we leak the output share we received from party m:
      - $\mathit{sendFinShare}(k,m,n) \rightsquigarrow \mathit{leakOutputShare}(k,n,m)_{\mathit{real}} \coloneqq \mathit{sendFinShare}(k,m,n)$

To determine the value of the output on wire k, we compute the sum of all output shares: for $0 \le m \le N+1$, the channel $\mathit{outputShare}_\Sigma(k,n,m)$ will hold the sum of all shares $\mathit{outputShare}(i,n,-)$ where the last index is less than or equal m. We define $\mathit{outputShare}_\Sigma(k,n,m)$ inductively as follows:

i. In the zero case we are summing a single share:
   - $\mathit{outputShare}(k,n,0) \rightsquigarrow \mathit{outputShare}_\Sigma(k,n,0) \coloneqq \mathit{outputShare}(k,n,0)$

ii. In the successor case we add the last share to the sum:
   - $\mathit{outputShare}_\Sigma(k,n,m), \mathit{outputShare}(k,n,m+1) \rightsquigarrow$
     $\mathit{outputShare}_\Sigma(k,n,m+1) \coloneqq \mathit{outputShare}_\Sigma(k,n,m) \oplus \mathit{outputShare}(k,n,m+1)$

Finally, we let the output on wire k be the sum of all the shares:

- $\mathit{outputShare}_\Sigma(k,n,N+1) \rightsquigarrow \mathit{out}(k,n) \coloneqq \mathit{outputShare}_\Sigma(k,n,N+1)$

and hide the intermediate channels

- $\mathit{outputShare}(k,n,m)$ for $0 \le m \le N+1$
- $\mathit{outputShare}_\Sigma(k,n,m)$ for $0 \le m \le N+1$

4.4 $\mathcal{P}^n_{\mathit{real}}$

The real-world protocol for party n is the composition of the following programs:

1. $\mathit{Init}(n)$
2. $\mathcal{C}(n, C, K+1)$
3. $\mathit{Fin}(n)$

followed by the hiding of the channels below:

- $\mathit{inputShare}(i,n)$ for $0 \le i \le I$
- $\mathit{share}(k,n)$ for $0 \le k \le K$

4.5 $\mathcal{OT}$

The OT functionality is the program $\mathcal{OT}(C, K+1)$, where for an arbitrary circuit C with L wires, we define $\mathcal{OT}(C, L)$ by induction on C as indicated below.

4.5.1 $\mathcal{OT}(\cdot, 0)$

We define $\mathcal{OT}(\cdot, 0)$ to be the empty program $\mathbf{1}_\emptyset$.

4.5.2 $\mathcal{OT}(C; \mathsf{input}(i), L+1)$

We define $\mathcal{OT}(C; \mathsf{input}(i), L+1)$ to be $\mathcal{OT}(C, L)$.

4.5.3 $\mathcal{OT}(C; l_1 \,\mathsf{XOR}\, l_2, L+1)$

We define $\mathcal{OT}(C; l_1 \,\mathsf{XOR}\, l_2, L+1)$ is $\mathcal{OT}(C, L)$.

4.5.4 $\mathcal{OT}(C; l_1 \,\mathsf{AND}\, l_2, L+1)$

We define $\mathcal{OT}(C; l_1 \,\mathsf{AND}\, l_2, L+1)$ to be the composition of the program $\mathcal{OT}(C, L)$ and the following program:

1. For each pair of parties $0 \leq n, m \leq N+1$:
   (a) If $n < m$ and $sender(n, m)$ is true or $m < n$ and $sender(m, n)$ is false – so n is the sender and m is the receiver – we perform the Oblivious Transfer:
      - $senOT_{00}(L, n, m), senOT_{01}(L, n, m), senOT_{10}(L, n, m), senOT_{11}(L, n, m),$ $recOT_1(L, n, m), recOT_2(L, n, m) \rightsquigarrow resOT(L, n, m) \coloneqq$ $\mathsf{if}\ recOT_1(L, n, m)\ \mathsf{then}\ \mathsf{if}\ recOT_2(L, n, m)\ \mathsf{then}\ senOT_{11}(L, n, m)\ \mathsf{else}\ senOT_{10}(L, n, m)$ $\mathsf{else}\ \mathsf{if}\ recOT_2(L, n, m)\ \mathsf{then}\ senOT_{01}(L, n, m)\ \mathsf{else}\ senOT_{00}(L, n, m)$

   (b) Otherwise we do nothing: $\mathsf{1}_\emptyset$

4.6 The real-world protocol

The real-world protocol is the composition of $N+2$ parties and the OT functionality:

1. $\mathcal{P}^n_{real}$ for $0 \leq n \leq N+1$
2. $\mathcal{OT}$

followed by the hiding of the following channel sets:

- the inputs of $\mathcal{OT}$
- the outputs of $\mathcal{OT}$
- $sendInitShare(i, n)$ for $0 \leq i \leq I$ and $0 \leq n \leq N+1$ with $n \neq owner(i)$
- $sendFinShare(k, n, m)$ for $0 \leq k \leq K$ and $0 \leq n, m \leq N+1$ with k declared as an output and $n \neq m$

5 Cleaning up the real-world protocol: eliminating intermediate channels

Eliminating the channels that serve to share information among the parties and the OT functionality, we can equivalently express the real-world protocol as the composition of the programs

- *Init* for the handling of inputs and input shares
- $\mathcal{D}(C, K+1)$ for the handling of shares on each wire of the ambient circuit C
- *Fin* for the handling of outputs and output shares

followed by the hiding of the channels

- $inputShare(i, n)$ for $0 \leq i \leq I$ and $0 \leq n \leq N+1$
- $share(k, n)$ for $0 \leq k \leq K$ and $0 \leq n \leq N+1$

We now describe each part in turn.

5.1 *Init*

The code for *Init* is as follows:

1. We leak the knowledge that the inputs have been received. For each input $0 \leq i \leq I$:
2. The owner of i informs the adversary that input i has been received:
   - $\mathit{in}(i) \rightsquigarrow \mathit{leakInputOk}(i)_{\mathit{real}} \coloneqq \star$
3. If applicable, we leak the value of an input. For each input $0 \leq i \leq I$:
   (a) If the owner of i is not corrupted, nothing is leaked: $\mathbf{1}_{\{\mathit{in}(i)\}}$
   (b) Otherwise the owner leaks the value of i:
      - $\mathit{in}(i) \rightsquigarrow \mathit{leakInput}(i)_{\mathit{real}} \coloneqq \mathit{in}(i)$
4. We randomly generate input shares for parties $0 \leq m \leq N$ (except for party $N+1$). For each input $0 \leq i \leq I$ and party $0 \leq m \leq N$:
   - $\mathit{in}(i) \rightsquigarrow \mathit{genShare}(i, m) \leftarrow \mathsf{rand}(\mathsf{bool})$
5. If applicable, we leak the input shares generated in the previous step. For each input $0 \leq i \leq I$ and party $0 \leq m \leq N$:
   (a) If the owner of i is not corrupted, nothing is leaked: $\mathbf{1}_{\{\mathit{genShare}(i,m)\}}$
   (b) Otherwise the owner leaks the share of i it generated for party m:
      - $\mathit{genShare}(i, m) \rightsquigarrow \mathit{leakGenShare}(i, m)_{\mathit{real}} \coloneqq \mathit{genShare}(i, m)$
6. We sum up the input shares generated for parties $0 \leq m \leq N$. For each input $0 \leq i \leq I$ and $0 \leq m \leq N$, we inductively define:
   (a) In the zero case we are summing up a single share:
      - $\mathit{genShare}(i, 0) \rightsquigarrow \mathit{genShare}_{\Sigma}(i, 0) \coloneqq \mathit{genShare}(i, 0)$
   (b) In the successor case we add the last share to the sum:
      - $\mathit{genShare}_{\Sigma}(i, m), \mathit{genShare}(i, m + 1) \rightsquigarrow$
        $\mathit{genShare}_{\Sigma}(i, m + 1) \coloneqq \mathit{genShare}_{\Sigma}(i, m) \oplus \mathit{genShare}(i, m + 1)$
7. We compute the input share for party $N + 1$. For each input $0 \leq i \leq I$:
   - $\mathit{in}(i), \mathit{genShare}_{\Sigma}(i, N) \rightsquigarrow \mathit{genShare}(i, N + 1) \coloneqq \mathit{in}(i) \oplus \mathit{genShare}_{\Sigma}(i, N)$
8. Everybody now records their input shares. For each input $0 \leq i \leq I$ and party $0 \leq m \leq N + 1$:
   - $\mathit{genShare}(i, m) \rightsquigarrow \mathit{inputShare}(i, m) \coloneqq \mathit{genShare}(i, m)$
9. If applicable, parties leak the input shares they received. For each input $0 \leq i \leq I$ and party $0 \leq m \leq N+1$:
   (a) If m is not corrupted or m is the owner of i, nothing is leaked: $\mathbf{1}_{\{\mathit{genShare}(i,m)\}}$
   (b) Otherwise we leak the share of i that party m received from the owner of i:
      - $\mathit{genShare}(i, m) \rightsquigarrow \mathit{leakInputShare}(i, m)_{\mathit{real}} \coloneqq \mathit{genShare}(i, m)$
10. At last we hide the intermediate channels
    - $\mathit{genShare}(i, n)$ for $0 \leq m \leq N + 1$
    - $\mathit{genShare}_{\Sigma}(i, n)$ for $0 \leq m \leq N$

5.2 $\mathcal{D}(C, L)$

We define $\mathcal{D}(C, L)$ for an arbitrary circuit C with L wires by induction on C.

5.2.1 $\mathcal{D}(\cdot, 0)$

If the circuit is empty we do nothing: $1_\emptyset$.

5.2.2 $\mathcal{D}(C; \mathsf{input}(i), L+1)$

If the last wire draws from the input i, the result is the composition of the program $\mathcal{D}(C, L)$ that computes each party's share on each wire $0 \le l < L$ and the program

- $\mathit{inputShare}(i, n) \rightsquigarrow \mathit{share}(L, n) := \mathit{inputShare}(i, n)$ for all $0 \le n \le N+1$

that sets the share for party n on wire L to the input share for party n of the input i.

5.2.3 $\mathcal{D}(C; l_1 \,\mathsf{XOR}\, l_2, L+1)$

If the last wire is an XOR gate of wires l_1 and l_2, the result is the composition of the program $\mathcal{D}(C, L)$ that computes each party's share on each wire $0 \le l < L$ and the program

- $\mathit{share}(l_1, n), \mathit{share}(l_2, n) \rightsquigarrow \mathit{share}(L, n) := \mathit{share}(l_1, n) \oplus \mathit{share}(l_2, n)$ for all $0 \le n \le N+1$

that sets the share for party n on wire L to be the sum of the shares for n on wires l_1 and l_2.

5.2.4 $\mathcal{D}(C; l_1 \,\mathsf{AND}\, l_2, L+1)$

If the last wire is an AND gate of wires l_1 and l_2, the result is the composition of the program $\mathcal{D}(C, L)$ that computes each party's share on each wire $0 \le l < L$ and the following program:

1. We generate a random bit for every OT exchange. For each pair of parties $0 \le n, m \le N+1$:
   (a) If $n < m$ and $\mathit{sender}(n, m)$ is true or $m < n$ and $\mathit{sender}(m, n)$ is false – so n is the sender and m is the receiver – we choose a bit at random:
      - $\mathit{share}(l_1, n), \mathit{share}(l_2, n) \rightsquigarrow \mathit{OTBitSen}(L, n, m) \leftarrow \mathsf{rand}(\mathsf{bool})$

   (b) Otherwise we do nothing: $1_{\{\mathit{share}(l_1,n), \mathit{share}(l_2,n)\}}$
2. Here we directly compute the bits we would have received as a result of the OT exchanges. For each pair of parties $0 \le n, m \le N+1$:
   (a) If $m < n$ and $\mathit{sender}(m, n)$ is true or $n < m$ and $\mathit{sender}(n, m)$ is false – so m is the sender and n is the receiver – we compute the bit we would have received from the OT exchange with party m:
      - $\mathit{OTBitSen}(L, m, n), \mathit{share}(l_1, n), \mathit{share}(l_2, n), \mathit{share}(l_1, m), \mathit{share}(l_2, m) \rightsquigarrow$
        $\mathit{OTBitRec}(L, n, m) := \big(\mathit{share}(l_1, m) * \mathit{share}(l_2, n) \oplus \mathit{share}(l_1, n) * \mathit{share}(l_2, m)\big) \oplus \mathit{OTBitSen}(L, m, n)$

   (b) Otherwise we do nothing: $1_{\{\mathit{share}(l_1,n), \mathit{share}(l_2,n), \mathit{share}(l_1,m), \mathit{share}(l_2,m)\}}$
3. We now compute a bit $\mathit{bit}(L, n, m)$ for each pair of parties $0 \le n, m \le N+1$ as follows:
   (a) If $m = n$, the resulting bit is the product of the n-th party's shares on wires l_1 and l_2:
      - $\mathit{share}(l_1, n), \mathit{share}(l_2, n) \rightsquigarrow \mathit{bit}(L, n, m) := \mathit{share}(l_1, n) * \mathit{share}(l_2, n)$

   (b) If $n < m$ and $\mathit{sender}(n, m)$ is true or $m < n$ and $\mathit{sender}(m, n)$ is false – so n is the sender and m is the receiver – the resulting bit is the random bit we generated in the first step for the OT exchange of party n with party m:
      - $\mathit{OTBitSen}(L, n, m), \mathit{share}(l_1, n), \mathit{share}(l_2, n) \rightsquigarrow \mathit{bit}(L, n, m) := \mathit{OTBitSen}(L, n, m)$

   (c) If $m < n$ and $\mathit{sender}(m, n)$ is true or $n < m$ and $\mathit{sender}(n, m)$ is false – so m is the sender and n is the receiver – the resulting bit is the bit we would have received as the result of the OT exchange between n and m:
      - $\mathit{OTBitRec}(L, m, n), \mathit{share}(l_1, n), \mathit{share}(l_2, n) \rightsquigarrow \mathit{bit}(L, n, m) := \mathit{OTBitRec}(L, n, m)$

4. If applicable, we leak the bits we generated for the OT exchanges. For each pair of parties $0 \leq n, m \leq N+1$:
   (a) If n is corrupted and either $n < m$ and $sender(n, m)$ is true or $m < n$ and $sender(m, n)$ is false – so n is the corrupted sender and m is the receiver – the value of $bit(L, n, m)$ is the randomly generated bit for the OT exchange between n and m, so we leak it:
      - $bit(L, n, m) \rightsquigarrow leakRandOTBitSen(L, n, m)_{real} := bit(L, n, m)$

   (b) Otherwise we leak nothing: $1_{\{bit(L,n,m)\}}$

5. If applicable, we leak the bits resulting from the OT exchanges. For each pair of parties $0 \leq n, m \leq N+1$:
   (a) If n corrupted and either $m < n$ and $sender(m, n)$ is true or $n < m$ and $sender(n, m)$ is false – so m is the sender and n is the corrupted receiver – the value of $bit(L, n, m)$ is the bit we would have received from the OT exchange between m and n, so we leak it:
      - $bit(L, m, n) \rightsquigarrow leakRandOTBitRec(L, m, n)_{real} := bit(L, n, m)$

   (b) Otherwise we leak nothing: $1_{\{bit(L,n,m)\}}$

6. We now sum up all the bits computed earlier to obtain the share of each party on wire L: for each party $0 \leq n \leq N+1$ and $0 \leq m \leq N+1$, the channel $bit_\Sigma(L, n, m)$ will hold the sum of all the bits $bit(L, n, -)$ where the last index is less than or equal to m. We define $bit_\Sigma(L, n, m)$ inductively as follows:
   (a) In the zero case we are summing a single bit:
      - $bit(L, n, 0) \rightsquigarrow bit_\Sigma(L, n, 0) := bit(L, n, 0)$

   (b) In the successor case we add the last bit to the sum:
      - $bit_\Sigma(L, n, m), bit(L, n, m+1) \rightsquigarrow bit_\Sigma(L, n, m+1) := bit_\Sigma(L, n, m) \oplus bit(L, n, m+1)$

7. We compute the share of each party on wire L. For each party $0 \leq n \leq N+1$:
   - $bit_\Sigma(L, n, N+1) \rightsquigarrow share(L, n) := bit_\Sigma(L, n, N+1)$

8. At last we hide the intermediate channels:
   - $OTBitSen(L, n, m)$ for $0 \leq m \leq N+1$ where $n < m$ and $sender(n, m)$ is true or $m < n$ and $sender(m, n)$ is false
   - $OTBitRec(L, n, m)$ for $0 \leq m \leq N+1$ where $m < n$ and $sender(m, n)$ is true or $n < m$ and $sender(n, m)$ is false
   - $bit(L, n, m)$ for $0 \leq m \leq N+1$
   - $bit_\Sigma(L, n, m)$ for $0 \leq m \leq N+1$

5.3 *Fin*

The code for *Fin* is as follows:

1. We first sum up the shares of all parties on each wire: for each wire $0 \leq k \leq K$ and $0 \leq m \leq N+1$, the channel $share_\Sigma(k, m)$ will hold the sum of all the shares $share(k, -)$ where the last index is less than or equal to m. We define $share_\Sigma(k, m)$ inductively as follows:
   (a) In the zero case we are summing up a single share:
      - $share(k, 0) \rightsquigarrow share_\Sigma(k, 0) := share(k, 0)$

   (b) In the successor case we add the last share to the sum:
      - $share_\Sigma(k, m), share(k, m+1) \rightsquigarrow share_\Sigma(k, m+1) := share_\Sigma(k, m) \oplus share(k, m+1)$

2. We let the value on each wire be the sum of all the shares. For each wire $0 \leq k \leq K$:
   - $share_\Sigma(k, N+1) \rightsquigarrow value(k) := share_\Sigma(k, N+1)$

3. We perform the outputs. For each wire $0 \le k \le K$ and party $0 \le n \le N+1$:
   (a) If k is not declared as an output, we do nothing: $\mathbf{1}_{\{value(k)\}}$
   (b) Otherwise we output the value on wire k:
      - $value(k) \rightsquigarrow out(k,n) \coloneqq value(k)$
4. If applicable, we leak each output share received. For each wire $0 \le k \le K$ and parties $0 \le n, m \le N+1$:
   (a) If k is not declared as an output or if n is not corrupted or if $m = n$, we do nothing: $\mathbf{1}_{\{share(k,m)\}}$
   (b) Otherwise party n leaks the share on wire k it received from party m:
      - $share(k,m) \rightsquigarrow leakOutputShare(k,n,m)_{real} \coloneqq share(k,m)$
5. Finally, we hide the intermediate channels
   - $value(k)$ for $0 \le k \le K$
   - $share_\Sigma(k,m)$ for $0 \le k \le K$ and $0 \le m \le N+1$

6 Cleaning up the real-world protocol: symmetry between senders and receivers

In the present form, the real-world protocol exhibits a certain form of asymmetry, which we now describe. Let us say parties n and m are computing an AND gate on wire L that has wires l_1, l_2 as inputs. Let n be the sender and m the receiver in the associated 1-out-of-4 OT exchange. In this scenario, n can compute its bit $bit(L,n,m)$ without waiting for the shares of m on l_1 and l_2 (n only needs its own shares on l_1 and l_2), whereas m needs both its own shares and the shares of n on l_1 and l_2 (as well as the value of $bit(L,n,m)$) to compute its bit $bit(L,m,n)$. In particular, if there is a party that only functions as a sender but never a receiver for any other party, then this party can finish computing its shares on *all* wires without needing the shares of any other party on any wire.

Here we formulate the real-world protocol in an equivalent form that corrects this asymmetry. We start by introducing channels $shareOk(k,n)$, which express that the share of party n on wire k has been computed. We recall that $\mathcal{D}(C, K+1)$ is the body of the real-world protocol; denote by $\mathcal{E}(C, K+1)$ the desired symmetric form of $\mathcal{D}(C, K+1)$ as defined below.

Let C be an arbitrary circuit with L wires. We now claim the following:

- Claim 1: The composition
  - *Init*
  - $\mathcal{D}(C,L)$
  - $\mathcal{B}(C,L)$
  - $share(k,n) \rightsquigarrow shareOk(k,n) \coloneqq \star$ for $0 \le k < L$ and $0 \le n \le N+1$

  rewrites to the composition
  - *Init*
  - $\mathcal{D}(C,L)$
  - $\mathcal{B}(C,L)$
  - $value(k) \rightsquigarrow shareOk(k,n) \coloneqq \star$ for $0 \le k < L$ and $0 \le n \le N+1$

  assuming the later hiding of the channels
  - $inputShare(k,n)$ for $0 \le k < L$ and $0 \le n \le N+1$
  - $share(k,n)$ for $0 \le k < L$ and $0 \le n \le N+1$
  - $value(k)$ for $0 \le k < L$

- $shareOk(k, n)$ for $0 \leq k < L$ and $0 \leq n \leq N+1$

- Claim 2: The composition
  - $Init$
  - $\mathcal{B}(C, L)$
  - $\mathcal{D}(C, L)$
  - $share(k, n) \rightsquigarrow shareOk(k, n) := \star$ for $0 \leq k < L$ and $0 \leq n \leq N+1$

  rewrites to the composition
  - $Init$
  - $\mathcal{E}(C, L)$
  - $\mathcal{B}(C, L)$
  - $share(k, n) \rightsquigarrow shareOk(k, n) := \star$ for $0 \leq k < L$ and $0 \leq n \leq N+1$

  assuming the later hiding of the channels
  - $inputShare(k, n)$ for $0 \leq k < L$ and $0 \leq n \leq N+1$
  - $share(k, n)$ for $0 \leq k < L$ and $0 \leq n \leq N+1$
  - $value(k)$ for $0 \leq k < L$
  - $shareOk(k, n)$ for $0 \leq k < L$ and $0 \leq n \leq N+1$

Using the second claim, the composition

- $Init$
- $\mathcal{D}(C, K+1)$
- $\mathcal{B}(C, L)$
- $share(k, n) \rightsquigarrow shareOk(k, n) := \star$ for $0 \leq k \leq K$ and $0 \leq n \leq N+1$

followed by the hiding of the channels

- $value(k)$ for $0 \leq k < L$
- $shareOk(k, n)$ for $0 \leq k \leq K$ and $0 \leq n \leq N+1$

rewrites assuming the later hiding of the channels

- $inputShare(k, n)$ for $0 \leq k \leq K$ and $0 \leq n \leq N+1$
- $share(k, n)$ for $0 \leq k \leq K$ and $0 \leq n \leq N+1$

to the composition of the programs

- $Init$
- $\mathcal{E}(C, K+1)$
- $\mathcal{B}(C, L)$
- $share(k, n) \rightsquigarrow shareOk(k, n) := \star$ for $0 \leq k \leq K$ and $0 \leq n \leq N+1$

followed by the hiding of the channels

- $value(k)$ for $0 \leq k < L$
- $shareOk(k, n)$ for $0 \leq k \leq K$ and $0 \leq n \leq N+1$

Thus the real-world protocol can be expressed equivalently as the composition

- *Init*
- $\mathcal{E}(C, K + 1)$
- *Fin*

followed by the hiding of the channels

- $\mathit{inputShare}(i, n)$ for $0 \leq i \leq I$ and $0 \leq n \leq N + 1$
- $\mathit{share}(k, n)$ for $0 \leq k \leq K$ and $0 \leq n \leq N + 1$

We now define $\mathcal{E}(C, L)$ and prove the two claims.

6.1 $\mathcal{E}(C, L)$

We define $\mathcal{E}(C, L)$ for an arbitrary circuit C with L wires by induction on C.

6.1.1 $\mathcal{E}(\cdot, 0)$

If the circuit is empty we do nothing: $\mathsf{1}_\emptyset$.

6.1.2 $\mathcal{E}(C;\, \mathsf{input}(i), L + 1)$

If the last wire draws from the input i, the result is the composition of the program $\mathcal{E}(C, L)$ and the program

- $\mathit{inputShare}(i, n) \rightsquigarrow \mathit{share}(L, n) \coloneqq \mathit{inputShare}(i, n)$ for all $0 \leq n \leq N + 1$

6.1.3 $\mathcal{E}(C;\, l_1 \,\mathsf{XOR}\, l_2, L + 1)$

If the last wire is an XOR gate of wires l_1 and l_2, the result is the composition of the program $\mathcal{E}(C, L)$ and the program

- $\mathit{share}(l_1, n), \mathit{share}(l_2, n) \rightsquigarrow \mathit{share}(L, n) \coloneqq \mathit{share}(l_1, n) \oplus \mathit{share}(l_2, n)$ for all $0 \leq n \leq N + 1$

6.1.4 $\mathcal{E}(C;\, l_1 \,\mathsf{AND}\, l_2, L + 1)$

If the last wire is an AND gate of wires l_1 and l_2, the result is the composition of the program $\mathcal{E}(C, L)$ and the following program:

1. We generate a random bit for every OT exchange. For each pair of parties $0 \leq n, m \leq N + 1$:
   (a) If $n < m$ and $\mathit{sender}(n, m)$ is true or $m < n$ and $\mathit{sender}(m, n)$ is false – so n is the sender and m is the receiver – we choose a bit at random:
      - $\mathit{share}(l_1, n), \mathit{share}(l_2, n), \mathit{share}(l_1, m), \mathit{share}(l_2, m) \rightsquigarrow \mathit{OTBitSen}(L, n, m) \leftarrow \mathsf{rand}(\mathsf{bool})$
   (b) Otherwise we do nothing: $\mathsf{1}_{\{\mathit{share}(l_1,n), \mathit{share}(l_2,n), \mathit{share}(l_1,m), \mathit{share}(l_2,m)\}}$
2. Here we directly compute the bits we would have received as a result of the OT exchanges. For each pair of parties $0 \leq n, m \leq N + 1$:
   (a) If $m < n$ and $\mathit{sender}(m, n)$ is true or $n < m$ and $\mathit{sender}(n, m)$ is false – so m is the sender and n is the receiver – we compute the bit we would have received from the OT exchange with party m:
      - $\mathit{OTBitSen}(L, m, n), \mathit{share}(l_1, n), \mathit{share}(l_2, n), \mathit{share}(l_1, m), \mathit{share}(l_2, m) \rightsquigarrow$ $\mathit{OTBitRec}(L, n, m) \coloneqq \big(\mathit{share}(l_1, m) * \mathit{share}(l_2, n) \oplus \mathit{share}(l_1, n) * \mathit{share}(l_2, m)\big) \oplus \mathit{OTBitSen}(L, m, n)$
   (b) Otherwise we do nothing: $\mathsf{1}_{\{\mathit{share}(l_1,n), \mathit{share}(l_2,n), \mathit{share}(l_1,m), \mathit{share}(l_2,m)\}}$
3. We now compute a bit $\mathit{bit}(L, n, m)$ for each pair of parties $0 \leq n, m \leq N + 1$ as follows:

(a) If $m = n$, the resulting bit is the product of the n-th party's shares on wires l_1 and l_2:

- $\mathit{share}(l_1, n), \mathit{share}(l_2, n) \rightsquigarrow \mathit{bit}(L, n, m) := \mathit{share}(l_1, n) * \mathit{share}(l_2, n)$

(b) If $n < m$ and $\mathit{sender}(n, m)$ is true or $m < n$ and $\mathit{sender}(m, n)$ is false – so n is the sender and m is the receiver – the resulting bit is the random bit we generated in the first step for the OT exchange of party n with party m:

- $\mathit{OTBitSen}(L, n, m), \mathit{share}(l_1, n), \mathit{share}(l_2, n) \rightsquigarrow \mathit{bit}(L, n, m) := \mathit{OTBitSen}(L, n, m)$

(c) If $m < n$ and $\mathit{sender}(m, n)$ is true or $n < m$ and $\mathit{sender}(n, m)$ is false – so m is the sender and n is the receiver – the resulting bit is the bit we would have received as the result of the OT exchange between n and m:

- $\mathit{OTBitRec}(L, m, n), \mathit{share}(l_1, n), \mathit{share}(l_2, n) \rightsquigarrow \mathit{bit}(L, n, m) := \mathit{OTBitRec}(L, n, m)$

4. If applicable, we leak the bits we generated for the OT exchanges. For each pair of parties $0 \leq n, m \leq N+1$:

(a) If n is corrupted and either $n < m$ and $\mathit{sender}(n, m)$ is true or $m < n$ and $\mathit{sender}(m, n)$ is false – so n is the corrupted sender and m is the receiver – the value of $\mathit{bit}(L, n, m)$ is the randomly generated bit for the OT exchange between n and m, so we leak it:

- $\mathit{bit}(L, n, m) \rightsquigarrow \mathit{leakRandOTBitSen}(L, n, m)_{\mathit{real}} := \mathit{bit}(L, n, m)$

(b) Otherwise we leak nothing: $1_{\{\mathit{bit}(L,n,m)\}}$

5. If applicable, we leak the bits resulting from the OT exchanges. For each pair of parties $0 \leq n, m \leq N+1$:

(a) If n corrupted and either $m < n$ and $\mathit{sender}(m, n)$ is true or $n < m$ and $\mathit{sender}(n, m)$ is false – so m is the sender and n is the corrupted receiver – the value of $\mathit{bit}(L, n, m)$ is the bit we would have received from the OT exchange between m and n, so we leak it:

- $\mathit{bit}(L, m, n) \rightsquigarrow \mathit{leakRandOTBitRec}(L, m, n)_{\mathit{real}} := \mathit{bit}(L, n, m)$

(b) Otherwise we leak nothing: $1_{\{\mathit{bit}(L,n,m)\}}$

6. We now sum up all the bits computed earlier to obtain the share of each party on wire L: for each party $0 \leq n \leq N+1$ and $0 \leq m \leq N+1$, the channel $\mathit{bit}_\Sigma(L, n, m)$ will hold the sum of all the bits $\mathit{bit}(L, n, -)$ where the last index is less than or equal to m. We define $\mathit{bit}_\Sigma(L, n, m)$ inductively as follows:

(a) In the zero case we are summing a single bit:

- $\mathit{bit}(L, n, 0) \rightsquigarrow \mathit{bit}_\Sigma(L, n, 0) := \mathit{bit}(L, n, 0)$

(b) In the successor case we add the last bit to the sum:

- $\mathit{bit}_\Sigma(L, n, m), \mathit{bit}(L, n, m+1) \rightsquigarrow \mathit{bit}_\Sigma(L, n, m+1) := \mathit{bit}_\Sigma(L, n, m) \oplus \mathit{bit}(L, n, m+1)$

7. We compute the share of each party on wire L. For each party $0 \leq n \leq N+1$:

- $\mathit{bit}_\Sigma(L, n, N+1) \rightsquigarrow \mathit{share}(L, n) := \mathit{bit}_\Sigma(L, n, N+1)$

8. At last we hide the intermediate channels:

- $\mathit{OTBitSen}(L, n, m)$ for $0 \leq m \leq N+1$ where $n < m$ and $\mathit{sender}(n, m)$ is true or $m < n$ and $\mathit{sender}(m, n)$ is false
- $\mathit{OTBitRec}(L, n, m)$ for $0 \leq m \leq N+1$ where $m < n$ and $\mathit{sender}(m, n)$ is true or $n < m$ and $\mathit{sender}(n, m)$ is false
- $\mathit{bit}(L, n, m)$ for $0 \leq m \leq N+1$
- $\mathit{bit}_\Sigma(L, n, m)$ for $0 \leq m \leq N+1$

6.2 Proving claim 1

We proceed induction on the circuit C.

6.2.1 Proving claim 1 for $\cdot$ and 0

If the circuit is empty we have nothing to prove.

6.2.2 Proving claim 1 for C; $\mathsf{input}(i)$ and $L+1$

If the last wire draws from the input i, we want to prove that the composition

- Init
- $\mathcal{D}(C, L)$
- $\mathcal{B}(C, L)$
- $\mathit{inputShare}(i, n) \rightsquigarrow \mathit{share}(L, n) \coloneqq \mathit{inputShare}(i, n)$ for all $0 \leq n \leq N+1$
- $\mathit{in}(i) \rightsquigarrow \mathit{value}(L) \coloneqq \mathit{in}(i)$
- $\mathit{share}(k, n) \rightsquigarrow \mathit{shareOk}(k, n) \coloneqq \star$ for $0 \leq k < L$ and $0 \leq n \leq N+1$
- $\mathit{share}(L, n) \rightsquigarrow \mathit{shareOk}(L, n) \coloneqq \star$ for $0 \leq n \leq N+1$

rewrites to the composition

- Init
- $\mathcal{D}(C, L)$
- $\mathcal{B}(C, L)$
- $\mathit{inputShare}(i, n) \rightsquigarrow \mathit{share}(L, n) \coloneqq \mathit{inputShare}(i, n)$ for all $0 \leq n \leq N+1$
- $\mathit{in}(i) \rightsquigarrow \mathit{value}(L) \coloneqq \mathit{in}(i)$
- $\mathit{value}(k) \rightsquigarrow \mathit{shareOk}(k, n) \coloneqq \star$ for $0 \leq k < L$ and $0 \leq n \leq N+1$
- $\mathit{value}(L) \rightsquigarrow \mathit{shareOk}(L, n) \coloneqq \star$ for $0 \leq n \leq N+1$

assuming the later hiding of the channels

- $\mathit{inputShare}(i, n)$ for $0 \leq i \leq I$ and $0 \leq n \leq N+1$
- $\mathit{share}(k, n)$ for $0 \leq k \leq L$ and $0 \leq n \leq N+1$
- $\mathit{value}(k)$ for $0 \leq k \leq L$
- $\mathit{shareOk}(k, n)$ for $0 \leq k \leq L$ and $0 \leq n \leq N+1$

To this end, assume the later hiding of the above channels. Then the composition

- Init
- $\mathcal{D}(C, L)$
- $\mathcal{B}(C, L)$
- $\mathit{inputShare}(i, n) \rightsquigarrow \mathit{share}(L, n) \coloneqq \mathit{inputShare}(i, n)$ for all $0 \leq n \leq N+1$
- $\mathit{in}(i) \rightsquigarrow \mathit{value}(L) \coloneqq \mathit{in}(i)$
- $\mathit{share}(k, n) \rightsquigarrow \mathit{shareOk}(k, n) \coloneqq \star$ for $0 \leq k < L$ and $0 \leq n \leq N+1$
- $\mathit{share}(L, n) \rightsquigarrow \mathit{shareOk}(L, n) \coloneqq \star$ for $0 \leq n \leq N+1$

rewrites to the composition

- $\mathcal{R}_1(L)$
- $\mathcal{D}(C, L)$
- $\mathcal{B}(C, L)$
- $\mathit{in}(i) \rightsquigarrow \mathit{value}(L) := \mathit{in}(i)$
- $\mathit{share}(k, n) \rightsquigarrow \mathit{shareOk}(k, n) := \star$ for $0 \leq k < L$ and $0 \leq n \leq N + 1$

where $\mathcal{R}_1(L)$ is the following program:

1. We leak the knowledge that the inputs have been received. For each input $0 \leq i \leq I$:
2. The owner of i informs the adversary that input i has been received:
   - $\mathit{in}(i) \rightsquigarrow \mathit{leakInputOk}(i)_{\mathit{real}} := \star$
3. If applicable, we leak the value of an input. For each input $0 \leq i \leq I$:
   (a) If the owner of i is not corrupted, nothing is leaked: $\mathsf{1}_{\{\mathit{in}(i)\}}$
   (b) Otherwise the owner leaks the value of i:
      - $\mathit{in}(i) \rightsquigarrow \mathit{leakInput}(i)_{\mathit{real}} := \mathit{in}(i)$
4. We randomly generate input shares for parties $0 \leq m \leq N$ (except for party $N+1$). For each input $0 \leq i \leq I$ and party $0 \leq m \leq N$:
   - $\mathit{in}(i) \rightsquigarrow \mathit{genShare}(i, m) \leftarrow \mathsf{rand}(\mathsf{bool})$
5. We record that the input shares for parties $0 \leq m \leq N$ have been generated. For each input $0 \leq i \leq I$ and party $0 \leq m \leq N$:
   - $\mathit{genShare}(i, m) \rightsquigarrow \mathit{genShareOk}(i, m) := \star$
6. If applicable, we leak the input shares generated in the previous step. For each input $0 \leq i \leq I$ and party $0 \leq m \leq N$:
   (a) If the owner of i is not corrupted, nothing is leaked: $\mathsf{1}_{\{\mathit{genShare}(i,m)\}}$
   (b) Otherwise the owner leaks the share of i it generated for party m:
      - $\mathit{genShare}(i, m) \rightsquigarrow \mathit{leakGenShare}(i, m)_{\mathit{real}} := \mathit{genShare}(i, m)$
7. We sum up the input shares generated for parties $0 \leq m \leq N$. For each input $0 \leq i \leq I$ and $0 \leq m \leq N$, we inductively define:
   (a) In the zero case we are summing up a single share:
      - $\mathit{genShare}(i, 0) \rightsquigarrow \mathit{genShare}_\Sigma(i, 0) := \mathit{genShare}(i, 0)$
   (b) In the successor case we add the last share to the sum:
      - $\mathit{genShare}_\Sigma(i, m), \mathit{genShare}(i, m + 1) \rightsquigarrow$
        $\mathit{genShare}_\Sigma(i, m + 1) := \mathit{genShare}_\Sigma(i, m) \oplus \mathit{genShare}(i, m + 1)$
8. We record that the sum of the input shares generated for parties $0 \leq m \leq N$ has been computed. For each input $0 \leq i \leq I$ and $0 \leq m \leq N$:
   - $\mathit{genShare}_\Sigma(i, m) \rightsquigarrow \mathit{genShareOk}_\Sigma(i, m) := \star$
9. We compute the input share for party $N + 1$. For each input $0 \leq i \leq I$:
   - $\mathit{in}(i), \mathit{genShare}_\Sigma(i, N) \rightsquigarrow \mathit{genShare}(i, N + 1) := \mathit{in}(i) \oplus \mathit{genShare}_\Sigma(i, N)$

10. We record that the input share for party $N+1$ has been computed. For each input $0 \le i \le I$:
    - $\mathit{genShare}(i, N+1) \rightsquigarrow \mathit{genShareOk}(i, N+1) \coloneqq \star$
11. Everybody now records their input shares. For each input $0 \le i \le I$ and party $0 \le m \le N+1$:
    - $\mathit{genShare}(i, m) \rightsquigarrow \mathit{inputShare}(i, m) \coloneqq \mathit{genShare}(i, m)$
12. We record that the input shares have been computed. For each input $0 \le i \le I$ and party $0 \le m \le N+1$:
    - $\mathit{inputShare}(i, m) \rightsquigarrow \mathit{inputShareOk}(i, m) \coloneqq \star$
13. If applicable, parties leak the input shares they received. For each input $0 \le i \le I$ and party $0 \le m \le N+1$:
    (a) If m is not corrupted or m is the owner of i, nothing is leaked: $1_{\{\mathit{genShare}(i,m)\}}$
    (b) Otherwise we leak the share of i that party m received from the owner of i:
       - $\mathit{genShare}(i, m) \rightsquigarrow \mathit{leakInputShare}(i, m)_{\mathit{real}} \coloneqq \mathit{genShare}(i, m)$
14. The share of each party on wire L is just the party's input share of i. For each party $0 \le n \le N+1$:
    - $\mathit{inputShare}(i, n) \rightsquigarrow \mathit{share}(L, n) \coloneqq \mathit{inputShare}(i, n)$
15. We record that the shares on wire L have been computed. For each party $0 \le n \le N+1$:
    - $\mathit{share}(L, n) \rightsquigarrow \mathit{shareOk}(L, n) \coloneqq \star$
16. At last we hide the intermediate channels
    - $\mathit{genShare}(i, n)$ for $0 \le m \le N+1$
    - $\mathit{genShareOk}(i, n)$ for $0 \le m \le N+1$
    - $\mathit{genShare}_{\Sigma}(i, n)$ for $0 \le m \le N$
    - $\mathit{genShareOk}_{\Sigma}(i, n)$ for $0 \le m \le N$
    - $\mathit{inputShareOk}(i, n)$ for $0 \le m \le N+1$

The program $\mathcal{R}_1(L)$ rewrites to the program $\mathcal{R}_2(L)$ below:

1. We leak the knowledge that the inputs have been received. For each input $0 \le i \le I$:
2. The owner of i informs the adversary that input i has been received:
    - $\mathit{in}(i) \rightsquigarrow \mathit{leakInputOk}(i)_{\mathit{real}} \coloneqq \star$
3. If applicable, we leak the value of an input. For each input $0 \le i \le I$:
    (a) If the owner of i is not corrupted, nothing is leaked: $1_{\{\mathit{in}(i)\}}$
    (b) Otherwise the owner leaks the value of i:
       - $\mathit{in}(i) \rightsquigarrow \mathit{leakInput}(i)_{\mathit{real}} \coloneqq \mathit{in}(i)$
4. We randomly generate input shares for parties $0 \le m \le N$ (except for party $N+1$). For each input $0 \le i \le I$ and party $0 \le m \le N$:
    - $\mathit{in}(i) \rightsquigarrow \mathit{genShare}(i, m) \leftarrow \mathsf{rand}(\mathsf{bool})$
5. We record that the input shares for parties $0 \le m \le N$ have been generated. For each input $0 \le i \le I$ and party $0 \le m \le N$:
    - $\mathit{in}(i) \rightsquigarrow \mathit{genShareOk}(i, m) \coloneqq \star$
6. If applicable, we leak the input shares generated in the previous step. For each input $0 \le i \le I$ and party $0 \le m \le N$:

(a) If the owner of i is not corrupted, nothing is leaked: $1_{\{genShare(i,m)\}}$

(b) Otherwise the owner leaks the share of i it generated for party m:

- $genShare(i,m) \rightsquigarrow leakGenShare(i,m)_{real} := genShare(i,m)$

7. We sum up the input shares generated for parties $0 \le m \le N$. For each input $0 \le i \le I$ and $0 \le m \le N$, we inductively define:

(a) In the zero case we are summing up a single share:

- $genShare(i,0) \rightsquigarrow genShare_{\Sigma}(i,0) := genShare(i,0)$

(b) In the successor case we add the last share to the sum:

- $genShare_{\Sigma}(i,m), genShare(i,m+1) \rightsquigarrow genShare_{\Sigma}(i,m+1) := genShare_{\Sigma}(i,m) \oplus genShare(i,m+1)$

8. We record that the sum of the input shares generated for parties $0 \le m \le N$ has been computed. For each input $0 \le i \le I$ and $0 \le m \le N$:

(a) In the zero case we have:

- $genShareOk(i,0) \rightsquigarrow genShare_{\Sigma}(i,0) := \star$

(b) In the successor case we have:

- $genShareOk_{\Sigma}(i,m), genShareOk(i,m+1) \rightsquigarrow genShareOk_{\Sigma}(i,m+1) := \star$

9. We compute the input share for party $N+1$. For each input $0 \le i \le I$:

- $in(i), genShare_{\Sigma}(i,N) \rightsquigarrow genShare(i,N+1) := in(i) \oplus genShare_{\Sigma}(i,N)$

10. We record that the input share for party $N+1$ has been computed. For each input $0 \le i \le I$:

- $in(i), genShareOk_{\Sigma}(i,N) \rightsquigarrow genShareOk(i,N+1) := \star$

11. Everybody now records their input shares. For each input $0 \le i \le I$ and party $0 \le m \le N+1$:

- $genShare(i,m) \rightsquigarrow inputShare(i,m) := genShare(i,m)$

12. We record that the input shares have been computed. For each input $0 \le i \le I$ and party $0 \le m \le N+1$:

- $genShareOk(i,m) \rightsquigarrow inputShareOk(i,m) := \star$

13. If applicable, parties leak the input shares they received. For each input $0 \le i \le I$ and party $0 \le m \le N+1$:

(a) If m is not corrupted or m is the owner of i, nothing is leaked: $1_{\{genShare(i,m)\}}$

(b) Otherwise we leak the share of i that party m received from the owner of i:

- $genShare(i,m) \rightsquigarrow leakInputShare(i,m)_{real} := genShare(i,m)$

14. The share of each party on wire L is just the party's input share of i. For each party $0 \le n \le N+1$:

- $inputShare(i,n) \rightsquigarrow share(L,n) := \star$

15. We record that the shares on wire L have been computed. For each party $0 \le n \le N+1$:

- $inputShareOk(i,n) \rightsquigarrow shareOk(L,n) := \star$

16. At last we hide the intermediate channels

- $genShare(i,n)$ for $0 \le m \le N+1$
- $genShareOk(i,n)$ for $0 \le m \le N+1$
- $genShare_{\Sigma}(i,n)$ for $0 \le m \le N$
- $genShareOk_{\Sigma}(i,n)$ for $0 \le m \le N$

- $\mathit{inputShareOk}(i, n)$ for $0 \leq m \leq N+1$

The program $\mathcal{R}_2(L)$ further rewrites to the program $\mathcal{R}_3(L)$ below:

1. We leak the knowledge that the inputs have been received. For each input $0 \leq i \leq I$:
2. The owner of i informs the adversary that input i has been received:
   - $\mathit{in}(i) \rightsquigarrow \mathit{leakInputOk}(i)_{\mathit{real}} := \star$
3. If applicable, we leak the value of an input. For each input $0 \leq i \leq I$:
   (a) If the owner of i is not corrupted, nothing is leaked: $\mathsf{1}_{\{\mathit{in}(i)\}}$
   (b) Otherwise the owner leaks the value of i:
      - $\mathit{in}(i) \rightsquigarrow \mathit{leakInput}(i)_{\mathit{real}} := \mathit{in}(i)$
4. We randomly generate input shares for parties $0 \leq m \leq N$ (except for party $N+1$). For each input $0 \leq i \leq I$ and party $0 \leq m \leq N$:
   - $\mathit{in}(i) \rightsquigarrow \mathit{genShare}(i, m) \leftarrow \mathsf{rand}(\mathsf{bool})$
5. We record that the input shares for parties $0 \leq m \leq N$ have been generated. For each input $0 \leq i \leq I$ and party $0 \leq m \leq N$:
   - $\mathit{in}(i) \rightsquigarrow \mathit{genShareOk}(i, m) := \star$
6. If applicable, we leak the input shares generated in the previous step. For each input $0 \leq i \leq I$ and party $0 \leq m \leq N$:
   (a) If the owner of i is not corrupted, nothing is leaked: $\mathsf{1}_{\{\mathit{genShare}(i,m)\}}$
   (b) Otherwise the owner leaks the share of i it generated for party m:
      - $\mathit{genShare}(i, m) \rightsquigarrow \mathit{leakGenShare}(i, m)_{\mathit{real}} := \mathit{genShare}(i, m)$
7. We sum up the input shares generated for parties $0 \leq m \leq N$. For each input $0 \leq i \leq I$ and $0 \leq m \leq N$, we inductively define:
   (a) In the zero case we are summing up a single share:
      - $\mathit{genShare}(i, 0) \rightsquigarrow \mathit{genShare}_{\Sigma}(i, 0) := \mathit{genShare}(i, 0)$
   (b) In the successor case we add the last share to the sum:
      - $\mathit{genShare}_{\Sigma}(i, m), \mathit{genShare}(i, m+1) \rightsquigarrow$
        $\mathit{genShare}_{\Sigma}(i, m+1) := \mathit{genShare}_{\Sigma}(i, m) \oplus \mathit{genShare}(i, m+1)$
8. We record that the sum of the input shares generated for parties $0 \leq m \leq N$ has been computed. For each input $0 \leq i \leq I$ and $0 \leq m \leq N$:
   - $\mathit{in}(i) \rightsquigarrow \mathit{genShare}_{\Sigma}(i, m) := \star$
9. We compute the input share for party $N+1$. For each input $0 \leq i \leq I$:
   - $\mathit{in}(i), \mathit{genShare}_{\Sigma}(i, N) \rightsquigarrow \mathit{genShare}(i, N+1) := \mathit{in}(i) \oplus \mathit{genShare}_{\Sigma}(i, N)$
10. We record that the input share for party $N+1$ has been computed. For each input $0 \leq i \leq I$:
    - $\mathit{in}(i) \rightsquigarrow \mathit{genShareOk}(i, N+1) := \star$
11. Everybody now records their input shares. For each input $0 \leq i \leq I$ and party $0 \leq m \leq N+1$:
    - $\mathit{genShare}(i, m) \rightsquigarrow \mathit{inputShare}(i, m) := \mathit{genShare}(i, m)$

12. We record that the input shares have been computed. For each input $0 \le i \le I$ and party $0 \le m \le N+1$:
    - $\mathit{in}(i) \rightsquigarrow \mathit{inputShareOk}(i,m) \coloneqq \star$
13. If applicable, parties leak the input shares they received. For each input $0 \le i \le I$ and party $0 \le m \le N+1$:
    (a) If m is not corrupted or m is the owner of i, nothing is leaked: $\mathbb{1}_{\{\mathit{genShare}(i,m)\}}$
    (b) Otherwise we leak the share of i that party m received from the owner of i:
        - $\mathit{genShare}(i,m) \rightsquigarrow \mathit{leakInputShare}(i,m)_{\mathit{real}} \coloneqq \mathit{genShare}(i,m)$
14. The share of each party on wire L is just the party's input share of i. For each party $0 \le n \le N+1$:
    - $\mathit{inputShare}(i,n) \rightsquigarrow \mathit{share}(L,n) \coloneqq \mathit{inputShare}(i,n)$
15. We record that the shares on wire L have been computed. For each party $0 \le n \le N+1$:
    - $\mathit{in}(i) \rightsquigarrow \mathit{shareOk}(L,n) \coloneqq \star$
16. At last we hide the intermediate channels
    - $\mathit{genShare}(i,n)$ for $0 \le m \le N+1$
    - $\mathit{genShareOk}(i,n)$ for $0 \le m \le N+1$
    - $\mathit{genShare}_{\Sigma}(i,n)$ for $0 \le m \le N$
    - $\mathit{genShareOk}_{\Sigma}(i,n)$ for $0 \le m \le N$
    - $\mathit{inputShareOk}(i,n)$ for $0 \le m \le N+1$

The composition

- $\mathcal{R}_3(L)$
- $\mathcal{D}(C,L)$
- $\mathcal{B}(C,L)$
- $\mathit{in}(i) \rightsquigarrow \mathit{value}(L) \coloneqq \mathit{in}(i)$
- $\mathit{share}(k,n) \rightsquigarrow \mathit{shareOk}(k,n) \coloneqq \star$ for $0 \le k < L$ and $0 \le n \le N+1$

now rewrites to the composition

- *Init*
- $\mathcal{D}(C,L)$
- $\mathcal{B}(C,L)$
- $\mathit{inputShare}(i,n) \rightsquigarrow \mathit{share}(L,n) \coloneqq \mathit{inputShare}(i,n)$ for $0 \le n \le N+1$
- $\mathit{in}(i) \rightsquigarrow \mathit{value}(L) \coloneqq \mathit{in}(i)$
- $\mathit{share}(k,n) \rightsquigarrow \mathit{shareOk}(k,n) \coloneqq \star$ for $0 \le k < L$ and $0 \le n \le N+1$
- $\mathit{value}(L) \rightsquigarrow \mathit{shareOk}(L,n) \coloneqq \star$

Using the induction hypothesis, the above rewrites to the composition

- *Init*
- $\mathcal{D}(C,L)$
- $\mathcal{B}(C,L)$

- $\mathit{inputShare}(i,n) \rightsquigarrow \mathit{share}(L,n) \coloneqq \mathit{inputShare}(i,n)$ for $0 \leq n \leq N+1$
- $\mathit{in}(i) \rightsquigarrow \mathit{value}(L) \coloneqq \mathit{in}(i)$
- $\mathit{value}(k) \rightsquigarrow \mathit{shareOk}(k,n) \coloneqq \star$ for $0 \leq k < L$ and $0 \leq n \leq N+1$
- $\mathit{value}(L) \rightsquigarrow \mathit{shareOk}(L,n) \coloneqq \star$

This finishes the proof.

6.2.3 Proving claim 1 for $C; l_1 \,\mathsf{XOR}\, l_2$ and $L+1$

If the last wire is an XOR gate of wires l_1 and l_2, we want to prove that the composition

- Init
- $\mathcal{D}(C,L)$
- $\mathcal{B}(C,L)$
- $\mathit{share}(l_1,n), \mathit{share}(l_2,n) \rightsquigarrow \mathit{share}(L,n) \coloneqq \mathit{share}(l_1,n) \oplus \mathit{share}(l_2,n)$ for all $0 \leq n \leq N+1$
- $\mathit{value}(l_1), \mathit{value}(l_2) \rightsquigarrow \mathit{value}(L) \coloneqq \mathit{value}(l_1) \oplus \mathit{value}(l_2)$
- $\mathit{share}(k,n) \rightsquigarrow \mathit{shareOk}(k,n) \coloneqq \star$ for $0 \leq k < L$ and $0 \leq n \leq N+1$
- $\mathit{share}(L,n) \rightsquigarrow \mathit{shareOk}(L,n) \coloneqq \star$ for $0 \leq n \leq N+1$

rewrites to the composition

- Init
- $\mathcal{D}(C,L)$
- $\mathcal{B}(C,L)$
- $\mathit{share}(l_1,n), \mathit{share}(l_2,n) \rightsquigarrow \mathit{share}(L,n) \coloneqq \mathit{share}(l_1,n) \oplus \mathit{share}(l_2,n)$ for all $0 \leq n \leq N+1$
- $\mathit{value}(l_1), \mathit{value}(l_2) \rightsquigarrow \mathit{value}(L) \coloneqq \mathit{value}(l_1) \oplus \mathit{value}(l_2)$
- $\mathit{value}(k) \rightsquigarrow \mathit{shareOk}(k,n) \coloneqq \star$ for $0 \leq k < L$ and $0 \leq n \leq N+1$
- $\mathit{value}(L) \rightsquigarrow \mathit{shareOk}(L,n) \coloneqq \star$ for $0 \leq n \leq N+1$

assuming the later hiding of the channels

- $\mathit{inputShare}(i,n)$ for $0 \leq i \leq I$ and $0 \leq n \leq N+1$
- $\mathit{share}(k,n)$ for $0 \leq k \leq L$ and $0 \leq n \leq N+1$
- $\mathit{value}(k)$ for $0 \leq k \leq L$
- $\mathit{shareOk}(k,n)$ for $0 \leq k \leq L$ and $0 \leq n \leq N+1$

To this end, assume the later hiding of the above channels. Then the composition

- Init
- $\mathcal{D}(C,L)$
- $\mathcal{B}(C,L)$
- $\mathit{share}(l_1,n), \mathit{share}(l_2,n) \rightsquigarrow \mathit{share}(L,n) \coloneqq \mathit{share}(l_1,n) \oplus \mathit{share}(l_2,n)$ for all $0 \leq n \leq N+1$

- $\mathit{value}(l_1), \mathit{value}(l_2) \rightsquigarrow \mathit{value}(L) \coloneqq \mathit{value}(l_1) \oplus \mathit{value}(l_2)$
- $\mathit{share}(k,n) \rightsquigarrow \mathit{shareOk}(k,n) \coloneqq \star$ for $0 \leq k < L$ and $0 \leq n \leq N+1$
- $\mathit{share}(L,n) \rightsquigarrow \mathit{shareOk}(L,n) \coloneqq \star$ for $0 \leq n \leq N+1$

rewrites to the composition

- Init
- $\mathcal{D}(C,L)$
- $\mathcal{B}(C,L)$
- $\mathit{share}(l_1,n), \mathit{share}(l_2,n) \rightsquigarrow \mathit{share}(L,n) \coloneqq \mathit{share}(l_1,n) \oplus \mathit{share}(l_2,n)$ for all $0 \leq n \leq N+1$
- $\mathit{value}(l_1), \mathit{value}(l_2) \rightsquigarrow \mathit{value}(L) \coloneqq \mathit{value}(l_1) \oplus \mathit{value}(l_2)$
- $\mathit{share}(k,n) \rightsquigarrow \mathit{shareOk}(k,n) \coloneqq \star$ for $0 \leq k < L$ and $0 \leq n \leq N+1$
- $\mathit{shareOk}(l_1,n), \mathit{shareOk}(l_2,n) \rightsquigarrow \mathit{shareOk}(L,n) \coloneqq \star$ for $0 \leq n \leq N+1$

Using the induction hypothesis, the above rewrites to the composition

- Init
- $\mathcal{D}(C,L)$
- $\mathcal{B}(C,L)$
- $\mathit{share}(l_1,n), \mathit{share}(l_2,n) \rightsquigarrow \mathit{share}(L,n) \coloneqq \mathit{share}(l_1,n) \oplus \mathit{share}(l_2,n)$ for all $0 \leq n \leq N+1$
- $\mathit{value}(l_1), \mathit{value}(l_2) \rightsquigarrow \mathit{value}(L) \coloneqq \mathit{value}(l_1) \oplus \mathit{value}(l_2)$
- $\mathit{value}(k) \rightsquigarrow \mathit{shareOk}(k,n) \coloneqq \star$ for $0 \leq k < L$ and $0 \leq n \leq N+1$
- $\mathit{shareOk}(l_1,n), \mathit{shareOk}(l_2,n) \rightsquigarrow \mathit{shareOk}(L,n) \coloneqq \star$ for $0 \leq n \leq N+1$

which further rewrites to the composition

- Init
- $\mathcal{D}(C,L)$
- $\mathcal{B}(C,L)$
- $\mathit{share}(l_1,n), \mathit{share}(l_2,n) \rightsquigarrow \mathit{share}(L,n) \coloneqq \mathit{share}(l_1,n) \oplus \mathit{share}(l_2,n)$ for all $0 \leq n \leq N+1$
- $\mathit{value}(l_1), \mathit{value}(l_2) \rightsquigarrow \mathit{value}(L) \coloneqq \mathit{value}(l_1) \oplus \mathit{value}(l_2)$
- $\mathit{value}(k) \rightsquigarrow \mathit{shareOk}(k,n) \coloneqq \star$ for $0 \leq k < L$ and $0 \leq n \leq N+1$
- $\mathit{value}(L) \rightsquigarrow \mathit{shareOk}(L,n) \coloneqq \star$ for $0 \leq n \leq N+1$

This finishes the proof.

6.2.4 Proving claim 1 for $C; l_1 \,\mathsf{AND}\, l_2$ and $L+1$

If the last wire is an AND gate of wires l_1 and l_2, let $\mathcal{Q}_1(l_1, l_2, L)$ be the following program:

1. We generate a random bit for every OT exchange. For each pair of parties $0 \leq n, m \leq N+1$:
   (a) If $n < m$ and $\mathit{sender}(n, m)$ is true or $m < n$ and $\mathit{sender}(m, n)$ is false – so n is the sender and m is the receiver – we choose a bit at random:
      - $\mathit{share}(l_1, n), \mathit{share}(l_2, n) \rightsquigarrow \mathit{OTBitSen}(L, n, m) \leftarrow \mathsf{rand}(\mathsf{bool})$

   (b) Otherwise we do nothing: $\mathbb{1}_{\{\mathit{share}(l_1,n),\mathit{share}(l_2,n)\}}$
2. Here we directly compute the bits we would have received as a result of the OT exchanges. For each pair of parties $0 \leq n, m \leq N+1$:
   (a) If $m < n$ and $\mathit{sender}(m, n)$ is true or $n < m$ and $\mathit{sender}(n, m)$ is false – so m is the sender and n is the receiver – we compute the bit we would have received from the OT exchange with party m:
      - $\mathit{OTBitSen}(L, m, n), \mathit{share}(l_1, n), \mathit{share}(l_2, n), \mathit{share}(l_1, m), \mathit{share}(l_2, m) \rightsquigarrow$ $\mathit{OTBitRec}(L, n, m) := \big(\mathit{share}(l_1, m) * \mathit{share}(l_2, n) \oplus \mathit{share}(l_1, n) * \mathit{share}(l_2, m)\big) \oplus \mathit{OTBitSen}(L, m, n)$

   (b) Otherwise we do nothing: $\mathbb{1}_{\{\mathit{share}(l_1,n),\mathit{share}(l_2,n),\mathit{share}(l_1,m),\mathit{share}(l_2,m)\}}$
3. We now compute a bit $\mathit{bit}(L, n, m)$ for each pair of parties $0 \leq n, m \leq N+1$ as follows:
   (a) If $m = n$, the resulting bit is the product of the n-th party's shares on wires l_1 and l_2:
      - $\mathit{share}(l_1, n), \mathit{share}(l_2, n) \rightsquigarrow \mathit{bit}(L, n, m) := \mathit{share}(l_1, n) * \mathit{share}(l_2, n)$

   (b) If $n < m$ and $\mathit{sender}(n, m)$ is true or $m < n$ and $\mathit{sender}(m, n)$ is false – so n is the sender and m is the receiver – the resulting bit is the random bit we generated in the first step for the OT exchange of party n with party m:
      - $\mathit{OTBitSen}(L, n, m), \mathit{share}(l_1, n), \mathit{share}(l_2, n) \rightsquigarrow \mathit{bit}(L, n, m) := \mathit{OTBitSen}(L, n, m)$

   (c) If $m < n$ and $\mathit{sender}(m, n)$ is true or $n < m$ and $\mathit{sender}(n, m)$ is false – so m is the sender and n is the receiver – the resulting bit is the bit we would have received as the result of the OT exchange between n and m:
      - $\mathit{OTBitRec}(L, m, n), \mathit{share}(l_1, n), \mathit{share}(l_2, n) \rightsquigarrow \mathit{bit}(L, n, m) := \mathit{OTBitRec}(L, n, m)$
4. If applicable, we leak the bits we generated for the OT exchanges. For each pair of parties $0 \leq n, m \leq N+1$:
   (a) If n is corrupted and either $n < m$ and $\mathit{sender}(n, m)$ is true or $m < n$ and $\mathit{sender}(m, n)$ is false – so n is the corrupted sender and m is the receiver – the value of $\mathit{bit}(L, n, m)$ is the randomly generated bit for the OT exchange between n and m, so we leak it:
      - $\mathit{bit}(L, n, m) \rightsquigarrow \mathit{leakRandOTBitSen}(L, n, m)_{\mathit{real}} := \mathit{bit}(L, n, m)$

   (b) Otherwise we leak nothing: $\mathbb{1}_{\{\mathit{bit}(L,n,m)\}}$
5. If applicable, we leak the bits resulting from the OT exchanges. For each pair of parties $0 \leq n, m \leq N+1$:
   (a) If n corrupted and either $m < n$ and $\mathit{sender}(m, n)$ is true or $n < m$ and $\mathit{sender}(n, m)$ is false – so m is the sender and n is the corrupted receiver – the value of $\mathit{bit}(L, n, m)$ is the bit we would have received from the OT exchange between m and n, so we leak it:
      - $\mathit{bit}(L, m, n) \rightsquigarrow \mathit{leakRandOTBitRec}(L, m, n)_{\mathit{real}} := \mathit{bit}(L, n, m)$

   (b) Otherwise we leak nothing: $\mathbb{1}_{\{\mathit{bit}(L,n,m)\}}$
6. We now sum up all the bits computed earlier to obtain the share of each party on wire L: for each party $0 \leq n \leq N+1$ and $0 \leq m \leq N+1$, the channel $\mathit{bit}_\Sigma(L, n, m)$ will hold the sum of all the bits $\mathit{bit}(L, n, -)$ where the last index is less than or equal to m. We define $\mathit{bit}_\Sigma(L, n, m)$ inductively as follows:
   (a) In the zero case we are summing a single bit:

- $bit(L, n, 0) \rightsquigarrow bit_\Sigma(L, n, 0) \coloneqq bit(L, n, 0)$

(b) In the successor case we add the last bit to the sum:

- $bit_\Sigma(L, n, m), bit(L, n, m+1) \rightsquigarrow bit_\Sigma(L, n, m+1) \coloneqq bit_\Sigma(L, n, m) \oplus bit(L, n, m+1)$

7. We compute the share of each party on wire L. For each party $0 \le n \le N+1$:
   - $bit_\Sigma(L, n, N+1) \rightsquigarrow share(L, n) \coloneqq bit_\Sigma(L, n, N+1)$
8. At last we hide the intermediate channels:
   - $OTBitSen(L, n, m)$ for $0 \le m \le N+1$ where $n < m$ and $sender(n, m)$ is true or $m < n$ and $sender(m, n)$ is false
   - $OTBitRec(L, n, m)$ for $0 \le m \le N+1$ where $m < n$ and $sender(m, n)$ is true or $n < m$ and $sender(n, m)$ is false
   - $bit(L, n, m)$ for $0 \le m \le N+1$
   - $bit_\Sigma(L, n, m)$ for $0 \le m \le N+1$

We now want to prove that the composition

- $Init$
- $\mathcal{D}(C, L)$
- $\mathcal{B}(C, L)$
- $\mathcal{Q}_1(l_1, l_2, L)$
- $value(l_1), value(l_2) \rightsquigarrow value(L) \coloneqq value(l_1) * value(l_2)$
- $share(k, n) \rightsquigarrow shareOk(k, n) \coloneqq \star$ for $0 \le k < L$ and $0 \le n \le N+1$
- $share(L, n) \rightsquigarrow shareOk(L, n) \coloneqq \star$ for $0 \le n \le N+1$

rewrites to the composition

- $Init$
- $\mathcal{D}(C, L)$
- $\mathcal{B}(C, L)$
- $\mathcal{Q}_1(l_1, l_2, L)$
- $value(l_1), value(l_2) \rightsquigarrow value(L) \coloneqq value(l_1) * value(l_2)$
- $value(k) \rightsquigarrow shareOk(k, n) \coloneqq \star$ for $0 \le k < L$ and $0 \le n \le N+1$
- $value(L) \rightsquigarrow shareOk(L, n) \coloneqq \star$ for $0 \le n \le N+1$

assuming the later hiding of the channels

- $inputShare(i, n)$ for $0 \le i \le I$ and $0 \le n \le N+1$
- $share(k, n)$ for $0 \le k \le L$ and $0 \le n \le N+1$
- $value(k)$ for $0 \le k \le L$
- $shareOk(k, n)$ for $0 \le k \le L$ and $0 \le n \le N+1$

To this end, assume the later hiding of the above channels. The composition

- $Init$

- $\mathcal{D}(C, L)$
- $\mathcal{B}(C, L)$
- $\mathcal{Q}_1(l_1, l_2, L)$
- $\mathit{value}(l_1), \mathit{value}(l_2) \rightsquigarrow \mathit{value}(L) \coloneqq \mathit{value}(l_1) * \mathit{value}(l_2)$
- $\mathit{share}(k, n) \rightsquigarrow \mathit{shareOk}(k, n) \coloneqq \star$ for $0 \leq k < L$ and $0 \leq n \leq N + 1$
- $\mathit{share}(L, n) \rightsquigarrow \mathit{shareOk}(L, n) \coloneqq \star$ for $0 \leq n \leq N + 1$

rewrites to the composition

- *Init*
- $\mathcal{D}(C, L)$
- $\mathcal{B}(C, L)$
- $\mathcal{Q}_2(l_1, l_2, L)$
- $\mathit{value}(l_1), \mathit{value}(l_2) \rightsquigarrow \mathit{value}(L) \coloneqq \mathit{value}(l_1) * \mathit{value}(l_2)$
- $\mathit{share}(k, n) \rightsquigarrow \mathit{shareOk}(k, n) \coloneqq \star$ for $0 \leq k < L$ and $0 \leq n \leq N + 1$

where $\mathcal{Q}_2(l_1, l_2, L)$ is the program below:

1. We generate a random bit for every OT exchange. For each pair of parties $0 \leq n, m \leq N + 1$:
   (a) If $n < m$ and $\mathit{sender}(n, m)$ is true or $m < n$ and $\mathit{sender}(m, n)$ is false – so n is the sender and m is the receiver – we choose a bit at random:
      - $\mathit{share}(l_1, n), \mathit{share}(l_2, n) \rightsquigarrow \mathit{OTBitSen}(L, n, m) \leftarrow \mathsf{rand}(\mathsf{bool})$

   (b) Otherwise we do nothing: $\mathsf{1}_{\{\mathit{share}(l_1,n), \mathit{share}(l_2,n)\}}$
2. We record that the bits for the OT exchanges have been generated. For each pair of parties $0 \leq n, m \leq N+1$:
   (a) If $n < m$ and $\mathit{sender}(n, m)$ is true or $m < n$ and $\mathit{sender}(m, n)$ is false – so n is the sender and m is the receiver – we record that $\mathit{OTBitSen}(L, n, m)$ has been generated:
      - $\mathit{OTBitSen}(L, n, m) \rightsquigarrow \mathit{OTBitSenOk}(L, n, m) \coloneqq \star$

   (b) Otherwise we do nothing: $\mathsf{1}_{\emptyset}$
3. Here we directly compute the bits we would have received as a result of the OT exchanges. For each pair of parties $0 \leq n, m \leq N + 1$:
   (a) If $m < n$ and $\mathit{sender}(m, n)$ is true or $n < m$ and $\mathit{sender}(n, m)$ is false – so m is the sender and n is the receiver – we compute the bit we would have received from the OT exchange with party m:
      - $\mathit{OTBitSen}(L, m, n), \mathit{share}(l_1, n), \mathit{share}(l_2, n), \mathit{share}(l_1, m), \mathit{share}(l_2, m) \rightsquigarrow$
        $\mathit{OTBitRec}(L, n, m) \coloneqq \big(\mathit{share}(l_1, m) * \mathit{share}(l_2, n) \oplus \mathit{share}(l_1, n) * \mathit{share}(l_2, m)\big) \oplus \mathit{OTBitSen}(L, m, n)$

   (b) Otherwise we do nothing: $\mathsf{1}_{\{\mathit{share}(l_1,n), \mathit{share}(l_2,n), \mathit{share}(l_1,m), \mathit{share}(l_2,m)\}}$
4. We record that the bits we would have received as a result of the OT exchanges have been computed. For each pair of parties $0 \leq n, m \leq N + 1$:
   (a) If $m < n$ and $\mathit{sender}(m, n)$ is true or $n < m$ and $\mathit{sender}(n, m)$ is false – so m is the sender and n is the receiver – we record that $\mathit{OTBitRec}(L, n, m)$ has been generated:
      - $\mathit{OTBitRec}(L, n, m) \rightsquigarrow \mathit{OTBitRecOk}(L, n, m) \coloneqq \star$

   (b) Otherwise we do nothing: $\mathsf{1}_{\emptyset}$

5. We now compute a bit $bit(L, n, m)$ for each pair of parties $0 \leq n, m \leq N + 1$ as follows:

   (a) If $m = n$, the resulting bit is the product of the n-th party's shares on wires l_1 and l_2:

   - $share(l_1, n), share(l_2, n) \rightsquigarrow bit(L, n, m) := share(l_1, n) * share(l_2, n)$

   (b) If $n < m$ and $sender(n, m)$ is true or $m < n$ and $sender(m, n)$ is false – so n is the sender and m is the receiver – the resulting bit is the random bit we generated in the first step for the OT exchange of party n with party m:

   - $OTBitSen(L, n, m), share(l_1, n), share(l_2, n) \rightsquigarrow bit(L, n, m) := OTBitSen(L, n, m)$

   (c) If $m < n$ and $sender(m, n)$ is true or $n < m$ and $sender(n, m)$ is false – so m is the sender and n is the receiver – the resulting bit is the bit we would have received as the result of the OT exchange between n and m:

   - $OTBitRec(L, m, n), share(l_1, n), share(l_2, n) \rightsquigarrow bit(L, n, m) := OTBitRec(L, n, m)$

6. We record that the computation of the bits in the previous step has been completed. For each pair of parties $0 \leq n, m \leq N + 1$:

   - $bit(L, n, m) \rightsquigarrow bitOk(L, n, m) := \star$

7. If applicable, we leak the bits we generated for the OT exchanges. For each pair of parties $0 \leq n, m \leq N+1$:

   (a) If n is corrupted and either $n < m$ and $sender(n, m)$ is true or $m < n$ and $sender(m, n)$ is false – so n is the corrupted sender and m is the receiver – the value of $bit(L, n, m)$ is the randomly generated bit for the OT exchange between n and m, so we leak it:

   - $bit(L, n, m) \rightsquigarrow leakRandOTBitSen(L, n, m)_{real} := bit(L, n, m)$

   (b) Otherwise we leak nothing: $1_{\{bit(L,n,m)\}}$

8. If applicable, we leak the bits resulting from the OT exchanges. For each pair of parties $0 \leq n, m \leq N + 1$:

   (a) If n corrupted and either $m < n$ and $sender(m, n)$ is true or $n < m$ and $sender(n, m)$ is false – so m is the sender and n is the corrupted receiver – the value of $bit(L, n, m)$ is the bit we would have received from the OT exchange between m and n, so we leak it:

   - $bit(L, m, n) \rightsquigarrow leakRandOTBitRec(L, m, n)_{real} := bit(L, n, m)$

   (b) Otherwise we leak nothing: $1_{\{bit(L,n,m)\}}$

9. We now sum up all the bits computed earlier to obtain the share of each party on wire L: for each party $0 \leq n \leq N + 1$ and $0 \leq m \leq N + 1$, the channel $bit_\Sigma(L, n, m)$ will hold the sum of all the bits $bit(L, n, -)$ where the last index is less than or equal to m. We define $bit_\Sigma(L, n, m)$ inductively as follows:

   (a) In the zero case we are summing a single bit:

   - $bit(L, n, 0) \rightsquigarrow bit_\Sigma(L, n, 0) := bit(L, n, 0)$

   (b) In the successor case we add the last bit to the sum:

   - $bit_\Sigma(L, n, m), bit(L, n, m + 1) \rightsquigarrow bit_\Sigma(L, n, m + 1) := bit_\Sigma(L, n, m) \oplus bit(L, n, m + 1)$

10. We record that the sum of the bits in the previous step has been computed. For each party $0 \leq n \leq N + 1$ and $0 \leq m \leq N + 1$:

    - $bit_\Sigma(L, n, m)\cdot \rightsquigarrow bitOk_\Sigma(L, n, m) := \star$

11. We compute the share of each party on wire L. For each party $0 \leq n \leq N + 1$:

    - $bit_\Sigma(L, n, N + 1) \rightsquigarrow share(L, n) := bit_\Sigma(L, n, N + 1)$

12. We record that the share of each party on wire L has been computed. For each party $0 \leq n \leq N + 1$:

    - $share(L, n) \rightsquigarrow shareOk(L, n) := \star$

13. At last we hide the intermediate channels:
    - $OTBitSen(L, n, m)$ for $0 \le m \le N+1$ where $n < m$ and $sender(n, m)$ is true or $m < n$ and $sender(m, n)$ is false
    - $OTBitSenOk(L, n, m)$ for $0 \le m \le N+1$ where either $n < m$ and $sender(n, m)$ is true or $m < n$ and $sender(m, n)$ is false
    - $OTBitRec(L, n, m)$ for $0 \le m \le N+1$ where $m < n$ and $sender(m, n)$ is true or $n < m$ and $sender(n, m)$ is false
    - $OTBitRecOk(L, n, m)$ for $0 \le m \le N+1$ where either $m < n$ and $sender(m, n)$ is true or $n < m$ and $sender(n, m)$ is false
    - $bit(L, n, m)$ for $0 \le m \le N+1$
    - $bitOk(L, n, m)$ for $0 \le m \le N+1$
    - $bit_\Sigma(L, n, m)$ for $0 \le m \le N+1$
    - $bitOk_\Sigma(L, n, m)$ for $0 \le m \le N+1$

The composition

- $Init$
- $\mathcal{D}(C, L)$
- $\mathcal{B}(C, L)$
- $\mathcal{Q}_2(l_1, l_2, L)$
- $value(l_1), value(l_2) \rightsquigarrow value(L) \coloneqq value(l_1) * value(l_2)$
- $share(k, n) \rightsquigarrow shareOk(k, n) \coloneqq \star$ for $0 \le k < L$ and $0 \le n \le N+1$

now rewrites to the composition

- $Init$
- $\mathcal{D}(C, L)$
- $\mathcal{B}(C, L)$
- $\mathcal{Q}_3(l_1, l_2, L)$
- $value(l_1), value(l_2) \rightsquigarrow value(L) \coloneqq value(l_1) * value(l_2)$
- $share(k, n) \rightsquigarrow shareOk(k, n) \coloneqq \star$ for $0 \le k < L$ and $0 \le n \le N+1$

where $\mathcal{Q}_3(l_1, l_2, L)$ is the following program:

1. We generate a random bit for every OT exchange. For each pair of parties $0 \le n, m \le N+1$:
    (a) If $n < m$ and $sender(n, m)$ is true or $m < n$ and $sender(m, n)$ is false – so n is the sender and m is the receiver – we choose a bit at random:
    - $share(l_1, n), share(l_2, n) \rightsquigarrow OTBitSen(L, n, m) \leftarrow \mathsf{rand}(\mathsf{bool})$

    (b) Otherwise we do nothing: $\mathbb{1}_{\{share(l_1,n), share(l_2,n)\}}$
2. We record that the bits for the OT exchanges have been generated. For each pair of parties $0 \le n, m \le N+1$:
    (a) If $n < m$ and $sender(n, m)$ is true or $m < n$ and $sender(m, n)$ is false – so n is the sender and m is the receiver – we record that $OTBitSen(L, n, m)$ has been generated:
    - $shareOk(l_1, n), shareOk(l_2, n) \rightsquigarrow OTBitSenOk(L, n, m) \coloneqq \star$

(b) Otherwise we do nothing: $1_{\{shareOk(l_1,n), shareOk(l_2,n)\}}$

3. Here we directly compute the bits we would have received as a result of the OT exchanges. For each pair of parties $0 \leq n, m \leq N+1$:

   (a) If $m < n$ and $sender(m,n)$ is true or $n < m$ and $sender(n,m)$ is false – so m is the sender and n is the receiver – we compute the bit we would have received from the OT exchange with party m:

   - $OTBitSen(L,m,n), share(l_1,n), share(l_2,n), share(l_1,m), share(l_2,m) \rightsquigarrow$ $OTBitRec(L,n,m) \coloneqq \big(share(l_1,m) * share(l_2,n) \oplus share(l_1,n) * share(l_2,m)\big) \oplus OTBitSen(L,m,n)$

   (b) Otherwise we do nothing: $1_{\{share(l_1,n), share(l_2,n), share(l_1,m), share(l_2,m)\}}$

4. We record that the bits we would have received as a result of the OT exchanges have been computed. For each pair of parties $0 \leq n, m \leq N+1$:

   (a) If $m < n$ and $sender(m,n)$ is true or $n < m$ and $sender(n,m)$ is false – so m is the sender and n is the receiver – we record that $OTBitRec(L,n,m)$ has been generated:

   - $OTBitSenOk(L,m,n), shareOk(l_1,n), shareOk(l_2,n), shareOk(l_1,m), shareOk(l_2,m) \rightsquigarrow$ $OTBitRecOk(L,n,m) \coloneqq \star$

   (b) Otherwise we do nothing: $1_{\{shareOk(l_1,n), shareOk(l_2,n), shareOk(l_1,m), shareOk(l_2,m)\}}$

5. We now compute a bit $bit(L,n,m)$ for each pair of parties $0 \leq n, m \leq N+1$ as follows:

   (a) If $m = n$, the resulting bit is the product of the n-th party's shares on wires l_1 and l_2:

   - $share(l_1,n), share(l_2,n) \rightsquigarrow bit(L,n,m) \coloneqq share(l_1,n) * share(l_2,n)$

   (b) If $n < m$ and $sender(n,m)$ is true or $m < n$ and $sender(m,n)$ is false – so n is the sender and m is the receiver – the resulting bit is the random bit we generated in the first step for the OT exchange of party n with party m:

   - $OTBitSen(L,n,m), share(l_1,n), share(l_2,n) \rightsquigarrow bit(L,n,m) \coloneqq OTBitSen(L,n,m)$

   (c) If $m < n$ and $sender(m,n)$ is true or $n < m$ and $sender(n,m)$ is false – so m is the sender and n is the receiver – the resulting bit is the bit we would have received as the result of the OT exchange between n and m:

   - $OTBitRec(L,m,n), share(l_1,n), share(l_2,n) \rightsquigarrow bit(L,n,m) \coloneqq OTBitRec(L,n,m)$

6. We record that the computation of the bits in the previous step has been completed. For each pair of parties $0 \leq n, m \leq N+1$:

   (a) If $m = n$:

   - $shareOk(l_1,n), shareOk(l_2,n) \rightsquigarrow bitOk(L,n,m) \coloneqq \star$

   (b) If $n < m$ and $sender(n,m)$ is true or $m < n$ and $sender(m,n)$ is false:

   - $OTBitSenOk(L,n,m), shareOk(l_1,n), shareOk(l_2,n) \rightsquigarrow bitOk(L,n,m) \coloneqq \star$

   (c) If $m < n$ and $sender(m,n)$ is true or $n < m$ and $sender(n,m)$ is false:

   - $OTBitRecOk(L,m,n), shareOk(l_1,n), shareOk(l_2,n) \rightsquigarrow bitOk(L,n,m) \coloneqq \star$

7. If applicable, we leak the bits we generated for the OT exchanges. For each pair of parties $0 \leq n, m \leq N+1$:

   (a) If n is corrupted and either $n < m$ and $sender(n,m)$ is true or $m < n$ and $sender(m,n)$ is false – so n is the corrupted sender and m is the receiver – the value of $bit(L,n,m)$ is the randomly generated bit for the OT exchange between n and m, so we leak it:

   - $bit(L,n,m) \rightsquigarrow leakRandOTBitSen(L,n,m)_{real} \coloneqq bit(L,n,m)$

   (b) Otherwise we leak nothing: $1_{\{bit(L,n,m)\}}$

8. If applicable, we leak the bits resulting from the OT exchanges. For each pair of parties $0 \leq n, m \leq N+1$:

(a) If n corrupted and either $m < n$ and $sender(m, n)$ is true or $n < m$ and $sender(n, m)$ is false – so m is the sender and n is the corrupted receiver – the value of $bit(L, n, m)$ is the bit we would have received from the OT exchange between m and n, so we leak it:

- $bit(L, m, n) \rightsquigarrow leakRandOTBitRec(L, m, n)_{real} \coloneqq bit(L, n, m)$

(b) Otherwise we leak nothing: $1_{\{bit(L,n,m)\}}$

9. We now sum up all the bits computed earlier to obtain the share of each party on wire L: for each party $0 \leq n \leq N + 1$ and $0 \leq m \leq N + 1$, the channel $bit_\Sigma(L, n, m)$ will hold the sum of all the bits $bit(L, n, -)$ where the last index is less than or equal to m. We define $bit_\Sigma(L, n, m)$ inductively as follows:

   (a) In the zero case we are summing a single bit:
   - $bit(L, n, 0) \rightsquigarrow bit_\Sigma(L, n, 0) \coloneqq bit(L, n, 0)$

   (b) In the successor case we add the last bit to the sum:
   - $bit_\Sigma(L, n, m), bit(L, n, m + 1) \rightsquigarrow bit_\Sigma(L, n, m + 1) \coloneqq bit_\Sigma(L, n, m) \oplus bit(L, n, m + 1)$

10. We record that the sum of the bits in the previous step has been computed. For each party $0 \leq n \leq N + 1$ and $0 \leq m \leq N + 1$, we define $bitOk_\Sigma(L, n, m)$ inductively as follows:

    (a) In the zero case we have:
    - $bitOk(L, n, 0) \rightsquigarrow bitOk_\Sigma(L, n, 0) \coloneqq \star$

    (b) In the successor case we have:
    - $bitOk_\Sigma(L, n, m), bitOk(L, n, m + 1) \rightsquigarrow bitOk_\Sigma(L, n, m + 1) \coloneqq \star$

11. We compute the share of each party on wire L. For each party $0 \leq n \leq N + 1$:
    - $bit_\Sigma(L, n, N + 1) \rightsquigarrow share(L, n) \coloneqq bit_\Sigma(L, n, N + 1)$

12. We record that the share of each party on wire L has been computed. For each party $0 \leq n \leq N + 1$:
    - $bitOk_\Sigma(L, n, N + 1) \rightsquigarrow shareOk(L, n) \coloneqq \star$

13. At last we hide the intermediate channels:
    - $OTBitSen(L, n, m)$ for $0 \leq m \leq N+1$ where $n < m$ and $sender(n, m)$ is true or $m < n$ and $sender(m, n)$ is false
    - $OTBitSenOk(L, n, m)$ for $0 \leq m \leq N + 1$ where either $n < m$ and $sender(n, m)$ is true or $m < n$ and $sender(m, n)$ is false
    - $OTBitRec(L, n, m)$ for $0 \leq m \leq N+1$ where $m < n$ and $sender(m, n)$ is true or $n < m$ and $sender(n, m)$ is false
    - $OTBitRecOk(L, n, m)$ for $0 \leq m \leq N + 1$ where either $m < n$ and $sender(m, n)$ is true or $n < m$ and $sender(n, m)$ is false
    - $bit(L, n, m)$ for $0 \leq m \leq N + 1$
    - $bitOk(L, n, m)$ for $0 \leq m \leq N + 1$
    - $bit_\Sigma(L, n, m)$ for $0 \leq m \leq N + 1$
    - $bitOk_\Sigma(L, n, m)$ for $0 \leq m \leq N + 1$

Using the induction hypothesis, the composition

- $Init$
- $\mathcal{D}(C, L)$
- $\mathcal{B}(C, L)$

- $\mathcal{Q}_3(l_1, l_2, L)$
- $value(l_1), value(l_2) \rightsquigarrow value(L) \coloneqq value(l_1) * value(l_2)$
- $share(k, n) \rightsquigarrow shareOk(k, n) \coloneqq \star$ for $0 \leq k < L$ and $0 \leq n \leq N+1$

rewrites to the composition

- *Init*
- $\mathcal{D}(C, L)$
- $\mathcal{B}(C, L)$
- $\mathcal{Q}_3(l_1, l_2, L)$
- $value(l_1), value(l_2) \rightsquigarrow value(L) \coloneqq value(l_1) * value(l_2)$
- $value(k) \rightsquigarrow shareOk(k, n) \coloneqq \star$ for $0 \leq k < L$ and $0 \leq n \leq N+1$

This further rewrites to the composition

- *Init*
- $\mathcal{D}(C, L)$
- $\mathcal{B}(C, L)$
- $\mathcal{Q}_4(l_1, l_2, L)$
- $value(l_1), value(l_2) \rightsquigarrow value(L) \coloneqq value(l_1) * value(l_2)$
- $value(k) \rightsquigarrow shareOk(k, n) \coloneqq \star$ for $0 \leq k < L$ and $0 \leq n \leq N+1$

where $\mathcal{Q}_4(l_1, l_2, L)$ is the program below:

1. We generate a random bit for every OT exchange. For each pair of parties $0 \leq n, m \leq N+1$:
   (a) If $n < m$ and $sender(n, m)$ is true or $m < n$ and $sender(m, n)$ is false – so n is the sender and m is the receiver – we choose a bit at random:
      - $share(l_1, n), share(l_2, n) \rightsquigarrow OTBitSen(L, n, m) \leftarrow \mathsf{rand}(\mathsf{bool})$

   (b) Otherwise we do nothing: $\mathbb{1}_{\{share(l_1,n), share(l_2,n)\}}$
2. We record that the bits for the OT exchanges have been generated. For each pair of parties $0 \leq n, m \leq N+1$:
   (a) If $n < m$ and $sender(n, m)$ is true or $m < n$ and $sender(m, n)$ is false – so n is the sender and m is the receiver – we record that $OTBitSen(L, n, m)$ has been generated:
      - $value(l_1), value(l_2) \rightsquigarrow OTBitSenOk(L, n, m) \coloneqq \star$

   (b) Otherwise we do nothing: $\mathbb{1}_{\{value(l_1), value(l_2)\}}$
3. Here we directly compute the bits we would have received as a result of the OT exchanges. For each pair of parties $0 \leq n, m \leq N+1$:
   (a) If $m < n$ and $sender(m, n)$ is true or $n < m$ and $sender(n, m)$ is false – so m is the sender and n is the receiver – we compute the bit we would have received from the OT exchange with party m:
      - $OTBitSen(L, m, n), share(l_1, n), share(l_2, n), share(l_1, m), share(l_2, m) \rightsquigarrow$
        $OTBitRec(L, n, m) \coloneqq \big(share(l_1, m) * share(l_2, n) \oplus share(l_1, n) * share(l_2, m)\big) \oplus OTBitSen(L, m, n)$

   (b) Otherwise we do nothing: $\mathbb{1}_{\{share(l_1,n), share(l_2,n), share(l_1,m), share(l_2,m)\}}$
4. We record that the bits we would have received as a result of the OT exchanges have been computed. For each pair of parties $0 \leq n, m \leq N+1$:

(a) If $m < n$ and $sender(m, n)$ is true or $n < m$ and $sender(n, m)$ is false – so m is the sender and n is the receiver – we record that $OTBitRec(L, n, m)$ has been generated:
   - $value(l_1), value(l_2) \rightsquigarrow OTBitRecOk(L, n, m) := \star$

(b) Otherwise we do nothing: $\mathsf{1}_{\{value(l_1), value(l_2)\}}$

5. We now compute a bit $bit(L, n, m)$ for each pair of parties $0 \leq n, m \leq N + 1$ as follows:

   (a) If $m = n$, the resulting bit is the product of the n-th party's shares on wires l_1 and l_2:
   - $share(l_1, n), share(l_2, n) \rightsquigarrow bit(L, n, m) := share(l_1, n) * share(l_2, n)$

   (b) If $n < m$ and $sender(n, m)$ is true or $m < n$ and $sender(m, n)$ is false – so n is the sender and m is the receiver – the resulting bit is the random bit we generated in the first step for the OT exchange of party n with party m:
   - $OTBitSen(L, n, m), share(l_1, n), share(l_2, n) \rightsquigarrow bit(L, n, m) := OTBitSen(L, n, m)$

   (c) If $m < n$ and $sender(m, n)$ is true or $n < m$ and $sender(n, m)$ is false – so m is the sender and n is the receiver – the resulting bit is the bit we would have received as the result of the OT exchange between n and m:
   - $OTBitRec(L, m, n), share(l_1, n), share(l_2, n) \rightsquigarrow bit(L, n, m) := OTBitRec(L, n, m)$

6. We record that the computation of the bits in the previous step has been completed. For each pair of parties $0 \leq n, m \leq N + 1$:
   - $value(l_1), value(l_2) \rightsquigarrow bitOk(L, n, m) := \star$

7. If applicable, we leak the bits we generated for the OT exchanges. For each pair of parties $0 \leq n, m \leq N+1$:

   (a) If n is corrupted and either $n < m$ and $sender(n, m)$ is true or $m < n$ and $sender(m, n)$ is false – so n is the corrupted sender and m is the receiver – the value of $bit(L, n, m)$ is the randomly generated bit for the OT exchange between n and m, so we leak it:
   - $bit(L, n, m) \rightsquigarrow leakRandOTBitSen(L, n, m)_{real} := bit(L, n, m)$

   (b) Otherwise we leak nothing: $\mathsf{1}_{\{bit(L,n,m)\}}$

8. If applicable, we leak the bits resulting from the OT exchanges. For each pair of parties $0 \leq n, m \leq N + 1$:

   (a) If n corrupted and either $m < n$ and $sender(m, n)$ is true or $n < m$ and $sender(n, m)$ is false – so m is the sender and n is the corrupted receiver – the value of $bit(L, n, m)$ is the bit we would have received from the OT exchange between m and n, so we leak it:
   - $bit(L, m, n) \rightsquigarrow leakRandOTBitRec(L, m, n)_{real} := bit(L, n, m)$

   (b) Otherwise we leak nothing: $\mathsf{1}_{\{bit(L,n,m)\}}$

9. We now sum up all the bits computed earlier to obtain the share of each party on wire L: for each party $0 \leq n \leq N + 1$ and $0 \leq m \leq N + 1$, the channel $bit_\Sigma(L, n, m)$ will hold the sum of all the bits $bit(L, n, -)$ where the last index is less than or equal to m. We define $bit_\Sigma(L, n, m)$ inductively as follows:

   (a) In the zero case we are summing a single bit:
   - $bit(L, n, 0) \rightsquigarrow bit_\Sigma(L, n, 0) := bit(L, n, 0)$

   (b) In the successor case we add the last bit to the sum:
   - $bit_\Sigma(L, n, m), bit(L, n, m + 1) \rightsquigarrow bit_\Sigma(L, n, m + 1) := bit_\Sigma(L, n, m) \oplus bit(L, n, m + 1)$

10. We record that the sum of the bits in the previous step has been computed. For each party $0 \leq n \leq N + 1$ and $0 \leq m \leq N + 1$:
    - $value(l_1), value(l_2) \rightsquigarrow bitOk_\Sigma(L, n, m) := \star$

11. We compute the share of each party on wire L. For each party $0 \leq n \leq N + 1$:

- $bit_\Sigma(L, n, N+1) \rightsquigarrow share(L, n) \coloneqq bit_\Sigma(L, n, N+1)$

12. We record that the share of each party on wire L has been computed. For each party $0 \le n \le N+1$:
    - $value(l_1), value(l_2) \rightsquigarrow shareOk(L, n) \coloneqq \star$
13. At last we hide the intermediate channels:
    - $OTBitSen(L, n, m)$ for $0 \le m \le N+1$ where $n < m$ and $sender(n, m)$ is true or $m < n$ and $sender(m, n)$ is false
    - $OTBitSenOk(L, n, m)$ for $0 \le m \le N+1$ where either $n < m$ and $sender(n, m)$ is true or $m < n$ and $sender(m, n)$ is false
    - $OTBitRec(L, n, m)$ for $0 \le m \le N+1$ where $m < n$ and $sender(m, n)$ is true or $n < m$ and $sender(n, m)$ is false
    - $OTBitRecOk(L, n, m)$ for $0 \le m \le N+1$ where either $m < n$ and $sender(m, n)$ is true or $n < m$ and $sender(n, m)$ is false
    - $bit(L, n, m)$ for $0 \le m \le N+1$
    - $bitOk(L, n, m)$ for $0 \le m \le N+1$
    - $bit_\Sigma(L, n, m)$ for $0 \le m \le N+1$
    - $bitOk_\Sigma(L, n, m)$ for $0 \le m \le N+1$

Finally, the composition

- $Init$
- $\mathcal{D}(C, L)$
- $\mathcal{B}(C, L)$
- $\mathcal{Q}_4(l_1, l_2, L)$
- $value(l_1), value(l_2) \rightsquigarrow value(L) \coloneqq value(l_1) * value(l_2)$
- $value(k) \rightsquigarrow shareOk(k, n) \coloneqq \star$ for $0 \le k < L$ and $0 \le n \le N+1$

rewrites to the composition

- $Init$
- $\mathcal{D}(C, L)$
- $\mathcal{B}(C, L)$
- $\mathcal{Q}_1(l_1, l_2, L)$
- $value(l_1), value(l_2) \rightsquigarrow value(L) \coloneqq value(l_1) * value(l_2)$
- $value(k) \rightsquigarrow shareOk(k, n) \coloneqq \star$ for $0 \le k < L$ and $0 \le n \le N+1$
- $value(L) \rightsquigarrow shareOk(L, n) \coloneqq \star$ for $0 \le n \le N+1$

This finishes the proof.

6.3 Proving claim 2

We proceed induction on the circuit C.

6.3.1 Proving claim 2 for $\cdot$ and 0

If the circuit is empty we have nothing to prove.

6.3.2 Proving claim 2 for C; $\mathsf{input}(i)$ and $L+1$

If the last wire draws from the input i, we want to prove that the composition

- Init
- $\mathcal{D}(C, L)$
- $\mathcal{B}(C, L)$
- $\mathit{inputShare}(i, n) \rightsquigarrow \mathit{share}(L, n) := \mathit{inputShare}(i, n)$ for all $0 \leq n \leq N+1$
- $\mathit{in}(i) \rightsquigarrow \mathit{value}(L) := \mathit{in}(i)$
- $\mathit{share}(k, n) \rightsquigarrow \mathit{shareOk}(k, n) := \star$ for $0 \leq k < L$ and $0 \leq n \leq N+1$
- $\mathit{share}(L, n) \rightsquigarrow \mathit{shareOk}(L, n) := \star$ for $0 \leq n \leq N+1$

rewrites to the composition

- Init
- $\mathcal{E}(C, L)$
- $\mathcal{B}(C, L)$
- $\mathit{inputShare}(i, n) \rightsquigarrow \mathit{share}(L, n) := \mathit{inputShare}(i, n)$ for all $0 \leq n \leq N+1$
- $\mathit{in}(i) \rightsquigarrow \mathit{value}(L) := \mathit{in}(i)$
- $\mathit{share}(k, n) \rightsquigarrow \mathit{shareOk}(k, n) := \star$ for $0 \leq k < L$ and $0 \leq n \leq N+1$
- $\mathit{share}(L, n) \rightsquigarrow \mathit{shareOk}(L, n) := \star$ for $0 \leq n \leq N+1$

assuming the later hiding of the channels

- $\mathit{inputShare}(i, n)$ for $0 \leq i \leq I$ and $0 \leq n \leq N+1$
- $\mathit{share}(k, n)$ for $0 \leq k \leq L$ and $0 \leq n \leq N+1$
- $\mathit{value}(k)n$ for $0 \leq k \leq L$ and $0 \leq n \leq N+1$
- $\mathit{shareOk}(k, n)$ for $0 \leq k \leq L$ and $0 \leq n \leq N+1$

This follows at once from the inductive hypothesis.

6.3.3 Proving claim 2 for C; $l_1 \,\mathsf{XOR}\, l_2$ and $L+1$

If the last wire is an XOR gate of wires l_1 and l_2, we want to prove that the composition

- Init
- $\mathcal{D}(C, L)$
- $\mathcal{B}(C, L)$
- $\mathit{share}(l_1, n), \mathit{share}(l_2, n) \rightsquigarrow \mathit{share}(L, n) := \mathit{share}(l_1, n) \oplus \mathit{share}(l_2, n)$ for all $0 \leq n \leq N+1$
- $\mathit{value}(l_1), \mathit{value}(l_2) \rightsquigarrow \mathit{value}(L) := \mathit{value}(l_1) \oplus \mathit{value}(l_2)$
- $\mathit{share}(k, n) \rightsquigarrow \mathit{shareOk}(k, n) := \star$ for $0 \leq k < L$ and $0 \leq n \leq N+1$
- $\mathit{share}(L, n) \rightsquigarrow \mathit{shareOk}(L, n) := \star$ for $0 \leq n \leq N+1$

rewrites to the composition

- Init
- $\mathcal{E}(C, L)$
- $\mathcal{B}(C, L)$
- $\mathit{share}(l_1, n), \mathit{share}(l_2, n) \rightsquigarrow \mathit{share}(L, n) \coloneqq \mathit{share}(l_1, n) \oplus \mathit{share}(l_2, n)$ for all $0 \leq n \leq N + 1$
- $\mathit{value}(l_1), \mathit{value}(l_2) \rightsquigarrow \mathit{value}(L) \coloneqq \mathit{value}(l_1) \oplus \mathit{value}(l_2)$
- $\mathit{share}(k, n) \rightsquigarrow \mathit{shareOk}(k, n) \coloneqq \star$ for $0 \leq k < L$ and $0 \leq n \leq N + 1$
- $\mathit{share}(L, n) \rightsquigarrow \mathit{shareOk}(L, n) \coloneqq \star$ for $0 \leq n \leq N + 1$

assuming the later hiding of the channels

- $\mathit{inputShare}(i, n)$ for $0 \leq i \leq I$ and $0 \leq n \leq N + 1$
- $\mathit{share}(k, n)$ for $0 \leq k \leq L$ and $0 \leq n \leq N + 1$
- $\mathit{value}(k)$ for $0 \leq k \leq L$ and $0 \leq n \leq N + 1$
- $\mathit{shareOk}(k, n)$ for $0 \leq k \leq L$ and $0 \leq n \leq N + 1$

This follows at once from the inductive hypothesis.

6.3.4 Proving claim 2 for $C; l_1\,\mathsf{AND}\,l_2$ and $L + 1$

If the last wire is an AND gate of wires l_1 and l_2, let $\mathcal{Q}_1(l_1, l_2, L)$ be the following program:

1. We generate a random bit for every OT exchange. For each pair of parties $0 \leq n, m \leq N + 1$:
   (a) If $n < m$ and $\mathit{sender}(n, m)$ is true or $m < n$ and $\mathit{sender}(m, n)$ is false – so n is the sender and m is the receiver – we choose a bit at random:
      - $\mathit{share}(l_1, n), \mathit{share}(l_2, n) \rightsquigarrow \mathit{OTBitSen}(L, n, m) \leftarrow \mathsf{rand}(\mathsf{bool})$

   (b) Otherwise we do nothing: $\mathbb{1}_{\{\mathit{share}(l_1,n), \mathit{share}(l_2,n)\}}$
2. Here we directly compute the bits we would have received as a result of the OT exchanges. For each pair of parties $0 \leq n, m \leq N + 1$:
   (a) If $m < n$ and $\mathit{sender}(m, n)$ is true or $n < m$ and $\mathit{sender}(n, m)$ is false – so m is the sender and n is the receiver – we compute the bit we would have received from the OT exchange with party m:
      - $\mathit{OTBitSen}(L, m, n), \mathit{share}(l_1, n), \mathit{share}(l_2, n), \mathit{share}(l_1, m), \mathit{share}(l_2, m) \rightsquigarrow$ $\mathit{OTBitRec}(L, n, m) \coloneqq \big(\mathit{share}(l_1, m) * \mathit{share}(l_2, n) \oplus \mathit{share}(l_1, n) * \mathit{share}(l_2, m)\big) \oplus \mathit{OTBitSen}(L, m, n)$

   (b) Otherwise we do nothing: $\mathbb{1}_{\{\mathit{share}(l_1,n), \mathit{share}(l_2,n), \mathit{share}(l_1,m), \mathit{share}(l_2,m)\}}$
3. We now compute a bit $\mathit{bit}(L, n, m)$ for each pair of parties $0 \leq n, m \leq N + 1$ as follows:
   (a) If $m = n$, the resulting bit is the product of the n-th party's shares on wires l_1 and l_2:
      - $\mathit{share}(l_1, n), \mathit{share}(l_2, n) \rightsquigarrow \mathit{bit}(L, n, m) \coloneqq \mathit{share}(l_1, n) * \mathit{share}(l_2, n)$

   (b) If $n < m$ and $\mathit{sender}(n, m)$ is true or $m < n$ and $\mathit{sender}(m, n)$ is false – so n is the sender and m is the receiver – the resulting bit is the random bit we generated in the first step for the OT exchange of party n with party m:
      - $\mathit{OTBitSen}(L, n, m), \mathit{share}(l_1, n), \mathit{share}(l_2, n) \rightsquigarrow \mathit{bit}(L, n, m) \coloneqq \mathit{OTBitSen}(L, n, m)$

   (c) If $m < n$ and $\mathit{sender}(m, n)$ is true or $n < m$ and $\mathit{sender}(n, m)$ is false – so m is the sender and n is the receiver – the resulting bit is the bit we would have received as the result of the OT exchange between n and m:

- $OTBitRec(L, m, n), share(l_1, n), share(l_2, n) \rightsquigarrow bit(L, n, m) \coloneqq OTBitRec(L, n, m)$

4. If applicable, we leak the bits we generated for the OT exchanges. For each pair of parties $0 \leq n, m \leq N+1$:
   (a) If n is corrupted and either $n < m$ and $sender(n, m)$ is true or $m < n$ and $sender(m, n)$ is false – so n is the corrupted sender and m is the receiver – the value of $bit(L, n, m)$ is the randomly generated bit for the OT exchange between n and m, so we leak it:
      - $bit(L, n, m) \rightsquigarrow leakRandOTBitSen(L, n, m)_{real} \coloneqq bit(L, n, m)$

   (b) Otherwise we leak nothing: $1_{\{bit(L,n,m)\}}$
5. If applicable, we leak the bits resulting from the OT exchanges. For each pair of parties $0 \leq n, m \leq N+1$:
   (a) If n corrupted and either $m < n$ and $sender(m, n)$ is true or $n < m$ and $sender(n, m)$ is false – so m is the sender and n is the corrupted receiver – the value of $bit(L, n, m)$ is the bit we would have received from the OT exchange between m and n, so we leak it:
      - $bit(L, m, n) \rightsquigarrow leakRandOTBitRec(L, m, n)_{real} \coloneqq bit(L, n, m)$

   (b) Otherwise we leak nothing: $1_{\{bit(L,n,m)\}}$
6. We now sum up all the bits computed earlier to obtain the share of each party on wire L: for each party $0 \leq n \leq N+1$ and $0 \leq m \leq N+1$, the channel $bit_\Sigma(L, n, m)$ will hold the sum of all the bits $bit(L, n, -)$ where the last index is less than or equal to m. We define $bit_\Sigma(L, n, m)$ inductively as follows:
   (a) In the zero case we are summing a single bit:
      - $bit(L, n, 0) \rightsquigarrow bit_\Sigma(L, n, 0) \coloneqq bit(L, n, 0)$

   (b) In the successor case we add the last bit to the sum:
      - $bit_\Sigma(L, n, m), bit(L, n, m+1) \rightsquigarrow bit_\Sigma(L, n, m+1) \coloneqq bit_\Sigma(L, n, m) \oplus bit(L, n, m+1)$
7. We compute the share of each party on wire L. For each party $0 \leq n \leq N+1$:
   - $bit_\Sigma(L, n, N+1+1) \rightsquigarrow share(L, n) \coloneqq bit_\Sigma(L, n, N+1+1)$
8. At last we hide the intermediate channels:
   - $OTBitSen(L, n, m)$ for $0 \leq m \leq N+1$ where $n < m$ and $sender(n, m)$ is true or $m < n$ and $sender(m, n)$ is false
   - $OTBitRec(L, n, m)$ for $0 \leq m \leq N+1$ where $m < n$ and $sender(m, n)$ is true or $n < m$ and $sender(n, m)$ is false
   - $bit(L, n, m)$ for $0 \leq m \leq N+1$
   - $bit_\Sigma(L, n, m)$ for $0 \leq m \leq N+1+1$

Furthermore, let $\mathcal{Q}_4(l_1, l_2, L)$ be the following program:

1. We generate a random bit for every OT exchange. For each pair of parties $0 \leq n, m \leq N+1$:
   (a) If $n < m$ and $sender(n, m)$ is true or $m < n$ and $sender(m, n)$ is false – so n is the sender and m is the receiver – we choose a bit at random:
      - $share(l_1, n), share(l_2, n), share(l_1, m), share(l_2, m) \rightsquigarrow OTBitSen(L, n, m) \leftarrow \mathsf{rand}(\mathsf{bool})$

   (b) Otherwise we do nothing: $1_{\{share(l_1,n), share(l_2,n), share(l_1,m), share(l_2,m)\}}$
2. Here we directly compute the bits we would have received as a result of the OT exchanges. For each pair of parties $0 \leq n, m \leq N+1$:
   (a) If $m < n$ and $sender(m, n)$ is true or $n < m$ and $sender(n, m)$ is false – so m is the sender and n is the receiver – we compute the bit we would have received from the OT exchange with party m:

- $\mathit{OTBitSen}(L, m, n), \mathit{share}(l_1, n), \mathit{share}(l_2, n), \mathit{share}(l_1, m), \mathit{share}(l_2, m) \rightsquigarrow$ $\mathit{OTBitRec}(L, n, m) := \big(\mathit{share}(l_1, m) * \mathit{share}(l_2, n) \oplus \mathit{share}(l_1, n) * \mathit{share}(l_2, m)\big) \oplus \mathit{OTBitSen}(L, m, n)$

(b) Otherwise we do nothing: $1_{\{\mathit{share}(l_1,n),\mathit{share}(l_2,n),\mathit{share}(l_1,m),\mathit{share}(l_2,m)\}}$

3. We now compute a bit $\mathit{bit}(L, n, m)$ for each pair of parties $0 \leq n, m \leq N + 1$ as follows:

   (a) If $m = n$, the resulting bit is the product of the n-th party's shares on wires l_1 and l_2:

   - $\mathit{share}(l_1, n), \mathit{share}(l_2, n) \rightsquigarrow \mathit{bit}(L, n, m) := \mathit{share}(l_1, n) * \mathit{share}(l_2, n)$

   (b) If $n < m$ and $\mathit{sender}(n, m)$ is true or $m < n$ and $\mathit{sender}(m, n)$ is false – so n is the sender and m is the receiver – the resulting bit is the random bit we generated in the first step for the OT exchange of party n with party m:

   - $\mathit{OTBitSen}(L, n, m), \mathit{share}(l_1, n), \mathit{share}(l_2, n) \rightsquigarrow \mathit{bit}(L, n, m) := \mathit{OTBitSen}(L, n, m)$

   (c) If $m < n$ and $\mathit{sender}(m, n)$ is true or $n < m$ and $\mathit{sender}(n, m)$ is false – so m is the sender and n is the receiver – the resulting bit is the bit we would have received as the result of the OT exchange between n and m:

   - $\mathit{OTBitRec}(L, m, n), \mathit{share}(l_1, n), \mathit{share}(l_2, n) \rightsquigarrow \mathit{bit}(L, n, m) := \mathit{OTBitRec}(L, n, m)$

4. If applicable, we leak the bits we generated for the OT exchanges. For each pair of parties $0 \leq n, m \leq N+1$:

   (a) If n is corrupted and either $n < m$ and $\mathit{sender}(n, m)$ is true or $m < n$ and $\mathit{sender}(m, n)$ is false – so n is the corrupted sender and m is the receiver – the value of $\mathit{bit}(L, n, m)$ is the randomly generated bit for the OT exchange between n and m, so we leak it:

   - $\mathit{bit}(L, n, m) \rightsquigarrow \mathit{leakRandOTBitSen}(L, n, m)_{\mathit{real}} := \mathit{bit}(L, n, m)$

   (b) Otherwise we leak nothing: $1_{\{\mathit{bit}(L,n,m)\}}$

5. If applicable, we leak the bits resulting from the OT exchanges. For each pair of parties $0 \leq n, m \leq N + 1$:

   (a) If n corrupted and either $m < n$ and $\mathit{sender}(m, n)$ is true or $n < m$ and $\mathit{sender}(n, m)$ is false – so m is the sender and n is the corrupted receiver – the value of $\mathit{bit}(L, n, m)$ is the bit we would have received from the OT exchange between m and n, so we leak it:

   - $\mathit{bit}(L, m, n) \rightsquigarrow \mathit{leakRandOTBitRec}(L, m, n)_{\mathit{real}} := \mathit{bit}(L, n, m)$

   (b) Otherwise we leak nothing: $1_{\{\mathit{bit}(L,n,m)\}}$

6. We now sum up all the bits computed earlier to obtain the share of each party on wire L: for each party $0 \leq n \leq N + 1$ and index $0 \leq m \leq N + 1 + 1$, the channel $\mathit{bit}_\Sigma(L, n, m)$ will hold the sum of all the bits $\mathit{bit}(L, n, -)$ where the last index is less than m. We define $\mathit{bit}_\Sigma(L, n, m)$ inductively as follows:

   (a) In the zero case, we are summing zero bits:

   - $\cdot \rightsquigarrow \mathit{bit}_\Sigma(L, n, 0) := \mathsf{false}$

   (b) In the successor case we add the last bit to the sum:

   - $\mathit{bit}_\Sigma(L, n, m), \mathit{bit}(L, n, m) \rightsquigarrow \mathit{bit}_\Sigma(L, n, m + 1) := \mathit{bit}_\Sigma(L, n, m) \oplus \mathit{bit}(L, n, m)$

7. We compute the share of each party on wire L. For each party $0 \leq n \leq N + 1$:

   - $\mathit{bit}_\Sigma(L, n, N + 1) \rightsquigarrow \mathit{share}(L, n) := \mathit{bit}_\Sigma(L, n, N + 1)$

8. At last we hide the intermediate channels:

   - $\mathit{OTBitSen}(L, n, m)$ for $0 \leq m \leq N+1$ where $n < m$ and $\mathit{sender}(n, m)$ is true or $m < n$ and $\mathit{sender}(m, n)$ is false
   - $\mathit{OTBitRec}(L, n, m)$ for $0 \leq m \leq N+1$ where $m < n$ and $\mathit{sender}(m, n)$ is true or $n < m$ and $\mathit{sender}(n, m)$ is false
   - $\mathit{bit}(L, n, m)$ for $0 \leq m \leq N + 1$

- $bit_\Sigma(L, n, m)$ for $0 \le m \le N+1$

We want to prove that the composition

- $Init$
- $\mathcal{D}(C, L)$
- $\mathcal{B}(C, L)$
- $\mathcal{Q}_1(l_1, l_2, L)$
- $value(l_1), value(l_2) \rightsquigarrow value(L) \coloneqq value(l_1) * value(l_2)$
- $share(k, n) \rightsquigarrow shareOk(k, n) \coloneqq \star$ for $0 \le k < L$ and $0 \le n \le N+1$
- $share(L, n) \rightsquigarrow shareOk(L, n) \coloneqq \star$ for $0 \le n \le N+1$

rewrites to the composition

- $Init$
- $\mathcal{E}(C, L)$
- $\mathcal{B}(C, L)$
- $\mathcal{Q}_4(l_1, l_2, L)$
- $value(l_1), value(l_2) \rightsquigarrow value(L) \coloneqq value(l_1) * value(l_2)$
- $share(k, n) \rightsquigarrow shareOk(k, n) \coloneqq \star$ for $0 \le k < L$ and $0 \le n \le N+1$
- $share(L, n) \rightsquigarrow shareOk(L, n) \coloneqq \star$ for $0 \le n \le N+1$

assuming the later hiding of the channels

- $inputShare(i, n)$ for $0 \le i \le I$ and $0 \le n \le N+1$
- $share(k, n)$ for $0 \le k \le L$ and $0 \le n \le N+1$
- $value(k)$ for $0 \le k \le L$
- $shareOk(k, n)$ for $0 \le k \le L$ and $0 \le n \le N+1$

To this end assume the later hiding of the above channels. The composition

- $Init$
- $\mathcal{D}(C, L)$
- $\mathcal{B}(C, L)$
- $\mathcal{Q}_1(l_1, l_2, L)$
- $value(l_1), value(l_2) \rightsquigarrow value(L) \coloneqq value(l_1) * value(l_2)$
- $share(k, n) \rightsquigarrow shareOk(k, n) \coloneqq \star$ for $0 \le k < L$ and $0 \le n \le N+1$
- $share(L, n) \rightsquigarrow shareOk(L, n) \coloneqq \star$ for $0 \le n \le N+1$

rewrites to the composition

- $Init$
- $\mathcal{D}(C, L)$

- $\mathcal{B}(C, L)$
- $\mathcal{Q}_2(l_1, l_2, L)$
- $value(l_1), value(l_2) \rightsquigarrow value(L) \coloneqq value(l_1) * value(l_2)$
- $share(k, n) \rightsquigarrow shareOk(k, n) \coloneqq \star$ for $0 \le k < L$ and $0 \le n \le N+1$
- $share(L, n) \rightsquigarrow shareOk(L, n) \coloneqq \star$ for $0 \le n \le N+1$

where $\mathcal{Q}_2(l_1, l_2, L)$ is the following program:

1. We generate a random bit for every OT exchange. For each pair of parties $0 \le n, m \le N+1$:
   (a) If $n < m$ and $sender(n, m)$ is true or $m < n$ and $sender(m, n)$ is false – so n is the sender and m is the receiver – we choose a bit at random:
      - $shareOk(l_1, n), shareOk(l_2, n) \rightsquigarrow OTBitSen(L, n, m) \leftarrow \mathsf{rand}(\mathsf{bool})$

   (b) Otherwise we do nothing: $1_{\{shareOk(l_1,n), shareOk(l_2,n)\}}$
2. Here we directly compute the bits we would have received as a result of the OT exchanges. For each pair of parties $0 \le n, m \le N+1$:
   (a) If $m < n$ and $sender(m, n)$ is true or $n < m$ and $sender(n, m)$ is false – so m is the sender and n is the receiver – we compute the bit we would have received from the OT exchange with party m:
      - $OTBitSen(L, m, n), share(l_1, n), share(l_2, n), share(l_1, m), share(l_2, m) \rightsquigarrow$
        $OTBitRec(L, n, m) \coloneqq \big(share(l_1, m) * share(l_2, n) \oplus share(l_1, n) * share(l_2, m)\big) \oplus OTBitSen(L, m, n)$

   (b) Otherwise we do nothing: $1_{\{share(l_1,n), share(l_2,n), share(l_1,m), share(l_2,m)\}}$
3. We now compute a bit $bit(L, n, m)$ for each pair of parties $0 \le n, m \le N+1$ as follows:
   (a) If $m = n$, the resulting bit is the product of the n-th party's shares on wires l_1 and l_2:
      - $share(l_1, n), share(l_2, n) \rightsquigarrow bit(L, n, m) \coloneqq share(l_1, n) * share(l_2, n)$

   (b) If $n < m$ and $sender(n, m)$ is true or $m < n$ and $sender(m, n)$ is false – so n is the sender and m is the receiver – the resulting bit is the random bit we generated in the first step for the OT exchange of party n with party m:
      - $OTBitSen(L, n, m), share(l_1, n), share(l_2, n) \rightsquigarrow bit(L, n, m) \coloneqq OTBitSen(L, n, m)$

   (c) If $m < n$ and $sender(m, n)$ is true or $n < m$ and $sender(n, m)$ is false – so m is the sender and n is the receiver – the resulting bit is the bit we would have received as the result of the OT exchange between n and m:
      - $OTBitRec(L, m, n), share(l_1, n), share(l_2, n) \rightsquigarrow bit(L, n, m) \coloneqq OTBitRec(L, n, m)$
4. If applicable, we leak the bits we generated for the OT exchanges. For each pair of parties $0 \le n, m \le N+1$:
   (a) If n is corrupted and either $n < m$ and $sender(n, m)$ is true or $m < n$ and $sender(m, n)$ is false – so n is the corrupted sender and m is the receiver – the value of $bit(L, n, m)$ is the randomly generated bit for the OT exchange between n and m, so we leak it:
      - $bit(L, n, m) \rightsquigarrow leakRandOTBitSen(L, n, m)_{real} \coloneqq bit(L, n, m)$

   (b) Otherwise we leak nothing: $1_{\{bit(L,n,m)\}}$
5. If applicable, we leak the bits resulting from the OT exchanges. For each pair of parties $0 \le n, m \le N+1$:
   (a) If n corrupted and either $m < n$ and $sender(m, n)$ is true or $n < m$ and $sender(n, m)$ is false – so m is the sender and n is the corrupted receiver – the value of $bit(L, n, m)$ is the bit we would have received from the OT exchange between m and n, so we leak it:
      - $bit(L, m, n) \rightsquigarrow leakRandOTBitRec(L, m, n)_{real} \coloneqq bit(L, n, m)$

(b) Otherwise we leak nothing: $1_{\{bit(L,n,m)\}}$

6. We now sum up all the bits computed earlier to obtain the share of each party on wire L: for each party $0 \le n \le N+1$ and $0 \le m \le N+1$, the channel $bit_\Sigma(L,n,m)$ will hold the sum of all the bits $bit(L,n,-)$ where the last index is less than or equal to m. We define $bit_\Sigma(L,n,m)$ inductively as follows:

   (a) In the zero case we are summing a single bit:
      - $bit(L,n,0) \rightsquigarrow bit_\Sigma(L,n,0) \coloneqq bit(L,n,0)$

   (b) In the successor case we add the last bit to the sum:
      - $bit_\Sigma(L,n,m), bit(L,n,m+1) \rightsquigarrow bit_\Sigma(L,n,m+1) \coloneqq bit_\Sigma(L,n,m) \oplus bit(L,n,m+1)$

7. We compute the share of each party on wire L. For each party $0 \le n \le N+1$:
   - $bit_\Sigma(L,n,N+1) \rightsquigarrow share(L,n) \coloneqq bit_\Sigma(L,n,N+1)$

8. At last we hide the intermediate channels:
   - $OTBitSen(L,n,m)$ for $0 \le m \le N+1$ where $n < m$ and $sender(n,m)$ is true or $m < n$ and $sender(m,n)$ is false
   - $OTBitRec(L,n,m)$ for $0 \le m \le N+1$ where $m < n$ and $sender(m,n)$ is true or $n < m$ and $sender(n,m)$ is false
   - $bit(L,n,m)$ for $0 \le m \le N+1$
   - $bit_\Sigma(L,n,m)$ for $0 \le m \le N+1$

Using claim 1, the composition

- $Init$
- $\mathcal{D}(C,L)$
- $\mathcal{B}(C,L)$
- $\mathcal{Q}_2(l_1,l_2,L)$
- $value(l_1), value(l_2) \rightsquigarrow value(L) \coloneqq value(l_1) * value(l_2)$
- $share(k,n) \rightsquigarrow shareOk(k,n) \coloneqq \star$ for $0 \le k < L$ and $0 \le n \le N+1$
- $share(L,n) \rightsquigarrow shareOk(L,n) \coloneqq \star$ for $0 \le n \le N+1$

rewrites to the composition

- $Init$
- $\mathcal{D}(C,L)$
- $\mathcal{B}(C,L)$
- $\mathcal{Q}_2(l_1,l_2,L)$
- $value(l_1), value(l_2) \rightsquigarrow value(L) \coloneqq value(l_1) * value(l_2)$
- $value(k) \rightsquigarrow shareOk(k,n) \coloneqq \star$ for $0 \le k < L$ and $0 \le n \le N+1$
- $share(L,n) \rightsquigarrow shareOk(L,n) \coloneqq \star$ for $0 \le n \le N+1$

This further rewrites to the composition

- $Init$

- $\mathcal{D}(C, L)$
- $\mathcal{B}(C, L)$
- $\mathcal{Q}_3(l_1, l_2, L)$
- $\mathit{value}(l_1), \mathit{value}(l_2) \rightsquigarrow \mathit{value}(L) := \mathit{value}(l_1) * \mathit{value}(l_2)$
- $\mathit{value}(k) \rightsquigarrow \mathit{shareOk}(k, n) := \star$ for $0 \leq k < L$ and $0 \leq n \leq N+1$
- $\mathit{share}(L, n) \rightsquigarrow \mathit{shareOk}(L, n) := \star$ for $0 \leq n \leq N+1$

where $\mathcal{Q}_3(l_1, l_2, L)$ is the following program:

1. We generate a random bit for every OT exchange. For each pair of parties $0 \leq n, m \leq N+1$:
   (a) If $n < m$ and $\mathit{sender}(n, m)$ is true or $m < n$ and $\mathit{sender}(m, n)$ is false – so n is the sender and m is the receiver – we choose a bit at random:
      - $\mathit{shareOk}(l_1, n), \mathit{shareOk}(l_2, n), \mathit{shareOk}(l_1, m), \mathit{shareOk}(l_2, m) \rightsquigarrow \mathit{OTBitSen}(L, n, m) \leftarrow \mathsf{rand}(\mathsf{bool})$

   (b) Otherwise we do nothing: $\mathbb{1}_{\{\mathit{shareOk}(l_1,n), \mathit{shareOk}(l_2,n), \mathit{shareOk}(l_1,m), \mathit{shareOk}(l_2,m)\}}$
2. Here we directly compute the bits we would have received as a result of the OT exchanges. For each pair of parties $0 \leq n, m \leq N+1$:
   (a) If $m < n$ and $\mathit{sender}(m, n)$ is true or $n < m$ and $\mathit{sender}(n, m)$ is false – so m is the sender and n is the receiver – we compute the bit we would have received from the OT exchange with party m:
      - $\mathit{OTBitSen}(L, m, n), \mathit{share}(l_1, n), \mathit{share}(l_2, n), \mathit{share}(l_1, m), \mathit{share}(l_2, m) \rightsquigarrow$ $\mathit{OTBitRec}(L, n, m) := \big(\mathit{share}(l_1, m) * \mathit{share}(l_2, n) \oplus \mathit{share}(l_1, n) * \mathit{share}(l_2, m)\big) \oplus \mathit{OTBitSen}(L, m, n)$

   (b) Otherwise we do nothing: $\mathbb{1}_{\{\mathit{share}(l_1,n), \mathit{share}(l_2,n), \mathit{share}(l_1,m), \mathit{share}(l_2,m)\}}$
3. We now compute a bit $\mathit{bit}(L, n, m)$ for each pair of parties $0 \leq n, m \leq N+1$ as follows:
   (a) If $m = n$, the resulting bit is the product of the n-th party's shares on wires l_1 and l_2:
      - $\mathit{share}(l_1, n), \mathit{share}(l_2, n) \rightsquigarrow \mathit{bit}(L, n, m) := \mathit{share}(l_1, n) * \mathit{share}(l_2, n)$

   (b) If $n < m$ and $\mathit{sender}(n, m)$ is true or $m < n$ and $\mathit{sender}(m, n)$ is false – so n is the sender and m is the receiver – the resulting bit is the random bit we generated in the first step for the OT exchange of party n with party m:
      - $\mathit{OTBitSen}(L, n, m), \mathit{share}(l_1, n), \mathit{share}(l_2, n) \rightsquigarrow \mathit{bit}(L, n, m) := \mathit{OTBitSen}(L, n, m)$

   (c) If $m < n$ and $\mathit{sender}(m, n)$ is true or $n < m$ and $\mathit{sender}(n, m)$ is false – so m is the sender and n is the receiver – the resulting bit is the bit we would have received as the result of the OT exchange between n and m:
      - $\mathit{OTBitRec}(L, m, n), \mathit{share}(l_1, n), \mathit{share}(l_2, n) \rightsquigarrow \mathit{bit}(L, n, m) := \mathit{OTBitRec}(L, n, m)$
4. If applicable, we leak the bits we generated for the OT exchanges. For each pair of parties $0 \leq n, m \leq N+1$:
   (a) If n is corrupted and either $n < m$ and $\mathit{sender}(n, m)$ is true or $m < n$ and $\mathit{sender}(m, n)$ is false – so n is the corrupted sender and m is the receiver – the value of $\mathit{bit}(L, n, m)$ is the randomly generated bit for the OT exchange between n and m, so we leak it:
      - $\mathit{bit}(L, n, m) \rightsquigarrow \mathit{leakRandOTBitSen}(L, n, m)_{\mathit{real}} := \mathit{bit}(L, n, m)$

   (b) Otherwise we leak nothing: $\mathbb{1}_{\{\mathit{bit}(L,n,m)\}}$
5. If applicable, we leak the bits resulting from the OT exchanges. For each pair of parties $0 \leq n, m \leq N+1$:
   (a) If n corrupted and either $m < n$ and $\mathit{sender}(m, n)$ is true or $n < m$ and $\mathit{sender}(n, m)$ is false – so m is the sender and n is the corrupted receiver – the value of $\mathit{bit}(L, n, m)$ is the bit we would have received from the OT exchange between m and n, so we leak it:

- $bit(L,m,n) \rightsquigarrow leakRandOTBitRec(L,m,n)_{real} := bit(L,n,m)$

(b) Otherwise we leak nothing: $1_{\{bit(L,n,m)\}}$

6. We now sum up all the bits computed earlier to obtain the share of each party on wire L: for each party $0 \leq n \leq N+1$ and $0 \leq m \leq N+1$, the channel $bit_\Sigma(L,n,m)$ will hold the sum of all the bits $bit(L,n,-)$ where the last index is less than or equal to m. We define $bit_\Sigma(L,n,m)$ inductively as follows:

   (a) In the zero case we are summing a single bit:
   - $bit(L,n,0) \rightsquigarrow bit_\Sigma(L,n,0) := bit(L,n,0)$

   (b) In the successor case we add the last bit to the sum:
   - $bit_\Sigma(L,n,m), bit(L,n,m+1) \rightsquigarrow bit_\Sigma(L,n,m+1) := bit_\Sigma(L,n,m) \oplus bit(L,n,m+1)$

7. We compute the share of each party on wire L. For each party $0 \leq n \leq N+1$:
   - $bit_\Sigma(L,n,N+1) \rightsquigarrow share(L,n) := bit_\Sigma(L,n,N+1)$

8. At last we hide the intermediate channels:
   - $OTBitSen(L,n,m)$ for $0 \leq m \leq N+1$ where $n < m$ and $sender(n,m)$ is true or $m < n$ and $sender(m,n)$ is false
   - $OTBitRec(L,n,m)$ for $0 \leq m \leq N+1$ where $m < n$ and $sender(m,n)$ is true or $n < m$ and $sender(n,m)$ is false
   - $bit(L,n,m)$ for $0 \leq m \leq N+1$
   - $bit_\Sigma(L,n,m)$ for $0 \leq m \leq N+1$

We now use claim 1 in the opposite direction: the composition

- $Init$
- $\mathcal{D}(C,L)$
- $\mathcal{B}(C,L)$
- $\mathcal{Q}_3(l_1,l_2,L)$
- $value(l_1), value(l_2) \rightsquigarrow value(L) := value(l_1) * value(l_2)$
- $value(k) \rightsquigarrow shareOk(k,n) := \star$ for $0 \leq k < L$ and $0 \leq n \leq N+1$
- $share(L,n) \rightsquigarrow shareOk(L,n) := \star$ for $0 \leq n \leq N+1$

rewrites to the composition

- $Init$
- $\mathcal{D}(C,L)$
- $\mathcal{B}(C,L)$
- $\mathcal{Q}_3(l_1,l_2,L)$
- $value(l_1), value(l_2) \rightsquigarrow value(L) := value(l_1) * value(l_2)$
- $share(k,n) \rightsquigarrow shareOk(k,n) := \star$ for $0 \leq k < L$ and $0 \leq n \leq N+1$
- $share(L,n) \rightsquigarrow shareOk(L,n) := \star$ for $0 \leq n \leq N+1$

which further rewrites to the composition

- $Init$

- $\mathcal{D}(C, L)$
- $\mathcal{B}(C, L)$
- $\mathcal{Q}_4(l_1, l_2, L)$
- $value(l_1), value(l_2) \rightsquigarrow value(L) \coloneqq value(l_1) * value(l_2)$
- $share(k, n) \rightsquigarrow shareOk(k, n) \coloneqq \star$ for $0 \leq k < L$ and $0 \leq n \leq N + 1$
- $share(L, n) \rightsquigarrow shareOk(L, n) \coloneqq \star$ for $0 \leq n \leq N + 1$

Using the induction hypothesis, the above rewrites to the composition

- $Init$
- $\mathcal{E}(C, L)$
- $\mathcal{B}(C, L)$
- $\mathcal{Q}_4(l_1, l_2, L)$
- $value(l_1), value(l_2) \rightsquigarrow value(L) \coloneqq value(l_1) * value(l_2)$
- $share(k, n) \rightsquigarrow shareOk(k, n) \coloneqq \star$ for $0 \leq k < L$ and $0 \leq n \leq N + 1$
- $share(L, n) \rightsquigarrow shareOk(L, n) \coloneqq \star$ for $0 \leq n \leq N + 1$

This finishes the proof.