IPDL Case Studies

Kristina Sojakova Mihai Codescu Joshua Gancher

May 31, 2023

Abstract

We present here the full proofs of our IPDL case studies: Authenticated-To-Secure Channel in Section 3; Diffie-Hellman Key Exchange (DHKE) in Section 4; DHKE + One-Time Pad in 5; El Gamal Encryption in Section 6; Oblivious Transfer: 1-Out-Of-2 Pre-Processing in Section 7; Multi-Party Coin Toss in Section 8; and Two-Party GMW Protocol in Section 9.

Acknowledgement

This project was funded through the NGI Assure Fund, a fund established by NLnet with financial support from the European Commission's Next Generation Internet programme, under the aegis of DG Communications Networks, Content and Technology under grant agreement No. 957073.

1 Symmetric-Key Encryption: CPA Security

Semantic security, also known as *security against chosen-plaintext attacks (CPA)*, for a symmetric-key encryption schema states that encoding q fixed messages with a secret key is computationally indistinguishable from encoding any other q messages. Or equivalently, encoding q fixed messages with a secret key is computationally indistinguishable from encoding the same message q times. We now show this equivalence in IPDL.

Formally, we assume types $\mathsf{key}, \mathsf{msg}, \mathsf{ctxt}$ of keys, messages, and ciphertexts, respectively; a chosen message $\mathsf{zeros} : 1 \to \mathsf{msg}$; a probabilistic key generation algorithm $\mathsf{gen}_{\mathsf{key}} : 1 \twoheadrightarrow \mathsf{key}$; and a probabilistic encryption algorithm $\mathsf{enc} : \mathsf{msg} \times \mathsf{key} \twoheadrightarrow \mathsf{ctxt}$ that takes a message and a key, and returns a ciphertext. We will write $\mathsf{enc}(m, k)$ in place of $\mathsf{enc}\ (m, k)$.

We can express the standard (*left-right*) version of the CPA cryptographic assumption as the following protocol-level axiom: in the channel context $\{\mathsf{Msg}_L(i) : \mathsf{msg}\}_i$, $\{\mathsf{Msg}_R(i) : \mathsf{msg}\}_i$, $\mathsf{Key} : \mathsf{key}$, $\{\mathsf{Enc}(i) : \mathsf{ctxt}\}_i$ where $i := 1, \ldots, q$, the protocol

- $\mathsf{Key} := \mathsf{samp}\ \mathsf{gen}_{\mathsf{key}}$
- $\mathsf{Enc}(i) := m_L \leftarrow \mathsf{Msg}_L(i);\ m_R \leftarrow \mathsf{Msg}_R(i);\ k \leftarrow \mathsf{Key};\ \mathsf{samp}\ \mathsf{enc}(m_L, k)$ for $0 \leqslant i < q$

with inputs $\mathsf{Msg}_L(-)$, $\mathsf{Msg}_R(-)$, outputs $\mathsf{Enc}(-)$, and an internal channel Key rewrites approximately to the protocol

- $\mathsf{Key} := \mathsf{samp}\ \mathsf{gen}_{\mathsf{key}}$
- $\mathsf{Enc}(i) := m_L \leftarrow \mathsf{Msg}_L(i);\ m_R \leftarrow \mathsf{Msg}_R(i);\ k \leftarrow \mathsf{Key};\ \mathsf{samp}\ \mathsf{enc}(m_R, k)$ for $0 \leqslant i < q$

The *chosen-message* version of CPA security is another protocol-level axiom: in the channel context $\{\mathsf{Msg}(i) : \mathsf{msg}\}_i, \mathsf{Key} : \mathsf{key}, \{\mathsf{Enc}(i) : \mathsf{ctxt}\}_i$ where $i := 1, \ldots, q$, the protocol

- $\mathsf{Key} := \mathsf{samp}\ \mathsf{gen}_{\mathsf{key}}$
- $\mathsf{Enc}(i) := m \leftarrow \mathsf{Msg}(i);\ k \leftarrow \mathsf{Key};\ \mathsf{samp}\ \mathsf{enc}(m, k)$ for $0 \leqslant i < q$

with inputs $\mathsf{Msg}(-)$, outputs $\mathsf{Enc}(-)$, and an internal channel Key rewrites approximately to the protocol

- $\mathsf{Key} := \mathsf{samp}\ \mathsf{gen}_{\mathsf{key}}$

- $\mathsf{Enc}(i) := m \leftarrow \mathsf{Msg}(i);\ k \leftarrow \mathsf{Key};\ \mathsf{samp}\ \mathsf{enc}(\mathsf{zeros}, k)$ for $0 \leqslant i < q$

To prove that these are equivalent, assume first the *left-right* version of CPA security. The protocol

- $\mathsf{Key} := \mathsf{samp}\ \mathsf{gen}_{\mathsf{key}}$
- $\mathsf{Enc}(i) := m \leftarrow \mathsf{Msg}(i);\ k \leftarrow \mathsf{Key};\ \mathsf{samp}\ \mathsf{enc}(m, k)$ for $0 \leqslant i < q$

can be factored as the composition of the protocol

- $\mathsf{Key} := \mathsf{samp}\ \mathsf{gen}_{\mathsf{key}}$
- $\mathsf{Enc}(i) := m_L \leftarrow \mathsf{Msg}_L(i);\ m_R \leftarrow \mathsf{Msg}_R(i);\ k \leftarrow \mathsf{Key};\ \mathsf{samp}\ \mathsf{enc}(m_L, k)$ for $0 \leqslant i < q$

(the left-hand side of our CPA assumption) and the protocol

- $\mathsf{Msg}_L(i) := \mathsf{read}\ \mathsf{Msg}(i)$ for $0 \leqslant i < q$
- $\mathsf{Msg}_R(i) := \mathsf{ret}\ \mathsf{zeros}$ for $0 \leqslant i < q$

followed by the hiding of the channels $\mathsf{Msg}_L(-)$, $\mathsf{Msg}_R(-)$.

Applying the CPA assumption yields the composition of the protocol

- $\mathsf{Key} := \mathsf{samp}\ \mathsf{gen}_{\mathsf{key}}$
- $\mathsf{Enc}(i) := m_L \leftarrow \mathsf{Msg}_L(i);\ m_R \leftarrow \mathsf{Msg}_R(i);\ k \leftarrow \mathsf{Key};\ \mathsf{samp}\ \mathsf{enc}(m_R, k)$ for $0 \leqslant i < q$

and the (unchanged) protocol

- $\mathsf{Msg}_L(i) := \mathsf{read}\ \mathsf{Msg}(i)$ for $0 \leqslant i < q$
- $\mathsf{Msg}_R(i) := \mathsf{ret}\ \mathsf{zeros}$ for $0 \leqslant i < q$

followed by the hiding of the channels $\mathsf{Msg}_L(-)$, $\mathsf{Msg}_R(-)$.

Reversing the factorization process yields the protocol

- $\mathsf{Key} := \mathsf{samp}\ \mathsf{gen}_{\mathsf{key}}$
- $\mathsf{Enc}(i) := m \leftarrow \mathsf{Msg}(i);\ k \leftarrow \mathsf{Key};\ \mathsf{samp}\ \mathsf{enc}(\mathsf{zeros}, k)$ for $0 \leqslant i < q$

as desired.

For the other direction, assume the *chosen-message* version of CPA security. The protocol

- $\mathsf{Key} := \mathsf{samp}\ \mathsf{gen}_{\mathsf{key}}$
- $\mathsf{Enc}(i) := m_L \leftarrow \mathsf{Msg}_L(i);\ m_R \leftarrow \mathsf{Msg}_R(i);\ k \leftarrow \mathsf{Key};\ \mathsf{samp}\ \mathsf{enc}(m_L, k)$ for $0 \leqslant i < q$

can be factored as the composition of the protocol

- $\mathsf{Key} := \mathsf{samp}\ \mathsf{gen}_{\mathsf{key}}$
- $\mathsf{Enc}(i) := m \leftarrow \mathsf{Msg}(i);\ k \leftarrow \mathsf{Key};\ \mathsf{samp}\ \mathsf{enc}(m, k)$ for $0 \leqslant i < q$

(the left-hand side of our CPA assumption) and the protocol

- $\mathsf{Msg}(i) := m_L \leftarrow \mathsf{Msg}_L(i);\ m_R \leftarrow \mathsf{Msg}_R(i);\ \mathsf{ret}\ m_L$ for $0 \leqslant i < q$

followed by the hiding of the channels $\mathsf{Msg}(-)$.

Applying the CPA assumption yields the composition of the protocol

- $\mathsf{Key} := \mathsf{samp}\ \mathsf{gen}_{\mathsf{key}}$
- $\mathsf{Enc}(i) := m \leftarrow \mathsf{Msg}(i);\ k \leftarrow \mathsf{Key};\ \mathsf{samp}\ \mathsf{enc}(\mathsf{zeros}, k)$ for $0 \leqslant i < q$

and the (unchanged) protocol

- $\mathsf{Msg}(i) := m_L \leftarrow \mathsf{Msg}_L(i);\ m_R \leftarrow \mathsf{Msg}_R(i);\ \mathsf{ret}\ m_L$ for $0 \leqslant i < q$

followed by the hiding of the channels $\mathsf{Msg}(-)$.

Substituting the channels $\mathsf{Msg}(-)$ away yields the protocol

- $\mathsf{Key} := \mathsf{samp}\ \mathsf{gen}_{\mathsf{key}}$
- $\mathsf{Enc}(i) := m_L \leftarrow \mathsf{Msg}_L(i);\ m_R \leftarrow \mathsf{Msg}_R(i);\ k \leftarrow \mathsf{Key};\ \mathsf{samp}\ \mathsf{enc}(\mathsf{zeros}, k)$ for $0 \leqslant i < q$

The above can be factored as the composition of the protocol

- $\mathsf{Key} := \mathsf{samp}\ \mathsf{gen}_{\mathsf{key}}$
- $\mathsf{Enc}(i) := m \leftarrow \mathsf{Msg}(i);\ k \leftarrow \mathsf{Key};\ \mathsf{samp}\ \mathsf{enc}(\mathsf{zeros}, k)$ for $0 \leqslant i < q$

(the right-hand side of our CPA assumption) and the protocol

- $\mathsf{Msg}(i) := m_L \leftarrow \mathsf{Msg}_L(i);\ m_R \leftarrow \mathsf{Msg}_R(i);\ \mathsf{ret}\ m_R$ for $0 \leqslant i < q$

followed by the hiding of the channels $\mathsf{Msg}(-)$.

Applying the CPA assumption again (this time from right to left) yields the composition of the protocol

- $\mathsf{Key} := \mathsf{samp}\ \mathsf{gen}_{\mathsf{key}}$
- $\mathsf{Enc}(i) := m \leftarrow \mathsf{Msg}(i);\ k \leftarrow \mathsf{Key};\ \mathsf{samp}\ \mathsf{enc}(m, k)$ for $0 \leqslant i < q$

and the (unchanged) protocol

- $\mathsf{Msg}(i) := m_L \leftarrow \mathsf{Msg}_L(i);\ m_R \leftarrow \mathsf{Msg}_R(i);\ \mathsf{ret}\ m_R$ for $0 \leqslant i < q$

followed by the hiding of the channels $\mathsf{Msg}(-)$.

Reversing the factorization process yields the protocol

- $\mathsf{Key} := \mathsf{samp}\ \mathsf{gen}_{\mathsf{key}}$
- $\mathsf{Enc}(i) := m_L \leftarrow \mathsf{Msg}_L(i);\ m_R \leftarrow \mathsf{Msg}_R(i);\ k \leftarrow \mathsf{Key};\ \mathsf{samp}\ \mathsf{enc}(m_R, k)$ for $0 \leqslant i < q$

as desired.

2 Symmetric-Key Encryption: CPA$-To-CPA Security

Many CPA-secure encryption schemas also satisfy the stronger property of having *pseudo-random ciphertexts in the presence of chosen-plaintext attacks (CPA$)*. This latter notion states that encoding q fixed messages with a secret key is computationally indistinguishable from generating q random ciphertexts. We now show that CPA$ security implies CPA security in IPDL.

Formally, we assume types $\mathsf{key}, \mathsf{msg}, \mathsf{ctxt}$ of keys, messages, and ciphertexts, respectively; a chosen message $\mathsf{zeros} : 1 \to \mathsf{msg}$; a uniform distribution $\mathsf{unif}_{\mathsf{ctxt}} : 1 \twoheadrightarrow \mathsf{ctxt}$ on ciphertexts; a probabilistic key generation algorithm $\mathsf{gen}_{\mathsf{key}} : 1 \twoheadrightarrow \mathsf{key}$; and a probabilistic encryption algorithm $\mathsf{enc} : \mathsf{msg} \times \mathsf{key} \twoheadrightarrow \mathsf{ctxt}$ that takes a message and a key, and returns a ciphertext. We will write $\mathsf{enc}(m, k)$ in place of $\mathsf{enc}\ (m, k)$.

We can express the CPA$ cryptographic assumption as the following protocol-level axiom: in the channel context $\{\mathsf{Msg}(i) : \mathsf{msg}\}_i$, $\mathsf{Key} : \mathsf{key}$, $\{\mathsf{Enc}(i) : \mathsf{ctxt}\}_i$ where $i := 1, \ldots, q$, the protocol

- $\mathsf{Key} := \mathsf{samp}\ \mathsf{gen}_{\mathsf{key}}$
- $\mathsf{Enc}(i) := m \leftarrow \mathsf{Msg}(i);\ k \leftarrow \mathsf{Key};\ \mathsf{samp}\ \mathsf{enc}(m, k)$ for $0 \leqslant i < q$

with inputs $\mathsf{Msg}(-)$, outputs $\mathsf{Enc}(-)$, and an internal channel Key rewrites approximately to the protocol

- $\mathsf{Enc}(i) := m \leftarrow \mathsf{Msg};\ \mathsf{samp}\ \mathsf{unif}_{\mathsf{ctxt}}$ for $0 \leqslant i < q$

We now show that the above axiom implies CPA security. The protocol

- $\mathsf{Key} := \mathsf{samp}\ \mathsf{gen}_{\mathsf{key}}$
- $\mathsf{Enc}(i) := m_L \leftarrow \mathsf{Msg}_L(i);\ m_R \leftarrow \mathsf{Msg}_R(i);\ k \leftarrow \mathsf{Key};\ \mathsf{samp}\ \mathsf{enc}(m_L, k)$ for $0 \leqslant i < q$

can be factored as the composition of the protocol

- $\mathsf{Key} := \mathsf{samp}\ \mathsf{gen}_{\mathsf{key}}$
- $\mathsf{Enc}(i) := m \leftarrow \mathsf{Msg}(i);\ k \leftarrow \mathsf{Key};\ \mathsf{samp}\ \mathsf{enc}(m, k)$ for $0 \leqslant i < q$

(the left-hand side of the CPA$ assumption) and the protocol

- $\mathsf{Msg}(i) := m_L \leftarrow \mathsf{Msg}_L(i);\ m_R \leftarrow \mathsf{Msg}_R(i);\ \mathsf{ret}\ m_L$ for $0 \leqslant i < q$

followed by the hiding of the channels $\mathsf{Msg}(-)$.

Applying the CPA$ assumption yields the composition of the protocol

- $\mathsf{Enc}(i) := m \leftarrow \mathsf{Msg};\ \mathsf{samp}\ \mathsf{unif}_{\mathsf{ctxt}}$ for $0 \leqslant i < q$

and the (unchanged) protocol

- $\mathsf{Msg}(i) := m_L \leftarrow \mathsf{Msg}_L(i);\ m_R \leftarrow \mathsf{Msg}_R(i);\ \mathsf{ret}\ m_L$ for $0 \leqslant i < q$

followed by the hiding of the channels $\mathsf{Msg}(-)$.

Substituting the channels $\mathsf{Msg}(-)$ away yields the protocol

- $\mathsf{Enc}(i) := m_L \leftarrow \mathsf{Msg}_L(i);\ m_R \leftarrow \mathsf{Msg}_R(i);\ \mathsf{samp}\ \mathsf{unif}_{\mathsf{ctxt}}$ for $0 \leqslant i < q$

The above can be factored as the composition of the protocol

- $\mathsf{Enc}(i) := m \leftarrow \mathsf{Msg}(i);\ \mathsf{samp}\ \mathsf{unif}_{\mathsf{ctxt}}$ for $0 \leqslant i < q$

(the right-hand side of the CPA$ assumption) and the protocol

- $\mathsf{Msg}(i) := m_L \leftarrow \mathsf{Msg}_L(i);\ m_R \leftarrow \mathsf{Msg}_R(i);\ \mathsf{ret}\ m_R$ for $0 \leqslant i < q$

followed by the hiding of the channels $\mathsf{Msg}(-)$.

Applying the CPA$ assumption again (this time from right to left) yields the composition of the protocol

- $\mathsf{Key} := \mathsf{samp}\ \mathsf{gen}_{\mathsf{key}}$
- $\mathsf{Enc}(i) := m \leftarrow \mathsf{Msg}(i);\ k \leftarrow \mathsf{Key};\ \mathsf{samp}\ \mathsf{enc}(m, k)$ for $0 \leqslant i < q$

and the (unchanged) protocol

- $\mathsf{Msg}(i) := m_L \leftarrow \mathsf{Msg}_L(i);\ m_R \leftarrow \mathsf{Msg}_R(i);\ \mathsf{ret}\ m_R$ for $0 \leqslant i < q$

followed by the hiding of the channels $\mathsf{Msg}(-)$.

Reversing the factorization process yields the protocol

- $\mathsf{Key} := \mathsf{samp}\ \mathsf{gen}_{\mathsf{key}}$
- $\mathsf{Enc}(i) := m_L \leftarrow \mathsf{Msg}_L(i);\ m_R \leftarrow \mathsf{Msg}_R(i);\ k \leftarrow \mathsf{Key};\ \mathsf{samp}\ \mathsf{enc}(m_R, k)$ for $0 \leqslant i < q$

as desired.

3 Symmetric-Key Encryption: Authenticated-To-Secure Channel

Alice wants to communicate q messages to Bob using an authenticated channel. The authenticated channel is not secure: it leaks each message to Eve, and waits to receive an ok message back from her before delivering the in-flight message. Thus, Eve cannot modify any of the messages but can read and delay them for any amount of time. To transmit information securely, Alice sends encryptions of her messages, which Bob decrypts using a shared key not known to Eve.

Formally, we assume types $\mathsf{key}, \mathsf{msg}, \mathsf{ctxt}$ of keys, messages, and ciphertexts, respectively; a chosen message $\mathsf{zeros} : 1 \to \mathsf{msg}$; a probabilistic key generation algorithm $\mathsf{gen}_{\mathsf{key}} : 1 \twoheadrightarrow \mathsf{key}$; a probabilistic encryption algorithm $\mathsf{enc} : \mathsf{msg} \times \mathsf{key} \twoheadrightarrow \mathsf{ctxt}$ that takes a message and a key, and returns a ciphertext; and a decryption algorithm $\mathsf{dec} : \mathsf{ctxt} \times \mathsf{key} \to \mathsf{msg}$ that takes a ciphertext and a key, and returns a message. We will write $\mathsf{enc}(m, k)$ and $\mathsf{dec}(c, k)$ in place of $\mathsf{enc}\ (m, k)$ and $\mathsf{dec}\ (c, k)$.

3.1 The Assumptions

The *correctness* assumption on the encryption schema states that encoding and decoding a single message with the same key yields the original message. We express this as a protocol-level axiom: in the channel context $\mathsf{In} : \mathsf{msg}, \mathsf{Key} : \mathsf{key}, \mathsf{Enc} : \mathsf{ctxt}, \mathsf{Dec} : \mathsf{msg}$ the protocol

- $\mathsf{Enc} := m \leftarrow \mathsf{In};\ k \leftarrow \mathsf{Key};\ \mathsf{samp}\ \mathsf{enc}(m, k)$
- $\mathsf{Dec} := c \leftarrow \mathsf{Enc};\ k \leftarrow \mathsf{Key};\ \mathsf{ret}\ \mathsf{dec}(c, k)$

with inputs In, Key and outputs Enc, Dec rewrites strictly to the protocol

- $\mathsf{Enc} := m \leftarrow \mathsf{In};\ k \leftarrow \mathsf{Key};\ \mathsf{samp}\ \mathsf{enc}(m, k)$
- $\mathsf{Dec} := \mathsf{read}\ \mathsf{In}$

Applying this assumption q times, we get that the protocol

- $\mathsf{Enc}(i) := m \leftarrow \mathsf{In}(i);\ k \leftarrow \mathsf{Key};\ \mathsf{samp}\ \mathsf{enc}(m, k)$ for $0 \leqslant i < q$
- $\mathsf{Dec}(i) := c \leftarrow \mathsf{Enc}(i);\ k \leftarrow \mathsf{Key};\ \mathsf{ret}\ \mathsf{dec}(c, k)$ for $0 \leqslant i < q$

rewrites strictly to the protocol

- $\mathsf{Enc}(i) := m \leftarrow \mathsf{In}(i);\ k \leftarrow \mathsf{Key};\ \mathsf{samp}\ \mathsf{enc}(m, k)$ for $0 \leqslant i < q$
- $\mathsf{Dec}(i) := \mathsf{read}\ \mathsf{In}(i)$ for $0 \leqslant i < q$

The CPA cryptographic assumption states that if the key is secret, encoding q fixed messages is computationally indistinguishable from encoding the chosen message q times: in the channel context $\{\mathsf{In}(i) : \mathsf{msg}\}_i$, $\mathsf{Key} : \mathsf{key}$, $\{\mathsf{Enc}(i) : \mathsf{ctxt}\}_i$ where $i := 1, \ldots, q$, the protocol

- $\mathsf{Key} := \mathsf{samp}\ \mathsf{gen}_{\mathsf{key}}$
- $\mathsf{Enc}(i) := m \leftarrow \mathsf{In}(i);\ k \leftarrow \mathsf{Key};\ \mathsf{samp}\ \mathsf{enc}(m, k)$ for $0 \leqslant i < q$

with inputs $\mathsf{In}(-)$, outputs $\mathsf{Enc}(-)$, and an internal channel Key rewrites approximately to the protocol

- $\mathsf{Key} := \mathsf{samp}\ \mathsf{gen}_{\mathsf{key}}$
- $\mathsf{Enc}(i) := m \leftarrow \mathsf{In}(i);\ k \leftarrow \mathsf{Key};\ \mathsf{samp}\ \mathsf{enc}(\mathsf{zeros}, k)$ for $0 \leqslant i < q$

3.2 The Ideal Functionality

The ideal functionality reads the input message, leaks a confirmation to Eve to signal that the message has been received, and, upon the approval from Eve, outputs the message:

- $\mathsf{LeakMsgRcvd}(i)^{\mathsf{id}}_{\mathsf{adv}} := m \leftarrow \mathsf{In}(i);\ \mathsf{ret}\ \checkmark$ for $0 \leqslant i < q$
- $\mathsf{Out}(i) := \_ \leftarrow \mathsf{OkMsg}(i)^{\mathsf{adv}}_{\mathsf{id}};\ \mathsf{read}\ \mathsf{In}(i)$ for $0 \leqslant i < q$

3.3 The Real Protocol

The real-world protocol consists of Alice, Bob, the key-generating functionality, and the authenticated channel. The functionality starts by calling the key generation algorithm:

- $\mathsf{Key} := \mathsf{samp}\ \mathsf{gen}_{\mathsf{key}}$

Alice encrypts each input with the provided key, samples a ciphertext from the resulting distribution, and sends it to the authenticated channel:

- $\mathsf{Send}(i) := m \leftarrow \mathsf{In}(i);\ k \leftarrow \mathsf{Key};\ \mathsf{samp}\ \mathsf{enc}(m, k)$

The authenticated channel leaks each ciphertext received from Alice to Eve, and, upon receiving the okay from Eve, forwards the ciphertext to Bob:

- $\mathsf{LeakCtxt}(i)^{\mathsf{net}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{Send}(i)$
- $\mathsf{Recv}(i) := \_ \leftarrow \mathsf{OkCtxt}(i)^{\mathsf{adv}}_{\mathsf{net}};\ \mathsf{read}\ \mathsf{Send}(i)$

Bob decrypts each ciphertext with the shared key and outputs the result:

- $\mathsf{Out}(i) := c \leftarrow \mathsf{Recv}(i);\ k \leftarrow \mathsf{Key};\ \mathsf{ret}\ \mathsf{dec}(c, k)$

Composing all of this together and hiding the internal communication yields the real-world protocol.

3.4 The Simulator

The simulator turns the adversarial inputs and outputs of the real world protocol into the adversarial inputs and outputs of the ideal functionality, thereby converting any adversary for the real-world protocol into an adversary for the ideal functionality. This means that the channels $\mathsf{LeakMsgRcvd}(-)^{\mathsf{id}}_{\mathsf{adv}}, \mathsf{OkCtxt}(-)^{\mathsf{adv}}_{\mathsf{net}}$ are inputs to the simulator and the channels $\mathsf{LeakCtxt}(-)^{\mathsf{net}}_{\mathsf{adv}}, \mathsf{OkMsg}(-)^{\mathsf{adv}}_{\mathsf{id}}$ are the outputs. Hence, upon receiving the empty message from the ideal functionality to indicate that a message has been received, the simulator must conjure up a ciphertext to leak to Eve. This is accomplished by generating a key and encrypting the chosen message:

- $\mathsf{Key} := \mathsf{samp}\ \mathsf{gen}_{\mathsf{key}}$
- $\mathsf{LeakCtxt}(i)^{\mathsf{net}}_{\mathsf{adv}} := \_ \leftarrow \mathsf{LeakMsgRcvd}(i)^{\mathsf{id}}_{\mathsf{adv}};\ k \leftarrow \mathsf{Key};\ \mathsf{samp}\ \mathsf{enc}(\mathsf{zeros}, k)$ for $0 \leqslant i < q$

Upon receiving the approval from Eve for the generated ciphertext, the simulator gives the approval to the functionality to output the message:

- $\mathsf{OkMsg}(i)^{\mathsf{adv}}_{\mathsf{id}} := \mathsf{read}\ \mathsf{OkCtxt}(i)^{\mathsf{adv}}_{\mathsf{net}}$ for $0 \leqslant i < q$

Putting this all together yields the following code for the simulator:

- $\mathsf{Key} := \mathsf{samp}\ \mathsf{gen}_{\mathsf{key}}$
- $\mathsf{LeakCtxt}(i)^{\mathsf{net}}_{\mathsf{adv}} := \_ \leftarrow \mathsf{LeakMsgRcvd}(i)^{\mathsf{id}}_{\mathsf{adv}};\ k \leftarrow \mathsf{Key};\ \mathsf{samp}\ \mathsf{enc}(\mathsf{zeros}, k)$ for $0 \leqslant i < q$
- $\mathsf{OkMsg}(i)^{\mathsf{adv}}_{\mathsf{id}} := \mathsf{read}\ \mathsf{OkCtxt}(i)^{\mathsf{adv}}_{\mathsf{net}}$ for $0 \leqslant i < q$

3.5 Real ≈ Ideal + Simulator

Composing the simulator with the ideal functionality and hiding the internal communication yields the following:

- $\mathsf{Key} := \mathsf{samp}\ \mathsf{gen}_{\mathsf{key}}$
- $\mathsf{LeakCtxt}(i)^{\mathsf{net}}_{\mathsf{adv}} := \_ \leftarrow \mathsf{LeakMsgRcvd}(i)^{\mathsf{id}}_{\mathsf{adv}};\ k \leftarrow \mathsf{Key};\ \mathsf{samp}\ \mathsf{enc}(\mathsf{zeros}, k)$ for $0 \leqslant i < q$
- $\mathsf{OkMsg}(i)^{\mathsf{adv}}_{\mathsf{id}} := \mathsf{read}\ \mathsf{OkCtxt}(i)^{\mathsf{adv}}_{\mathsf{net}}$ for $0 \leqslant i < q$
- $\mathsf{LeakMsgRcvd}(i)^{\mathsf{id}}_{\mathsf{adv}} := m \leftarrow \mathsf{In}(i);\ \mathsf{ret}\ \checkmark$ for $0 \leqslant i < q$
- $\mathsf{Out}(i) := \_ \leftarrow \mathsf{OkMsg}(i)^{\mathsf{adv}}_{\mathsf{id}};\ \mathsf{read}\ \mathsf{In}(i)$ for $0 \leqslant i < q$

The internal channels $\mathsf{LeakMsgRcvd}(-)^{\mathsf{id}}_{\mathsf{adv}}$ and $\mathsf{OkMsg}(-)^{\mathsf{adv}}_{\mathsf{id}}$ that originally served as a line of communication for Eve can now be substituted away:

- $\mathsf{Key} := \mathsf{samp}\ \mathsf{gen}_{\mathsf{key}}$
- $\mathsf{LeakCtxt}(i)^{\mathsf{net}}_{\mathsf{adv}} := m \leftarrow \mathsf{In}(i);\ k \leftarrow \mathsf{Key};\ \mathsf{samp}\ \mathsf{enc}(\mathsf{zeros}, k)$ for $0 \leqslant i < q$
- $\mathsf{Out}(i) := \_ \leftarrow \mathsf{OkCtxt}(i)^{\mathsf{adv}}_{\mathsf{net}};\ \mathsf{read}\ \mathsf{In}(i)$ for $0 \leqslant i < q$

Next we move on to simplifying the real protocol. Explicitly, we have the code below:

- $\mathsf{Key} := \mathsf{samp}\ \mathsf{gen}_{\mathsf{key}}$
- $\mathsf{Send}(i) := m \leftarrow \mathsf{In}(i);\ k \leftarrow \mathsf{Key};\ \mathsf{samp}\ \mathsf{enc}(m,k)$ for $0 \leqslant i < q$
- $\mathsf{LeakCtxt}(i)^{\mathsf{net}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{Send}(i)$ for $0 \leqslant i < q$
- $\mathsf{Recv}(i) := \_ \leftarrow \mathsf{OkCtxt}(i)^{\mathsf{adv}}_{\mathsf{net}};\ \mathsf{read}\ \mathsf{Send}(i)$ for $0 \leqslant i < q$
- $\mathsf{Out}(i) := c \leftarrow \mathsf{Recv}(i);\ k \leftarrow \mathsf{Key};\ \mathsf{ret}\ \mathsf{dec}(c,k)$ for $0 \leqslant i < q$

We first substitute away the internal channels $\mathsf{Recv}(-)$:

- $\mathsf{Key} := \mathsf{samp}\ \mathsf{gen}_{\mathsf{key}}$
- $\mathsf{Send}(i) := m \leftarrow \mathsf{In}(i);\ k \leftarrow \mathsf{Key};\ \mathsf{samp}\ \mathsf{enc}(m,k)$ for $0 \leqslant i < q$
- $\mathsf{LeakCtxt}(i)^{\mathsf{net}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{Send}(i)$ for $0 \leqslant i < q$
- $\mathsf{Out}(i) := \_ \leftarrow \mathsf{OkCtxt}(i)^{\mathsf{adv}}_{\mathsf{net}};\ c \leftarrow \mathsf{Send}(i);\ k \leftarrow \mathsf{Key};\ \mathsf{ret}\ \mathsf{dec}(c,k)$ for $0 \leqslant i < q$

Next we conceptually separate the encryption and decryption actions from the message-passing in the real-world by introducing new internal channels $\mathsf{Enc}(-)$ and $\mathsf{Dec}(-)$:

- $\mathsf{Key} := \mathsf{samp}\ \ \mathsf{gen}_{\mathsf{key}}$
- $\mathsf{Enc}(i) := m \leftarrow \mathsf{In}(i);\ k \leftarrow \mathsf{Key};\ \mathsf{samp}\ \mathsf{enc}(m,k)$ for $0 \leqslant i < q$
- $\mathsf{Send}(i) := \mathsf{read}\ \mathsf{Enc}(i)$ for $0 \leqslant i < q$
- $\mathsf{LeakCtxt}(i)^{\mathsf{net}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{Send}(i)$ for $0 \leqslant i < q$
- $\mathsf{Dec}(i) := c \leftarrow \mathsf{Send}(i);\ k \leftarrow \mathsf{Key};\ \mathsf{ret}\ \mathsf{dec}(c,k)$ for $0 \leqslant i < q$
- $\mathsf{Out}(i) := \_ \leftarrow \mathsf{OkCtxt}(i)^{\mathsf{adv}}_{\mathsf{net}};\ \mathsf{read}\ \mathsf{Dec}(i)$ for $0 \leqslant i < q$

We can now substitute away the internal channels $\mathsf{Send}(-)$ as well:

- $\mathsf{Key} := \mathsf{samp}\ \mathsf{gen}_{\mathsf{key}}$
- $\mathsf{Enc}(i) := m \leftarrow \mathsf{In}(i);\ k \leftarrow \mathsf{Key};\ \mathsf{samp}\ \mathsf{enc}(m,k)$ for $0 \leqslant i < q$
- $\mathsf{LeakCtxt}(i)^{\mathsf{net}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{Enc}(i)$ for $0 \leqslant i < q$
- $\mathsf{Dec}(i) := c \leftarrow \mathsf{Enc}(i);\ k \leftarrow \mathsf{Key};\ \mathsf{ret}\ \mathsf{dec}(c,k)$ for $0 \leqslant i < q$
- $\mathsf{Out}(i) := \_ \leftarrow \mathsf{OkCtxt}(i)^{\mathsf{adv}}_{\mathsf{net}};\ \mathsf{read}\ \mathsf{Dec}(i)$ for $0 \leqslant i < q$

As we observed earlier, the subprotocol

- $\mathsf{Enc}(i) := m \leftarrow \mathsf{In}(i);\ k \leftarrow \mathsf{Key};\ \mathsf{samp}\ \mathsf{enc}(m,k)$ for $0 \leqslant i < q$
- $\mathsf{Dec}(i) := c \leftarrow \mathsf{Enc}(i);\ k \leftarrow \mathsf{Key};\ \mathsf{ret}\ \mathsf{dec}(c,k)$ for $0 \leqslant i < q$

strictly rewrites to the following protocol snippet:

- $\mathsf{Enc}(i) := m \leftarrow \mathsf{In}(i);\ k \leftarrow \mathsf{Key};\ \mathsf{samp}\ \mathsf{enc}(m,k)$ for $0 \leqslant i < q$
- $\mathsf{Dec}(i) := \mathsf{read}\ \mathsf{In}(i)$ for $0 \leqslant i < q$

Our original protocol thus becomes:

- $\mathsf{Key} := \mathsf{samp}\ \mathsf{gen}_{\mathsf{key}}$
- $\mathsf{Enc}(i) := m \leftarrow \mathsf{In}(i);\ k \leftarrow \mathsf{Key};\ \mathsf{samp}\ \mathsf{enc}(m, k)$ for $0 \leqslant i < q$
- $\mathsf{LeakCtxt}(i)^{\mathsf{net}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{Enc}(i)$ for $0 \leqslant i < q$
- $\mathsf{Dec}(i) := \mathsf{read}\ \mathsf{In}(i)$ for $0 \leqslant i < q$
- $\mathsf{Out}(i) := \_ \leftarrow \mathsf{OkCtxt}(i)^{\mathsf{adv}}_{\mathsf{net}};\ \mathsf{read}\ \mathsf{Dec}(i)$ for $0 \leqslant i < q$

Since the channel Key is only used in the channels $\mathsf{Enc}(-)$, we can extract the following subprotocol, where Key is hidden:

- $\mathsf{Key} := \mathsf{samp}\ \mathsf{gen}_{\mathsf{key}}$
- $\mathsf{Enc}(i) := m \leftarrow \mathsf{In}(i);\ k \leftarrow \mathsf{Key};\ \mathsf{samp}\ \mathsf{enc}(m, k)$ for $0 \leqslant i < q$

The CPA assumption allows us to approximately rewrite the above protocol snippet to

- $\mathsf{Key} := \mathsf{samp}\ \mathsf{gen}_{\mathsf{key}}$
- $\mathsf{Enc}(i) := m \leftarrow \mathsf{In}(i);\ k \leftarrow \mathsf{Key};\ \mathsf{samp}\ \mathsf{enc}(\mathsf{zeros}, k)$ for $0 \leqslant i < q$

Plugging this back into the original protocol yields the following:

- $\mathsf{Key} := \mathsf{samp}\ \mathsf{gen}_{\mathsf{key}}$
- $\mathsf{Enc}(i) := m \leftarrow \mathsf{In}(i);\ k \leftarrow \mathsf{Key};\ \mathsf{samp}\ \mathsf{enc}(\mathsf{zeros}, k)$ for $0 \leqslant i < q$
- $\mathsf{LeakCtxt}(i)^{\mathsf{net}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{Enc}(i)$ for $0 \leqslant i < q$
- $\mathsf{Dec}(i) := \mathsf{read}\ \mathsf{In}(i)$ for $0 \leqslant i < q$
- $\mathsf{Out}(i) := \_ \leftarrow \mathsf{OkCtxt}(i)^{\mathsf{adv}}_{\mathsf{net}};\ \mathsf{read}\ \mathsf{Dec}(i)$ for $0 \leqslant i < q$

Finally, we can fold away the internal channels $\mathsf{Enc}(-)$ and $\mathsf{Dec}(-)$:

- $\mathsf{Key} := \mathsf{samp}\ \mathsf{gen}_{\mathsf{key}}$
- $\mathsf{LeakCtxt}(i)^{\mathsf{net}}_{\mathsf{adv}} := m \leftarrow \mathsf{In}(i);\ k \leftarrow \mathsf{Key};\ \mathsf{samp}\ \mathsf{enc}(\mathsf{zeros}, k)$ for $0 \leqslant i < q$
- $\mathsf{Out}(i) := \_ \leftarrow \mathsf{OkCtxt}(i)^{\mathsf{adv}}_{\mathsf{net}};\ \mathsf{read}\ \mathsf{In}(i)$ for $0 \leqslant i < q$

This is precisely the simplified composition of the ideal functionality and the simulator from the beginning of this section.

4 Diffie-Hellman Key Exchange (DHKE)

In symmetric-key encryption, the sender (Alice) and the receiver (Bob) need to agree on a shared secret key. One such key-agreement protocol is the Diffie-Hellman Key Exchange (DHKE), which we now prove secure in IPDL.

Formally, we assume a type key of secret keys (elements of $\{0, \ldots, p-1\}$, where p is a prime); a type msg of messages, also serving as public keys (elements of a cyclic group $G = \{g^0, \ldots, g^{p-1}\}$); a uniform distribution $\mathsf{unif}_{\mathsf{key}} : 1 \twoheadrightarrow \mathsf{key}$ on keys; a uniform distribution $\mathsf{unif}_{\mathsf{msg}} : 1 \twoheadrightarrow \mathsf{msg}$ on messages; the generator $\mathsf{g} : 1 \to \mathsf{msg}$ of G; and the exponentiation function $\mathsf{exp} : \mathsf{msg} \times \mathsf{key} \to \mathsf{msg}$ that raises an element of G to the power of $k \in \{0, \ldots, p-1\}$. We will write m^k in place of $\mathsf{exp}\ (m, k)$.

4.1 The Assumptions

At the level of expressions, we only need to know that exponents commute:

- $k : \mathsf{key}, l : \mathsf{key} \vdash (\mathsf{g}^l)^k = (\mathsf{g}^k)^l : \mathsf{msg}$, and

At the level of distributions, we need to know that sampling a random secret key k from $\mathsf{unif}_{\mathsf{key}}$ and returning g^k is the same as sampling from $\mathsf{unif}_{\mathsf{msg}}$ directly (a consequence of $\mathsf{unif}_{\mathsf{key}}$ being uniform),

- $\cdot \vdash \left(k \leftarrow \mathsf{unif}_{\mathsf{key}};\ 1[\mathsf{g}^k]\right) = \mathsf{unif}_{\mathsf{msg}} : \mathsf{msg}$

Finally, at the level of protocols we need the *decisional Diffie-Hellman (DDH)* cryptographic assumption: as long as the secret keys k, l are generated uniformly, even if the adversary knows the values g^k and g^l, they will be unable to distinguish $(g^k)^l$ from a uniformly generated element of G. The corresponding protocol-level axiom states that in the channel context $\mathsf{DDH}_3 : \mathsf{msg} \times (\mathsf{msg} \times \mathsf{msg})$, the single-reaction no-input protocol

- $\mathsf{DDH}_3 := k_A \leftarrow \mathsf{unif}_{\mathsf{key}};\ k_B \leftarrow \mathsf{unif}_{\mathsf{key}};\ \mathsf{ret}\ \left(\mathsf{g}^{k_A}, \left(\mathsf{g}^{k_B}, (\mathsf{g}^{k_A})^{k_B}\right)\right)$

rewrites approximately to the protocol

- $\mathsf{DDH}_3 := k_A \leftarrow \mathsf{unif}_{\mathsf{key}};\ k_B \leftarrow \mathsf{unif}_{\mathsf{key}};\ k_R \leftarrow \mathsf{unif}_{\mathsf{key}};\ \mathsf{ret}\ \left(\mathsf{g}^{k_A}, \left(\mathsf{g}^{k_B}, \mathsf{g}^{k_R}\right)\right)$

4.2 The Ideal Functionality

The ideal functionality for the key exchange generates the shared key randomly, and, upon the approval from the adversary, sends it to the respective party:

- $\mathsf{SharedKey} := \mathsf{samp}\ \mathsf{unif}_{\mathsf{msg}}$
- $\mathsf{Key}(\mathsf{Alice}) := \_ \leftarrow \mathsf{OkKey}(\mathsf{Alice})^{\mathsf{adv}}_{\mathsf{id}};\ \mathsf{read}\ \mathsf{SharedKey}$
- $\mathsf{Key}(\mathsf{Bob}) := \_ \leftarrow \mathsf{OkKey}(\mathsf{Bob})^{\mathsf{adv}}_{\mathsf{id}};\ \mathsf{read}\ \mathsf{SharedKey}$

Here the channel $\mathsf{SharedKey}$ is internal.

4.3 The Real Protocol

The real-world protocol consists of Alice, Bob, and two authenticated channels. Alice randomly generates a secret key k_A and sends the value g^{k_A} as her public key to Bob using one of the authenticated channels:

- $\mathsf{SecretKey}(\mathsf{Alice}) := \mathsf{samp}\ \mathsf{unif}_{\mathsf{key}}$
- $\mathsf{Send}(\mathsf{Alice}, \mathsf{Bob}) := k_A \leftarrow \mathsf{SecretKey}(\mathsf{Alice});\ \mathsf{ret}\ \mathsf{g}^{k_A}$

Symmetrically, Bob generates a secret key k_B and sends the value g^{k_B} to Alice using the second authenticated channel:

- $\mathsf{SecretKey}(\mathsf{Bob}) := \mathsf{samp}\ \mathsf{unif}_{\mathsf{key}}$
- $\mathsf{Send}(\mathsf{Bob}, \mathsf{Alice}) := k_B \leftarrow \mathsf{SecretKey}(\mathsf{Bob});\ \mathsf{ret}\ \mathsf{g}^{k_B}$

Each authenticated channel leaks the corresponding public key to the adversary and waits for their approval before forwarding the key to the other party:

- $\mathsf{LeakPublicKey}(\mathsf{Alice})^{\mathsf{adv}}_{\mathsf{net}} := \mathsf{read}\ \mathsf{Send}(\mathsf{Alice}, \mathsf{Bob})$
- $\mathsf{LeakPublicKey}(\mathsf{Bob})^{\mathsf{adv}}_{\mathsf{net}} := \mathsf{read}\ \mathsf{Send}(\mathsf{Bob}, \mathsf{Alice})$
- $\mathsf{Recv}(\mathsf{Alice}, \mathsf{Bob}) := \_ \leftarrow \mathsf{OkPublicKey}(\mathsf{Alice})^{\mathsf{adv}}_{\mathsf{net}};\ \mathsf{read}\ \mathsf{Send}(\mathsf{Alice}, \mathsf{Bob})$
- $\mathsf{Recv}(\mathsf{Bob}, \mathsf{Alice}) := \_ \leftarrow \mathsf{OkPublicKey}(\mathsf{Bob})^{\mathsf{adv}}_{\mathsf{net}};\ \mathsf{read}\ \mathsf{Send}(\mathsf{Bob}, \mathsf{Alice})$

Upon receiving Bob's public key g^{k_B}, Alice computes the shared key as the value $(g^{k_B})^{k_A}$:

- $\mathsf{Key}(\mathsf{Alice}) := p_B \leftarrow \mathsf{Recv}(\mathsf{Bob}, \mathsf{Alice});\ k_A \leftarrow \mathsf{SecretKey}(\mathsf{Alice});\ \mathsf{ret}\ p_B^{k_A}$

Analogously, upon receiving Alice's public key g^{k_A}, Bob computes the shared key as the value $(g^{k_A})^{k_B}$:

- $\mathsf{Key}(\mathsf{Bob}) := p_A \leftarrow \mathsf{Recv}(\mathsf{Alice}, \mathsf{Bob});\ k_B \leftarrow \mathsf{SecretKey}(\mathsf{Bob});\ \mathsf{ret}\ p_A^{k_B}$

Thus, we have the following code for Alice:

- $\mathsf{SecretKey}(\mathsf{Alice}) := \mathsf{samp}\ \mathsf{unif}_{\mathsf{key}}$
- $\mathsf{Send}(\mathsf{Alice}, \mathsf{Bob}) := k_A \leftarrow \mathsf{SecretKey}(\mathsf{Alice});\ \mathsf{ret}\ \mathsf{g}^{k_A}$
- $\mathsf{Key}(\mathsf{Alice}) := p_B \leftarrow \mathsf{Recv}(\mathsf{Bob}, \mathsf{Alice});\ k_A \leftarrow \mathsf{SecretKey}(\mathsf{Alice});\ \mathsf{ret}\ p_B^{k_A}$

The code for Bob has the symmetric form:

- $\mathsf{SecretKey}(\mathsf{Bob}) := \mathsf{samp}\ \mathsf{unif}_{\mathsf{key}}$
- $\mathsf{Send}(\mathsf{Bob}, \mathsf{Alice}) := k_B \leftarrow \mathsf{SecretKey}(\mathsf{Bob});\ \mathsf{ret}\ \mathsf{g}^{k_B}$
- $\mathsf{Key}(\mathsf{Bob}) := p_A \leftarrow \mathsf{Recv}(\mathsf{Alice}, \mathsf{Bob});\ k_B \leftarrow \mathsf{SecretKey}(\mathsf{Bob});\ \mathsf{ret}\ p_A^{k_B}$

At last we have the two authenticated channels: from Alice to Bob,

- $\mathsf{LeakPublicKey}(\mathsf{Alice})_{\mathsf{net}}^{\mathsf{adv}} := \mathsf{read}\ \mathsf{Send}(\mathsf{Alice}, \mathsf{Bob})$
- $\mathsf{Recv}(\mathsf{Alice}, \mathsf{Bob}) := \_ \leftarrow \mathsf{OkPublicKey}(\mathsf{Alice})_{\mathsf{net}}^{\mathsf{adv}};\ \mathsf{read}\ \mathsf{Send}(\mathsf{Alice}, \mathsf{Bob})$

and from Bob to Alice:

- $\mathsf{LeakPublicKey}(\mathsf{Bob})_{\mathsf{net}}^{\mathsf{adv}} := \mathsf{read}\ \mathsf{Send}(\mathsf{Bob}, \mathsf{Alice})$
- $\mathsf{Recv}(\mathsf{Bob}, \mathsf{Alice}) := \_ \leftarrow \mathsf{OkPublicKey}(\mathsf{Bob})_{\mathsf{net}}^{\mathsf{adv}};\ \mathsf{read}\ \mathsf{Send}(\mathsf{Bob}, \mathsf{Alice})$

Composing all of this together and hiding the internal communication yields the Diffie-Hellman Key Exchange protocol.

4.4 The Simulator

The channels $\mathsf{OkPublicKey}(\mathsf{Alice})_{\mathsf{net}}^{\mathsf{adv}}$ and $\mathsf{OkPublicKey}(\mathsf{Bob})_{\mathsf{net}}^{\mathsf{adv}}$ are inputs to the simulator, whereas the channels $\mathsf{LeakPublicKey}(\mathsf{Alice})_{\mathsf{net}}^{\mathsf{adv}}$, $\mathsf{LeakPublicKey}(\mathsf{Bob})_{\mathsf{net}}^{\mathsf{adv}}$ and $\mathsf{OkKey}(\mathsf{Alice})_{\mathsf{id}}^{\mathsf{adv}}$, $\mathsf{OkKey}(\mathsf{Bob})_{\mathsf{id}}^{\mathsf{adv}}$ are the outputs.

The simulator constructs the public keys for Alice and Bob exactly as in the real protocol: it randomly generates the respective secret keys k_A and k_B, and computes the corresponding public keys as g^{k_A} and g^{k_B}. In the real-world Diffie-Hellman Key Exchange, Alice can only construct the shared key once the adversary approves the forwarding of Bob's public key to her. Hence, the ideal functionality is only allowed to forward the shared key to Alice once the approval for Bob's public key clears, and vice versa.

- $\mathsf{SecretKey}(\mathsf{Alice}) := \mathsf{samp}\ \mathsf{unif}_{\mathsf{key}}$
- $\mathsf{SecretKey}(\mathsf{Bob}) := \mathsf{samp}\ \mathsf{unif}_{\mathsf{key}}$
- $\mathsf{PublicKey}(\mathsf{Alice}) := k_A \leftarrow \mathsf{SecretKey}(\mathsf{Alice});\ \mathsf{ret}\ \mathsf{g}^{k_A}$
- $\mathsf{PublicKey}(\mathsf{Bob}) := k_B \leftarrow \mathsf{SecretKey}(\mathsf{Bob});\ \mathsf{ret}\ \mathsf{g}^{k_B}$
- $\mathsf{LeakPublicKey}(\mathsf{Alice})_{\mathsf{net}}^{\mathsf{adv}} := \mathsf{read}\ \mathsf{PublicKey}(\mathsf{Alice})$
- $\mathsf{LeakPublicKey}(\mathsf{Bob})_{\mathsf{net}}^{\mathsf{adv}} := \mathsf{read}\ \mathsf{PublicKey}(\mathsf{Bob})$
- $\mathsf{OkKey}(\mathsf{Alice})_{\mathsf{id}}^{\mathsf{adv}} := \mathsf{read}\ \mathsf{OkPublicKey}(\mathsf{Bob})_{\mathsf{net}}^{\mathsf{adv}}$
- $\mathsf{OkKey}(\mathsf{Bob})_{\mathsf{id}}^{\mathsf{adv}} := \mathsf{read}\ \mathsf{OkPublicKey}(\mathsf{Alice})_{\mathsf{net}}^{\mathsf{adv}}$

In the above, the channels $\mathsf{SecretKey}(\mathsf{Alice})$, $\mathsf{SecretKey}(\mathsf{Bob})$ and $\mathsf{PublicKey}(\mathsf{Alice})$, $\mathsf{PublicKey}(\mathsf{Bob})$ are internal.

4.5 Real ≈ Ideal + Simulator

Plugging the simulator into the ideal functionality and substituting away the internal channels $\mathsf{OkKey(Alice)}^{\mathsf{adv}}_{\mathsf{id}}$, $\mathsf{OkKey(Bob)}^{\mathsf{adv}}_{\mathsf{id}}$ that originally served as a line of communication for the adversary yields the following:

- $\mathsf{SecretKey(Alice)} := \mathsf{samp}\ \mathsf{unif}_{\mathsf{key}}$
- $\mathsf{SecretKey(Bob)} := \mathsf{samp}\ \mathsf{unif}_{\mathsf{key}}$
- $\mathsf{SharedKey} := \mathsf{samp}\ \mathsf{unif}_{\mathsf{msg}}$
- $\mathsf{PublicKey(Alice)} := k_A \leftarrow \mathsf{SecretKey(Alice)};\ \mathsf{ret}\ \mathsf{g}^{k_A}$
- $\mathsf{PublicKey(Bob)} := k_B \leftarrow \mathsf{SecretKey(Bob)};\ \mathsf{ret}\ \mathsf{g}^{k_B}$
- $\mathsf{LeakPublicKey(Alice)}^{\mathsf{adv}}_{\mathsf{net}} := \mathsf{read}\ \mathsf{PublicKey(Alice)}$
- $\mathsf{LeakPublicKey(Bob)}^{\mathsf{adv}}_{\mathsf{net}} := \mathsf{read}\ \mathsf{PublicKey(Bob)}$
- $\mathsf{Key(Alice)} := \_ \leftarrow \mathsf{OkPublicKey(Bob)}^{\mathsf{adv}}_{\mathsf{net}};\ \mathsf{read}\ \mathsf{SharedKey}$
- $\mathsf{Key(Bob)} := \_ \leftarrow \mathsf{OkPublicKey(Alice)}^{\mathsf{adv}}_{\mathsf{net}};\ \mathsf{read}\ \mathsf{SharedKey}$

Next we move on to simplifying the real protocol. Explicitly, we have the code below:

- $\mathsf{SecretKey(Alice)} := \mathsf{samp}\ \mathsf{unif}_{\mathsf{key}}$
- $\mathsf{SecretKey(Bob)} := \mathsf{samp}\ \mathsf{unif}_{\mathsf{key}}$
- $\mathsf{Send(Alice, Bob)} := k_A \leftarrow \mathsf{SecretKey(Alice)};\ \mathsf{ret}\ \mathsf{g}^{k_A}$
- $\mathsf{Send(Bob, Alice)} := k_B \leftarrow \mathsf{SecretKey(Bob)};\ \mathsf{ret}\ \mathsf{g}^{k_B}$
- $\mathsf{LeakPublicKey(Alice)}^{\mathsf{adv}}_{\mathsf{net}} := \mathsf{read}\ \mathsf{Send(Alice, Bob)}$
- $\mathsf{LeakPublicKey(Bob)}^{\mathsf{adv}}_{\mathsf{net}} := \mathsf{read}\ \mathsf{Send(Bob, Alice)}$
- $\mathsf{Recv(Alice, Bob)} := \_ \leftarrow \mathsf{OkPublicKey(Alice)}^{\mathsf{adv}}_{\mathsf{net}};\ \mathsf{read}\ \mathsf{Send(Alice, Bob)}$
- $\mathsf{Recv(Bob, Alice)} := \_ \leftarrow \mathsf{OkPublicKey(Bob)}^{\mathsf{adv}}_{\mathsf{net}};\ \mathsf{read}\ \mathsf{Send(Bob, Alice)}$
- $\mathsf{Key(Alice)} := p_B \leftarrow \mathsf{Recv(Bob, Alice)};\ k_A \leftarrow \mathsf{SecretKey(Alice)};\ \mathsf{ret}\ p_B^{k_A}$
- $\mathsf{Key(Bob)} := p_A \leftarrow \mathsf{Recv(Alice, Bob)};\ k_B \leftarrow \mathsf{SecretKey(Bob)};\ \mathsf{ret}\ p_A^{k_B}$

The internal channels $\mathsf{Send(Alice, Bob)}$, $\mathsf{Send(Bob, Alice)}$ and $\mathsf{Recv(Alice, Bob)}$, $\mathsf{Recv(Bob, Alice)}$ can be substituted away:

- $\mathsf{SecretKey(Alice)} := \mathsf{samp}\ \mathsf{unif}_{\mathsf{key}}$
- $\mathsf{SecretKey(Bob)} := \mathsf{samp}\ \mathsf{unif}_{\mathsf{key}}$
- $\mathsf{LeakPublicKey(Alice)}^{\mathsf{adv}}_{\mathsf{net}} := k_A \leftarrow \mathsf{SecretKey(Alice)};\ \mathsf{ret}\ \mathsf{g}^{k_A}$
- $\mathsf{LeakPublicKey(Bob)}^{\mathsf{adv}}_{\mathsf{net}} := k_B \leftarrow \mathsf{SecretKey(Bob)};\ \mathsf{ret}\ \mathsf{g}^{k_B}$
- $\mathsf{Key(Alice)} := \_ \leftarrow \mathsf{OkPublicKey(Bob)}^{\mathsf{adv}}_{\mathsf{net}};\ k_B \leftarrow \mathsf{SecretKey(Bob)};\ k_A \leftarrow \mathsf{SecretKey(Alice)};\ \mathsf{ret}\ (\mathsf{g}^{k_B})^{k_A}$
- $\mathsf{Key(Bob)} := \_ \leftarrow \mathsf{OkPublicKey(Alice)}^{\mathsf{adv}}_{\mathsf{net}};\ k_A \leftarrow \mathsf{SecretKey(Alice)};\ k_B \leftarrow \mathsf{SecretKey(Bob)};\ \mathsf{ret}\ (\mathsf{g}^{k_A})^{k_B}$

Exponents commute by assumption, so we can rewrite the channel $\mathsf{Key(Alice)}$ equivalently as

- $\mathsf{SecretKey(Alice)} := \mathsf{samp}\ \mathsf{unif}_{\mathsf{key}}$

- SecretKey(Bob) := samp $\mathsf{unif}_{\mathsf{key}}$
- $\mathsf{LeakPublicKey(Alice)}^{\mathsf{adv}}_{\mathsf{net}}$:= $k_A \leftarrow$ SecretKey(Alice); ret g^{k_A}
- $\mathsf{LeakPublicKey(Bob)}^{\mathsf{adv}}_{\mathsf{net}}$:= $k_B \leftarrow$ SecretKey(Bob); ret g^{k_B}
- Key(Alice) := _ $\leftarrow$ $\mathsf{OkPublicKey(Bob)}^{\mathsf{adv}}_{\mathsf{net}}$; $k_B \leftarrow$ SecretKey(Bob); $k_A \leftarrow$ SecretKey(Alice); ret $(\mathsf{g}^{k_A})^{k_B}$
- Key(Bob) := _ $\leftarrow$ $\mathsf{OkPublicKey(Alice)}^{\mathsf{adv}}_{\mathsf{net}}$; $k_A \leftarrow$ SecretKey(Alice); $k_B \leftarrow$ SecretKey(Bob); ret $(\mathsf{g}^{k_A})^{k_B}$

After rearranging, the channels Key(Alice) and Key(Bob) construct the exact same shared key:

- SecretKey(Alice) := samp $\mathsf{unif}_{\mathsf{key}}$
- SecretKey(Bob) := samp $\mathsf{unif}_{\mathsf{key}}$
- $\mathsf{LeakPublicKey(Alice)}^{\mathsf{adv}}_{\mathsf{net}}$:= $k_A \leftarrow$ SecretKey(Alice); ret g^{k_A}
- $\mathsf{LeakPublicKey(Bob)}^{\mathsf{adv}}_{\mathsf{net}}$:= $k_B \leftarrow$ SecretKey(Bob); ret g^{k_B}
- Key(Alice) := _ $\leftarrow$ $\mathsf{OkPublicKey(Bob)}^{\mathsf{adv}}_{\mathsf{net}}$; $k_A \leftarrow$ SecretKey(Alice); $k_B \leftarrow$ SecretKey(Bob); ret $(\mathsf{g}^{k_A})^{k_B}$
- Key(Bob) := _ $\leftarrow$ $\mathsf{OkPublicKey(Alice)}^{\mathsf{adv}}_{\mathsf{net}}$; $k_A \leftarrow$ SecretKey(Alice); $k_B \leftarrow$ SecretKey(Bob); ret $(\mathsf{g}^{k_A})^{k_B}$

We can extract the key construction into a new internal channel SharedKey:

- SecretKey(Alice) := samp $\mathsf{unif}_{\mathsf{key}}$
- SecretKey(Bob) := samp $\mathsf{unif}_{\mathsf{key}}$
- SharedKey := $k_A \leftarrow$ SecretKey(Alice); $k_B \leftarrow$ SecretKey(Bob); ret $(\mathsf{g}^{k_A})^{k_B}$
- $\mathsf{LeakPublicKey(Alice)}^{\mathsf{adv}}_{\mathsf{net}}$:= $k_A \leftarrow$ SecretKey(Alice); ret g^{k_A}
- $\mathsf{LeakPublicKey(Bob)}^{\mathsf{adv}}_{\mathsf{net}}$:= $k_B \leftarrow$ SecretKey(Bob); ret g^{k_B}
- Key(Alice) := _ $\leftarrow$ $\mathsf{OkPublicKey(Bob)}^{\mathsf{adv}}_{\mathsf{net}}$; read SharedKey
- Key(Bob) := _ $\leftarrow$ $\mathsf{OkPublicKey(Alice)}^{\mathsf{adv}}_{\mathsf{net}}$; read SharedKey

Similarly, we can extract the public key construction for Alice and Bob into new internal channels PublicKey(Alice) and PublicKey(Bob):

- SecretKey(Alice) := samp $\mathsf{unif}_{\mathsf{key}}$
- SecretKey(Bob) := samp $\mathsf{unif}_{\mathsf{key}}$
- SharedKey := $k_A \leftarrow$ SecretKey(Alice); $k_B \leftarrow$ SecretKey(Bob); ret $(\mathsf{g}^{k_A})^{k_B}$
- PublicKey(Alice) := $k_A \leftarrow$ SecretKey(Alice); ret g^{k_A}
- PublicKey(Bob) := $k_B \leftarrow$ SecretKey(Bob); ret g^{k_B}
- $\mathsf{LeakPublicKey(Alice)}^{\mathsf{adv}}_{\mathsf{net}}$:= read PublicKey(Alice)
- $\mathsf{LeakPublicKey(Bob)}^{\mathsf{adv}}_{\mathsf{net}}$:= read PublicKey(Bob)
- Key(Alice) := _ $\leftarrow$ $\mathsf{OkPublicKey(Bob)}^{\mathsf{adv}}_{\mathsf{net}}$; read SharedKey
- Key(Bob) := _ $\leftarrow$ $\mathsf{OkPublicKey(Alice)}^{\mathsf{adv}}_{\mathsf{net}}$; read SharedKey

Adding a gratuitous dependency on the channel SecretKey(Bob) into the channel PublicKey(Alice), and, analogously, on the channel SecretKey(Alice) into the channel PublicKey(Bob) yields

- $\mathsf{SecretKey(Alice)} := \mathsf{samp}\ \mathsf{unif}_{\mathsf{key}}$
- $\mathsf{SecretKey(Bob)} := \mathsf{samp}\ \mathsf{unif}_{\mathsf{key}}$
- $\mathsf{SharedKey} := k_A \leftarrow \mathsf{SecretKey(Alice)};\ k_B \leftarrow \mathsf{SecretKey(Bob)};\ \mathsf{ret}\ (\mathsf{g}^{k_A})^{k_B}$
- $\mathsf{PublicKey(Alice)} := k_A \leftarrow \mathsf{SecretKey(Alice)};\ k_B \leftarrow \mathsf{SecretKey(Bob)};\ \mathsf{ret}\ \mathsf{g}^{k_A}$
- $\mathsf{PublicKey(Bob)} := k_A \leftarrow \mathsf{SecretKey(Alice)};\ k_B \leftarrow \mathsf{SecretKey(Bob)};\ \mathsf{ret}\ \mathsf{g}^{k_B}$
- $\mathsf{LeakPublicKey(Alice)}^{\mathsf{adv}}_{\mathsf{net}} := \mathsf{read\ PublicKey(Alice)}$
- $\mathsf{LeakPublicKey(Bob)}^{\mathsf{adv}}_{\mathsf{net}} := \mathsf{read\ PublicKey(Bob)}$
- $\mathsf{Key(Alice)} := \_ \leftarrow \mathsf{OkPublicKey(Bob)}^{\mathsf{adv}}_{\mathsf{net}};\ \mathsf{read\ SharedKey}$
- $\mathsf{Key(Bob)} := \_ \leftarrow \mathsf{OkPublicKey(Alice)}^{\mathsf{adv}}_{\mathsf{net}};\ \mathsf{read\ SharedKey}$

We can now extract the DDH triple into a new internal channel DDH_3:

- $\mathsf{SecretKey(Alice)} := \mathsf{samp}\ \mathsf{unif}_{\mathsf{key}}$
- $\mathsf{SecretKey(Bob)} := \mathsf{samp}\ \mathsf{unif}_{\mathsf{key}}$
- $\mathsf{DDH}_3 := k_A \leftarrow \mathsf{SecretKey(Alice)};\ k_B \leftarrow \mathsf{SecretKey(Bob)};\ \mathsf{ret}\ \left(\mathsf{g}^{k_A}, \left(\mathsf{g}^{k_B}, (\mathsf{g}^{k_A})^{k_B}\right)\right)$
- $\mathsf{SharedKey} := \left(k_A, \left(k_B, k\right)\right) \leftarrow \mathsf{DDH}_3;\ \mathsf{ret}\ k$
- $\mathsf{PublicKey(Alice)} := \left(k_A, \left(k_B, k\right)\right) \leftarrow \mathsf{DDH}_3;\ \mathsf{ret}\ k_A$
- $\mathsf{PublicKey(Bob)} := \left(k_A, \left(k_B, k\right)\right) \leftarrow \mathsf{DDH}_3;\ \mathsf{ret}\ k_B$
- $\mathsf{LeakPublicKey(Alice)}^{\mathsf{adv}}_{\mathsf{net}} := \mathsf{read\ PublicKey(Alice)}$
- $\mathsf{LeakPublicKey(Bob)}^{\mathsf{adv}}_{\mathsf{net}} := \mathsf{read\ PublicKey(Bob)}$
- $\mathsf{Key(Alice)} := \_ \leftarrow \mathsf{OkPublicKey(Bob)}^{\mathsf{adv}}_{\mathsf{net}};\ \mathsf{read\ SharedKey}$
- $\mathsf{Key(Bob)} := \_ \leftarrow \mathsf{OkPublicKey(Alice)}^{\mathsf{adv}}_{\mathsf{net}};\ \mathsf{read\ SharedKey}$

The internal channels SecretKey(Alice) and SecretKey(Bob) are now only used in the channel DDH_3, so we can fold them in:

- $\mathsf{DDH}_3 := k_A \leftarrow \mathsf{unif}_{\mathsf{key}};\ k_B \leftarrow \mathsf{unif}_{\mathsf{key}};\ \mathsf{ret}\ \left(\mathsf{g}^{k_A}, \left(\mathsf{g}^{k_B}, (\mathsf{g}^{k_A})^{k_B}\right)\right)$
- $\mathsf{SharedKey} := \left(k_A, \left(k_B, k\right)\right) \leftarrow \mathsf{DDH}_3;\ \mathsf{ret}\ k$
- $\mathsf{PublicKey(Alice)} := \left(k_A, \left(k_B, k\right)\right) \leftarrow \mathsf{DDH}_3;\ \mathsf{ret}\ k_A$
- $\mathsf{PublicKey(Bob)} := \left(k_A, \left(k_B, k\right)\right) \leftarrow \mathsf{DDH}_3;\ \mathsf{ret}\ k_B$
- $\mathsf{LeakPublicKey(Alice)}^{\mathsf{adv}}_{\mathsf{net}} := \mathsf{read\ PublicKey(Alice)}$
- $\mathsf{LeakPublicKey(Bob)}^{\mathsf{adv}}_{\mathsf{net}} := \mathsf{read\ PublicKey(Bob)}$
- $\mathsf{Key(Alice)} := \_ \leftarrow \mathsf{OkPublicKey(Bob)}^{\mathsf{adv}}_{\mathsf{net}};\ \mathsf{read\ SharedKey}$
- $\mathsf{Key(Bob)} := \_ \leftarrow \mathsf{OkPublicKey(Alice)}^{\mathsf{adv}}_{\mathsf{net}};\ \mathsf{read\ SharedKey}$

Using the DDH assumption, we can approximately rewrite the above protocol to

- $\mathsf{DDH}_3 := k_A \leftarrow \mathsf{unif}_{\mathsf{key}};\ k_B \leftarrow \mathsf{unif}_{\mathsf{key}};\ k_R \leftarrow \mathsf{unif}_{\mathsf{key}};\ \mathsf{ret}\ \left(\mathsf{g}^{k_A}, \left(\mathsf{g}^{k_B}, \mathsf{g}^{k_R}\right)\right)$
- $\mathsf{SharedKey} := \left(k_A, \left(k_B, k\right)\right) \leftarrow \mathsf{DDH}_3;\ \mathsf{ret}\ k$

- $\mathsf{PublicKey(Alice)} := \big(k_A, \big(k_B, k\big)\big) \leftarrow \mathsf{DDH}_3;\ \mathsf{ret}\ k_A$
- $\mathsf{PublicKey(Bob)} := \big(k_A, \big(k_B, k\big)\big) \leftarrow \mathsf{DDH}_3;\ \mathsf{ret}\ k_B$
- $\mathsf{LeakPublicKey(Alice)}^{\mathsf{adv}}_{\mathsf{net}} := \mathsf{read\ PublicKey(Alice)}$
- $\mathsf{LeakPublicKey(Bob)}^{\mathsf{adv}}_{\mathsf{net}} := \mathsf{read\ PublicKey(Bob)}$
- $\mathsf{Key(Alice)} := \_ \leftarrow \mathsf{OkPublicKey(Bob)}^{\mathsf{adv}}_{\mathsf{net}};\ \mathsf{read\ SharedKey}$
- $\mathsf{Key(Bob)} := \_ \leftarrow \mathsf{OkPublicKey(Alice)}^{\mathsf{adv}}_{\mathsf{net}};\ \mathsf{read\ SharedKey}$

Unfolding the three samplings from the channel DDH_3 into new internal channels $\mathsf{SecretKey(Alice)}$, $\mathsf{SecretKey(Bob)}$, Key yields

- $\mathsf{Key} := \mathsf{samp\ unif}_{\mathsf{key}}$
- $\mathsf{SecretKey(Alice)} := \mathsf{samp\ unif}_{\mathsf{key}}$
- $\mathsf{SecretKey(Bob)} := \mathsf{samp\ unif}_{\mathsf{key}}$
- $\mathsf{DDH}_3 := k_A \leftarrow \mathsf{SecretKey(Alice)};\ k_B \leftarrow \mathsf{SecretKey(Bob)};\ k_R \leftarrow \mathsf{Key};\ \mathsf{ret}\ \big(\mathsf{g}^{k_A}, \big(\mathsf{g}^{k_B}, \mathsf{g}^{k_R}\big)\big)$
- $\mathsf{SharedKey} := \big(k_A, \big(k_B, k\big)\big) \leftarrow \mathsf{DDH}_3;\ \mathsf{ret}\ k$
- $\mathsf{PublicKey(Alice)} := \big(k_A, \big(k_B, k\big)\big) \leftarrow \mathsf{DDH}_3;\ \mathsf{ret}\ k_A$
- $\mathsf{PublicKey(Bob)} := \big(k_A, \big(k_B, k\big)\big) \leftarrow \mathsf{DDH}_3;\ \mathsf{ret}\ k_B$
- $\mathsf{LeakPublicKey(Alice)}^{\mathsf{adv}}_{\mathsf{net}} := \mathsf{read\ PublicKey(Alice)}$
- $\mathsf{LeakPublicKey(Bob)}^{\mathsf{adv}}_{\mathsf{net}} := \mathsf{read\ PublicKey(Bob)}$
- $\mathsf{Key(Alice)} := \_ \leftarrow \mathsf{OkPublicKey(Bob)}^{\mathsf{adv}}_{\mathsf{net}};\ \mathsf{read\ SharedKey}$
- $\mathsf{Key(Bob)} := \_ \leftarrow \mathsf{OkPublicKey(Alice)}^{\mathsf{adv}}_{\mathsf{net}};\ \mathsf{read\ SharedKey}$

The internal channel DDH_3 can now be substituted away:

- $\mathsf{Key} := \mathsf{samp\ unif}_{\mathsf{key}}$
- $\mathsf{SecretKey(Alice)} := \mathsf{samp\ unif}_{\mathsf{key}}$
- $\mathsf{SecretKey(Bob)} := \mathsf{samp\ unif}_{\mathsf{key}}$
- $\mathsf{SharedKey} := k_A \leftarrow \mathsf{SecretKey(Alice)};\ k_B \leftarrow \mathsf{SecretKey(Bob)};\ k_R \leftarrow \mathsf{Key};\ \mathsf{ret}\ \mathsf{g}^{k_R}$
- $\mathsf{PublicKey(Alice)} := k_A \leftarrow \mathsf{SecretKey(Alice)};\ k_B \leftarrow \mathsf{SecretKey(Bob)};\ k_R \leftarrow \mathsf{Key};\ \mathsf{ret}\ \mathsf{g}^{k_A}$
- $\mathsf{PublicKey(Bob)} := k_A \leftarrow \mathsf{SecretKey(Alice)};\ k_B \leftarrow \mathsf{SecretKey(Bob)};\ k_R \leftarrow \mathsf{Key};\ \mathsf{ret}\ \mathsf{g}^{k_B}$
- $\mathsf{LeakPublicKey(Alice)}^{\mathsf{adv}}_{\mathsf{net}} := \mathsf{read\ PublicKey(Alice)}$
- $\mathsf{LeakPublicKey(Bob)}^{\mathsf{adv}}_{\mathsf{net}} := \mathsf{read\ PublicKey(Bob)}$
- $\mathsf{Key(Alice)} := \_ \leftarrow \mathsf{OkPublicKey(Bob)}^{\mathsf{adv}}_{\mathsf{net}};\ \mathsf{read\ SharedKey}$
- $\mathsf{Key(Bob)} := \_ \leftarrow \mathsf{OkPublicKey(Alice)}^{\mathsf{adv}}_{\mathsf{net}};\ \mathsf{read\ SharedKey}$

Dropping the gratuitous dependencies – on channels $\mathsf{SecretKey(Alice)}$, $\mathsf{SecretKey(Bob)}$ in the channel $\mathsf{SharedKey}$, on channels $\mathsf{SecretKey(Bob)}$, Key in the channel $\mathsf{PublicKey(Alice)}$, and on channels $\mathsf{SecretKey(Alice)}$, Key in the channel $\mathsf{PublicKey(Bob)}$ – yields the following:

- Key := samp $\mathsf{unif}_{\mathsf{key}}$
- SecretKey(Alice) := samp $\mathsf{unif}_{\mathsf{key}}$
- SecretKey(Bob) := samp $\mathsf{unif}_{\mathsf{key}}$
- SharedKey := $k_R \leftarrow$ Key; ret g^{k_R}
- PublicKey(Alice) := $k_A \leftarrow$ SecretKey(Alice); ret g^{k_A}
- PublicKey(Bob) := $k_B \leftarrow$ SecretKey(Bob); ret g^{k_B}
- LeakPublicKey(Alice)$^{\mathsf{adv}}_{\mathsf{net}}$:= read PublicKey(Alice)
- LeakPublicKey(Bob)$^{\mathsf{adv}}_{\mathsf{net}}$:= read PublicKey(Bob)
- Key(Alice) := $\_ \leftarrow$ OkPublicKey(Bob)$^{\mathsf{adv}}_{\mathsf{net}}$; read SharedKey
- Key(Bob) := $\_ \leftarrow$ OkPublicKey(Alice)$^{\mathsf{adv}}_{\mathsf{net}}$; read SharedKey

Since the channel Key is now only used in the channel SharedKey, we can fold it in:

- SecretKey(Alice) := samp $\mathsf{unif}_{\mathsf{key}}$
- SecretKey(Bob) := samp $\mathsf{unif}_{\mathsf{key}}$
- SharedKey := $k_R \leftarrow \mathsf{unif}_{\mathsf{key}}$; ret g^{k_R}
- PublicKey(Alice) := $k_A \leftarrow$ SecretKey(Alice); ret g^{k_A}
- PublicKey(Bob) := $k_B \leftarrow$ SecretKey(Bob); ret g^{k_B}
- LeakPublicKey(Alice)$^{\mathsf{adv}}_{\mathsf{net}}$:= read PublicKey(Alice)
- LeakPublicKey(Bob)$^{\mathsf{adv}}_{\mathsf{net}}$:= read PublicKey(Bob)
- Key(Alice) := $\_ \leftarrow$ OkPublicKey(Bob)$^{\mathsf{adv}}_{\mathsf{net}}$; read SharedKey
- Key(Bob) := $\_ \leftarrow$ OkPublicKey(Alice)$^{\mathsf{adv}}_{\mathsf{net}}$; read SharedKey

By assumption, sampling a random secret key k_R from $\mathsf{unif}_{\mathsf{key}}$ and returning g^{k_R} is the same as sampling from $\mathsf{unif}_{\mathsf{msg}}$ directly, so we can simplify the channel SharedKey as follows:

- SecretKey(Alice) := samp $\mathsf{unif}_{\mathsf{key}}$
- SecretKey(Bob) := samp $\mathsf{unif}_{\mathsf{key}}$
- SharedKey := samp $\mathsf{unif}_{\mathsf{msg}}$
- PublicKey(Alice) := $k_A \leftarrow$ SecretKey(Alice); ret g^{k_A}
- PublicKey(Bob) := $k_B \leftarrow$ SecretKey(Bob); ret g^{k_B}
- LeakPublicKey(Alice)$^{\mathsf{adv}}_{\mathsf{net}}$:= read PublicKey(Alice)
- LeakPublicKey(Bob)$^{\mathsf{adv}}_{\mathsf{net}}$:= read PublicKey(Bob)
- Key(Alice) := $\_ \leftarrow$ OkPublicKey(Bob)$^{\mathsf{adv}}_{\mathsf{net}}$; read SharedKey
- Key(Bob) := $\_ \leftarrow$ OkPublicKey(Alice)$^{\mathsf{adv}}_{\mathsf{net}}$; read SharedKey

This is precisely the simplified composition of the ideal functionality and the simulator from the beginning of this section.

5 Diffie-Hellman Key Exchange (DHKE) + One-Time Pad (OTP)

We now use the Diffie-Hellman Key Exchange protocol from Section 4 to turn an authenticated channel into a one-time pad that delivers a single secret message from Alice to Bob.

Formally, we assume a type key of secret keys (elements of $\{0, \ldots, p-1\}$, where p is a prime); a type msg of messages, also serving as public keys (elements of a cyclic group $G = \{g^0, \ldots, g^{p-1}\}$); a uniform distribution $\mathsf{unif}_{\mathsf{key}} : 1 \twoheadrightarrow \mathsf{key}$ on keys; a uniform distribution $\mathsf{unif}_{\mathsf{msg}} : 1 \twoheadrightarrow \mathsf{msg}$ on messages; the generator $\mathsf{g} : 1 \to \mathsf{msg}$ of G; the group multiplication function $\mathsf{mul} : \mathsf{msg} \times \mathsf{msg} \to \mathsf{msg}$, where we write $m * k$ in place of $\mathsf{mul}\ (m, k)$; the group inverse function $\mathsf{inv} : \mathsf{msg} \to \mathsf{msg}$, where we write k^{-1} in place of $\mathsf{inv}\ k$; and the exponentiation function $\mathsf{exp} : \mathsf{msg} \times \mathsf{key} \to \mathsf{msg}$ that raises an element of G to the power of $k \in \{0, \ldots, p-1\}$. We will write m^k in place of $\mathsf{exp}\ (m, k)$.

We will structure our proof modularly: in the first step, we establish that the Diffie-Hellman Key Exchange can be replaced with an idealization. In the second step, we prove that the resulting One-Time Pad protocol itself reduces to an idealization.

5.1 The Assumptions

In addition to the assumptions from Section 4, we will need the fact that inverses cancel on the right:

- $m : \mathsf{msg}, k : \mathsf{msg} \vdash (m * k) * k^{-1} = m : \mathsf{msg}$

Additionally, we need to know that that the distribution $\mathsf{unif}_{\mathsf{msg}}$ on messages is invariant under the operation of multiplication with a fixed message (a consequence of $\mathsf{unif}_{\mathsf{msg}}$ being uniform):

- $m : \mathsf{msg} \vdash \big(k \leftarrow \mathsf{unif}_{\mathsf{msg}};\ 1[m * k]\big) = \mathsf{unif}_{\mathsf{msg}} : \mathsf{msg}$

5.2 The Ideal Functionality

The ideal functionality reads the input message, leaks a confirmation to the adversary to signal that the message has been received, and, upon the approval from the adversary, outputs the message:

- $\mathsf{LeakMsgRcvd}^{\mathsf{id}}_{\mathsf{adv}} := m \leftarrow \mathsf{In};\ \mathsf{ret}\ \checkmark$
- $\mathsf{Out} := \_ \leftarrow \mathsf{OkMsg}^{\mathsf{adv}}_{\mathsf{id}};\ \mathsf{read}\ \mathsf{In}$

5.3 The Real Protocol

The real-world protocol consists of Alice, Bob, and three authenticated channels: two for carrying out the Diffie-Hellman Key Exchange, as described in Section 4, and one for communicating the secret message from Alice to Bob. The complete code for Alice is the composition of Alice's key exchange part with the single-reaction protocol

- $\mathsf{Send} := m \leftarrow \mathsf{In};\ k_A \leftarrow \mathsf{Key}(\mathsf{Alice});\ \mathsf{ret}\ m * k_A$

In the above, Alice encrypts the input message by multiplying it with the shared key produced by the key exchange, and sends the resulting ciphertext to Bob via the third authenticated channel. The complete code for Bob is the composition of Bob's key exchange part with the single-reaction protocol

- $\mathsf{Out} := c \leftarrow \mathsf{Recv};\ k_B \leftarrow \mathsf{Key}(\mathsf{Bob});\ \mathsf{ret}\ c * k_B^{-1}$

In the above, Bob decrypts the ciphertext received from Alice by multiplying it with the inverse of the shared key, and outputs the result. For communicating the ciphertext from Alice to Bob through the adversary, we have the third authenticated channel:

- $\mathsf{LeakCtxt}^{\mathsf{net}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{Send}$
- $\mathsf{Recv} := \_ \leftarrow \mathsf{OkCtxt}^{\mathsf{adv}}_{\mathsf{net}};\ \mathsf{read}\ \mathsf{Send}$

The real protocol is a composition of the two parties and the three authenticated channels, followed by the hiding of the internal communication.

5.4 The Simulator

The inputs to the simulator are channels $\mathsf{OkPublicKey(Alice)}^{\mathsf{adv}}_{\mathsf{net}}$, $\mathsf{OkPublicKey(Bob)}^{\mathsf{adv}}_{\mathsf{net}}$, $\mathsf{OkCtxt}^{\mathsf{adv}}_{\mathsf{net}}$, $\mathsf{LeakMsgRcvd}^{\mathsf{id}}_{\mathsf{adv}}$, whereas the outputs are channels $\mathsf{LeakPublicKey(Alice)}^{\mathsf{adv}}_{\mathsf{net}}$, $\mathsf{LeakPublicKey(Bob)}^{\mathsf{adv}}_{\mathsf{net}}$, $\mathsf{LeakCtxt}^{\mathsf{net}}_{\mathsf{adv}}$, $\mathsf{OkMsg}^{\mathsf{adv}}_{\mathsf{id}}$. We define the simulator as the composition of the key exchange simulator and the protocol

- $\mathsf{LeakCtxt}^{\mathsf{net}}_{\mathsf{adv}} := \_ \leftarrow \mathsf{LeakMsgRcvd}^{\mathsf{id}}_{\mathsf{adv}};\ \_ \leftarrow \mathsf{OkKey(Alice)}^{\mathsf{adv}}_{\mathsf{id}};\ \mathsf{samp\ unif}_{\mathsf{msg}}$
- $\mathsf{OkMsg}^{\mathsf{adv}}_{\mathsf{id}} := \_ \leftarrow \mathsf{OkCtxt}^{\mathsf{adv}}_{\mathsf{net}};\ \_ \leftarrow \mathsf{OkKey(Alice)}^{\mathsf{adv}}_{\mathsf{id}};\ \_ \leftarrow \mathsf{OkKey(Bob)}^{\mathsf{adv}}_{\mathsf{id}};\ \mathsf{ret}\ \checkmark$

followed by the hiding of the channels $\mathsf{OkKey(Alice)}^{\mathsf{adv}}_{\mathsf{id}}$ and $\mathsf{OkKey(Bob)}^{\mathsf{adv}}_{\mathsf{id}}$.

5.5 Real ≈ Ideal + Simulator: One-Time Pad

Plugging the simulator into the ideal functionality and substituting away the internal channels $\mathsf{LeakMsgRcvd}^{\mathsf{id}}_{\mathsf{adv}}$ and $\mathsf{OkMsg}^{\mathsf{adv}}_{\mathsf{id}}$ that originally served as a line of communication for the adversary yields the composition of the key exchange simulator with the protocol

- $\mathsf{LeakCtxt}^{\mathsf{net}}_{\mathsf{adv}} := m \leftarrow \mathsf{In};\ \_ \leftarrow \mathsf{OkKey(Alice)}^{\mathsf{adv}}_{\mathsf{id}};\ \mathsf{samp\ unif}_{\mathsf{msg}}$
- $\mathsf{Out} := \_ \leftarrow \mathsf{OkCtxt}^{\mathsf{adv}}_{\mathsf{net}};\ \_ \leftarrow \mathsf{OkKey(Alice)}^{\mathsf{adv}}_{\mathsf{id}};\ \_ \leftarrow \mathsf{OkKey(Bob)}^{\mathsf{adv}}_{\mathsf{id}};\ \mathsf{read\ In}$

followed by the hiding of the channels $\mathsf{OkKey(Alice)}^{\mathsf{adv}}_{\mathsf{id}}$ and $\mathsf{OkKey(Bob)}^{\mathsf{adv}}_{\mathsf{id}}$.

Next we move on to simplifying the real protocol. The real protocol can be refactored as the composition of the key exchange part and the protocol

- $\mathsf{Send} := m \leftarrow \mathsf{In};\ k_A \leftarrow \mathsf{Key(Alice)};\ \mathsf{ret}\ m * k_A$
- $\mathsf{LeakCtxt}^{\mathsf{net}}_{\mathsf{adv}} := \mathsf{read\ Send}$
- $\mathsf{Recv} := \_ \leftarrow \mathsf{OkCtxt}^{\mathsf{adv}}_{\mathsf{net}};\ \mathsf{read\ Send}$
- $\mathsf{Out} := c \leftarrow \mathsf{Recv};\ k_B \leftarrow \mathsf{Key(Bob)};\ \mathsf{ret}\ c * k_B^{-1}$

where the channels Send and Recv are internal, followed by the hiding of the channels Key(Alice) and Key(Bob).

The channels Send and Recv can be substituted away, turning our protocol into the composition of the key exchange part and the protocol

- $\mathsf{LeakCtxt}^{\mathsf{net}}_{\mathsf{adv}} := m \leftarrow \mathsf{In};\ k_A \leftarrow \mathsf{Key(Alice)};\ \mathsf{ret}\ m * k_A$
- $\mathsf{Out} := \_ \leftarrow \mathsf{OkCtxt}^{\mathsf{adv}}_{\mathsf{net}};\ m \leftarrow \mathsf{In};\ k_A \leftarrow \mathsf{Key(Alice)};\ k_B \leftarrow \mathsf{Key(Bob)};\ \mathsf{ret}\ (m * k_A) * k_B^{-1}$

followed by the hiding of the channels Key(Alice) and Key(Bob).

We can approximately rewrite the key exchange part as the composition of the key exchange idealization and the key exchange simulator, followed by the hiding of the channels $\mathsf{OkKey(Alice)}^{\mathsf{adv}}_{\mathsf{id}}$ and $\mathsf{OkKey(Bob)}^{\mathsf{adv}}_{\mathsf{id}}$. Thus, we can express the real protocol as the composition of the key exchange idealization, the key exchange simulator, and the protocol

- $\mathsf{LeakCtxt}^{\mathsf{net}}_{\mathsf{adv}} := m \leftarrow \mathsf{In};\ k_A \leftarrow \mathsf{Key(Alice)};\ \mathsf{ret}\ m * k_A$
- $\mathsf{Out} := \_ \leftarrow \mathsf{OkCtxt}^{\mathsf{adv}}_{\mathsf{net}};\ m \leftarrow \mathsf{In};\ k_A \leftarrow \mathsf{Key(Alice)};\ k_B \leftarrow \mathsf{Key(Bob)};\ \mathsf{ret}\ (m * k_A) * k_B^{-1}$

all followed by the hiding of the channels $\mathsf{OkKey(Alice)}^{\mathsf{adv}}_{\mathsf{id}}$, $\mathsf{OkKey(Bob)}^{\mathsf{adv}}_{\mathsf{id}}$, and Key(Alice), Key(Bob).

Composing the above protocol snippet with the key exchange idealization, and substituting away the internal channels Key(Alice) and Key(Bob) that originally served as a line of communication between the key exchange part and the one-time pad yields the protocol

- $\mathsf{SharedKey} := \mathsf{samp\ unif}_{\mathsf{msg}}$
- $\mathsf{LeakCtxt}^{\mathsf{net}}_{\mathsf{adv}} := m \leftarrow \mathsf{In};\ \_ \leftarrow \mathsf{OkKey(Alice)}^{\mathsf{adv}}_{\mathsf{id}};\ k \leftarrow \mathsf{SharedKey};\ \mathsf{ret}\ m * k$
- $\mathsf{Out} := \_ \leftarrow \mathsf{OkCtxt}^{\mathsf{adv}}_{\mathsf{net}};\ m \leftarrow \mathsf{In};\ \_ \leftarrow \mathsf{OkKey(Alice)}^{\mathsf{adv}}_{\mathsf{id}};\ \_ \leftarrow \mathsf{OkKey(Bob)}^{\mathsf{adv}}_{\mathsf{id}};\ k \leftarrow \mathsf{SharedKey};\ \mathsf{ret}\ (m * k) * k^{-1}$

where the channel $\mathsf{SharedKey}$ is internal. Canceling out k with its inverse in the channel Out yields

- $\mathsf{Out} := \_ \leftarrow \mathsf{OkCtxt}^{\mathsf{adv}}_{\mathsf{net}};\ m \leftarrow \mathsf{In};\ \_ \leftarrow \mathsf{OkKey}(\mathsf{Alice})^{\mathsf{adv}}_{\mathsf{id}};\ \_ \leftarrow \mathsf{OkKey}(\mathsf{Bob})^{\mathsf{adv}}_{\mathsf{id}};\ k \leftarrow \mathsf{SharedKey};\ \mathsf{ret}\ m$

Dropping the gratuitous dependency on the channel $\mathsf{SharedKey}$ yields

- $\mathsf{Out} := \_ \leftarrow \mathsf{OkCtxt}^{\mathsf{adv}}_{\mathsf{net}};\ m \leftarrow \mathsf{In};\ \_ \leftarrow \mathsf{OkKey}(\mathsf{Alice})^{\mathsf{adv}}_{\mathsf{id}};\ \_ \leftarrow \mathsf{OkKey}(\mathsf{Bob})^{\mathsf{adv}}_{\mathsf{id}};\ \mathsf{ret}\ m$

which simplifies to

- $\mathsf{Out} := \_ \leftarrow \mathsf{OkCtxt}^{\mathsf{adv}}_{\mathsf{net}};\ \_ \leftarrow \mathsf{OkKey}(\mathsf{Alice})^{\mathsf{adv}}_{\mathsf{id}};\ \_ \leftarrow \mathsf{OkKey}(\mathsf{Bob})^{\mathsf{adv}}_{\mathsf{id}};\ \mathsf{read}\ \mathsf{In}$

Summarizing, the cleaned-up version of the real protocol is the composition of the the key exchange simulator and the protocol

- $\mathsf{SharedKey} := \mathsf{samp}\ \mathsf{unif}_{\mathsf{msg}}$
- $\mathsf{LeakCtxt}^{\mathsf{net}}_{\mathsf{adv}} := m \leftarrow \mathsf{In};\ \_ \leftarrow \mathsf{OkKey}(\mathsf{Alice})^{\mathsf{adv}}_{\mathsf{id}};\ k \leftarrow \mathsf{SharedKey};\ \mathsf{ret}\ m * k$
- $\mathsf{Out} := \_ \leftarrow \mathsf{OkCtxt}^{\mathsf{adv}}_{\mathsf{net}};\ \_ \leftarrow \mathsf{OkKey}(\mathsf{Alice})^{\mathsf{adv}}_{\mathsf{id}};\ \_ \leftarrow \mathsf{OkKey}(\mathsf{Bob})^{\mathsf{adv}}_{\mathsf{id}};\ \mathsf{read}\ \mathsf{In}$

where the channel $\mathsf{SharedKey}$ is internal, followed by the hiding of the channels $\mathsf{OkKey}(\mathsf{Alice})^{\mathsf{adv}}_{\mathsf{id}}$ and $\mathsf{OkKey}(\mathsf{Bob})^{\mathsf{adv}}_{\mathsf{id}}$.

Since the channel $\mathsf{SharedKey}$ is now only used in the channel $\mathsf{LeakCtxt}^{\mathsf{net}}_{\mathsf{adv}}$, we can fold it in:

- $\mathsf{LeakCtxt}^{\mathsf{net}}_{\mathsf{adv}} := m \leftarrow \mathsf{In};\ \_ \leftarrow \mathsf{OkKey}(\mathsf{Alice})^{\mathsf{adv}}_{\mathsf{id}};\ k \leftarrow \mathsf{unif}_{\mathsf{msg}};\ \mathsf{ret}\ m * k$
- $\mathsf{Out} := \_ \leftarrow \mathsf{OkCtxt}^{\mathsf{adv}}_{\mathsf{net}};\ \_ \leftarrow \mathsf{OkKey}(\mathsf{Alice})^{\mathsf{adv}}_{\mathsf{id}};\ \_ \leftarrow \mathsf{OkKey}(\mathsf{Bob})^{\mathsf{adv}}_{\mathsf{id}};\ \mathsf{read}\ \mathsf{In}$

By assumption, the distribution $\mathsf{unif}_{\mathsf{msg}}$ on messages is invariant under the operation of multiplication with a fixed message, so we can simplify the channel $\mathsf{LeakCtxt}^{\mathsf{net}}_{\mathsf{adv}}$ as follows:

- $\mathsf{LeakCtxt}^{\mathsf{net}}_{\mathsf{adv}} := m \leftarrow \mathsf{In};\ \_ \leftarrow \mathsf{OkKey}(\mathsf{Alice})^{\mathsf{adv}}_{\mathsf{id}};\ \mathsf{samp}\ \mathsf{unif}_{\mathsf{msg}}$

Thus, the final version of the real protocol is the composition of the key exchange simulator and the protocol

- $\mathsf{LeakCtxt}^{\mathsf{net}}_{\mathsf{adv}} := m \leftarrow \mathsf{In};\ \_ \leftarrow \mathsf{OkKey}(\mathsf{Alice})^{\mathsf{adv}}_{\mathsf{id}};\ \mathsf{samp}\ \mathsf{unif}_{\mathsf{msg}}$
- $\mathsf{Out} := \_ \leftarrow \mathsf{OkCtxt}^{\mathsf{adv}}_{\mathsf{net}};\ \_ \leftarrow \mathsf{OkKey}(\mathsf{Alice})^{\mathsf{adv}}_{\mathsf{id}};\ \_ \leftarrow \mathsf{OkKey}(\mathsf{Bob})^{\mathsf{adv}}_{\mathsf{id}};\ \mathsf{read}\ \mathsf{In}$

followed by the hiding of the channels $\mathsf{OkKey}(\mathsf{Alice})^{\mathsf{adv}}_{\mathsf{id}}$ and $\mathsf{OkKey}(\mathsf{Bob})^{\mathsf{adv}}_{\mathsf{id}}$. But this is precisely the simplified composition of the ideal functionality and the simulator from the beginning of this section.

6 Public-Key Encryption: El Gamal

The El Gamal protocol is a CPA-secure public-key encryption schema based on the Diffie-Hellman Key Exchange. As in our previous case studies, Alice wants to communicate q messages to Bob using an authenticated channel. In the symmetric-key setting, the two parties had to agree on a secret shared key that was used both for encryption and decryption. In the public-key setting of El Gamal, Bob generates two keys: the public key is used for encryption and is known to both Alice and the adversary, whereas the secret key is used for decryption and known only to Bob.

Formally, we assume a type key of secret keys (elements of $\{0, \ldots, p-1\}$, where p is a prime); a type msg of messages, also serving as public keys (elements of a cyclic group $G = \{g^0, \ldots, g^{p-1}\}$); a uniform distribution $\mathsf{unif}_{\mathsf{key}} : 1 \twoheadrightarrow \mathsf{key}$ on keys; a uniform distribution $\mathsf{unif}_{\mathsf{msg}} : 1 \twoheadrightarrow \mathsf{msg}$ on messages; the generator $\mathsf{g} : 1 \to \mathsf{msg}$ of G; the group multiplication function $\mathsf{mul} : \mathsf{msg} \times \mathsf{msg} \to \mathsf{msg}$, where we write $m * k$ in place of $\mathsf{mul}\ (m, k)$; the group inverse function $\mathsf{inv} : \mathsf{msg} \to \mathsf{msg}$, where we write k^{-1} in place of $\mathsf{inv}\ k$; and the exponentiation function $\mathsf{exp} : \mathsf{msg} \times \mathsf{key} \to \mathsf{msg}$ that raises an element of G to the power of $k \in \{0, \ldots, p-1\}$. We will write m^k in place of $\mathsf{exp}\ (m, k)$.

6.1 The Assumptions

At the level of expressions, we only need to know that exponents commute and inverses cancel on the right:

- $k : \mathsf{key}, l : \mathsf{key} \vdash (\mathsf{g}^l)^k = (\mathsf{g}^k)^l : \mathsf{msg}$, and
- $m : \mathsf{msg}, k : \mathsf{key} \vdash (m * k) * k^{-1} = m : \mathsf{msg}$

At the level of distributions, we need to know that sampling a random secret key k from $\mathsf{unif}_{\mathsf{key}}$ and returning g^k is the same as sampling from $\mathsf{unif}_{\mathsf{msg}}$ directly (a consequence of $\mathsf{unif}_{\mathsf{key}}$ being uniform),

- $\cdot \vdash \left(k \leftarrow \mathsf{unif}_{\mathsf{key}};\ 1[\mathsf{g}^k]\right) = \mathsf{unif}_{\mathsf{msg}} : \mathsf{msg}$

and that the distribution $\mathsf{unif}_{\mathsf{msg}}$ on messages is invariant under the operation of multiplication with a fixed message (a consequence of $\mathsf{unif}_{\mathsf{msg}}$ being uniform),

- $m : \mathsf{msg} \vdash \left(k \leftarrow \mathsf{unif}_{\mathsf{msg}};\ 1[m * k]\right) = \mathsf{unif}_{\mathsf{msg}} : \mathsf{msg}$.

Finally, at the level of protocols we need the *decisional Diffie-Hellman (DDH)* cryptographic assumption: as long as the secret keys k, l are generated uniformly, even if the adversary knows the values g^k and g^l, they will be unable to distinguish $(g^k)^l$ from a uniformly generated element of G. The corresponding protocol-level axiom states that in the channel context $\mathsf{DDH}_3 : \mathsf{msg} \times (\mathsf{msg} \times \mathsf{msg})$, the single-reaction no-input protocol

- $\mathsf{DDH}_3 := k_B \leftarrow \mathsf{unif}_{\mathsf{key}};\ k_A \leftarrow \mathsf{unif}_{\mathsf{key}};\ \mathsf{ret}\ \left(\mathsf{g}^{k_B}, \left(\mathsf{g}^{k_A}, (\mathsf{g}^{k_B})^{k_A}\right)\right)$

rewrites approximately to the protocol

- $\mathsf{DDH}_3 := k_B \leftarrow \mathsf{unif}_{\mathsf{key}};\ k_A \leftarrow \mathsf{unif}_{\mathsf{key}};\ k_R \leftarrow \mathsf{unif}_{\mathsf{key}};\ \mathsf{ret}\ \left(\mathsf{g}^{k_B}, \left(\mathsf{g}^{k_A}, \mathsf{g}^{k_R}\right)\right)$

We now show that if the DDH assumption holds, the protocol

- $\mathsf{PublicKey} := k_B \leftarrow \mathsf{unif}_{\mathsf{key}};\ \mathsf{ret}\ \mathsf{g}^{k_B}$
- $\mathsf{DDH}_2 := p_B \leftarrow \mathsf{PublicKey};\ k_A \leftarrow \mathsf{unif}_{\mathsf{key}};\ \mathsf{ret}\ \left(\mathsf{g}^{k_A}, p_B^{k_A}\right)$

rewrites approximately to the protocol

- $\mathsf{PublicKey} := k_B \leftarrow \mathsf{unif}_{\mathsf{key}};\ \mathsf{ret}\ \mathsf{g}^{k_B}$
- $\mathsf{DDH}_2 := k_A \leftarrow \mathsf{unif}_{\mathsf{key}};\ k_R \leftarrow \mathsf{unif}_{\mathsf{key}};\ \mathsf{ret}\ \left(\mathsf{g}^{k_A}, \mathsf{g}^{k_R}\right)$

To show this, we first unfold the key samplings from the channels PublicKey and DDH_2 into new internal channels SecretKey and SecretEphemeralKey, respectively:

- $\mathsf{SecretKey} := \mathsf{samp}\ \mathsf{unif}_{\mathsf{key}}$
- $\mathsf{SecretEphemeralKey} := \mathsf{unif}_{\mathsf{key}}$
- $\mathsf{PublicKey} := k_B \leftarrow \mathsf{SecretKey};\ \mathsf{ret}\ \mathsf{g}^{k_B}$
- $\mathsf{DDH}_2 := p_B \leftarrow \mathsf{PublicKey};\ k_A \leftarrow \mathsf{SecretEphemeralKey};\ \mathsf{ret}\ \left(\mathsf{g}^{k_A}, p_B^{k_A}\right)$

Next we substitute the channel PublicKey into the channel DDH_2, yielding

- $\mathsf{SecretKey} := \mathsf{samp}\ \mathsf{unif}_{\mathsf{key}}$
- $\mathsf{SecretEphemeralKey} := \mathsf{samp}\ \mathsf{unif}_{\mathsf{key}}$
- $\mathsf{PublicKey} := k_B \leftarrow \mathsf{SecretKey};\ \mathsf{ret}\ \mathsf{g}^{k_B}$
- $\mathsf{DDH}_2 := k_B \leftarrow \mathsf{SecretKey};\ k_A \leftarrow \mathsf{SecretEphemeralKey};\ \mathsf{ret}\ \left(\mathsf{g}^{k_A}, (\mathsf{g}^{k_B})^{k_A}\right)$

We subsequently add a gratuitous dependency on the channel SecretEphemeralKey into the channel PublicKey:

- $\mathsf{SecretKey} := \mathsf{samp}\ \mathsf{unif}_{\mathsf{key}}$
- $\mathsf{SecretEphemeralKey} := \mathsf{samp}\ \mathsf{unif}_{\mathsf{key}}$
- $\mathsf{PublicKey} := k_B \leftarrow \mathsf{SecretKey};\ k_A \leftarrow \mathsf{SecretEphemeralKey};\ \mathsf{ret}\ \mathsf{g}^{k_B}$
- $\mathsf{DDH}_2 := k_B \leftarrow \mathsf{SecretKey};\ k_A \leftarrow \mathsf{SecretEphemeralKey};\ \mathsf{ret}\ \left(\mathsf{g}^{k_A}, (\mathsf{g}^{k_B})^{k_A}\right)$

We can now extract the DDH triple into a new internal channel DDH_3:

- $\mathsf{SecretKey} := \mathsf{samp}\ \mathsf{unif}_{\mathsf{key}}$
- $\mathsf{SecretEphemeralKey} := \mathsf{samp}\ \mathsf{unif}_{\mathsf{key}}$
- $\mathsf{DDH}_3 := k_B \leftarrow \mathsf{SecretKey};\ k_A \leftarrow \mathsf{SecretEphemeralKey};\ \mathsf{ret}\ \left(\mathsf{g}^{k_B}, \left(\mathsf{g}^{k_A}, (\mathsf{g}^{k_B})^{k_A}\right)\right)$
- $\mathsf{PublicKey} := \left(p_B, ddh_2\right) \leftarrow \mathsf{DDH}_3;\ \mathsf{ret}\ p_B$
- $\mathsf{DDH}_2 := \left(p_B, ddh_2\right) \leftarrow \mathsf{DDH}_3;\ \mathsf{ret}\ ddh_2$

The internal channels SecretKey and SecretEphemeralKey are now only used in the channel DDH_3, so we can fold them in:

- $\mathsf{DDH}_3 := k_B \leftarrow \mathsf{unif}_{\mathsf{key}};\ k_A \leftarrow \mathsf{unif}_{\mathsf{key}};\ \mathsf{ret}\ \left(\mathsf{g}^{k_B}, \left(\mathsf{g}^{k_A}, (\mathsf{g}^{k_B})^{k_A}\right)\right)$
- $\mathsf{PublicKey} := \left(p_B, ddh_2\right) \leftarrow \mathsf{DDH}_3;\ \mathsf{ret}\ p_B$
- $\mathsf{DDH}_2 := \left(p_B, ddh_2\right) \leftarrow \mathsf{DDH}_3;\ \mathsf{ret}\ ddh_2$

Using the DDH assumption, we can approximately rewrite the above protocol to

- $\mathsf{DDH}_3 := k_B \leftarrow \mathsf{unif}_{\mathsf{key}};\ k_A \leftarrow \mathsf{unif}_{\mathsf{key}};\ k_R \leftarrow \mathsf{unif}_{\mathsf{key}};\ \mathsf{ret}\ \left(\mathsf{g}^{k_B}, \left(\mathsf{g}^{k_A}, \mathsf{g}^{k_R}\right)\right)$
- $\mathsf{PublicKey} := \left(p_B, ddh_2\right) \leftarrow \mathsf{DDH}_3;\ \mathsf{ret}\ p_B$
- $\mathsf{DDH}_2 := \left(p_B, ddh_2\right) \leftarrow \mathsf{DDH}_3;\ \mathsf{ret}\ ddh_2$

Unfolding the three samplings from the channel DDH_3 into new internal channels SecretKey, SecretEphemeralKey, Key yields

- $\mathsf{SecretKey} := \mathsf{samp}\ \mathsf{unif}_{\mathsf{key}}$
- $\mathsf{SecretEphemeralKey} := \mathsf{samp}\ \mathsf{unif}_{\mathsf{key}}$
- $\mathsf{Key} := \mathsf{samp}\ \mathsf{unif}_{\mathsf{key}}$
- $\mathsf{DDH}_3 := k_B \leftarrow \mathsf{SecretKey};\ k_A \leftarrow \mathsf{SecretEphemeralKey};\ k_R \leftarrow \mathsf{Key};\ \mathsf{ret}\ \left(\mathsf{g}^{k_B}, \left(\mathsf{g}^{k_A}, \mathsf{g}^{k_R}\right)\right)$
- $\mathsf{PublicKey} := \left(p_B, ddh_2\right) \leftarrow \mathsf{DDH}_3;\ \mathsf{ret}\ p_B$
- $\mathsf{DDH}_2 := \left(p_B, ddh_2\right) \leftarrow \mathsf{DDH}_3;\ \mathsf{ret}\ ddh_2$

The internal channel DDH_3 can now be substituted away:

- $\mathsf{SecretKey} := \mathsf{samp}\ \mathsf{unif}_{\mathsf{key}}$
- $\mathsf{SecretEphemeralKey} := \mathsf{samp}\ \mathsf{unif}_{\mathsf{key}}$
- $\mathsf{Key} := \mathsf{samp}\ \mathsf{unif}_{\mathsf{key}}$
- $\mathsf{PublicKey} := k_B \leftarrow \mathsf{SecretKey};\ k_A \leftarrow \mathsf{SecretEphemeralKey};\ k_R \leftarrow \mathsf{Key};\ \mathsf{ret}\ \mathsf{g}^{k_B}$
- $\mathsf{DDH}_2 := k_B \leftarrow \mathsf{SecretKey};\ k_A \leftarrow \mathsf{SecretEphemeralKey};\ k_R \leftarrow \mathsf{Key};\ \mathsf{ret}\ \left(\mathsf{g}^{k_A}, \mathsf{g}^{k_R}\right)$

After dropping the gratuitous dependencies – on channels SecretEphemeralKey, Key in the channel PublicKey, and channel SecretKey in the channel DDH_2 – we end up with the following:

- $\mathsf{SecretKey} := \mathsf{samp}\ \mathsf{unif}_{\mathsf{key}}$
- $\mathsf{SecretEphemeralKey} := \mathsf{samp}\ \mathsf{unif}_{\mathsf{key}}$
- $\mathsf{Key} := \mathsf{samp}\ \mathsf{unif}_{\mathsf{key}}$
- $\mathsf{PublicKey} := k_B \leftarrow \mathsf{SecretKey};\ \mathsf{ret}\ \mathsf{g}^{k_B}$
- $\mathsf{DDH}_2 := k_A \leftarrow \mathsf{SecretEphemeralKey};\ k_R \leftarrow \mathsf{Key};\ \mathsf{ret}\ \left(\mathsf{g}^{k_A}, \mathsf{g}^{k_R}\right)$

The channel SecretKey is now only used in the channel PublicKey, and the channels SecretEphemeralKey and Key are only used in the channel DDH_2. Performing the appropriate foldings results in the protocol

- $\mathsf{PublicKey} := k_B \leftarrow \mathsf{unif}_{\mathsf{key}};\ \mathsf{ret}\ \mathsf{g}^{k_B}$
- $\mathsf{DDH}_2 := k_A \leftarrow \mathsf{unif}_{\mathsf{key}};\ k_R \leftarrow \mathsf{unif}_{\mathsf{key}};\ \mathsf{ret}\ \left(\mathsf{g}^{k_A}, \mathsf{g}^{k_R}\right)$

and we are done. Generalizing this argument to q sessions, we have just shown that the protocol

- $\mathsf{PublicKey} := k_B \leftarrow \mathsf{unif}_{\mathsf{key}};\ \mathsf{ret}\ \mathsf{g}^{k_B}$
- $\mathsf{DDH}_2(i) := p_B \leftarrow \mathsf{PublicKey};\ k_A \leftarrow \mathsf{unif}_{\mathsf{key}};\ \mathsf{ret}\ \left(\mathsf{g}^{k_A}, p_B^{k_A}\right)\ \text{for}\ 0 \leqslant i < q$

rewrites approximately to the protocol

- $\mathsf{PublicKey} := k_B \leftarrow \mathsf{unif}_{\mathsf{key}};\ \mathsf{ret}\ \mathsf{g}^{k_B}$
- $\mathsf{DDH}_2(i) := k_A \leftarrow \mathsf{unif}_{\mathsf{key}};\ k_R \leftarrow \mathsf{unif}_{\mathsf{key}};\ \mathsf{ret}\ \left(\mathsf{g}^{k_A}, \mathsf{g}^{k_R}\right)\ \text{for}\ 0 \leqslant i < q$

6.2 The Ideal Functionality

The ideal functionality reads the input message, leaks a confirmation to the adversary to signal that the message has been received, and, upon the approval from the adversary, outputs the message:

- $\mathsf{LeakMsgRcvd}(i)^{\mathsf{id}}_{\mathsf{adv}} := m \leftarrow \mathsf{In}(i);\ \mathsf{ret}\ \checkmark\ \text{for}\ 0 \leqslant i < q$
- $\mathsf{Out}(i) := \_ \leftarrow \mathsf{OkMsg}(i)^{\mathsf{adv}}_{\mathsf{id}};\ \mathsf{read}\ \mathsf{In}(i)\ \text{for}\ 0 \leqslant i < q$

6.3 The Real Protocol

The real-world protocol consists of Alice, Bob, and an authenticated channel. Bob randomly generates a secret key k_B visible only to him:

- $\mathsf{SecretKey} := \mathsf{samp}\ \mathsf{unif}_{\mathsf{key}}$

He computes the value g^{k_B} as the public key visible to everybody, including the adversary; we model the adversary's access to the public key by an explicit leakage function:

- $\mathsf{PublicKey} := k_B \leftarrow \mathsf{SecretKey};\ \mathsf{ret}\ \mathsf{g}^{k_B}$
- $\mathsf{LeakPublicKey}^{\mathsf{rec}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{PublicKey}$

Using the public key g^{k_B}, Alice encrypts each message by randomly generating a secret ephemeral key k_A and constructing the ciphertext in two parts. The first is the public ephemeral key g^{k_A}, and the second is the encoding of the input message by multiplying it with the session key $(g^{k_B})^{k_A}$:

- $\mathsf{SecretEphemeralKey}(i) := \mathsf{samp}\ \mathsf{unif}_{\mathsf{key}}\ \text{for}\ 0 \leqslant i < q$
- $\mathsf{PublicEphemeralKey}(i) := k_A \leftarrow \mathsf{SecretEphemeralKey}(i);\ \mathsf{ret}\ \mathsf{g}^{k_A}$

- $\mathsf{SessionKey}(i) := p_B \leftarrow \mathsf{PublicKey};\ k_A \leftarrow \mathsf{SecretEphemeralKey}(i);\ \mathsf{ret}\ p_B^{k_A}$ for $0 \leqslant i < q$
- $\mathsf{Send}(i) := m \leftarrow \mathsf{In}(i);\ k \leftarrow \mathsf{SessionKey}(i);\ p_A \leftarrow \mathsf{PublicEphemeralKey}(i);\ \mathsf{ret}\ \left(p_A, m * k\right)$ for $0 \leqslant i < q$

The authenticated channel leaks each ciphertext to the adversary and waits for their approval before forwarding it to Bob:

- $\mathsf{LeakCtxt}(i)^{\mathsf{adv}}_{\mathsf{net}} := \mathsf{read}\ \mathsf{Send}(i)$ for $0 \leqslant i < q$
- $\mathsf{Recv}(i) := \_ \leftarrow \mathsf{OkCtxt}(i)^{\mathsf{adv}}_{\mathsf{net}};\ \mathsf{read}\ \mathsf{Send}(i)$ for $0 \leqslant i < q$

At last, Bob decrypts the ciphertext by multiplying the message encoding (the second half) by the inverse of the session key $(g^{k_A})^{k_B}$. He constructs the session key from the the public ephemeral key (the first half) and his own secret key k_B:

- $\mathsf{Out}(i) := \left(p_A, c\right) \leftarrow \mathsf{Recv}(i);\ k_B \leftarrow \mathsf{SecretKey};\ \mathsf{ret}\ c * \left(p_A^{k_B}\right)^{-1}$ for $0 \leqslant i < q$

Thus, we have the following code for Alice:

- $\mathsf{SecretEphemeralKey}(i) := \mathsf{samp}\ \mathsf{unif}_{\mathsf{key}}$ for $0 \leqslant i < q$
- $\mathsf{PublicEphemeralKey}(i) := k_A \leftarrow \mathsf{SecretEphemeralKey}(i);\ \mathsf{ret}\ \mathsf{g}^{k_A}$ for $0 \leqslant i < q$
- $\mathsf{SessionKey}(i) := p_B \leftarrow \mathsf{PublicKey};\ k_A \leftarrow \mathsf{SecretEphemeralKey}(i);\ \mathsf{ret}\ p_B^{k_A}$ for $0 \leqslant i < q$
- $\mathsf{Send}(i) := m \leftarrow \mathsf{In}(i);\ k \leftarrow \mathsf{SessionKey}(i);\ p_A \leftarrow \mathsf{PublicEphemeralKey}(i);\ \mathsf{ret}\ \left(p_A, m * k\right)$ for $0 \leqslant i < q$

The code for Bob is below:

- $\mathsf{SecretKey} := \mathsf{samp}\ \mathsf{unif}_{\mathsf{key}}$
- $\mathsf{PublicKey} := k_B \leftarrow \mathsf{SecretKey};\ \mathsf{ret}\ \mathsf{g}^{k_B}$
- $\mathsf{LeakPublicKey}^{\mathsf{rec}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{PublicKey}$
- $\mathsf{Out}(i) := \left(p_A, c\right) \leftarrow \mathsf{Recv}(i);\ k_B \leftarrow \mathsf{SecretKey};\ \mathsf{ret}\ c * \left(p_A^{k_B}\right)^{-1}$ for $0 \leqslant i < q$

Finally, we recall the code for the authenticated channel:

- $\mathsf{LeakCtxt}(i)^{\mathsf{adv}}_{\mathsf{net}} := \mathsf{read}\ \mathsf{Send}(i)$ for $0 \leqslant i < q$
- $\mathsf{Recv}(i) := \_ \leftarrow \mathsf{OkCtxt}(i)^{\mathsf{adv}}_{\mathsf{net}};\ \mathsf{read}\ \mathsf{Send}(i)$ for $0 \leqslant i < q$

Composing all of this together and hiding the internal communication yields the real-world protocol.

6.4 The Simulator

The channels $\mathsf{OkCtxt}(-)^{\mathsf{adv}}_{\mathsf{net}}$, $\mathsf{LeakMsgRcvd}(-)^{\mathsf{id}}_{\mathsf{adv}}$ are inputs to the simulator, whereas the channels $\mathsf{OkMsg}(i)^{\mathsf{adv}}_{\mathsf{id}}$, $\mathsf{LeakPublicKey}^{\mathsf{rec}}_{\mathsf{adv}}$, $\mathsf{LeakCtxt}(-)^{\mathsf{adv}}_{\mathsf{net}}$ are the outputs. The simulator constructs the public key and the public ephemeral key exactly as in the real world: by randomly generating the secret key k_A and the secret ephemeral key k_B, and computing g^{k_A} and g^{k_B}. Upon receiving the empty message from the ideal functionality to indicate that a message has been received, the simulator leaks the ciphertext by pairing up the public ephemeral key with a random element of G:

- $\mathsf{SecretKey} := \mathsf{samp}\ \mathsf{unif}_{\mathsf{key}}$
- $\mathsf{PublicKey} := k_B \leftarrow \mathsf{SecretKey};\ \mathsf{ret}\ \mathsf{g}^{k_B}$
- $\mathsf{LeakPublicKey}^{\mathsf{rec}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{PublicKey}$
- $\mathsf{SecretEphemeralKey}(i) := \mathsf{samp}\ \mathsf{unif}_{\mathsf{key}}$ for $0 \leqslant i < q$
- $\mathsf{PublicEphemeralKey}(i) := k_A \leftarrow \mathsf{SecretEphemeralKey}(i);\ \mathsf{ret}\ \mathsf{g}^{k_A}$ for $0 \leqslant i < q$
- $\mathsf{LeakCtxt}(i)^{\mathsf{adv}}_{\mathsf{net}} := \_ \leftarrow \mathsf{LeakMsgRcvd}(i)^{\mathsf{id}}_{\mathsf{adv}};\ c \leftarrow \mathsf{unif}_{\mathsf{msg}};\ p_A \leftarrow \mathsf{PublicEphemeralKey}(i);\ \mathsf{ret}\ \left(p_A, c\right)$ for $0 \leqslant i < q$
- $\mathsf{OkMsg}(i)^{\mathsf{adv}}_{\mathsf{id}} := \mathsf{read}\ \mathsf{OkCtxt}(i)^{\mathsf{adv}}_{\mathsf{net}}$ for $0 \leqslant i < q$

6.5 Real ≈ Ideal + Simulator

Plugging the simulator into the ideal functionality and substituting away the internal channels $\mathsf{LeakMsgRcvd}(-)^{\mathsf{id}}_{\mathsf{adv}}$ and $\mathsf{OkMsg}(-)^{\mathsf{adv}}_{\mathsf{id}}$ that originally served as a line of communication for the adversary yields the following:

- $\mathsf{SecretKey} := \mathsf{samp}\ \mathsf{unif}_{\mathsf{key}}$
- $\mathsf{PublicKey} := k_B \leftarrow \mathsf{SecretKey};\ \mathsf{ret}\ \mathsf{g}^{k_B}$
- $\mathsf{LeakPublicKey}^{\mathsf{rec}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{PublicKey}$
- $\mathsf{SecretEphemeralKey}(i) := \mathsf{samp}\ \mathsf{unif}_{\mathsf{key}}$ for $0 \leqslant i < q$
- $\mathsf{PublicEphemeralKey}(i) := k_A \leftarrow \mathsf{SecretEphemeralKey}(i);\ \mathsf{ret}\ \mathsf{g}^{k_A}$ for $0 \leqslant i < q$
- $\mathsf{LeakCtxt}(i)^{\mathsf{adv}}_{\mathsf{net}} := \_ \leftarrow \mathsf{In}(i);\ c \leftarrow \mathsf{unif}_{\mathsf{msg}};\ p_A \leftarrow \mathsf{PublicEphemeralKey}(i);\ \mathsf{ret}\ \left(p_A, c\right)$ for $0 \leqslant i < q$
- $\mathsf{Out}(i) := \_ \leftarrow \mathsf{OkCtxt}(i)^{\mathsf{adv}}_{\mathsf{net}};\ \mathsf{read}\ \mathsf{In}(i)$ for $0 \leqslant i < q$

Next we move on to simplifying the real protocol. Explicitly, we have the code below:

- $\mathsf{SecretKey} := \mathsf{samp}\ \mathsf{unif}_{\mathsf{key}}$
- $\mathsf{PublicKey} := k_B \leftarrow \mathsf{SecretKey};\ \mathsf{ret}\ \mathsf{g}^{k_B}$
- $\mathsf{LeakPublicKey}^{\mathsf{rec}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{PublicKey}$
- $\mathsf{SecretEphemeralKey}(i) := \mathsf{samp}\ \mathsf{unif}_{\mathsf{key}}$ for $0 \leqslant i < q$
- $\mathsf{PublicEphemeralKey}(i) := k_A \leftarrow \mathsf{SecretEphemeralKey}(i);\ \mathsf{ret}\ \mathsf{g}^{k_A}$ for $0 \leqslant i < q$
- $\mathsf{SessionKey}(i) := p_B \leftarrow \mathsf{PublicKey};\ k_A \leftarrow \mathsf{SecretEphemeralKey}(i);\ \mathsf{ret}\ p_B^{k_A}$ for $0 \leqslant i < q$
- $\mathsf{Send}(i) := m \leftarrow \mathsf{In}(i);\ k \leftarrow \mathsf{SessionKey}(i);\ p_A \leftarrow \mathsf{PublicEphemeralKey}(i);\ \mathsf{ret}\ \left(p_A, m * k\right)$ for $0 \leqslant i < q$
- $\mathsf{LeakCtxt}(i)^{\mathsf{adv}}_{\mathsf{net}} := \mathsf{read}\ \mathsf{Send}(i)$ for $0 \leqslant i < q$
- $\mathsf{Recv}(i) := \_ \leftarrow \mathsf{OkCtxt}(i)^{\mathsf{adv}}_{\mathsf{net}};\ \mathsf{read}\ \mathsf{Send}(i)$ for $0 \leqslant i < q$
- $\mathsf{Out}(i) := \left(p_A, c\right) \leftarrow \mathsf{Recv}(i);\ k_B \leftarrow \mathsf{SecretKey};\ \mathsf{ret}\ c * \left(p_A^{k_B}\right)^{-1}$ for $0 \leqslant i < q$

We first substitute away the internal channels $\mathsf{Send}(-)$ and $\mathsf{Recv}(-)$:

- $\mathsf{SecretKey} := \mathsf{samp}\ \mathsf{unif}_{\mathsf{key}}$
- $\mathsf{PublicKey} := k_B \leftarrow \mathsf{SecretKey};\ \mathsf{ret}\ \mathsf{g}^{k_B}$
- $\mathsf{LeakPublicKey}^{\mathsf{rec}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{PublicKey}$
- $\mathsf{SecretEphemeralKey}(i) := \mathsf{samp}\ \mathsf{unif}_{\mathsf{key}}$ for $0 \leqslant i < q$
- $\mathsf{PublicEphemeralKey}(i) := k_A \leftarrow \mathsf{SecretEphemeralKey}(i);\ \mathsf{ret}\ \mathsf{g}^{k_A}$ for $0 \leqslant i < q$
- $\mathsf{SessionKey}(i) := p_B \leftarrow \mathsf{PublicKey};\ k_A \leftarrow \mathsf{SecretEphemeralKey}(i);\ \mathsf{ret}\ p_B^{k_A}$ for $0 \leqslant i < q$
- $\mathsf{LeakCtxt}(i)^{\mathsf{adv}}_{\mathsf{net}} := m \leftarrow \mathsf{In}(i);\ k \leftarrow \mathsf{SessionKey}(i);\ p_A \leftarrow \mathsf{PublicEphemeralKey}(i);\ \mathsf{ret}\ \left(p_A, m * k\right)$ for $0 \leqslant i < q$
- $\mathsf{Out}(i) := \_ \leftarrow \mathsf{OkCtxt}(i)^{\mathsf{adv}}_{\mathsf{net}};\ m \leftarrow \mathsf{In}(i);\ k \leftarrow \mathsf{SessionKey}(i);\ p_A \leftarrow \mathsf{PublicEphemeralKey}(i);\ k_B \leftarrow \mathsf{SecretKey};$ $\mathsf{ret}\ (m * k) * \left(p_A^{k_B}\right)^{-1}$ for $0 \leqslant i < q$

Substituting the channels $\mathsf{PublicKey}$, $\mathsf{SessionKey}(i)$, $\mathsf{PublicEphemeralKey}(i)$ into the channel $\mathsf{Out}(i)$ yields

- $\mathsf{Out}(i) := \_ \leftarrow \mathsf{OkCtxt}(i)^{\mathsf{adv}}_{\mathsf{net}};\ m \leftarrow \mathsf{In}(i);\ k_B \leftarrow \mathsf{SecretKey};\ k_A \leftarrow \mathsf{SecretEphemeralKey}(i);$
  $\mathsf{ret}\ \left(m * (\mathsf{g}^{k_B})^{k_A}\right) * \left((\mathsf{g}^{k_A})^{k_B}\right)^{-1}$ for $0 \leqslant i < q$

Since exponents commute, we can rewrite the channel $\mathsf{Out}(i)$ equivalently as

- $\mathsf{Out}(i) := \_ \leftarrow \mathsf{OkCtxt}(i)^{\mathsf{adv}}_{\mathsf{net}};\ m \leftarrow \mathsf{In}(i);\ k_B \leftarrow \mathsf{SecretKey};\ k_A \leftarrow \mathsf{SecretEphemeralKey}(i);$
  $\mathsf{ret}\ \left(m * (\mathsf{g}^{k_A})^{k_B}\right) * \left((\mathsf{g}^{k_A})^{k_B}\right)^{-1}$ for $0 \leqslant i < q$

We can now cancel out the two applications of $*$ to get:

- $\mathsf{Out}(i) := \_ \leftarrow \mathsf{OkCtxt}(i)^{\mathsf{adv}}_{\mathsf{net}};\ m \leftarrow \mathsf{In}(i);\ k_B \leftarrow \mathsf{SecretKey};\ k_A \leftarrow \mathsf{SecretEphemeralKey}(i);\ \mathsf{ret}\ m$ for $0 \leqslant i < q$

Dropping the gratuitous dependencies on the channels $\mathsf{SecretKey}$ and $\mathsf{SecretEphemeralKey}(i)$ yields

- $\mathsf{Out}(i) := \_ \leftarrow \mathsf{OkCtxt}(i)^{\mathsf{adv}}_{\mathsf{net}};\ m \leftarrow \mathsf{In}(i);\ \mathsf{ret}\ m$ for $0 \leqslant i < q$

which simplifies to

- $\mathsf{Out}(i) := \_ \leftarrow \mathsf{OkCtxt}(i)^{\mathsf{adv}}_{\mathsf{net}};\ \mathsf{read}\ \mathsf{In}(i)$ for $0 \leqslant i < q$

To summarize, our cleaned-up version of the real protocol looks as follows:

- $\mathsf{SecretKey} := \mathsf{samp}\ \mathsf{unif}_{\mathsf{key}}$
- $\mathsf{PublicKey} := k_B \leftarrow \mathsf{SecretKey};\ \mathsf{ret}\ \mathsf{g}^{k_B}$
- $\mathsf{LeakPublicKey}^{\mathsf{rec}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{PublicKey}$
- $\mathsf{SecretEphemeralKey}(i) := \mathsf{samp}\ \mathsf{unif}_{\mathsf{key}}$ for $0 \leqslant i < q$
- $\mathsf{PublicEphemeralKey}(i) := k_A \leftarrow \mathsf{SecretEphemeralKey}(i);\ \mathsf{ret}\ \mathsf{g}^{k_A}$ for $0 \leqslant i < q$
- $\mathsf{SessionKey}(i) := p_B \leftarrow \mathsf{PublicKey};\ k_A \leftarrow \mathsf{SecretEphemeralKey}(i);\ \mathsf{ret}\ p_B^{k_A}$ for $0 \leqslant i < q$
- $\mathsf{LeakCtxt}(i)^{\mathsf{adv}}_{\mathsf{net}} := m \leftarrow \mathsf{In}(i);\ k \leftarrow \mathsf{SessionKey}(i);\ p_A \leftarrow \mathsf{PublicEphemeralKey}(i);\ \mathsf{ret}\ \left(p_A, m * k\right)$ for $0 \leqslant i < q$
- $\mathsf{Out}(i) := \_ \leftarrow \mathsf{OkCtxt}(i)^{\mathsf{adv}}_{\mathsf{net}};\ \mathsf{read}\ \mathsf{In}(i)$ for $0 \leqslant i < q$

The channel $\mathsf{SecretKey}$ is now only used in $\mathsf{PublicKey}$, so we can fold it in:

- $\mathsf{PublicKey} := k_B \leftarrow \mathsf{unif}_{\mathsf{key}};\ \mathsf{ret}\ \mathsf{g}^{k_B}$
- $\mathsf{LeakPublicKey}^{\mathsf{rec}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{PublicKey}$
- $\mathsf{SecretEphemeralKey}(i) := \mathsf{samp}\ \mathsf{unif}_{\mathsf{key}}$ for $0 \leqslant i < q$
- $\mathsf{PublicEphemeralKey}(i) := k_A \leftarrow \mathsf{SecretEphemeralKey}(i);\ \mathsf{ret}\ \mathsf{g}^{k_A}$ for $0 \leqslant i < q$
- $\mathsf{SessionKey}(i) := p_B \leftarrow \mathsf{PublicKey};\ k_A \leftarrow \mathsf{SecretEphemeralKey}(i);\ \mathsf{ret}\ p_B^{k_A}$ for $0 \leqslant i < q$
- $\mathsf{LeakCtxt}(i)^{\mathsf{adv}}_{\mathsf{net}} := m \leftarrow \mathsf{In}(i);\ k \leftarrow \mathsf{SessionKey}(i);\ p_A \leftarrow \mathsf{PublicEphemeralKey}(i);\ \mathsf{ret}\ \left(p_A, m * k\right)$ for $0 \leqslant i < q$
- $\mathsf{Out}(i) := \_ \leftarrow \mathsf{OkCtxt}(i)^{\mathsf{adv}}_{\mathsf{net}};\ \mathsf{read}\ \mathsf{In}(i)$ for $0 \leqslant i < q$

Adding a gratuitous dependency on the channel $\mathsf{PublicKey}$ into the channel $\mathsf{PublicEphemeralKey}(i)$ yields

- $\mathsf{PublicKey} := k_B \leftarrow \mathsf{unif}_{\mathsf{key}};\ \mathsf{ret}\ \mathsf{g}^{k_B}$
- $\mathsf{LeakPublicKey}^{\mathsf{rec}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{PublicKey}$
- $\mathsf{SecretEphemeralKey}(i) := \mathsf{samp}\ \mathsf{unif}_{\mathsf{key}}$ for $0 \leqslant i < q$
- $\mathsf{PublicEphemeralKey}(i) := p_B \leftarrow \mathsf{PublicKey};\ k_A \leftarrow \mathsf{SecretEphemeralKey}(i);\ \mathsf{ret}\ \mathsf{g}^{k_A}$ for $0 \leqslant i < q$

- $\mathsf{SessionKey}(i) := p_B \leftarrow \mathsf{PublicKey};\ k_A \leftarrow \mathsf{SecretEphemeralKey}(i);\ \mathsf{ret}\ p_B^{k_A}$ for $0 \leqslant i < q$
- $\mathsf{LeakCtxt}(i)^{\mathsf{adv}}_{\mathsf{net}} := m \leftarrow \mathsf{In}(i);\ k \leftarrow \mathsf{SessionKey}(i);\ p_A \leftarrow \mathsf{PublicEphemeralKey}(i);\ \mathsf{ret}\ \left(p_A, m * k\right)$ for $0 \leqslant i < q$
- $\mathsf{Out}(i) := \_ \leftarrow \mathsf{OkCtxt}(i)^{\mathsf{adv}}_{\mathsf{net}};\ \mathsf{read}\ \mathsf{In}(i)$ for $0 \leqslant i < q$

We can now combine the construction of the public ephemeral key and the session key into a new internal channel $\mathsf{DDH}_2(i)$:

- $\mathsf{PublicKey} := k_B \leftarrow \mathsf{unif}_{\mathsf{key}};\ \mathsf{ret}\ \mathsf{g}^{k_B}$
- $\mathsf{LeakPublicKey}^{\mathsf{rec}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{PublicKey}$
- $\mathsf{SecretEphemeralKey}(i) := \mathsf{samp}\ \mathsf{unif}_{\mathsf{key}}$ for $0 \leqslant i < q$
- $\mathsf{DDH}_2(i) := p_B \leftarrow \mathsf{PublicKey};\ k_A \leftarrow \mathsf{SecretEphemeralKey}(i);\ \mathsf{ret}\ \left(\mathsf{g}^{k_A}, p_B^{k_A}\right)$ for $0 \leqslant i < q$
- $\mathsf{PublicEphemeralKey}(i) := \left(p_A, k\right) \leftarrow \mathsf{DDH}_2(i);\ \mathsf{ret}\ p_A$ for $0 \leqslant i < q$
- $\mathsf{SessionKey}(i) := \left(p_A, k\right) \leftarrow \mathsf{DDH}_2(i);\ \mathsf{ret}\ k$ for $0 \leqslant i < q$
- $\mathsf{LeakCtxt}(i)^{\mathsf{adv}}_{\mathsf{net}} := m \leftarrow \mathsf{In}(i);\ k \leftarrow \mathsf{SessionKey}(i);\ p_A \leftarrow \mathsf{PublicEphemeralKey}(i);\ \mathsf{ret}\ \left(p_A, m * k\right)$ for $0 \leqslant i < q$
- $\mathsf{Out}(i) := \_ \leftarrow \mathsf{OkCtxt}(i)^{\mathsf{adv}}_{\mathsf{net}};\ \mathsf{read}\ \mathsf{In}(i)$ for $0 \leqslant i < q$

The channel $\mathsf{SecretEphemeralKey}(i)$ is now only used in $\mathsf{DDH}_2(i)$, so we can fold it in:

- $\mathsf{PublicKey} := k_B \leftarrow \mathsf{unif}_{\mathsf{key}};\ \mathsf{ret}\ \mathsf{g}^{k_B}$
- $\mathsf{LeakPublicKey}^{\mathsf{rec}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{PublicKey}$
- $\mathsf{DDH}_2(i) := p_B \leftarrow \mathsf{PublicKey};\ k_A \leftarrow \mathsf{unif}_{\mathsf{key}};\ \mathsf{ret}\ \left(\mathsf{g}^{k_A}, p_B^{k_A}\right)$ for $0 \leqslant i < q$
- $\mathsf{PublicEphemeralKey}(i) := \left(p_A, k\right) \leftarrow \mathsf{DDH}_2(i);\ \mathsf{ret}\ p_A$ for $0 \leqslant i < q$
- $\mathsf{SessionKey}(i) := \left(p_A, k\right) \leftarrow \mathsf{DDH}_2(i);\ \mathsf{ret}\ k$ for $0 \leqslant i < q$
- $\mathsf{LeakCtxt}(i)^{\mathsf{adv}}_{\mathsf{net}} := m \leftarrow \mathsf{In}(i);\ k \leftarrow \mathsf{SessionKey}(i);\ p_A \leftarrow \mathsf{PublicEphemeralKey}(i);\ \mathsf{ret}\ \left(p_A, m * k\right)$ for $0 \leqslant i < q$
- $\mathsf{Out}(i) := \_ \leftarrow \mathsf{OkCtxt}(i)^{\mathsf{adv}}_{\mathsf{net}};\ \mathsf{read}\ \mathsf{In}(i)$ for $0 \leqslant i < q$

As we observed earlier, the protocol snippet

- $\mathsf{PublicKey} := k_B \leftarrow \mathsf{unif}_{\mathsf{key}};\ \mathsf{ret}\ \mathsf{g}^{k_B}$
- $\mathsf{DDH}_2(i) := p_B \leftarrow \mathsf{PublicKey};\ k_A \leftarrow \mathsf{unif}_{\mathsf{key}};\ \mathsf{ret}\ \left(\mathsf{g}^{k_A}, p_B^{k_A}\right)$ for $0 \leqslant i < q$

rewrites approximately to

- $\mathsf{PublicKey} := k_B \leftarrow \mathsf{unif}_{\mathsf{key}};\ \mathsf{ret}\ \mathsf{g}^{k_B}$
- $\mathsf{DDH}_2(i) := k_A \leftarrow \mathsf{unif}_{\mathsf{key}};\ k_R \leftarrow \mathsf{unif}_{\mathsf{key}};\ \mathsf{ret}\ \left(\mathsf{g}^{k_A}, \mathsf{g}^{k_R}\right)$ for $0 \leqslant i < q$

Our real-world protocol therefore approximately rewrites to the protocol

- $\mathsf{PublicKey} := k_B \leftarrow \mathsf{unif}_{\mathsf{key}};\ \mathsf{ret}\ \mathsf{g}^{k_B}$
- $\mathsf{LeakPublicKey}^{\mathsf{rec}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{PublicKey}$
- $\mathsf{DDH}_2(i) := k_A \leftarrow \mathsf{unif}_{\mathsf{key}};\ k_R \leftarrow \mathsf{unif}_{\mathsf{key}};\ \mathsf{ret}\ \left(\mathsf{g}^{k_A}, \mathsf{g}^{k_R}\right)$ for $0 \leqslant i < q$
- $\mathsf{PublicEphemeralKey}(i) := \left(p_A, k\right) \leftarrow \mathsf{DDH}_2(i);\ \mathsf{ret}\ p_A$ for $0 \leqslant i < q$

- $\mathsf{SessionKey}(i) := \big(p_A, k\big) \leftarrow \mathsf{DDH}_2(i);\ \mathsf{ret}\ k$ for $0 \leqslant i < q$
- $\mathsf{LeakCtxt}(i)^{\mathsf{adv}}_{\mathsf{net}} := m \leftarrow \mathsf{In}(i);\ k \leftarrow \mathsf{SessionKey}(i);\ p_A \leftarrow \mathsf{PublicEphemeralKey}(i);\ \mathsf{ret}\ \big(p_A, m * k\big)$ for $0 \leqslant i < q$
- $\mathsf{Out}(i) := \_ \leftarrow \mathsf{OkCtxt}(i)^{\mathsf{adv}}_{\mathsf{net}};\ \mathsf{read}\ \mathsf{In}(i)$ for $0 \leqslant i < q$

Unfolding the two samplings from $\mathsf{DDH}_2(i)$ into new internal channels $\mathsf{SecretEphemeralKey}(i)$ and $\mathsf{Key}(i)$ yields

- $\mathsf{PublicKey} := k_B \leftarrow \mathsf{unif}_{\mathsf{key}};\ \mathsf{ret}\ \mathsf{g}^{k_B}$
- $\mathsf{LeakPublicKey}^{\mathsf{rec}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{PublicKey}$
- $\mathsf{SecretEphemeralKey}(i) := \mathsf{samp}\ \mathsf{unif}_{\mathsf{key}}$ for $0 \leqslant i < q$
- $\mathsf{Key}(i) := \mathsf{samp}\ \mathsf{unif}_{\mathsf{key}}$ for $0 \leqslant i < q$
- $\mathsf{DDH}_2(i) := k_A \leftarrow \mathsf{SecretEphemeralKey}(i);\ k_R \leftarrow \mathsf{Key}(i);\ \mathsf{ret}\ \big(\mathsf{g}^{k_A}, \mathsf{g}^{k_R}\big)$ for $0 \leqslant i < q$
- $\mathsf{PublicEphemeralKey}(i) := \big(p_A, k\big) \leftarrow \mathsf{DDH}_2(i);\ \mathsf{ret}\ p_A$ for $0 \leqslant i < q$
- $\mathsf{SessionKey}(i) := \big(p_A, k\big) \leftarrow \mathsf{DDH}_2(i);\ \mathsf{ret}\ k$ for $0 \leqslant i < q$
- $\mathsf{LeakCtxt}(i)^{\mathsf{adv}}_{\mathsf{net}} := m \leftarrow \mathsf{In}(i);\ k \leftarrow \mathsf{SessionKey}(i);\ p_A \leftarrow \mathsf{PublicEphemeralKey}(i);\ \mathsf{ret}\ \big(p_A, m * k\big)$ for $0 \leqslant i < q$
- $\mathsf{Out}(i) := \_ \leftarrow \mathsf{OkCtxt}(i)^{\mathsf{adv}}_{\mathsf{net}};\ \mathsf{read}\ \mathsf{In}(i)$ for $0 \leqslant i < q$

The channel $\mathsf{DDH}_2(i)$ can now be substituted away:

- $\mathsf{PublicKey} := k_B \leftarrow \mathsf{unif}_{\mathsf{key}};\ \mathsf{ret}\ \mathsf{g}^{k_B}$
- $\mathsf{LeakPublicKey}^{\mathsf{rec}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{PublicKey}$
- $\mathsf{SecretEphemeralKey}(i) := \mathsf{samp}\ \mathsf{unif}_{\mathsf{key}}$ for $0 \leqslant i < q$
- $\mathsf{Key}(i) := \mathsf{samp}\ \mathsf{unif}_{\mathsf{key}}$ for $0 \leqslant i < q$
- $\mathsf{PublicEphemeralKey}(i) := k_A \leftarrow \mathsf{SecretEphemeralKey}(i);\ k_R \leftarrow \mathsf{Key}(i);\ \mathsf{ret}\ \mathsf{g}^{k_A}$ for $0 \leqslant i < q$
- $\mathsf{SessionKey}(i) := k_A \leftarrow \mathsf{SecretEphemeralKey}(i);\ k_R \leftarrow \mathsf{Key}(i);\ \mathsf{ret}\ \mathsf{g}^{k_R}$ for $0 \leqslant i < q$
- $\mathsf{LeakCtxt}(i)^{\mathsf{adv}}_{\mathsf{net}} := m \leftarrow \mathsf{In}(i);\ k \leftarrow \mathsf{SessionKey}(i);\ p_A \leftarrow \mathsf{PublicEphemeralKey}(i);\ \mathsf{ret}\ \big(p_A, m * k\big)$ for $0 \leqslant i < q$
- $\mathsf{Out}(i) := \_ \leftarrow \mathsf{OkCtxt}(i)^{\mathsf{adv}}_{\mathsf{net}};\ \mathsf{read}\ \mathsf{In}(i)$ for $0 \leqslant i < q$

After dropping the gratuitous dependencies – on channel $\mathsf{Key}(i)$ from the channel $\mathsf{PublicEphemeralKey}(i)$, and channel $\mathsf{SecretEphemeralKey}(i)$ from the channel $\mathsf{SessionKey}(i)$ – we end up with the following:

- $\mathsf{PublicKey} := k_B \leftarrow \mathsf{unif}_{\mathsf{key}};\ \mathsf{ret}\ \mathsf{g}^{k_B}$
- $\mathsf{LeakPublicKey}^{\mathsf{rec}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{PublicKey}$
- $\mathsf{SecretEphemeralKey}(i) := \mathsf{samp}\ \mathsf{unif}_{\mathsf{key}}$ for $0 \leqslant i < q$
- $\mathsf{Key}(i) := \mathsf{samp}\ \mathsf{unif}_{\mathsf{key}}$ for $0 \leqslant i < q$
- $\mathsf{PublicEphemeralKey}(i) := k_A \leftarrow \mathsf{SecretEphemeralKey}(i);\ \mathsf{ret}\ \mathsf{g}^{k_A}$ for $0 \leqslant i < q$
- $\mathsf{SessionKey}(i) := k_R \leftarrow \mathsf{Key}(i);\ \mathsf{ret}\ \mathsf{g}^{k_R}$ for $0 \leqslant i < q$
- $\mathsf{LeakCtxt}(i)^{\mathsf{adv}}_{\mathsf{net}} := m \leftarrow \mathsf{In}(i);\ k \leftarrow \mathsf{SessionKey}(i);\ p_A \leftarrow \mathsf{PublicEphemeralKey}(i);\ \mathsf{ret}\ \big(p_A, m * k\big)$ for $0 \leqslant i < q$
- $\mathsf{Out}(i) := \_ \leftarrow \mathsf{OkCtxt}(i)^{\mathsf{adv}}_{\mathsf{net}};\ \mathsf{read}\ \mathsf{In}(i)$ for $0 \leqslant i < q$

We now unfold the sampling from PublicKey into a new internal channel SecretKey:

- SecretKey := samp $\mathsf{unif}_{\mathsf{key}}$
- PublicKey := $k_B \leftarrow$ SecretKey; ret g^{k_B}
- $\mathsf{LeakPublicKey}^{\mathsf{rec}}_{\mathsf{adv}}$:= read PublicKey
- SecretEphemeralKey(i) := samp $\mathsf{unif}_{\mathsf{key}}$ for $0 \leqslant i < q$
- Key(i) := samp $\mathsf{unif}_{\mathsf{key}}$ for $0 \leqslant i < q$
- PublicEphemeralKey(i) := $k_A \leftarrow$ SecretEphemeralKey(i); ret g^{k_A} for $0 \leqslant i < q$
- SessionKey(i) := $k_R \leftarrow$ Key(i); ret g^{k_R} for $0 \leqslant i < q$
- $\mathsf{LeakCtxt}(i)^{\mathsf{adv}}_{\mathsf{net}}$:= $m \leftarrow \mathsf{In}(i)$; $k \leftarrow$ SessionKey(i); $p_A \leftarrow$ PublicEphemeralKey(i); ret $\left(p_A, m * k\right)$ for $0 \leqslant i < q$
- $\mathsf{Out}(i)$:= $\_ \leftarrow \mathsf{OkCtxt}(i)^{\mathsf{adv}}_{\mathsf{net}}$; read $\mathsf{In}(i)$ for $0 \leqslant i < q$

The channel Key(i) is now only used in SessionKey(i), so we can fold it in:

- SecretKey := samp $\mathsf{unif}_{\mathsf{key}}$
- PublicKey := $k_B \leftarrow$ SecretKey; ret g^{k_B}
- $\mathsf{LeakPublicKey}^{\mathsf{rec}}_{\mathsf{adv}}$:= read PublicKey
- SecretEphemeralKey(i) := samp $\mathsf{unif}_{\mathsf{key}}$ for $0 \leqslant i < q$
- PublicEphemeralKey(i) := $k_A \leftarrow$ SecretEphemeralKey(i); ret g^{k_A} for $0 \leqslant i < q$
- SessionKey(i) := $k_R \leftarrow \mathsf{unif}_{\mathsf{key}}$; ret g^{k_R} for $0 \leqslant i < q$
- $\mathsf{LeakCtxt}(i)^{\mathsf{adv}}_{\mathsf{net}}$:= $m \leftarrow \mathsf{In}(i)$; $k \leftarrow$ SessionKey(i); $p_A \leftarrow$ PublicEphemeralKey(i); ret $\left(p_A, m * k\right)$ for $0 \leqslant i < q$
- $\mathsf{Out}(i)$:= $\_ \leftarrow \mathsf{OkCtxt}(i)^{\mathsf{adv}}_{\mathsf{net}}$; read $\mathsf{In}(i)$ for $0 \leqslant i < q$

By assumption, sampling a random secret key k_R from $\mathsf{unif}_{\mathsf{key}}$ and returning g^{k_R} is the same as sampling from $\mathsf{unif}_{\mathsf{msg}}$ directly, so we can simplify the channel SessionKey(i) as follows:

- SecretKey := samp $\mathsf{unif}_{\mathsf{key}}$
- PublicKey := $k_B \leftarrow$ SecretKey; ret g^{k_B}
- $\mathsf{LeakPublicKey}^{\mathsf{rec}}_{\mathsf{adv}}$:= read PublicKey
- SecretEphemeralKey(i) := samp $\mathsf{unif}_{\mathsf{key}}$ for $0 \leqslant i < q$
- PublicEphemeralKey(i) := $k_A \leftarrow$ SecretEphemeralKey(i); ret g^{k_A} for $0 \leqslant i < q$
- SessionKey(i) := samp $\mathsf{unif}_{\mathsf{msg}}$ for $0 \leqslant i < q$
- $\mathsf{LeakCtxt}(i)^{\mathsf{adv}}_{\mathsf{net}}$:= $m \leftarrow \mathsf{In}(i)$; $k \leftarrow$ SessionKey(i); $p_A \leftarrow$ PublicEphemeralKey(i); ret $\left(p_A, m * k\right)$ for $0 \leqslant i < q$
- $\mathsf{Out}(i)$:= $\_ \leftarrow \mathsf{OkCtxt}(i)^{\mathsf{adv}}_{\mathsf{net}}$; read $\mathsf{In}(i)$ for $0 \leqslant i < q$

We can now extract the encoding of the input message into a new internal channel Enc(i):

- SecretKey := samp $\mathsf{unif}_{\mathsf{key}}$
- PublicKey := $k_B \leftarrow$ SecretKey; ret g^{k_B}

- $\mathsf{LeakPublicKey}^{\mathsf{rec}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{PublicKey}$
- $\mathsf{SecretEphemeralKey}(i) := \mathsf{samp}\ \mathsf{unif}_{\mathsf{key}}$ for $0 \leqslant i < q$
- $\mathsf{PublicEphemeralKey}(i) := k_A \leftarrow \mathsf{SecretEphemeralKey}(i);\ \mathsf{ret}\ \mathsf{g}^{k_A}$ for $0 \leqslant i < q$
- $\mathsf{SessionKey}(i) := \mathsf{samp}\ \mathsf{unif}_{\mathsf{msg}}$ for $0 \leqslant i < q$
- $\mathsf{Enc}(i) := m \leftarrow \mathsf{In}(i);\ k \leftarrow \mathsf{SessionKey}(i);\ \mathsf{ret}\ m * k$ for $0 \leqslant i < q$
- $\mathsf{LeakCtxt}(i)^{\mathsf{adv}}_{\mathsf{net}} := c \leftarrow \mathsf{Enc}(i);\ p_A \leftarrow \mathsf{PublicEphemeralKey}(i);\ \mathsf{ret}\ \left(p_A, c\right)$ for $0 \leqslant i < q$
- $\mathsf{Out}(i) := \_ \leftarrow \mathsf{OkCtxt}(i)^{\mathsf{adv}}_{\mathsf{net}};\ \mathsf{read}\ \mathsf{In}(i)$ for $0 \leqslant i < q$

At this point, the channel $\mathsf{SessionKey}(i)$ is only used in $\mathsf{Enc}(i)$, so we can fold it in:

- $\mathsf{SecretKey} := \mathsf{samp}\ \mathsf{unif}_{\mathsf{key}}$
- $\mathsf{PublicKey} := k_B \leftarrow \mathsf{SecretKey};\ \mathsf{ret}\ \mathsf{g}^{k_B}$
- $\mathsf{LeakPublicKey}^{\mathsf{rec}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{PublicKey}$
- $\mathsf{SecretEphemeralKey}(i) := \mathsf{samp}\ \mathsf{unif}_{\mathsf{key}}$ for $0 \leqslant i < q$
- $\mathsf{PublicEphemeralKey}(i) := k_A \leftarrow \mathsf{SecretEphemeralKey}(i);\ \mathsf{ret}\ \mathsf{g}^{k_A}$ for $0 \leqslant i < q$
- $\mathsf{Enc}(i) := m \leftarrow \mathsf{In}(i);\ k \leftarrow \mathsf{unif}_{\mathsf{key}};\ \mathsf{ret}\ m * k$ for $0 \leqslant i < q$
- $\mathsf{LeakCtxt}(i)^{\mathsf{adv}}_{\mathsf{net}} := c \leftarrow \mathsf{Enc}(i);\ p_A \leftarrow \mathsf{PublicEphemeralKey}(i);\ \mathsf{ret}\ \left(p_A, c\right)$ for $0 \leqslant i < q$
- $\mathsf{Out}(i) := \_ \leftarrow \mathsf{OkCtxt}(i)^{\mathsf{adv}}_{\mathsf{net}};\ \mathsf{read}\ \mathsf{In}(i)$ for $0 \leqslant i < q$

By assumption, the distribution $\mathsf{unif}_{\mathsf{msg}}$ on messages is invariant under the operation of multiplication with a fixed message, so we can simplify the channel $\mathsf{Enc}(i)$ as follows:

- $\mathsf{SecretKey} := \mathsf{samp}\ \mathsf{unif}_{\mathsf{key}}$
- $\mathsf{PublicKey} := k_B \leftarrow \mathsf{SecretKey};\ \mathsf{ret}\ \mathsf{g}^{k_B}$
- $\mathsf{LeakPublicKey}^{\mathsf{rec}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{PublicKey}$
- $\mathsf{SecretEphemeralKey}(i) := \mathsf{samp}\ \mathsf{unif}_{\mathsf{key}}$ for $0 \leqslant i < q$
- $\mathsf{PublicEphemeralKey}(i) := k_A \leftarrow \mathsf{SecretEphemeralKey}(i);\ \mathsf{ret}\ \mathsf{g}^{k_A}$ for $0 \leqslant i < q$
- $\mathsf{Enc}(i) := \_ \leftarrow \mathsf{In}(i);\ \mathsf{samp}\ \mathsf{unif}_{\mathsf{key}}$ for $0 \leqslant i < q$
- $\mathsf{LeakCtxt}(i)^{\mathsf{adv}}_{\mathsf{net}} := c \leftarrow \mathsf{Enc}(i);\ p_A \leftarrow \mathsf{PublicEphemeralKey}(i);\ \mathsf{ret}\ \left(p_A, c\right)$ for $0 \leqslant i < q$
- $\mathsf{Out}(i) := \_ \leftarrow \mathsf{OkCtxt}(i)^{\mathsf{adv}}_{\mathsf{net}};\ \mathsf{read}\ \mathsf{In}(i)$ for $0 \leqslant i < q$

The channel $\mathsf{Enc}(i)$ is now only used in $\mathsf{LeakCtxt}(i)^{\mathsf{adv}}_{\mathsf{net}}$, so we can fold it in:

- $\mathsf{SecretKey} := \mathsf{samp}\ \mathsf{unif}_{\mathsf{key}}$
- $\mathsf{PublicKey} := k_B \leftarrow \mathsf{SecretKey};\ \mathsf{ret}\ \mathsf{g}^{k_B}$
- $\mathsf{LeakPublicKey}^{\mathsf{rec}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{PublicKey}$
- $\mathsf{SecretEphemeralKey}(i) := \mathsf{samp}\ \mathsf{unif}_{\mathsf{key}}$ for $0 \leqslant i < q$
- $\mathsf{PublicEphemeralKey}(i) := k_A \leftarrow \mathsf{SecretEphemeralKey}(i);\ \mathsf{ret}\ \mathsf{g}^{k_A}$ for $0 \leqslant i < q$
- $\mathsf{LeakCtxt}(i)^{\mathsf{adv}}_{\mathsf{net}} := \_ \leftarrow \mathsf{In}(i);\ c \leftarrow \mathsf{unif}_{\mathsf{key}};\ p_A \leftarrow \mathsf{PublicEphemeralKey}(i);\ \mathsf{ret}\ \left(p_A, c\right)$ for $0 \leqslant i < q$
- $\mathsf{Out}(i) := \_ \leftarrow \mathsf{OkCtxt}(i)^{\mathsf{adv}}_{\mathsf{net}};\ \mathsf{read}\ \mathsf{In}(i)$ for $0 \leqslant i < q$

But this is precisely the simplified composition of the ideal functionality and the simulator from the beginning of this section.

7 Oblivious Transfer: 1-Out-Of-2 Pre-Processing

In this case study, Alice and Bob carry out a 1-Out-Of-2 Oblivious Transfer (OT) separated into an *offline* phase, where Alice and Bob exchange a key using a single idealized 1-Out-Of-2 OT instance, and an *online* phase that relies on the shared key and requires no cryptographic assumptions at all, thereby being very fast. We prove the protocol semi-honest secure in the case when the receiver is corrupt. Formally, we assume a type msg of messages; a coin-flip distribution $\mathsf{flip} : 1 \twoheadrightarrow \mathsf{Bool}$; a uniform distribution $\mathsf{unif}_{\mathsf{msg}} : 1 \twoheadrightarrow \mathsf{msg}$ on messages; a Boolean xor function $\oplus_{\mathsf{Bool}} : \mathsf{Bool} \times \mathsf{Bool} \to \mathsf{Bool}$; and a bitwise xor function $\oplus_{\mathsf{msg}} : \mathsf{msg} \times \mathsf{msg} \to \mathsf{msg}$ on messages. By an abuse of notation, we will write both $\oplus_{\mathsf{Bool}}(x, y)$ and $\oplus_{\mathsf{msg}}(x, y)$ as $x \oplus y$.

7.1 The Assumptions

At the expression level, we assume the standard definition for the Boolean xor function, as captured, *e.g.*, by the axioms

- $x : \mathsf{Bool}, y : \mathsf{Bool} \vdash x \oplus x = \mathsf{false} : \mathsf{Bool}$,
- $x : \mathsf{Bool}, y : \mathsf{Bool} \vdash x \oplus \mathsf{false} = x : \mathsf{Bool}$, and
- $x : \mathsf{Bool}, y : \mathsf{Bool} \vdash \mathsf{false} \oplus x = x : \mathsf{Bool}$.

Furthermore, we assume that the operation of bitwise xor with a fixed message is self-inverse:

- $x : \mathsf{msg}, y : \mathsf{msg} \vdash (x \oplus y) \oplus y = x : \mathsf{msg}$.

At the distribution level, we assume that the distribution $\mathsf{unif}_{\mathsf{msg}}$ on messages is invariant under the operation of xor-ing with a fixed message (a consequence of $\mathsf{unif}_{\mathsf{msg}}$ being uniform):

- $x : \mathsf{msg} \vdash \big(y \leftarrow \mathsf{unif}_{\mathsf{msg}};\ 1[x \oplus y]\big) = \mathsf{unif}_{\mathsf{msg}} : \mathsf{msg}$.

7.2 The Ideal Functionality

In its basic form, the ideal functionality reads two messages m_0, m_1 from the sender, and one Boolean c from the receiver, and outputs the following message:

$$\begin{cases} m_0 & \text{if } c = \mathsf{false} \\ m_1 & \text{if } c = \mathsf{true} \end{cases}$$

In code:

- $\mathsf{Out} := m_0 \leftarrow \mathsf{Msg}(0);\ m_1 \leftarrow \mathsf{Msg}(1);\ c \leftarrow \mathsf{Choice};\ \mathsf{if}\ c\ \mathsf{then\ ret}\ m_1\ \mathsf{else\ ret}\ m_0$

Since the sender is honest, only the timing information for their messages is leaked:

- $\mathsf{MsgRcvd}(0)^{\mathsf{id}}_{\mathsf{adv}} := m_0 \leftarrow \mathsf{Msg}(0);\ \mathsf{ret}\ \checkmark$
- $\mathsf{MsgRcvd}(1)^{\mathsf{id}}_{\mathsf{adv}} := m_1 \leftarrow \mathsf{Msg}(1);\ \mathsf{ret}\ \checkmark$

Since the receiver is semi-honest, the choice as well as selected message are leaked to the adversary:

- $\mathsf{Choice}^{\mathsf{id}}_{\mathsf{adv}} := \mathsf{read\ Choice}$
- $\mathsf{Out}^{\mathsf{id}}_{\mathsf{adv}} := \mathsf{read\ Out}$

7.3 The Real Protocol

For the offline phase, we assume an ideal OT functionality. Alice randomly generates a new pair of messages, to be treated as keys, and sends them to the OT functionality:

- $\mathsf{Key}(0) := \mathsf{samp}\ \mathsf{unif}_{\mathsf{msg}}$
- $\mathsf{Key}(1) := \mathsf{samp}\ \mathsf{unif}_{\mathsf{msg}}$
- $\mathsf{OTMsg}(0) := \mathsf{read}\ \mathsf{Key}(0)$
- $\mathsf{OTMsg}(1) := \mathsf{read}\ \mathsf{Key}(1)$

Bob flips a coin to decide which key he will ask for and informs the adversary:

- $\mathsf{Flip} := \mathsf{samp}\ \mathsf{flip}$
- $\mathsf{Flip}^{\mathsf{rec}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{Flip}$

He subsequently sends his choice to the OT functionality:

- $\mathsf{OTChoice} := \mathsf{read}\ \mathsf{Flip}$

The OT functionality selects the corresponding key and sends it to Bob, accompanied by the requisite leakages:

- $\mathsf{OTOut} := m_0 \leftarrow \mathsf{OTMsg}(0);\ m_1 \leftarrow \mathsf{OTMsg}(1);\ c \leftarrow \mathsf{OTChoice};$ $\mathsf{if}\ c\ \mathsf{then\ ret}\ m_1\ \mathsf{else\ ret}\ m_0$
- $\mathsf{OTMsgRcvd}(0)^{\mathsf{ot}}_{\mathsf{adv}} := m_0 \leftarrow \mathsf{OTMsg}(0);\ \mathsf{ret}\ \checkmark$
- $\mathsf{OTMsgRcvd}(1)^{\mathsf{ot}}_{\mathsf{adv}} := m_1 \leftarrow \mathsf{OTMsg}(1);\ \mathsf{ret}\ \checkmark$
- $\mathsf{OTChoice}^{\mathsf{ot}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{OTChoice}$
- $\mathsf{OTOut}^{\mathsf{ot}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{OTOut}$

Bob stores the result of the OT exchange as the key shared between him and Alice:

- $\mathsf{SharedKey} := \mathsf{read}\ \mathsf{OTOut}$

This ends the offline phase. The online phase starts by Bob's informing the adversary about his choice of message:

- $\mathsf{Choice}^{\mathsf{rec}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{Choice}$

Bob subsequently encrypts this choice by xor-ing it with the shared key established in the pre-processing phase, and sends the encryption to Alice while leaking its value:

- $\mathsf{ChoiceEnc} := f \leftarrow \mathsf{Flip};\ c \leftarrow \mathsf{Choice};\ \mathsf{ret}\ f \oplus c$
- $\mathsf{ChoiceEnc}^{\mathsf{rec}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{ChoiceEnc}$

Upon receiving Bob's encrypted choice, Alice encrypts her messages by bitwise xor-ing them with the keys - either their own respective keys in case Bob's encrypted choice is false, or the mutually-swapped keys if Bob's encrypted choice is true:

- $\mathsf{MsgEnc}(0) := m_0 \leftarrow \mathsf{Msg}(0);\ m_1 \leftarrow \mathsf{Msg}(1);\ k_0 \leftarrow \mathsf{Key}(0);\ k_1 \leftarrow \mathsf{Key}(1);\ e \leftarrow \mathsf{ChoiceEnc};$ $\mathsf{if}\ e\ \mathsf{then\ ret}\ m_0 \oplus k_1\ \mathsf{else\ ret}\ m_0 \oplus k_0$
- $\mathsf{MsgEnc}(1) := m_0 \leftarrow \mathsf{Msg}(0);\ m_1 \leftarrow \mathsf{Msg}(1);\ k_0 \leftarrow \mathsf{Key}(0);\ k_1 \leftarrow \mathsf{Key}(1);\ e \leftarrow \mathsf{ChoiceEnc};$ $\mathsf{if}\ e\ \mathsf{then\ ret}\ m_1 \oplus k_0\ \mathsf{else\ ret}\ m_1 \oplus k_1$

After receiving Alice's encrypted messages, Bob leaks them to the adversary:

- $\mathsf{MsgEnc}(0)^{\mathsf{rec}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{MsgEnc}(0)$

- $\mathsf{MsgEnc}(1)^{\mathsf{rec}}_{\mathsf{adv}}$:= read $\mathsf{MsgEnc}(1)$

He then selects the encryption of the message he wants, decrypts it by xor-ing it with the shared key, and outputs the result while leaking its value:

- $\mathsf{Out} := e_0 \leftarrow \mathsf{MsgEnc}(0);\ e_1 \leftarrow \mathsf{MsgEnc}(1);\ k \leftarrow \mathsf{SharedKey};\ c \leftarrow \mathsf{Choice};$ if c then ret $e_1 \oplus k$ else ret $e_0 \oplus k$
- $\mathsf{Out}^{\mathsf{rec}}_{\mathsf{adv}}$:= read Out

Thus, we have the following code for Alice:

- $\mathsf{Key}(0)$:= samp $\mathsf{unif}_{\mathsf{msg}}$
- $\mathsf{Key}(1)$:= samp $\mathsf{unif}_{\mathsf{msg}}$
- $\mathsf{OTMsg}(0)$:= read $\mathsf{Key}(0)$
- $\mathsf{OTMsg}(1)$:= read $\mathsf{Key}(1)$
- $\mathsf{MsgEnc}(0) := m_0 \leftarrow \mathsf{Msg}(0);\ m_1 \leftarrow \mathsf{Msg}(1);\ k_0 \leftarrow \mathsf{Key}(0);\ k_1 \leftarrow \mathsf{Key}(1);\ e \leftarrow \mathsf{ChoiceEnc};$ if e then ret $m_0 \oplus k_1$ else ret $m_0 \oplus k_0$
- $\mathsf{MsgEnc}(1) := m_0 \leftarrow \mathsf{Msg}(0);\ m_1 \leftarrow \mathsf{Msg}(1);\ k_0 \leftarrow \mathsf{Key}(0);\ k_1 \leftarrow \mathsf{Key}(1);\ e \leftarrow \mathsf{ChoiceEnc};$ if e then ret $m_1 \oplus k_0$ else ret $m_1 \oplus k_1$

The code for Bob has the following form:

- Flip := samp flip
- $\mathsf{Flip}^{\mathsf{rec}}_{\mathsf{adv}}$:= read Flip
- OTChoice := read Flip
- SharedKey := read OTOut
- $\mathsf{Choice}^{\mathsf{rec}}_{\mathsf{adv}}$:= read Choice
- $\mathsf{ChoiceEnc} := f \leftarrow \mathsf{Flip};\ c \leftarrow \mathsf{Choice};$ ret $f \oplus c$
- $\mathsf{ChoiceEnc}^{\mathsf{rec}}_{\mathsf{adv}}$:= read ChoiceEnc
- $\mathsf{MsgEnc}(0)^{\mathsf{rec}}_{\mathsf{adv}}$:= read $\mathsf{MsgEnc}(0)$
- $\mathsf{MsgEnc}(1)^{\mathsf{rec}}_{\mathsf{adv}}$:= read $\mathsf{MsgEnc}(1)$
- $\mathsf{Out} := e_0 \leftarrow \mathsf{MsgEnc}(0);\ e_1 \leftarrow \mathsf{MsgEnc}(1);\ k \leftarrow \mathsf{SharedKey};\ c \leftarrow \mathsf{Choice};$ if c then ret $e_1 \oplus k$ else ret $e_0 \oplus k$
- $\mathsf{Out}^{\mathsf{rec}}_{\mathsf{adv}}$:= read Out

Finally, we recall the code for the OT functionality:

- $\mathsf{OTOut} := m_0 \leftarrow \mathsf{OTMsg}(0);\ m_1 \leftarrow \mathsf{OTMsg}(1);\ c \leftarrow \mathsf{OTChoice};$ if c then ret m_1 else ret m_0
- $\mathsf{OTMsgRcvd}(0)^{\mathsf{ot}}_{\mathsf{adv}} := m_0 \leftarrow \mathsf{OTMsg}(0);$ ret ✓
- $\mathsf{OTMsgRcvd}(1)^{\mathsf{ot}}_{\mathsf{adv}} := m_1 \leftarrow \mathsf{OTMsg}(1);$ ret ✓
- $\mathsf{OTChoice}^{\mathsf{ot}}_{\mathsf{adv}}$:= read OTChoice
- $\mathsf{OTOut}^{\mathsf{ot}}_{\mathsf{adv}}$:= read OTOut

Composing all of this together and hiding the internal communication yields the real-world protocol.

7.4 Real = Ideal + Simulator

Our goal is to simplify the real protocol until it becomes clear how to separate it out into the ideal functionality part and the simulator part. We recall the code:

- $\mathsf{Key}(0) := \mathsf{samp}\ \mathsf{unif}_{\mathsf{msg}}$
- $\mathsf{Key}(1) := \mathsf{samp}\ \mathsf{unif}_{\mathsf{msg}}$
- $\mathsf{Flip} := \mathsf{samp}\ \mathsf{flip}$
- $\mathsf{Flip}^{\mathsf{rec}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{Flip}$
- $\mathsf{OTMsg}(0) := \mathsf{read}\ \mathsf{Key}(0)$
- $\mathsf{OTMsg}(1) := \mathsf{read}\ \mathsf{Key}(1)$
- $\mathsf{OTChoice} := \mathsf{read}\ \mathsf{Flip}$
- $\mathsf{OTOut} := m_0 \leftarrow \mathsf{OTMsg}(0);\ m_1 \leftarrow \mathsf{OTMsg}(1);\ c \leftarrow \mathsf{OTChoice};$
  $\mathsf{if}\ c\ \mathsf{then\ ret}\ m_1\ \mathsf{else\ ret}\ m_0$
- $\mathsf{OTMsgRcvd}(0)^{\mathsf{ot}}_{\mathsf{adv}} := m_0 \leftarrow \mathsf{OTMsg}(0);\ \mathsf{ret}\ \checkmark$
- $\mathsf{OTMsgRcvd}(1)^{\mathsf{ot}}_{\mathsf{adv}} := m_1 \leftarrow \mathsf{OTMsg}(1);\ \mathsf{ret}\ \checkmark$
- $\mathsf{OTChoice}^{\mathsf{ot}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{OTChoice}$
- $\mathsf{OTOut}^{\mathsf{ot}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{OTOut}$
- $\mathsf{SharedKey} := \mathsf{read}\ \mathsf{OTOut}$
- $\mathsf{Choice}^{\mathsf{rec}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{Choice}$
- $\mathsf{ChoiceEnc} := f \leftarrow \mathsf{Flip};\ c \leftarrow \mathsf{Choice};\ \mathsf{ret}\ f \oplus c$
- $\mathsf{ChoiceEnc}^{\mathsf{rec}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{ChoiceEnc}$
- $\mathsf{MsgEnc}(0) := m_0 \leftarrow \mathsf{Msg}(0);\ m_1 \leftarrow \mathsf{Msg}(1);\ k_0 \leftarrow \mathsf{Key}(0);\ k_1 \leftarrow \mathsf{Key}(1);\ e \leftarrow \mathsf{ChoiceEnc};$
  $\mathsf{if}\ e\ \mathsf{then\ ret}\ m_0 \oplus k_1\ \mathsf{else\ ret}\ m_0 \oplus k_0$
- $\mathsf{MsgEnc}(1) := m_0 \leftarrow \mathsf{Msg}(0);\ m_1 \leftarrow \mathsf{Msg}(1);\ k_0 \leftarrow \mathsf{Key}(0);\ k_1 \leftarrow \mathsf{Key}(1);\ e \leftarrow \mathsf{ChoiceEnc};$
  $\mathsf{if}\ e\ \mathsf{then\ ret}\ m_1 \oplus k_0\ \mathsf{else\ ret}\ m_1 \oplus k_1$
- $\mathsf{MsgEnc}(0)^{\mathsf{rec}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{MsgEnc}(0)$
- $\mathsf{MsgEnc}(1)^{\mathsf{rec}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{MsgEnc}(1)$
- $\mathsf{Out} := e_0 \leftarrow \mathsf{MsgEnc}(0);\ e_1 \leftarrow \mathsf{MsgEnc}(1);\ k \leftarrow \mathsf{SharedKey};\ c \leftarrow \mathsf{Choice};$
  $\mathsf{if}\ c\ \mathsf{then\ ret}\ e_1 \oplus k\ \mathsf{else\ ret}\ e_0 \oplus k$
- $\mathsf{Out}^{\mathsf{rec}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{Out}$

We start by eliminating all of the internal OT channels:

- $\mathsf{Key}(0) := \mathsf{samp}\ \mathsf{unif}_{\mathsf{msg}}$
- $\mathsf{Key}(1) := \mathsf{samp}\ \mathsf{unif}_{\mathsf{msg}}$
- $\mathsf{Flip} := \mathsf{samp}\ \mathsf{flip}$
- $\mathsf{Flip}^{\mathsf{rec}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{Flip}$
- $\mathsf{OTMsgRcvd}(0)^{\mathsf{ot}}_{\mathsf{adv}} := k_0 \leftarrow \mathsf{Key}(0);\ \mathsf{ret}\ \checkmark$

- $\mathsf{OTMsgRcvd}(1)^{\mathsf{ot}}_{\mathsf{adv}} := k_1 \leftarrow \mathsf{Key}(1);\ \mathsf{ret}\ \checkmark$
- $\mathsf{OTChoice}^{\mathsf{ot}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{Flip}$
- $\mathsf{OTOut}^{\mathsf{ot}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{SharedKey}$
- $\mathsf{SharedKey} := k_0 \leftarrow \mathsf{Key}(0);\ k_1 \leftarrow \mathsf{Key}(1);\ f \leftarrow \mathsf{Flip};$
  $\mathsf{if}\ f\ \mathsf{then}\ \mathsf{ret}\ k_1\ \mathsf{else}\ \mathsf{ret}\ k_0$
- $\mathsf{Choice}^{\mathsf{rec}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{Choice}$
- $\mathsf{ChoiceEnc} := f \leftarrow \mathsf{Flip};\ c \leftarrow \mathsf{Choice};\ \mathsf{ret}\ f \oplus c$
- $\mathsf{ChoiceEnc}^{\mathsf{rec}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{ChoiceEnc}$
- $\mathsf{MsgEnc}(0) := m_0 \leftarrow \mathsf{Msg}(0);\ m_1 \leftarrow \mathsf{Msg}(1);\ k_0 \leftarrow \mathsf{Key}(0);\ k_1 \leftarrow \mathsf{Key}(1);\ e \leftarrow \mathsf{ChoiceEnc};$
  $\mathsf{if}\ e\ \mathsf{then}\ \mathsf{ret}\ m_0 \oplus k_1\ \mathsf{else}\ \mathsf{ret}\ m_0 \oplus k_0$
- $\mathsf{MsgEnc}(1) := m_0 \leftarrow \mathsf{Msg}(0);\ m_1 \leftarrow \mathsf{Msg}(1);\ k_0 \leftarrow \mathsf{Key}(0);\ k_1 \leftarrow \mathsf{Key}(1);\ e \leftarrow \mathsf{ChoiceEnc};$
  $\mathsf{if}\ e\ \mathsf{then}\ \mathsf{ret}\ m_1 \oplus k_0\ \mathsf{else}\ \mathsf{ret}\ m_1 \oplus k_1$
- $\mathsf{MsgEnc}(0)^{\mathsf{rec}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{MsgEnc}(0)$
- $\mathsf{MsgEnc}(1)^{\mathsf{rec}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{MsgEnc}(1)$
- $\mathsf{Out} := e_0 \leftarrow \mathsf{MsgEnc}(0);\ e_1 \leftarrow \mathsf{MsgEnc}(1);\ k \leftarrow \mathsf{SharedKey};\ c \leftarrow \mathsf{Choice};$
  $\mathsf{if}\ c\ \mathsf{then}\ \mathsf{ret}\ e_1 \oplus k\ \mathsf{else}\ \mathsf{ret}\ e_0 \oplus k$
- $\mathsf{Out}^{\mathsf{rec}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{Out}$

Substituting the channel $\mathsf{ChoiceEnc}$ into $\mathsf{MsgEnc}(0)$ and $\mathsf{MsgEnc}(1)$ yields

- $\mathsf{MsgEnc}(0) := m_0 \leftarrow \mathsf{Msg}(0);\ m_1 \leftarrow \mathsf{Msg}(1);\ k_0 \leftarrow \mathsf{Key}(0);\ k_1 \leftarrow \mathsf{Key}(1);\ f \leftarrow \mathsf{Flip};\ c \leftarrow \mathsf{Choice};$
  $\mathsf{if}\ f \oplus c\ \mathsf{then}\ \mathsf{ret}\ m_0 \oplus k_1\ \mathsf{else}\ \mathsf{ret}\ m_0 \oplus k_0$
- $\mathsf{MsgEnc}(1) := m_0 \leftarrow \mathsf{Msg}(0);\ m_1 \leftarrow \mathsf{Msg}(1);\ k_0 \leftarrow \mathsf{Key}(0);\ k_1 \leftarrow \mathsf{Key}(1);\ f \leftarrow \mathsf{Flip};\ c \leftarrow \mathsf{Choice};$
  $\mathsf{if}\ f \oplus c\ \mathsf{then}\ \mathsf{ret}\ m_0 \oplus k_1\ \mathsf{else}\ \mathsf{ret}\ m_0 \oplus k_0$

Introducing a branching on each of the Boolean channels Flip and Choice, and simplifying $f \oplus c$ according to the definition of $\oplus$ yields the following:

- $\mathsf{MsgEnc}(0) := m_0 \leftarrow \mathsf{Msg}(0);\ m_1 \leftarrow \mathsf{Msg}(1);\ k_0 \leftarrow \mathsf{Key}(0);\ k_1 \leftarrow \mathsf{Key}(1);\ f \leftarrow \mathsf{Flip};\ c \leftarrow \mathsf{Choice};$
  $\mathsf{if}\ f\ \mathsf{then}\ \big(\mathsf{if}\ c\ \mathsf{then}\ \mathsf{ret}\ m_0 \oplus k_0\ \mathsf{else}\ \mathsf{ret}\ m_0 \oplus k_1\big)\ \mathsf{else}\ \big(\mathsf{if}\ c\ \mathsf{then}\ \mathsf{ret}\ m_0 \oplus k_1\ \mathsf{else}\ \mathsf{ret}\ m_0 \oplus k_0\big)$
- $\mathsf{MsgEnc}(1) := m_0 \leftarrow \mathsf{Msg}(0);\ m_1 \leftarrow \mathsf{Msg}(1);\ k_0 \leftarrow \mathsf{Key}(0);\ k_1 \leftarrow \mathsf{Key}(1);\ f \leftarrow \mathsf{Flip};\ c \leftarrow \mathsf{Choice};$
  $\mathsf{if}\ f\ \mathsf{then}\ \big(\mathsf{if}\ c\ \mathsf{then}\ \mathsf{ret}\ m_1 \oplus k_1\ \mathsf{else}\ \mathsf{ret}\ m_1 \oplus k_0\big)\ \mathsf{else}\ \big(\mathsf{if}\ c\ \mathsf{then}\ \mathsf{ret}\ m_1 \oplus k_0\ \mathsf{else}\ \mathsf{ret}\ m_1 \oplus k_1\big)$

Substituting the channel $\mathsf{SharedKey}$ into Out yields

- $\mathsf{Out} := e_0 \leftarrow \mathsf{MsgEnc}(0);\ e_1 \leftarrow \mathsf{MsgEnc}(1);\ k_0 \leftarrow \mathsf{Key}(0);\ k_1 \leftarrow \mathsf{Key}(1);\ f \leftarrow \mathsf{Flip};\ c \leftarrow \mathsf{Choice};$
  $\mathsf{if}\ f\ \mathsf{then}\ \big(\mathsf{if}\ c\ \mathsf{then}\ \mathsf{ret}\ e_1 \oplus k_1\ \mathsf{else}\ \mathsf{ret}\ e_0 \oplus k_1\big)\ \mathsf{else}\ \big(\mathsf{if}\ c\ \mathsf{then}\ \mathsf{ret}\ e_1 \oplus k_0\ \mathsf{else}\ \mathsf{ret}\ e_0 \oplus k_0\big)$

Further substituting the channels $\mathsf{MsgEnc}(0)$ and $\mathsf{MsgEnc}(1)$ into Out yields

- $\mathsf{Out} := m_0 \leftarrow \mathsf{Msg}(0);\ m_1 \leftarrow \mathsf{Msg}(1);\ k_0 \leftarrow \mathsf{Key}(0);\ k_1 \leftarrow \mathsf{Key}(1);\ f \leftarrow \mathsf{Flip};\ c \leftarrow \mathsf{Choice};$
  $\mathsf{if}\ f\ \mathsf{then}\ \mathsf{if}\ c\ \mathsf{then}\ \mathsf{ret}\ (m_1 \oplus k_1) \oplus k_1\ \mathsf{else}\ \mathsf{ret}\ (m_0 \oplus k_1) \oplus k_1$
  $\quad\mathsf{else}\ \mathsf{if}\ c\ \mathsf{then}\ \mathsf{ret}\ (m_1 \oplus k_0) \oplus k_0\ \mathsf{else}\ \mathsf{ret}\ (m_0 \oplus k_0) \oplus k_0$

We can now cancel out the two applications of $\oplus$ since they are mutually inverse by assumption:

- $\mathsf{Out} := m_0 \leftarrow \mathsf{Msg}(0);\ m_1 \leftarrow \mathsf{Msg}(1);\ k_0 \leftarrow \mathsf{Key}(0);\ k_1 \leftarrow \mathsf{Key}(1);\ f \leftarrow \mathsf{Flip};\ c \leftarrow \mathsf{Choice};$
  $\mathsf{if}\ f\ \mathsf{then}\ \big(\mathsf{if}\ c\ \mathsf{then}\ \mathsf{ret}\ m_1\ \mathsf{else}\ \mathsf{ret}\ m_0\big)\ \mathsf{else}\ \big(\mathsf{if}\ c\ \mathsf{then}\ \mathsf{ret}\ m_1\ \mathsf{else}\ \mathsf{ret}\ m_0\big)$

We can now drop the gratuitous dependencies on channels $\mathsf{Key}(0)$, $\mathsf{Key}(1)$:

- $\mathsf{Out} := m_0 \leftarrow \mathsf{Msg}(0);\ m_1 \leftarrow \mathsf{Msg}(1);\ f \leftarrow \mathsf{Flip};\ c \leftarrow \mathsf{Choice};$
  $\mathsf{if}\ f\ \mathsf{then}\ \big(\mathsf{if}\ c\ \mathsf{then}\ \mathsf{ret}\ m_1\ \mathsf{else}\ \mathsf{ret}\ m_0\big)\ \mathsf{else}\ \big(\mathsf{if}\ c\ \mathsf{then}\ \mathsf{ret}\ m_1\ \mathsf{else}\ \mathsf{ret}\ m_0\big)$

Since both branches are the same, we can also not flip:

- $\mathsf{Out} := m_0 \leftarrow \mathsf{Msg}(0);\ m_1 \leftarrow \mathsf{Msg}(1);\ f \leftarrow \mathsf{Flip};\ c \leftarrow \mathsf{Choice};$
  $\mathsf{if}\ c\ \mathsf{then}\ \mathsf{ret}\ m_1\ \mathsf{else}\ \mathsf{ret}\ m_0$

After dropping the gratuitous dependency on the channel Flip, we get

- $\mathsf{Out} := m_0 \leftarrow \mathsf{Msg}(0);\ m_1 \leftarrow \mathsf{Msg}(1);\ c \leftarrow \mathsf{Choice};$
  $\mathsf{if}\ c\ \mathsf{then}\ \mathsf{ret}\ m_1\ \mathsf{else}\ \mathsf{ret}\ m_0$

Summarizing, the cleaned-up version of the real protocol is below:

- $\mathsf{Key}(0) := \mathsf{samp}\ \mathsf{unif}_{\mathsf{msg}}$
- $\mathsf{Key}(1) := \mathsf{samp}\ \mathsf{unif}_{\mathsf{msg}}$
- $\mathsf{Flip} := \mathsf{samp}\ \mathsf{flip}$
- $\mathsf{Flip}^{\mathsf{rec}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{Flip}$
- $\mathsf{OTMsgRcvd}(0)^{\mathsf{ot}}_{\mathsf{adv}} := k_0 \leftarrow \mathsf{Key}(0);\ \mathsf{ret}\ \checkmark$
- $\mathsf{OTMsgRcvd}(1)^{\mathsf{ot}}_{\mathsf{adv}} := k_1 \leftarrow \mathsf{Key}(1);\ \mathsf{ret}\ \checkmark$
- $\mathsf{OTChoice}^{\mathsf{ot}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{Flip}$
- $\mathsf{OTOut}^{\mathsf{ot}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{SharedKey}$
- $\mathsf{SharedKey} := k_0 \leftarrow \mathsf{Key}(0);\ k_1 \leftarrow \mathsf{Key}(1);\ f \leftarrow \mathsf{Flip};$
  $\mathsf{if}\ f\ \mathsf{then}\ \mathsf{ret}\ k_1\ \mathsf{else}\ \mathsf{ret}\ k_0$
- $\mathsf{Choice}^{\mathsf{rec}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{Choice}$
- $\mathsf{ChoiceEnc} := f \leftarrow \mathsf{Flip};\ c \leftarrow \mathsf{Choice};\ \mathsf{ret}\ f \oplus c$
- $\mathsf{ChoiceEnc}^{\mathsf{rec}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{ChoiceEnc}$
- $\mathsf{MsgEnc}(0) := m_0 \leftarrow \mathsf{Msg}(0);\ m_1 \leftarrow \mathsf{Msg}(1);\ k_0 \leftarrow \mathsf{Key}(0);\ k_1 \leftarrow \mathsf{Key}(1);\ f \leftarrow \mathsf{Flip};\ c \leftarrow \mathsf{Choice};$
  $\mathsf{if}\ f\ \mathsf{then}\ \big(\mathsf{if}\ c\ \mathsf{then}\ \mathsf{ret}\ m_0 \oplus k_0\ \mathsf{else}\ \mathsf{ret}\ m_0 \oplus k_1\big)\ \mathsf{else}\ \big(\mathsf{if}\ c\ \mathsf{then}\ \mathsf{ret}\ m_0 \oplus k_1\ \mathsf{else}\ \mathsf{ret}\ m_0 \oplus k_0\big)$
- $\mathsf{MsgEnc}(1) := m_0 \leftarrow \mathsf{Msg}(0);\ m_1 \leftarrow \mathsf{Msg}(1);\ k_0 \leftarrow \mathsf{Key}(0);\ k_1 \leftarrow \mathsf{Key}(1);\ f \leftarrow \mathsf{Flip};\ c \leftarrow \mathsf{Choice};$
  $\mathsf{if}\ f\ \mathsf{then}\ \big(\mathsf{if}\ c\ \mathsf{then}\ \mathsf{ret}\ m_1 \oplus k_1\ \mathsf{else}\ \mathsf{ret}\ m_1 \oplus k_0\big)\ \mathsf{else}\ \big(\mathsf{if}\ c\ \mathsf{then}\ \mathsf{ret}\ m_1 \oplus k_0\ \mathsf{else}\ \mathsf{ret}\ m_1 \oplus k_1\big)$
- $\mathsf{MsgEnc}(0)^{\mathsf{rec}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{MsgEnc}(0)$
- $\mathsf{MsgEnc}(1)^{\mathsf{rec}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{MsgEnc}(1)$
- $\mathsf{Out} := m_0 \leftarrow \mathsf{Msg}(0);\ m_1 \leftarrow \mathsf{Msg}(1);\ c \leftarrow \mathsf{Choice};$
  $\mathsf{if}\ c\ \mathsf{then}\ \mathsf{ret}\ m_1\ \mathsf{else}\ \mathsf{ret}\ m_0$
- $\mathsf{Out}^{\mathsf{rec}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{Out}$

Since both keys are generated from the same distribution, the coin flip that distinguishes between them can be eliminated (*"decoupling"*). To show this, we introduce an internal channel KeyPair that constructs the pair of two keys, where the first one is shared and the second one is private:

- $\mathsf{Key}(0) := \mathsf{samp}\ \mathsf{unif}_{\mathsf{msg}}$

- $\mathsf{Key}(1) := \mathsf{samp}\ \mathsf{unif}_{\mathsf{msg}}$
- $\mathsf{Flip} := \mathsf{samp}\ \mathsf{flip}$
- $\mathsf{Flip}^{\mathsf{rec}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{Flip}$
- $\mathsf{OTMsgRcvd}(0)^{\mathsf{ot}}_{\mathsf{adv}} := k_0 \leftarrow \mathsf{Key}(0);\ \mathsf{ret}\ \checkmark$
- $\mathsf{OTMsgRcvd}(1)^{\mathsf{ot}}_{\mathsf{adv}} := k_1 \leftarrow \mathsf{Key}(1);\ \mathsf{ret}\ \checkmark$
- $\mathsf{OTChoice}^{\mathsf{ot}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{Flip}$
- $\mathsf{OTOut}^{\mathsf{ot}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{SharedKey}$
- $\mathsf{KeyPair} := k_0 \leftarrow \mathsf{Key}(0);\ k_1 \leftarrow \mathsf{Key}(1);\ f \leftarrow \mathsf{Flip};$ if f then ret (k_1, k_0) else ret (k_0, k_1)
- $\mathsf{SharedKey} := \big(k_s, k_p\big) \leftarrow \mathsf{KeyPair};\ \mathsf{ret}\ k_s$
- $\mathsf{Choice}^{\mathsf{rec}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{Choice}$
- $\mathsf{ChoiceEnc} := f \leftarrow \mathsf{Flip};\ c \leftarrow \mathsf{Choice};\ \mathsf{ret}\ f \oplus c$
- $\mathsf{ChoiceEnc}^{\mathsf{rec}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{ChoiceEnc}$
- $\mathsf{MsgEnc}(0) := m_0 \leftarrow \mathsf{Msg}(0);\ m_1 \leftarrow \mathsf{Msg}(1);\ \big(k_s, k_p\big) \leftarrow \mathsf{KeyPair};\ c \leftarrow \mathsf{Choice};$ if c then ret $m_0 \oplus k_p$ else ret $m_0 \oplus k_s$
- $\mathsf{MsgEnc}(1) := m_0 \leftarrow \mathsf{Msg}(0);\ m_1 \leftarrow \mathsf{Msg}(1);\ \big(k_s, k_p\big) \leftarrow \mathsf{KeyPair};\ c \leftarrow \mathsf{Choice};$ if c then ret $m_1 \oplus k_s$ else ret $m_1 \oplus k_p$
- $\mathsf{MsgEnc}(0)^{\mathsf{rec}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{MsgEnc}(0)$
- $\mathsf{MsgEnc}(1)^{\mathsf{rec}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{MsgEnc}(1)$
- $\mathsf{Out} := m_0 \leftarrow \mathsf{Msg}(0);\ m_1 \leftarrow \mathsf{Msg}(1);\ c \leftarrow \mathsf{Choice};$ if c then ret m_1 else ret m_0
- $\mathsf{Out}^{\mathsf{rec}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{Out}$

The internal channels $\mathsf{Key}(0)$ and $\mathsf{Key}(1)$ are now only used in the channel KeyPair. We can therefore fold them in:

- $\mathsf{Flip} := \mathsf{samp}\ \mathsf{flip}$
- $\mathsf{Flip}^{\mathsf{rec}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{Flip}$
- $\mathsf{OTMsgRcvd}(0)^{\mathsf{ot}}_{\mathsf{adv}} := k_0 \leftarrow \mathsf{Key}(0);\ \mathsf{ret}\ \checkmark$
- $\mathsf{OTMsgRcvd}(1)^{\mathsf{ot}}_{\mathsf{adv}} := k_1 \leftarrow \mathsf{Key}(1);\ \mathsf{ret}\ \checkmark$
- $\mathsf{OTChoice}^{\mathsf{ot}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{Flip}$
- $\mathsf{OTOut}^{\mathsf{ot}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{SharedKey}$
- $\mathsf{KeyPair} := k_0 \leftarrow \mathsf{unif}_{\mathsf{msg}};\ k_1 \leftarrow \mathsf{unif}_{\mathsf{msg}};\ f \leftarrow \mathsf{Flip};$ if f then ret (k_1, k_0) else ret (k_0, k_1)
- $\mathsf{SharedKey} := \big(k_s, k_p\big) \leftarrow \mathsf{KeyPair};\ \mathsf{ret}\ k_s$
- $\mathsf{Choice}^{\mathsf{rec}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{Choice}$
- $\mathsf{ChoiceEnc} := f \leftarrow \mathsf{Flip};\ c \leftarrow \mathsf{Choice};\ \mathsf{ret}\ f \oplus c$
- $\mathsf{ChoiceEnc}^{\mathsf{rec}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{ChoiceEnc}$

- $\mathsf{MsgEnc}(0) := m_0 \leftarrow \mathsf{Msg}(0);\ m_1 \leftarrow \mathsf{Msg}(1);\ (k_s, k_p) \leftarrow \mathsf{KeyPair};\ c \leftarrow \mathsf{Choice};$ if c then ret $m_0 \oplus k_p$ else ret $m_0 \oplus k_s$
- $\mathsf{MsgEnc}(1) := m_0 \leftarrow \mathsf{Msg}(0);\ m_1 \leftarrow \mathsf{Msg}(1);\ (k_s, k_p) \leftarrow \mathsf{KeyPair};\ c \leftarrow \mathsf{Choice};$ if c then ret $m_1 \oplus k_s$ else ret $m_1 \oplus k_p$
- $\mathsf{MsgEnc}(0)^{\mathsf{rec}}_{\mathsf{adv}} := \text{read } \mathsf{MsgEnc}(0)$
- $\mathsf{MsgEnc}(1)^{\mathsf{rec}}_{\mathsf{adv}} := \text{read } \mathsf{MsgEnc}(1)$
- $\mathsf{Out} := m_0 \leftarrow \mathsf{Msg}(0);\ m_1 \leftarrow \mathsf{Msg}(1);\ c \leftarrow \mathsf{Choice};$ if c then ret m_1 else ret m_0
- $\mathsf{Out}^{\mathsf{rec}}_{\mathsf{adv}} := \text{read } \mathsf{Out}$

Rearranging the order of bindings inside KeyPair yields

- $\mathsf{KeyPair} := f \leftarrow \mathsf{Flip};\ k_0 \leftarrow \mathsf{unif}_{\mathsf{msg}};\ k_1 \leftarrow \mathsf{unif}_{\mathsf{msg}};$ if f then ret (k_1, k_0) else ret (k_0, k_1)

Since sampling and branching are interchangeable, the three reaction snippets

- $k_0 \leftarrow \mathsf{unif}_{\mathsf{msg}};\ k_1 \leftarrow \mathsf{unif}_{\mathsf{msg}};$ if f then ret (k_1, k_0) else ret (k_0, k_1)
- if f then $\big(k_0 \leftarrow \mathsf{unif}_{\mathsf{msg}};\ k_1 \leftarrow \mathsf{unif}_{\mathsf{msg}};$ ret $(k_1, k_0)\big)$ else $\big(k_0 \leftarrow \mathsf{unif}_{\mathsf{msg}};\ k_1 \leftarrow \mathsf{unif}_{\mathsf{msg}};$ ret $(k_0, k_1)\big)$
- if f then $\big(k_1 \leftarrow \mathsf{unif}_{\mathsf{msg}};\ k_0 \leftarrow \mathsf{unif}_{\mathsf{msg}};$ ret $(k_1, k_0)\big)$ else $\big(k_0 \leftarrow \mathsf{unif}_{\mathsf{msg}};\ k_1 \leftarrow \mathsf{unif}_{\mathsf{msg}};$ ret $(k_0, k_1)\big)$

are equivalent. But the last snippet amounts to doing the same thing either way, so we might just as well not flip:

- $\mathsf{KeyPair} := f \leftarrow \mathsf{Flip};\ k_0 \leftarrow \mathsf{unif}_{\mathsf{msg}};\ k_1 \leftarrow \mathsf{unif}_{\mathsf{msg}};$ ret (k_0, k_1)

Getting rid of the gratuitous dependency on the channel Flip gives us

- $\mathsf{KeyPair} := k_0 \leftarrow \mathsf{unif}_{\mathsf{msg}};\ k_1 \leftarrow \mathsf{unif}_{\mathsf{msg}};$ ret (k_0, k_1)

Unfolding the samplings back thus yields the following protocol:

- $\mathsf{Key}(0) := \text{samp } \mathsf{unif}_{\mathsf{msg}}$
- $\mathsf{Key}(1) := \text{samp } \mathsf{unif}_{\mathsf{msg}}$
- $\mathsf{Flip} := \text{samp flip}$
- $\mathsf{Flip}^{\mathsf{rec}}_{\mathsf{adv}} := \text{read Flip}$
- $\mathsf{OTMsgRcvd}(0)^{\mathsf{ot}}_{\mathsf{adv}} := k_0 \leftarrow \mathsf{Key}(0);$ ret ✓
- $\mathsf{OTMsgRcvd}(1)^{\mathsf{ot}}_{\mathsf{adv}} := k_1 \leftarrow \mathsf{Key}(1);$ ret ✓
- $\mathsf{OTChoice}^{\mathsf{ot}}_{\mathsf{adv}} := \text{read Flip}$
- $\mathsf{OTOut}^{\mathsf{ot}}_{\mathsf{adv}} := \text{read SharedKey}$
- $\mathsf{KeyPair} := k_0 \leftarrow \mathsf{Key}(0);\ k_1 \leftarrow \mathsf{Key}(1);$ ret (k_0, k_1)
- $\mathsf{SharedKey} := (k_s, k_p) \leftarrow \mathsf{KeyPair};$ ret k_s
- $\mathsf{Choice}^{\mathsf{rec}}_{\mathsf{adv}} := \text{read Choice}$
- $\mathsf{ChoiceEnc} := f \leftarrow \mathsf{Flip};\ c \leftarrow \mathsf{Choice};$ ret $f \oplus c$
- $\mathsf{ChoiceEnc}^{\mathsf{rec}}_{\mathsf{adv}} := \text{read ChoiceEnc}$

- $\mathsf{MsgEnc}(0) := m_0 \leftarrow \mathsf{Msg}(0);\ m_1 \leftarrow \mathsf{Msg}(1);\ (k_s, k_p) \leftarrow \mathsf{KeyPair};\ c \leftarrow \mathsf{Choice};$
  if c then ret $m_0 \oplus k_p$ else ret $m_0 \oplus k_s$
- $\mathsf{MsgEnc}(1) := m_0 \leftarrow \mathsf{Msg}(0);\ m_1 \leftarrow \mathsf{Msg}(1);\ (k_s, k_p) \leftarrow \mathsf{KeyPair};\ c \leftarrow \mathsf{Choice};$
  if c then ret $m_1 \oplus k_s$ else ret $m_1 \oplus k_p$
- $\mathsf{MsgEnc}(0)^{\mathsf{rec}}_{\mathsf{adv}} := \text{read } \mathsf{MsgEnc}(0)$
- $\mathsf{MsgEnc}(1)^{\mathsf{rec}}_{\mathsf{adv}} := \text{read } \mathsf{MsgEnc}(1)$
- $\mathsf{Out} := m_0 \leftarrow \mathsf{Msg}(0);\ m_1 \leftarrow \mathsf{Msg}(1);\ c \leftarrow \mathsf{Choice};$
  if c then ret m_1 else ret m_0
- $\mathsf{Out}^{\mathsf{rec}}_{\mathsf{adv}} := \text{read Out}$

The internal channel KeyPair can now be substituted away:

- $\mathsf{Key}(0) := \text{samp unif}_{\mathsf{msg}}$
- $\mathsf{Key}(1) := \text{samp unif}_{\mathsf{msg}}$
- $\mathsf{Flip} := \text{samp flip}$
- $\mathsf{Flip}^{\mathsf{rec}}_{\mathsf{adv}} := \text{read Flip}$
- $\mathsf{OTMsgRcvd}(0)^{\mathsf{ot}}_{\mathsf{adv}} := k_0 \leftarrow \mathsf{Key}(0);\ \text{ret } \checkmark$
- $\mathsf{OTMsgRcvd}(1)^{\mathsf{ot}}_{\mathsf{adv}} := k_1 \leftarrow \mathsf{Key}(1);\ \text{ret } \checkmark$
- $\mathsf{OTChoice}^{\mathsf{ot}}_{\mathsf{adv}} := \text{read Flip}$
- $\mathsf{OTOut}^{\mathsf{ot}}_{\mathsf{adv}} := \text{read SharedKey}$
- $\mathsf{SharedKey} := \text{read } \mathsf{Key}(0)$
- $\mathsf{Choice}^{\mathsf{rec}}_{\mathsf{adv}} := \text{read Choice}$
- $\mathsf{ChoiceEnc} := f \leftarrow \mathsf{Flip};\ c \leftarrow \mathsf{Choice};\ \text{ret } f \oplus c$
- $\mathsf{ChoiceEnc}^{\mathsf{rec}}_{\mathsf{adv}} := \text{read ChoiceEnc}$
- $\mathsf{MsgEnc}(0) := m_0 \leftarrow \mathsf{Msg}(0);\ m_1 \leftarrow \mathsf{Msg}(1);\ k_0 \leftarrow \mathsf{Key}(0);\ k_1 \leftarrow \mathsf{Key}(1);\ c \leftarrow \mathsf{Choice};$
  if c then ret $m_0 \oplus k_1$ else ret $m_0 \oplus k_0$
- $\mathsf{MsgEnc}(1) := m_0 \leftarrow \mathsf{Msg}(0);\ m_1 \leftarrow \mathsf{Msg}(1);\ k_0 \leftarrow \mathsf{Key}(0);\ k_1 \leftarrow \mathsf{Key}(1);\ c \leftarrow \mathsf{Choice};$
  if c then ret $m_1 \oplus k_0$ else ret $m_1 \oplus k_1$
- $\mathsf{MsgEnc}(0)^{\mathsf{rec}}_{\mathsf{adv}} := \text{read } \mathsf{MsgEnc}(0)$
- $\mathsf{MsgEnc}(1)^{\mathsf{rec}}_{\mathsf{adv}} := \text{read } \mathsf{MsgEnc}(1)$
- $\mathsf{Out} := m_0 \leftarrow \mathsf{Msg}(0);\ m_1 \leftarrow \mathsf{Msg}(1);\ c \leftarrow \mathsf{Choice};$
  if c then ret m_1 else ret m_0
- $\mathsf{Out}^{\mathsf{rec}}_{\mathsf{adv}} := \text{read Out}$

The second key is now only referenced in the channels $\mathsf{MsgEnc}(0)$ and $\mathsf{MsgEnc}(1)$, where we use it to encrypt either the first or the second message, respectively. This encryption process can be extracted out into a new internal channel PrivateMsg:

- $\mathsf{Key}(0) := \text{samp unif}_{\mathsf{msg}}$

- $\mathsf{Key}(1) := \mathsf{samp}\ \mathsf{unif}_{\mathsf{msg}}$
- $\mathsf{Flip} := \mathsf{samp}\ \mathsf{flip}$
- $\mathsf{Flip}^{\mathsf{rec}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{Flip}$
- $\mathsf{OTMsgRcvd}(0)^{\mathsf{ot}}_{\mathsf{adv}} := k_0 \leftarrow \mathsf{Key}(0);\ \mathsf{ret}\ \checkmark$
- $\mathsf{OTMsgRcvd}(1)^{\mathsf{ot}}_{\mathsf{adv}} := k_1 \leftarrow \mathsf{Key}(1);\ \mathsf{ret}\ \checkmark$
- $\mathsf{OTChoice}^{\mathsf{ot}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{Flip}$
- $\mathsf{OTOut}^{\mathsf{ot}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{SharedKey}$
- $\mathsf{SharedKey} := \mathsf{read}\ \mathsf{Key}(0)$
- $\mathsf{Choice}^{\mathsf{rec}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{Choice}$
- $\mathsf{ChoiceEnc} := f \leftarrow \mathsf{Flip};\ c \leftarrow \mathsf{Choice};\ \mathsf{ret}\ f \oplus c$
- $\mathsf{ChoiceEnc}^{\mathsf{rec}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{ChoiceEnc}$
- $\mathsf{PrivateMsg} := m_0 \leftarrow \mathsf{Msg}(0);\ m_1 \leftarrow \mathsf{Msg}(1);\ k_1 \leftarrow \mathsf{Key}(1);\ c \leftarrow \mathsf{Choice};$
  $\mathsf{if}\ c\ \mathsf{then}\ \mathsf{ret}\ m_0 \oplus k_1\ \mathsf{else}\ \mathsf{ret}\ m_1 \oplus k_1$
- $\mathsf{MsgEnc}(0) := m_0 \leftarrow \mathsf{Msg}(0);\ m_1 \leftarrow \mathsf{Msg}(1);\ k_0 \leftarrow \mathsf{Key}(0);\ m_p \leftarrow \mathsf{PrivateMsg};\ c \leftarrow \mathsf{Choice};$
  $\mathsf{if}\ c\ \mathsf{then}\ \mathsf{ret}\ m_p\ \mathsf{else}\ \mathsf{ret}\ m_0 \oplus k_0$
- $\mathsf{MsgEnc}(1) := m_0 \leftarrow \mathsf{Msg}(0);\ m_1 \leftarrow \mathsf{Msg}(1);\ k_0 \leftarrow \mathsf{Key}(0);\ m_p \leftarrow \mathsf{PrivateMsg};\ c \leftarrow \mathsf{Choice};$
  $\mathsf{if}\ c\ \mathsf{then}\ \mathsf{ret}\ m_1 \oplus k_0\ \mathsf{else}\ \mathsf{ret}\ m_p$
- $\mathsf{MsgEnc}(0)^{\mathsf{rec}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{MsgEnc}(0)$
- $\mathsf{MsgEnc}(1)^{\mathsf{rec}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{MsgEnc}(1)$
- $\mathsf{Out} := m_0 \leftarrow \mathsf{Msg}(0);\ m_1 \leftarrow \mathsf{Msg}(1);\ c \leftarrow \mathsf{Choice};$
  $\mathsf{if}\ c\ \mathsf{then}\ \mathsf{ret}\ m_1\ \mathsf{else}\ \mathsf{ret}\ m_0$
- $\mathsf{Out}^{\mathsf{rec}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{Out}$

We can now fold the internal channel $\mathsf{Key}(1)$ into the channel PrivateMsg:

- $\mathsf{Key}(0) := \mathsf{samp}\ \mathsf{unif}_{\mathsf{msg}}$
- $\mathsf{Flip} := \mathsf{samp}\ \mathsf{flip}$
- $\mathsf{Flip}^{\mathsf{rec}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{Flip}$
- $\mathsf{OTMsgRcvd}(0)^{\mathsf{ot}}_{\mathsf{adv}} := k_0 \leftarrow \mathsf{Key}(0);\ \mathsf{ret}\ \checkmark$
- $\mathsf{OTMsgRcvd}(1)^{\mathsf{ot}}_{\mathsf{adv}} := k_1 \leftarrow \mathsf{Key}(1);\ \mathsf{ret}\ \checkmark$
- $\mathsf{OTChoice}^{\mathsf{ot}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{Flip}$
- $\mathsf{OTOut}^{\mathsf{ot}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{SharedKey}$
- $\mathsf{SharedKey} := \mathsf{read}\ \mathsf{Key}(0)$
- $\mathsf{Choice}^{\mathsf{rec}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{Choice}$
- $\mathsf{ChoiceEnc} := f \leftarrow \mathsf{Flip};\ c \leftarrow \mathsf{Choice};\ \mathsf{ret}\ f \oplus c$
- $\mathsf{ChoiceEnc}^{\mathsf{rec}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{ChoiceEnc}$

- PrivateMsg $:= m_0 \leftarrow \mathsf{Msg}(0);\ m_1 \leftarrow \mathsf{Msg}(1);\ k_1 \leftarrow \mathsf{unif}_{\mathsf{msg}};\ c \leftarrow \mathsf{Choice};$
  if c then ret $m_0 \oplus k_1$ else ret $m_1 \oplus k_1$
- $\mathsf{MsgEnc}(0) := m_0 \leftarrow \mathsf{Msg}(0);\ m_1 \leftarrow \mathsf{Msg}(1);\ k_0 \leftarrow \mathsf{Key}(0);\ m_p \leftarrow \mathsf{PrivateMsg};\ c \leftarrow \mathsf{Choice};$
  if c then ret m_p else ret $m_0 \oplus k_0$
- $\mathsf{MsgEnc}(1) := m_0 \leftarrow \mathsf{Msg}(0);\ m_1 \leftarrow \mathsf{Msg}(1);\ k_0 \leftarrow \mathsf{Key}(0);\ m_p \leftarrow \mathsf{PrivateMsg};\ c \leftarrow \mathsf{Choice};$
  if c then ret $m_1 \oplus k_0$ else ret m_p
- $\mathsf{MsgEnc}(0)^{\mathsf{rec}}_{\mathsf{adv}} :=$ read $\mathsf{MsgEnc}(0)$
- $\mathsf{MsgEnc}(1)^{\mathsf{rec}}_{\mathsf{adv}} :=$ read $\mathsf{MsgEnc}(1)$
- $\mathsf{Out} := m_0 \leftarrow \mathsf{Msg}(0);\ m_1 \leftarrow \mathsf{Msg}(1);\ c \leftarrow \mathsf{Choice};$
  if c then ret m_1 else ret m_0
- $\mathsf{Out}^{\mathsf{rec}}_{\mathsf{adv}} :=$ read Out

Rearranging the order of bindings inside PrivateMsg yields

- PrivateMsg $:= m_0 \leftarrow \mathsf{Msg}(0);\ m_1 \leftarrow \mathsf{Msg}(1);\ c \leftarrow \mathsf{Choice};\ k_1 \leftarrow \mathsf{unif}_{\mathsf{msg}};$
  if c then ret $m_0 \oplus k_1$ else ret $m_1 \oplus k_1$

Since sampling and branching are interchangeable, and by assumption $\mathsf{unif}_{\mathsf{msg}}$ is invariant under xor-ing with a fixed message, the three reaction snippets

- $k_1 \leftarrow \mathsf{unif}_{\mathsf{msg}};$ if c then ret $m_0 \oplus k_1$ else ret $m_1 \oplus k_1$
- if c then $\left(k_1 \leftarrow \mathsf{unif}_{\mathsf{msg}};\ \text{ret } m_0 \oplus k_1\right)$ else $\left(k_1 \leftarrow \mathsf{unif}_{\mathsf{msg}};\ \text{ret } m_1 \oplus k_1\right)$
- if c then samp $\mathsf{unif}_{\mathsf{msg}}$ else samp $\mathsf{unif}_{\mathsf{msg}}$

are equivalent. So we may just as well not branch:

- PrivateMsg $:= m_0 \leftarrow \mathsf{Msg}(0);\ m_1 \leftarrow \mathsf{Msg}(1);\ c \leftarrow \mathsf{Choice};$ samp $\mathsf{unif}_{\mathsf{msg}}$

Unfolding the sampling back gives us:

- $\mathsf{Key}(0) :=$ samp $\mathsf{unif}_{\mathsf{msg}}$
- $\mathsf{Key}(1) :=$ samp $\mathsf{unif}_{\mathsf{msg}}$
- Flip := samp flip
- $\mathsf{Flip}^{\mathsf{rec}}_{\mathsf{adv}} :=$ read Flip
- $\mathsf{OTMsgRcvd}(0)^{\mathsf{ot}}_{\mathsf{adv}} := k_0 \leftarrow \mathsf{Key}(0);$ ret ✓
- $\mathsf{OTMsgRcvd}(1)^{\mathsf{ot}}_{\mathsf{adv}} := k_1 \leftarrow \mathsf{Key}(1);$ ret ✓
- $\mathsf{OTChoice}^{\mathsf{ot}}_{\mathsf{adv}} :=$ read Flip
- $\mathsf{OTOut}^{\mathsf{ot}}_{\mathsf{adv}} :=$ read SharedKey
- SharedKey := read $\mathsf{Key}(0)$
- $\mathsf{Choice}^{\mathsf{rec}}_{\mathsf{adv}} :=$ read Choice
- ChoiceEnc $:= f \leftarrow$ Flip; $c \leftarrow$ Choice; ret $f \oplus c$
- $\mathsf{ChoiceEnc}^{\mathsf{rec}}_{\mathsf{adv}} :=$ read ChoiceEnc
- PrivateMsg $:= m_0 \leftarrow \mathsf{Msg}(0);\ m_1 \leftarrow \mathsf{Msg}(1);\ c \leftarrow \mathsf{Choice};$ read $\mathsf{Key}(1)$

- $\mathsf{MsgEnc}(0) := m_0 \leftarrow \mathsf{Msg}(0);\ m_1 \leftarrow \mathsf{Msg}(1);\ k_0 \leftarrow \mathsf{Key}(0);\ m_p \leftarrow \mathsf{PrivateMsg};\ c \leftarrow \mathsf{Choice};$
  if c then ret m_p else ret $m_0 \oplus k_0$
- $\mathsf{MsgEnc}(1) := m_0 \leftarrow \mathsf{Msg}(0);\ m_1 \leftarrow \mathsf{Msg}(1);\ k_0 \leftarrow \mathsf{Key}(0);\ m_p \leftarrow \mathsf{PrivateMsg};\ c \leftarrow \mathsf{Choice};$
  if c then ret $m_1 \oplus k_0$ else ret m_p
- $\mathsf{MsgEnc}(0)^{\mathsf{rec}}_{\mathsf{adv}} := \text{read } \mathsf{MsgEnc}(0)$
- $\mathsf{MsgEnc}(1)^{\mathsf{rec}}_{\mathsf{adv}} := \text{read } \mathsf{MsgEnc}(1)$
- $\mathsf{Out} := m_0 \leftarrow \mathsf{Msg}(0);\ m_1 \leftarrow \mathsf{Msg}(1);\ c \leftarrow \mathsf{Choice};$
  if c then ret m_1 else ret m_0
- $\mathsf{Out}^{\mathsf{rec}}_{\mathsf{adv}} := \text{read } \mathsf{Out}$

The internal channel PrivateMsg can now be substituted away, yielding the final version of the real protocol:

- $\mathsf{Key}(0) := \text{samp unif}_{\mathsf{msg}}$
- $\mathsf{Key}(1) := \text{samp unif}_{\mathsf{msg}}$
- $\mathsf{Flip} := \text{samp flip}$
- $\mathsf{Flip}^{\mathsf{rec}}_{\mathsf{adv}} := \text{read } \mathsf{Flip}$
- $\mathsf{OTMsgRcvd}(0)^{\mathsf{ot}}_{\mathsf{adv}} := k_0 \leftarrow \mathsf{Key}(0);\ \text{ret } \checkmark$
- $\mathsf{OTMsgRcvd}(1)^{\mathsf{ot}}_{\mathsf{adv}} := k_1 \leftarrow \mathsf{Key}(1);\ \text{ret } \checkmark$
- $\mathsf{OTChoice}^{\mathsf{ot}}_{\mathsf{adv}} := \text{read } \mathsf{Flip}$
- $\mathsf{OTOut}^{\mathsf{ot}}_{\mathsf{adv}} := \text{read } \mathsf{SharedKey}$
- $\mathsf{SharedKey} := \text{read } \mathsf{Key}(0)$
- $\mathsf{Choice}^{\mathsf{rec}}_{\mathsf{adv}} := \text{read } \mathsf{Choice}$
- $\mathsf{ChoiceEnc} := f \leftarrow \mathsf{Flip};\ c \leftarrow \mathsf{Choice};\ \text{ret } f \oplus c$
- $\mathsf{ChoiceEnc}^{\mathsf{rec}}_{\mathsf{adv}} := \text{read } \mathsf{ChoiceEnc}$
- $\mathsf{MsgEnc}(0) := m_0 \leftarrow \mathsf{Msg}(0);\ m_1 \leftarrow \mathsf{Msg}(1);\ k_0 \leftarrow \mathsf{Key}(0);\ k_1 \leftarrow \mathsf{Key}(1);\ c \leftarrow \mathsf{Choice};$
  if c then ret k_1 else ret $m_0 \oplus k_0$
- $\mathsf{MsgEnc}(1) := m_0 \leftarrow \mathsf{Msg}(0);\ m_1 \leftarrow \mathsf{Msg}(1);\ k_0 \leftarrow \mathsf{Key}(0);\ k_1 \leftarrow \mathsf{Key}(1);\ c \leftarrow \mathsf{Choice};$
  if c then ret $m_1 \oplus k_0$ else ret k_1
- $\mathsf{MsgEnc}(0)^{\mathsf{rec}}_{\mathsf{adv}} := \text{read } \mathsf{MsgEnc}(0)$
- $\mathsf{MsgEnc}(1)^{\mathsf{rec}}_{\mathsf{adv}} := \text{read } \mathsf{MsgEnc}(1)$
- $\mathsf{Out} := m_0 \leftarrow \mathsf{Msg}(0);\ m_1 \leftarrow \mathsf{Msg}(1);\ c \leftarrow \mathsf{Choice};$
  if c then ret m_1 else ret m_0
- $\mathsf{Out}^{\mathsf{rec}}_{\mathsf{adv}} := \text{read } \mathsf{Out}$

7.5 The Simulator

The channel

- $\mathsf{Out} := m_0 \leftarrow \mathsf{Msg}(0);\ m_1 \leftarrow \mathsf{Msg}(1);\ c \leftarrow \mathsf{Choice};$
  $\mathsf{if}\ c\ \mathsf{then\ ret}\ m_1\ \mathsf{else\ ret}\ m_0$

can now be separated out as coming from the functionality, and the remainder of the protocol is turned into the simulator below. Plugging in the simulator into the ideal functionality and substituting away the internal channels $\mathsf{Choice}^{\mathsf{id}}_{\mathsf{adv}}$ and $\mathsf{Out}^{\mathsf{id}}_{\mathsf{adv}}$ that originally served as a line of communication for the adversary yields the final version of the real protocol, as desired.

- $\mathsf{Key}(0) := \mathsf{samp\ unif}_{\mathsf{msg}}$
- $\mathsf{Key}(1) := \mathsf{samp\ unif}_{\mathsf{msg}}$
- $\mathsf{Flip} := \mathsf{samp\ flip}$
- $\mathsf{Flip}^{\mathsf{rec}}_{\mathsf{adv}} := \mathsf{read\ Flip}$
- $\mathsf{OTMsgRcvd}(0)^{\mathsf{ot}}_{\mathsf{adv}} := k_0 \leftarrow \mathsf{Key}(0);\ \mathsf{ret}\ \checkmark$
- $\mathsf{OTMsgRcvd}(1)^{\mathsf{ot}}_{\mathsf{adv}} := k_1 \leftarrow \mathsf{Key}(1);\ \mathsf{ret}\ \checkmark$
- $\mathsf{OTChoice}^{\mathsf{ot}}_{\mathsf{adv}} := \mathsf{read\ Flip}$
- $\mathsf{OTOut}^{\mathsf{ot}}_{\mathsf{adv}} := \mathsf{read\ SharedKey}$
- $\mathsf{SharedKey} := \mathsf{read}\ \mathsf{Key}(0)$
- $\mathsf{Choice}^{\mathsf{rec}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{Choice}^{\mathsf{id}}_{\mathsf{adv}}$
- $\mathsf{ChoiceEnc} := f \leftarrow \mathsf{Flip};\ c \leftarrow \mathsf{Choice}^{\mathsf{id}}_{\mathsf{adv}};\ \mathsf{ret}\ f \oplus c$
- $\mathsf{ChoiceEnc}^{\mathsf{rec}}_{\mathsf{adv}} := \mathsf{read\ ChoiceEnc}$
- $\mathsf{MsgEnc}(0) := m \leftarrow \mathsf{Out}^{\mathsf{id}}_{\mathsf{adv}};\ k_0 \leftarrow \mathsf{Key}(0);\ k_1 \leftarrow \mathsf{Key}(1);\ c \leftarrow \mathsf{Choice}^{\mathsf{id}}_{\mathsf{adv}};$
  $\mathsf{if}\ c\ \mathsf{then\ ret}\ k_1\ \mathsf{else\ ret}\ m \oplus k_0$
- $\mathsf{MsgEnc}(1) := m \leftarrow \mathsf{Out}^{\mathsf{id}}_{\mathsf{adv}};\ k_0 \leftarrow \mathsf{Key}(0);\ k_1 \leftarrow \mathsf{Key}(1);\ c \leftarrow \mathsf{Choice}^{\mathsf{id}}_{\mathsf{adv}};$
  $\mathsf{if}\ c\ \mathsf{then\ ret}\ m \oplus k_0\ \mathsf{else\ ret}\ k_1$
- $\mathsf{MsgEnc}(0)^{\mathsf{rec}}_{\mathsf{adv}} := \mathsf{read\ MsgEnc}(0)$
- $\mathsf{MsgEnc}(1)^{\mathsf{rec}}_{\mathsf{adv}} := \mathsf{read\ MsgEnc}(1)$
- $\mathsf{Out}^{\mathsf{rec}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{Out}^{\mathsf{id}}_{\mathsf{adv}}$

8 Multi-Party Coin Toss

In this section we implement a protocol where $n+2$ parties labeled $0, \ldots, n+1$ reach a Boolean consensus. We prove the protocol secure against a malicious attacker in the case when the last party is honest and any other party is arbitrarily honest or corrupt. Formally, we assume a coin-flip distribution $\mathsf{flip} : 1 \twoheadrightarrow \mathsf{Bool}$ and a Boolean sum function $\oplus : \mathsf{Bool} \times \mathsf{Bool} \to \mathsf{Bool}$, where we write $x \oplus y$ in place of $\oplus\,(x, y)$.

8.1 The Assumptions

At the expression level, we assume that the operation of Boolean sum with a fixed bit is self-inverse:

- $x : \mathsf{Bool}, y : \mathsf{Bool} \vdash (x \oplus y) \oplus y = x : \mathsf{Bool}$.

At the distribution level, we assume that the distribution flip on bits is invariant under the operation of Boolean sum with a fixed bit (as is indeed the case when flip is uniform):

- $x : \mathsf{Bool} \vdash \big(y \leftarrow \mathsf{flip};\ 1[x \oplus y]\big) = \mathsf{flip} : \mathsf{Bool}$

8.2 The Ideal Protocol

The ideal functionality generates a random Boolean, leaks it to the adversary, and, upon the approval from the adversary, outputs it on behalf of every honest party:

- $\mathsf{Flip} := \mathsf{samp}\ \mathsf{flip}$
- $\mathsf{LeakFlip}^{\mathsf{id}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{Flip}$
- $\begin{cases} \mathsf{Out}(i) := \_ \leftarrow \mathsf{Ok}^{\mathsf{adv}}_{\mathsf{id}};\ \mathsf{read}\ \mathsf{Flip} & \text{if } 0 \leqslant i \leqslant n+1 \text{ honest} \\ \mathsf{Out}(i) := \mathsf{read}\ \mathsf{Out}(i) & \text{otherwise} \end{cases}$

The output of every corrupted party diverges, since in the malicious setting the external outputs of corrupted parties provide no useful information.

8.3 The Real Protocol

We assume that each party has an associated *commitment functionality* that broadcasts information, and that all broadcast communication is visible to the adversary. At the start of the protocol, each honest party i commits to a randomly generated Boolean and sends it to its commitment functionality:

- $\mathsf{Commit}(i) := \mathsf{samp}\ \mathsf{flip}$

In the malicious setting, we assume that the adversary supplies inputs to each corrupted party in lieu of the party's own internal computation. Thus, each corrupted party i commits to the Boolean of the adversary's choice:

- $\mathsf{Commit}(i) := \mathsf{read}\ \mathsf{AdvCommit}^{\mathsf{adv}}_{\mathsf{party}(i)}$

To uniformly cover all cases, we assume channels $\mathsf{AdvCommit}^{\mathsf{adv}}_{\mathsf{party}(i)}$ as inputs to the real protocol, for all $0 \leqslant i \leqslant n+1$ even if i is honest; in this case the corresponding input simply goes unused.

Upon receiving the commit from the party, each commitment functionality broadcasts the fact that a commit happened – but not its value – to everybody, including the adversary:

- $\mathsf{Committed}(i) := c_i \leftarrow \mathsf{Commit}(i);\ \mathsf{ret}\ \checkmark$
- $\mathsf{LeakCommitted}(i)^{\mathsf{comm}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{Committed}(i)$

Each honest party i inductively keeps track of all parties that have already committed:

- $\begin{cases} \mathsf{AllCommitted}(i,0) := \mathsf{ret}\ \checkmark \\ \mathsf{AllCommitted}(i,j+1) := \_ \leftarrow \mathsf{AllCommitted}(i,j);\ c_j \leftarrow \mathsf{Committed}(j);\ \mathsf{ret}\ \checkmark & \text{for } 0 \leqslant j \leqslant n+1 \end{cases}$

After all parties have committed, each honest party lets the commitment functionality open its commit for everybody else to see:

- $\mathsf{Open}(i) := \_ \leftarrow \mathsf{AllCommitted}(i, n+2);\ \mathsf{ret}\ \checkmark$

A corrupted party i opens its commit when the adversary says so:

- $\mathsf{Open}(i) := \mathsf{read}\ \mathsf{AdvOpen}^{\mathsf{adv}}_{\mathsf{party}(i)}$

We again assume channels $\mathsf{AdvOpen}^{\mathsf{adv}}_{\mathsf{party}(i)}$ as inputs to the real protocol for all $0 \leqslant i \leqslant n+1$.

Upon receiving the party's decision to open the commit, each commitment functionality broadcasts the value of the commit to everybody, including the adversary:

- $\mathsf{Opened}(i) := \_ \leftarrow \mathsf{Open}(i);\ \mathsf{read}\ \mathsf{Commit}(i)$
- $\mathsf{LeakOpened}(i)^{\mathsf{comm}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{Opened}(i)$

Each honest party i inductively sums up the commits of all parties once they have been opened:

- $\begin{cases} \mathsf{SumOpened}(i,0) := \mathsf{ret\ false} \\ \mathsf{SumOpened}(i,j+1) := x_j \leftarrow \mathsf{SumOpened}(i,j);\ o_j \leftarrow \mathsf{Opened}(j);\ \mathsf{ret}\ x_j \oplus o_j \quad \text{for } 0 \leqslant j \leqslant n+1 \end{cases}$

Finally, each honest party i outputs the consensus - the Boolean sum of all commits:

- $\mathsf{Out}(i) := \mathsf{read\ SumOpened}(i, n+2)$

The output of each corrupted party i diverges:

- $\mathsf{Out}(i) := \mathsf{read\ Out}(i)$

Thus, we have the following code for each honest party i:

- $\mathsf{Commit}(i) := \mathsf{samp\ flip}$
- $\begin{cases} \mathsf{AllCommitted}(i,0) := \mathsf{ret}\ \checkmark \\ \mathsf{AllCommitted}(i,j+1) := \_ \leftarrow \mathsf{AllCommitted}(i,j);\ c_j \leftarrow \mathsf{Committed}(j);\ \mathsf{ret}\ \checkmark \quad \text{for } 0 \leqslant j \leqslant n+1 \end{cases}$
- $\mathsf{Open}(i) := \_ \leftarrow \mathsf{AllCommitted}(i, n+2);\ \mathsf{ret}\ \checkmark$
- $\begin{cases} \mathsf{SumOpened}(i,0) := \mathsf{ret\ false} \\ \mathsf{SumOpened}(i,j+1) := x_j \leftarrow \mathsf{SumOpened}(i,j);\ o_j \leftarrow \mathsf{Opened}(j);\ \mathsf{ret}\ x_j \oplus o_j \quad \text{for } 0 \leqslant j \leqslant n+1 \end{cases}$
- $\mathsf{Out}(i) := \mathsf{read\ SumOpened}(i, n+2)$

The code for a corrupted party i has the following form:

- $\mathsf{Commit}(i) := \mathsf{read\ AdvCommit}^{\mathsf{adv}}_{\mathsf{party}(i)}$
- $\mathsf{Open}(i) := \mathsf{read\ AdvOpen}^{\mathsf{adv}}_{\mathsf{party}(i)}$
- $\mathsf{Out}(i) := \mathsf{read\ Out}(i)$

Finally, the code for the commitment functionality for party i is below:

- $\mathsf{Committed}(i) := c_i \leftarrow \mathsf{Commit}(i);\ \mathsf{ret}\ \checkmark$
- $\mathsf{LeakCommitted}(i)^{\mathsf{comm}}_{\mathsf{adv}} := \mathsf{read\ Committed}(i)$
- $\mathsf{Opened}(i) := \_ \leftarrow \mathsf{Open}(i);\ \mathsf{read\ Commit}(i)$
- $\mathsf{LeakOpened}(i)^{\mathsf{comm}}_{\mathsf{adv}} := \mathsf{read\ Opened}(i)$

Composing all of the above together and hiding the internal communication yields the real protocol.

8.4 The Simulator

In the real protocol, the consensus is the Boolean sum of all parties' commits. The simulator, however, gets the value of the consensus from the ideal functionality. To preserve the invariant that the consensus is the sum of all commits, we adjust the last party's commit: it is no longer a random Boolean, but rather the sum of all other commits plus the consensus. Hence, in the simulator, the last commit only happens after all the other commits, unlike in the real world where the last commit has no dependencies. This is okay – the last party is by assumption honest, so there is no leakage that would need to happen right away – but requires some care. Specifically, the announcement that the last party committed must be independent of the timing of the other commits, so we cannot let it actually depend on the last commit as it does in the real world. Instead, we manually postulate no dependencies. The simulator gives the ok message to the functionality once all the commits (except the last, which we explicitly construct) and all the requests to open have been made.

- $\begin{cases} \mathsf{Commit}(i) := \mathsf{samp\ flip} & \text{if } 0 \leqslant i \leqslant n \text{ honest} \\ \mathsf{Commit}(i) := \mathsf{read\ AdvCommit}^{\mathsf{adv}}_{\mathsf{party}(i)} & \text{otherwise} \end{cases}$

- $\mathsf{LastCommit} := x_{n+1} \leftarrow \mathsf{SumCommit}(n+1);\ f \leftarrow \mathsf{LeakFlip}^{\mathsf{id}}_{\mathsf{adv}};\ \mathsf{ret}\ x_{n+1} \oplus f$
- $\begin{cases} \mathsf{SumCommit}(0) := \mathsf{ret\ false} \\ \mathsf{SumCommit}(j+1) := x_j \leftarrow \mathsf{SumCommit}(j);\ c_j \leftarrow \mathsf{Commit}(j);\ \mathsf{ret}\ x_j \oplus c_j \quad \text{for } 0 \leqslant j \leqslant n \end{cases}$
- $\mathsf{SumCommit}(n+2) := x_{n+1} \leftarrow \mathsf{SumCommit}(n+1);\ c_{n+1} \leftarrow \mathsf{LastCommit};\ \mathsf{ret}\ x_{n+1} \oplus c_{n+1}$
- $\mathsf{Committed}(i) := c_j \leftarrow \mathsf{Commit}(i);\ \mathsf{ret}\ \checkmark \text{ for } 0 \leqslant i \leqslant n$
- $\mathsf{Committed}(n+1) := \mathsf{ret}\ \checkmark$
- $\mathsf{LeakCommitted}(i)^{\mathsf{comm}}_{\mathsf{adv}} := \mathsf{read\ Committed}(i) \text{ for } 0 \leqslant i \leqslant n+1$
- $\begin{cases} \mathsf{Open}(i) := x_{n+1} \leftarrow \mathsf{SumCommit}(n+2);\ \mathsf{ret}\ \checkmark & \text{if } 0 \leqslant i \leqslant n+1 \text{ honest} \\ \mathsf{Open}(i) := \mathsf{read\ AdvOpen}^{\mathsf{adv}}_{\mathsf{party}(i)} & \text{otherwise} \end{cases}$
- $\begin{cases} \mathsf{AllOpen}(0) := \mathsf{ret}\ \checkmark \\ \mathsf{AllOpen}(j+1) := \_ \leftarrow \mathsf{AllOpen}(j);\ \_ \leftarrow \mathsf{Open}(j);\ \mathsf{ret}\ \checkmark \quad \text{for } 0 \leqslant j \leqslant n+1 \end{cases}$
- $\mathsf{Opened}(i) := \_ \leftarrow \mathsf{Open}(i);\ \mathsf{read\ Commit}(i) \text{ for } 0 \leqslant i \leqslant n+1$
- $\mathsf{LeakOpened}(i)^{\mathsf{comm}}_{\mathsf{adv}} := \mathsf{read\ Opened}(i) \text{ for } 0 \leqslant i \leqslant n+1$
- $\mathsf{Ok}^{\mathsf{adv}}_{\mathsf{id}} := \_ \leftarrow \mathsf{AllOpen}(n+2);\ x_{n+1} \leftarrow \mathsf{SumCommit}(n+1); \mathsf{ret}\ \checkmark$

8.5 Real = Ideal + Simulator

In the real protocol, the composition of all commitment functionalities has the following form:

- $\mathsf{Committed}(i) := c_i \leftarrow \mathsf{Commit}(i);\ \mathsf{ret}\ \checkmark \text{ for } 0 \leqslant i \leqslant n+1$
- $\mathsf{LeakCommitted}(i)^{\mathsf{comm}}_{\mathsf{adv}} := \mathsf{read\ Committed}(i) \text{ for } 0 \leqslant i \leqslant n+1$
- $\mathsf{Opened}(i) := \_ \leftarrow \mathsf{Open}(i);\ \mathsf{read\ Commit}(i) \text{ for } 0 \leqslant i \leqslant n+1$
- $\mathsf{LeakOpened}(i)^{\mathsf{comm}}_{\mathsf{adv}} := \mathsf{read\ Opened}(i) \text{ for } 0 \leqslant i \leqslant n+1$

Currently, each honest party i keeps its own track of who committed. This is of course unnecessary, as each party has the same information, so we can add new internal channels $\mathsf{AllCommitted}(-)$ that inductively keep a global track of commitment:

- $\mathsf{Committed}(i) := c_i \leftarrow \mathsf{Commit}(i);\ \mathsf{ret}\ \checkmark \text{ for } 0 \leqslant i \leqslant n+1$
- $\mathsf{LeakCommitted}(i)^{\mathsf{comm}}_{\mathsf{adv}} := \mathsf{read\ Committed}(i) \text{ for } 0 \leqslant i \leqslant n+1$
- $\begin{cases} \mathsf{AllCommitted}(0) := \mathsf{ret}\ \checkmark \\ \mathsf{AllCommitted}(j+1) := \_ \leftarrow \mathsf{AllCommitted}(j);\ c_j \leftarrow \mathsf{Committed}(j);\ \mathsf{ret}\ \checkmark \quad \text{for } 0 \leqslant j \leqslant n+1 \end{cases}$
- $\mathsf{Opened}(i) := \_ \leftarrow \mathsf{Open}(i);\ \mathsf{read\ Commit}(i) \text{ for } 0 \leqslant i \leqslant n+1$
- $\mathsf{LeakOpened}(i)^{\mathsf{comm}}_{\mathsf{adv}} := \mathsf{read\ Opened}(i) \text{ for } 0 \leqslant i \leqslant n+1$

In the presence of the above, we can inductively rewrite the code of each honest party i to the following:

- $\mathsf{Commit}(i) := \mathsf{samp\ flip}$
- $\mathsf{AllCommitted}(i,j) := \mathsf{read\ AllCommitted}(j) \text{ for } 0 \leqslant j \leqslant n+2$
- $\mathsf{Open}(i) := \_ \leftarrow \mathsf{AllCommitted}(i, n+2);\ \mathsf{ret}\ \checkmark$

- $\begin{cases} \mathsf{SumOpened}(i,0) := \mathsf{ret\ false} \\ \mathsf{SumOpened}(i,j+1) := x_j \leftarrow \mathsf{SumOpened}(i,j);\ o_j \leftarrow \mathsf{Opened}(j);\ \mathsf{ret}\ x_j \oplus o_j \quad \text{for } 0 \leqslant j \leqslant n+1 \end{cases}$
- $\mathsf{Out}(i) := \mathsf{read\ SumOpened}(i, n+2)$

After substituting the channel $\mathsf{AllCommitted}(i, n+2)$ into $\mathsf{Open}(i)$, the internal channels $\mathsf{AllCommitted}(i,-)$ become unused and we can eliminate them entirely:

- $\mathsf{Commit}(i) := \mathsf{samp\ flip}$
- $\mathsf{Open}(i) := \_ \leftarrow \mathsf{AllCommitted}(n+2);\ \mathsf{ret}\ \checkmark$
- $\begin{cases} \mathsf{SumOpened}(i,0) := \mathsf{ret\ false} \\ \mathsf{SumOpened}(i,j+1) := x_j \leftarrow \mathsf{SumOpened}(i,j);\ o_j \leftarrow \mathsf{Opened}(j);\ \mathsf{ret}\ x_j \oplus o_j \quad \text{for } 0 \leqslant j \leqslant n+1 \end{cases}$
- $\mathsf{Out}(i) := \mathsf{read\ SumOpened}(i, n+2)$

By the same token, we can add new internal channels $\mathsf{SumOpened}(-)$ to the composition of functionalities that inductively keep a global track of the sum of all commits once they have been opened:

- $\mathsf{Committed}(i) := c_i \leftarrow \mathsf{Commit}(i);\ \mathsf{ret}\ \checkmark \text{ for } 0 \leqslant i \leqslant n+1$
- $\mathsf{LeakCommitted}(i)^{\mathsf{comm}}_{\mathsf{adv}} := \mathsf{read\ Committed}(i) \text{ for } 0 \leqslant i \leqslant n+1$
- $\begin{cases} \mathsf{AllCommitted}(0) := \mathsf{ret}\ \checkmark \\ \mathsf{AllCommitted}(j+1) := \_ \leftarrow \mathsf{AllCommitted}(j);\ c_j \leftarrow \mathsf{Committed}(j);\ \mathsf{ret}\ \checkmark \quad \text{for } 0 \leqslant j \leqslant n+1 \end{cases}$
- $\mathsf{Opened}(i) := \_ \leftarrow \mathsf{Open}(i);\ \mathsf{read\ Commit}(i) \text{ for } 0 \leqslant i \leqslant n+1$
- $\mathsf{LeakOpened}(i)^{\mathsf{comm}}_{\mathsf{adv}} := \mathsf{read\ Opened}(i) \text{ for } 0 \leqslant i \leqslant n+1$
- $\begin{cases} \mathsf{SumOpened}(0) := \mathsf{ret\ false} \\ \mathsf{SumOpened}(j+1) := x_j \leftarrow \mathsf{SumOpened}(j);\ o_j \leftarrow \mathsf{Opened}(j);\ \mathsf{ret}\ x_j \oplus o_j \quad \text{for } 0 \leqslant j \leqslant n+1 \end{cases}$

In the presence of the above, we can inductively rewrite the code of each honest party i to the following:

- $\mathsf{Commit}(i) := \mathsf{samp\ flip}$
- $\mathsf{Open}(i) := \_ \leftarrow \mathsf{AllCommitted}(n+2);\ \mathsf{ret}\ \checkmark$
- $\mathsf{SumOpened}(i,j) := \mathsf{read\ SumOpened}(j) \text{ for } 0 \leqslant j \leqslant n+2$
- $\mathsf{Out}(i) := \mathsf{read\ SumOpened}(i, n+2)$

After substituting the channel $\mathsf{SumOpened}(i, n+2)$ into $\mathsf{Out}(i)$, the internal channels $\mathsf{SumOpened}(i,-)$ become unused and we can eliminate them entirely:

- $\mathsf{Commit}(i) := \mathsf{samp\ flip}$
- $\mathsf{Open}(i) := \_ \leftarrow \mathsf{AllCommitted}(n+2);\ \mathsf{ret}\ \checkmark$
- $\mathsf{Out}(i) := \mathsf{read\ SumOpened}(n+2)$

The combined code for the real protocol after the aforementioned changes is thus as follows:

- $\begin{cases} \mathsf{Commit}(i) := \mathsf{samp\ flip} & \text{if } 0 \leqslant i \leqslant n+1 \text{ honest} \\ \mathsf{Commit}(i) := \mathsf{read\ AdvCommit}^{\mathsf{adv}}_{\mathsf{party}(i)} & \text{otherwise} \end{cases}$
- $\mathsf{Committed}(i) := c_i \leftarrow \mathsf{Commit}(i);\ \mathsf{ret}\ \checkmark \text{ for } 0 \leqslant i \leqslant n+1$

- $\mathsf{LeakCommitted}(i)^{\mathsf{comm}}_{\mathsf{adv}} := \text{read } \mathsf{Committed}(i) \text{ for } 0 \leqslant i \leqslant n+1$
- $\begin{cases} \mathsf{AllCommitted}(0) := \text{ret } \checkmark \\ \mathsf{AllCommitted}(j+1) := \_ \leftarrow \mathsf{AllCommitted}(j);\ c_j \leftarrow \mathsf{Committed}(j);\ \text{ret } \checkmark & \text{for } 0 \leqslant j \leqslant n+1 \end{cases}$
- $\begin{cases} \mathsf{Open}(i) := \_ \leftarrow \mathsf{AllCommitted}(n+2);\ \text{ret } \checkmark & \text{if } 0 \leqslant i \leqslant n+1 \text{ honest} \\ \mathsf{Open}(i) := \text{read } \mathsf{AdvOpen}^{\mathsf{adv}}_{\mathsf{party}(i)} & \text{otherwise} \end{cases}$
- $\mathsf{Opened}(i) := \_ \leftarrow \mathsf{Open}(i);\ \text{read } \mathsf{Commit}(i) \text{ for } 0 \leqslant i \leqslant n+1$
- $\mathsf{LeakOpened}(i)^{\mathsf{comm}}_{\mathsf{adv}} := \text{read } \mathsf{Opened}(i) \text{ for } 0 \leqslant i \leqslant n+1$
- $\begin{cases} \mathsf{SumOpened}(0) := \text{ret false} \\ \mathsf{SumOpened}(j+1) := x_j \leftarrow \mathsf{SumOpened}(j);\ o_j \leftarrow \mathsf{Opened}(j);\ \text{ret } x_j \oplus o_j & \text{for } 0 \leqslant j \leqslant n+1 \end{cases}$
- $\begin{cases} \mathsf{Out}(i) := \text{read } \mathsf{SumOpened}(n+2) & \text{if } 0 \leqslant i \leqslant n+1 \text{ honest} \\ \mathsf{Out}(i) := \text{read } \mathsf{Out}(i) & \text{otherwise} \end{cases}$

Instead of summing up the commits once they have been opened, we can sum them up at the beginning, as done in the simulator, using new internal channels $\mathsf{SumCommit}(-)$:

- $\begin{cases} \mathsf{Commit}(i) := \text{samp flip} & \text{if } 0 \leqslant i \leqslant n+1 \text{ honest} \\ \mathsf{Commit}(i) := \text{read } \mathsf{AdvCommit}^{\mathsf{adv}}_{\mathsf{party}(i)} & \text{otherwise} \end{cases}$
- $\begin{cases} \mathsf{SumCommit}(0) := \text{ret false} \\ \mathsf{SumCommit}(j+1) := x_j \leftarrow \mathsf{SumCommit}(j);\ c_j \leftarrow \mathsf{Commit}(j);\ \text{ret } x_j \oplus c_j & \text{for } 0 \leqslant j \leqslant n+1 \end{cases}$
- $\mathsf{Committed}(i) := c_i \leftarrow \mathsf{Commit}(i);\ \text{ret } \checkmark \text{ for } 0 \leqslant i \leqslant n+1$
- $\mathsf{LeakCommitted}(i)^{\mathsf{comm}}_{\mathsf{adv}} := \text{read } \mathsf{Committed}(i) \text{ for } 0 \leqslant i \leqslant n+1$
- $\begin{cases} \mathsf{AllCommitted}(0) := \text{ret } \checkmark \\ \mathsf{AllCommitted}(j+1) := \_ \leftarrow \mathsf{AllCommitted}(j);\ c_j \leftarrow \mathsf{Committed}(j);\ \text{ret } \checkmark & \text{for } 0 \leqslant j \leqslant n+1 \end{cases}$
- $\begin{cases} \mathsf{Open}(i) := \_ \leftarrow \mathsf{AllCommitted}(n+2);\ \text{ret } \checkmark & \text{if } 0 \leqslant i \leqslant n+1 \text{ honest} \\ \mathsf{Open}(i) := \text{read } \mathsf{AdvOpen}^{\mathsf{adv}}_{\mathsf{party}(i)} & \text{otherwise} \end{cases}$
- $\mathsf{Opened}(i) := \_ \leftarrow \mathsf{Open}(i);\ \text{read } \mathsf{Commit}(i) \text{ for } 0 \leqslant i \leqslant n+1$
- $\mathsf{LeakOpened}(i)^{\mathsf{comm}}_{\mathsf{adv}} := \text{read } \mathsf{Opened}(i) \text{ for } 0 \leqslant i \leqslant n+1$
- $\begin{cases} \mathsf{SumOpened}(0) := \text{ret false} \\ \mathsf{SumOpened}(j+1) := x_j \leftarrow \mathsf{SumOpened}(j);\ o_j \leftarrow \mathsf{Opened}(j);\ \text{ret } x_j \oplus o_j & \text{for } 0 \leqslant j \leqslant n+1 \end{cases}$
- $\begin{cases} \mathsf{Out}(i) := \text{read } \mathsf{SumOpened}(n+2) & \text{if } 0 \leqslant i \leqslant n+1 \text{ honest} \\ \mathsf{Out}(i) := \text{read } \mathsf{Out}(i) & \text{otherwise} \end{cases}$

In the presence of these new channels, the channels $\mathsf{AllCommitted}(-)$ can be simplified:

- $\begin{cases} \mathsf{Commit}(i) := \text{samp flip} & \text{if } 0 \leqslant i \leqslant n+1 \text{ honest} \\ \mathsf{Commit}(i) := \text{read } \mathsf{AdvCommit}^{\mathsf{adv}}_{\mathsf{party}(i)} & \text{otherwise} \end{cases}$
- $\begin{cases} \mathsf{SumCommit}(0) := \text{ret false} \\ \mathsf{SumCommit}(j+1) := x_j \leftarrow \mathsf{SumCommit}(j);\ c_j \leftarrow \mathsf{Commit}(j);\ \text{ret } x_j \oplus c_j & \text{for } 0 \leqslant j \leqslant n+1 \end{cases}$

- $\mathsf{Committed}(i) := c_i \leftarrow \mathsf{Commit}(i);\ \mathsf{ret}\ \checkmark$ for $0 \leqslant i \leqslant n+1$
- $\mathsf{LeakCommitted}(i)^{\mathsf{comm}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{Committed}(i)$ for $0 \leqslant i \leqslant n+1$
- $\mathsf{AllCommitted}(j) := c_j \leftarrow \mathsf{SumCommit}(j);\ \mathsf{ret}\ \checkmark$ for $0 \leqslant j \leqslant n+2$
- $\begin{cases} \mathsf{Open}(i) := \_ \leftarrow \mathsf{AllCommitted}(n+2);\ \mathsf{ret}\ \checkmark & \text{if } 0 \leqslant i \leqslant n+1 \text{ honest} \\ \mathsf{Open}(i) := \mathsf{read}\ \mathsf{AdvOpen}^{\mathsf{adv}}_{\mathsf{party}(i)} & \text{otherwise} \end{cases}$
- $\mathsf{Opened}(i) := \_ \leftarrow \mathsf{Open}(i);\ \mathsf{read}\ \mathsf{Commit}(i)$ for $0 \leqslant i \leqslant n+1$
- $\mathsf{LeakOpened}(i)^{\mathsf{comm}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{Opened}(i)$ for $0 \leqslant i \leqslant n+1$
- $\begin{cases} \mathsf{SumOpened}(0) := \mathsf{ret}\ \mathsf{false} \\ \mathsf{SumOpened}(j+1) := x_j \leftarrow \mathsf{SumOpened}(j);\ o_j \leftarrow \mathsf{Opened}(j);\ \mathsf{ret}\ x_j \oplus o_j & \text{for } 0 \leqslant j \leqslant n+1 \end{cases}$
- $\begin{cases} \mathsf{Out}(i) := \mathsf{read}\ \mathsf{SumOpened}(n+2) & \text{if } 0 \leqslant i \leqslant n+1 \text{ honest} \\ \mathsf{Out}(i) := \mathsf{read}\ \mathsf{Out}(i) & \text{otherwise} \end{cases}$

After substituting the channel $\mathsf{AllCommitted}(n+2)$ into the channels $\mathsf{Open}(i)$ for $0 \leqslant i \leqslant n+1$ honest, the internal channels $\mathsf{AllCommitted}(-)$ become unused and we can eliminate them entirely:

- $\begin{cases} \mathsf{Commit}(i) := \mathsf{samp}\ \mathsf{flip} & \text{if } 0 \leqslant i \leqslant n+1 \text{ honest} \\ \mathsf{Commit}(i) := \mathsf{read}\ \mathsf{AdvCommit}^{\mathsf{adv}}_{\mathsf{party}(i)} & \text{otherwise} \end{cases}$
- $\begin{cases} \mathsf{SumCommit}(0) := \mathsf{ret}\ \mathsf{false} \\ \mathsf{SumCommit}(j+1) := x_j \leftarrow \mathsf{SumCommit}(j);\ c_j \leftarrow \mathsf{Commit}(j);\ \mathsf{ret}\ x_j \oplus c_j & \text{for } 0 \leqslant j \leqslant n+1 \end{cases}$
- $\mathsf{Committed}(i) := c_i \leftarrow \mathsf{Commit}(i);\ \mathsf{ret}\ \checkmark$ for $0 \leqslant i \leqslant n+1$
- $\mathsf{LeakCommitted}(i)^{\mathsf{comm}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{Committed}(i)$ for $0 \leqslant i \leqslant n+1$
- $\begin{cases} \mathsf{Open}(i) := x_{n+2} \leftarrow \mathsf{SumCommit}(n+2);\ \mathsf{ret}\ \checkmark & \text{if } 0 \leqslant i \leqslant n+1 \text{ honest} \\ \mathsf{Open}(i) := \mathsf{read}\ \mathsf{AdvOpen}^{\mathsf{adv}}_{\mathsf{party}(i)} & \text{otherwise} \end{cases}$
- $\mathsf{Opened}(i) := \_ \leftarrow \mathsf{Open}(i);\ \mathsf{read}\ \mathsf{Commit}(i)$ for $0 \leqslant i \leqslant n+1$
- $\mathsf{LeakOpened}(i)^{\mathsf{comm}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{Opened}(i)$ for $0 \leqslant i \leqslant n+1$
- $\begin{cases} \mathsf{SumOpened}(0) := \mathsf{ret}\ \mathsf{false} \\ \mathsf{SumOpened}(j+1) := x_j \leftarrow \mathsf{SumOpened}(j);\ o_j \leftarrow \mathsf{Opened}(j);\ \mathsf{ret}\ x_j \oplus o_j & \text{for } 0 \leqslant j \leqslant n+1 \end{cases}$
- $\begin{cases} \mathsf{Out}(i) := \mathsf{read}\ \mathsf{SumOpened}(n+2) & \text{if } 0 \leqslant i \leqslant n+1 \text{ honest} \\ \mathsf{Out}(i) := \mathsf{read}\ \mathsf{Out}(i) & \text{otherwise} \end{cases}$

Proceeding further, we can keep track of the decisions to open the commits just as the simulator does, using new internal channels $\mathsf{AllOpen}(-)$:

- $\begin{cases} \mathsf{Commit}(i) := \mathsf{samp}\ \mathsf{flip} & \text{if } 0 \leqslant i \leqslant n+1 \text{ honest} \\ \mathsf{Commit}(i) := \mathsf{read}\ \mathsf{AdvCommit}^{\mathsf{adv}}_{\mathsf{party}(i)} & \text{otherwise} \end{cases}$
- $\begin{cases} \mathsf{SumCommit}(0) := \mathsf{ret}\ \mathsf{false} \\ \mathsf{SumCommit}(j+1) := x_j \leftarrow \mathsf{SumCommit}(j);\ c_j \leftarrow \mathsf{Commit}(j);\ \mathsf{ret}\ x_j \oplus c_j & \text{for } 0 \leqslant j \leqslant n+1 \end{cases}$
- $\mathsf{Committed}(i) := c_i \leftarrow \mathsf{Commit}(i);\ \mathsf{ret}\ \checkmark$ for $0 \leqslant i \leqslant n+1$

- $\mathsf{LeakCommitted}(i)^{\mathsf{comm}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{Committed}(i)$ for $0 \leqslant i \leqslant n+1$
- $\begin{cases} \mathsf{Open}(i) := x_{n+2} \leftarrow \mathsf{SumCommit}(n+2);\ \mathsf{ret}\ \checkmark & \text{if } 0 \leqslant i \leqslant n+1 \text{ honest} \\ \mathsf{Open}(i) := \mathsf{read}\ \mathsf{AdvOpen}^{\mathsf{adv}}_{\mathsf{party}(i)} & \text{otherwise} \end{cases}$
- $\begin{cases} \mathsf{AllOpen}(0) := \mathsf{ret}\ \checkmark \\ \mathsf{AllOpen}(j+1) := \_ \leftarrow \mathsf{AllOpen}(j);\ \_ \leftarrow \mathsf{Open}(j);\ \mathsf{ret}\ \checkmark & \text{for } 0 \leqslant j \leqslant n+1 \end{cases}$
- $\mathsf{Opened}(i) := \_ \leftarrow \mathsf{Open}(i);\ \mathsf{read}\ \mathsf{Commit}(i)$ for $0 \leqslant i \leqslant n+1$
- $\mathsf{LeakOpened}(i)^{\mathsf{comm}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{Opened}(i)$ for $0 \leqslant i \leqslant n+1$
- $\begin{cases} \mathsf{SumOpened}(0) := \mathsf{ret}\ \mathsf{false} \\ \mathsf{SumOpened}(j+1) := x_j \leftarrow \mathsf{SumOpened}(j);\ o_j \leftarrow \mathsf{Opened}(j);\ \mathsf{ret}\ x_j \oplus o_j & \text{for } 0 \leqslant j \leqslant n+1 \end{cases}$
- $\begin{cases} \mathsf{Out}(i) := \mathsf{read}\ \mathsf{SumOpened}(n+2) & \text{if } 0 \leqslant i \leqslant n+1 \text{ honest} \\ \mathsf{Out}(i) := \mathsf{read}\ \mathsf{Out}(i) & \text{otherwise} \end{cases}$

In the presence of these new channels, the channels $\mathsf{SumOpened}(-)$ can be simplified:

- $\begin{cases} \mathsf{Commit}(i) := \mathsf{samp}\ \mathsf{flip} & \text{if } 0 \leqslant i \leqslant n+1 \text{ honest} \\ \mathsf{Commit}(i) := \mathsf{read}\ \mathsf{AdvCommit}^{\mathsf{adv}}_{\mathsf{party}(i)} & \text{otherwise} \end{cases}$
- $\begin{cases} \mathsf{SumCommit}(0) := \mathsf{ret}\ \mathsf{false} \\ \mathsf{SumCommit}(j+1) := x_j \leftarrow \mathsf{SumCommit}(j);\ c_j \leftarrow \mathsf{Commit}(j);\ \mathsf{ret}\ x_j \oplus c_j & \text{for } 0 \leqslant j \leqslant n+1 \end{cases}$
- $\mathsf{Committed}(i) := c_i \leftarrow \mathsf{Commit}(i);\ \mathsf{ret}\ \checkmark$ for $0 \leqslant i \leqslant n+1$
- $\mathsf{LeakCommitted}(i)^{\mathsf{comm}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{Committed}(i)$ for $0 \leqslant i \leqslant n+1$
- $\begin{cases} \mathsf{Open}(i) := x_{n+2} \leftarrow \mathsf{SumCommit}(n+2);\ \mathsf{ret}\ \checkmark & \text{if } 0 \leqslant i \leqslant n+1 \text{ honest} \\ \mathsf{Open}(i) := \mathsf{read}\ \mathsf{AdvOpen}^{\mathsf{adv}}_{\mathsf{party}(i)} & \text{otherwise} \end{cases}$
- $\begin{cases} \mathsf{AllOpen}(0) := \mathsf{ret}\ \checkmark \\ \mathsf{AllOpen}(j+1) := \_ \leftarrow \mathsf{AllOpen}(j);\ \_ \leftarrow \mathsf{Open}(j);\ \mathsf{ret}\ \checkmark & \text{for } 0 \leqslant j \leqslant n+1 \end{cases}$
- $\mathsf{Opened}(i) := \_ \leftarrow \mathsf{Open}(i);\ \mathsf{read}\ \mathsf{Commit}(i)$ for $0 \leqslant i \leqslant n+1$
- $\mathsf{LeakOpened}(i)^{\mathsf{comm}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{Opened}(i)$ for $0 \leqslant i \leqslant n+1$
- $\mathsf{SumOpened}(j) := \_ \leftarrow \mathsf{AllOpen}(j);\ \mathsf{read}\ \mathsf{SumCommit}(j)$ for $0 \leqslant j \leqslant n+2$
- $\begin{cases} \mathsf{Out}(i) := \mathsf{read}\ \mathsf{SumOpened}(n+2) & \text{if } 0 \leqslant i \leqslant n+1 \text{ honest} \\ \mathsf{Out}(i) := \mathsf{read}\ \mathsf{Out}(i) & \text{otherwise} \end{cases}$

After substituting the channel $\mathsf{SumOpened}(n+2)$ into the channels $\mathsf{Out}(i)$ for $0 \leqslant i \leqslant n$ honest, the internal channels $\mathsf{SumOpened}(-)$ become unused and we can eliminate them entirely:

- $\begin{cases} \mathsf{Commit}(i) := \mathsf{samp}\ \mathsf{flip} & \text{if } 0 \leqslant i \leqslant n+1 \text{ honest} \\ \mathsf{Commit}(i) := \mathsf{read}\ \mathsf{AdvCommit}^{\mathsf{adv}}_{\mathsf{party}(i)} & \text{otherwise} \end{cases}$
- $\begin{cases} \mathsf{SumCommit}(0) := \mathsf{ret}\ \mathsf{false} \\ \mathsf{SumCommit}(j+1) := x_j \leftarrow \mathsf{SumCommit}(j);\ c_j \leftarrow \mathsf{Commit}(j);\ \mathsf{ret}\ x_j \oplus c_j & \text{for } 0 \leqslant j \leqslant n+1 \end{cases}$
- $\mathsf{Committed}(i) := c_i \leftarrow \mathsf{Commit}(i);\ \mathsf{ret}\ \checkmark$ for $0 \leqslant i \leqslant n+1$

- $\mathsf{LeakCommitted}(i)^{\mathsf{comm}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{Committed}(i)$ for $0 \leqslant i \leqslant n+1$
- $\begin{cases} \mathsf{Open}(i) := x_{n+2} \leftarrow \mathsf{SumCommit}(n+2);\ \mathsf{ret}\ \checkmark & \text{if } 0 \leqslant i \leqslant n+1 \text{ honest} \\ \mathsf{Open}(i) := \mathsf{read}\ \mathsf{AdvOpen}^{\mathsf{adv}}_{\mathsf{party}(i)} & \text{otherwise} \end{cases}$
- $\begin{cases} \mathsf{AllOpen}(0) := \mathsf{ret}\ \checkmark \\ \mathsf{AllOpen}(j+1) := \_ \leftarrow \mathsf{AllOpen}(j);\ \_ \leftarrow \mathsf{Open}(j);\ \mathsf{ret}\ \checkmark & \text{for } 0 \leqslant j \leqslant n+1 \end{cases}$
- $\mathsf{Opened}(i) := \_ \leftarrow \mathsf{Open}(i);\ \mathsf{read}\ \mathsf{Commit}(i)$ for $0 \leqslant i \leqslant n+1$
- $\mathsf{LeakOpened}(i)^{\mathsf{comm}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{Opened}(i)$ for $0 \leqslant i \leqslant n+1$
- $\begin{cases} \mathsf{Out}(i) := \_ \leftarrow \mathsf{AllOpen}(n+2);\ \mathsf{read}\ \mathsf{SumCommit}(n+2) & \text{if } 0 \leqslant i \leqslant n+1 \text{ honest} \\ \mathsf{Out}(i) := \mathsf{read}\ \mathsf{Out}(i) & \text{otherwise} \end{cases}$

This is the cleaned-up version of the real protocol. Plugging the simulator into the ideal protocol and substituting away the channels $\mathsf{LeakFlip}^{\mathsf{id}}_{\mathsf{adv}}$ and $\mathsf{Ok}^{\mathsf{adv}}_{\mathsf{id}}$ that have now become internal yields the following:

- $\mathsf{Flip} := \mathsf{samp}\ \mathsf{flip}$
- $\begin{cases} \mathsf{Commit}(i) := \mathsf{samp}\ \mathsf{flip} & \text{if } 0 \leqslant i \leqslant n \text{ honest} \\ \mathsf{Commit}(i) := \mathsf{read}\ \mathsf{AdvCommit}^{\mathsf{adv}}_{\mathsf{party}(i)} & \text{otherwise} \end{cases}$
- $\mathsf{LastCommit} := x_{n+1} \leftarrow \mathsf{SumCommit}(n+1);\ f \leftarrow \mathsf{Flip};\ \mathsf{ret}\ x_{n+1} \oplus f$
- $\begin{cases} \mathsf{SumCommit}(0) := \mathsf{ret}\ \mathsf{false} \\ \mathsf{SumCommit}(j+1) := x_j \leftarrow \mathsf{SumCommit}(j);\ c_j \leftarrow \mathsf{Commit}(j);\ \mathsf{ret}\ x_j \oplus c_j & \text{for } 0 \leqslant j \leqslant n \end{cases}$
- $\mathsf{SumCommit}(n+2) := x_{n+1} \leftarrow \mathsf{SumCommit}(n+1);\ c_{n+1} \leftarrow \mathsf{LastCommit};\ \mathsf{ret}\ x_{n+1} \oplus c_{n+1}$
- $\mathsf{Committed}(i) := c_i \leftarrow \mathsf{Commit}(i);\ \mathsf{ret}\ \checkmark$ for $0 \leqslant i \leqslant n$
- $\mathsf{Committed}(n+1) := \mathsf{ret}\ \checkmark$
- $\mathsf{LeakCommitted}(i)^{\mathsf{comm}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{Committed}(i)$ for $0 \leqslant i \leqslant n+1$
- $\begin{cases} \mathsf{Open}(i) := x_{n+2} \leftarrow \mathsf{SumCommit}(n+2);\ \mathsf{ret}\ \checkmark & \text{if } 0 \leqslant i \leqslant n+1 \text{ honest} \\ \mathsf{Open}(i) := \mathsf{read}\ \mathsf{AdvOpen}^{\mathsf{adv}}_{\mathsf{party}(i)} & \text{otherwise} \end{cases}$
- $\begin{cases} \mathsf{AllOpen}(0) := \mathsf{ret}\ \checkmark \\ \mathsf{AllOpen}(j+1) := \_ \leftarrow \mathsf{AllOpen}(j);\ \_ \leftarrow \mathsf{Open}(j);\ \mathsf{ret}\ \checkmark & \text{for } 0 \leqslant j \leqslant n+1 \end{cases}$
- $\mathsf{Opened}(i) := \_ \leftarrow \mathsf{Open}(i);\ \mathsf{read}\ \mathsf{Commit}(i)$ for $0 \leqslant i \leqslant n+1$
- $\mathsf{LeakOpened}(i)^{\mathsf{comm}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{Opened}(i)$ for $0 \leqslant i \leqslant n+1$
- $\begin{cases} \mathsf{Out}(i) := \_ \leftarrow \mathsf{AllOpen}(n+2);\ x_{n+1} \leftarrow \mathsf{SumCommit}(n+1);\ \mathsf{read}\ \mathsf{Flip} & \text{if } 0 \leqslant i \leqslant n+1 \text{ honest} \\ \mathsf{Out}(i) := \mathsf{read}\ \mathsf{Out}(i) & \text{otherwise} \end{cases}$

Substituting the channel $\mathsf{LastCommit}$ into the channel $\mathsf{SumCommit}(n+2)$ yields:

- $\mathsf{SumCommit}(n+2) := x_{n+1} \leftarrow \mathsf{SumCommit}(n+1);\ f \leftarrow \mathsf{Flip};\ \mathsf{ret}\ x_{n+1} \oplus (x_{n+1} \oplus f)$

By assumption, we can cancel out the Boolean sum:

- $\mathsf{SumCommit}(n+2) := x_{n+1} \leftarrow \mathsf{SumCommit}(n+1);\ \mathsf{read}\ \mathsf{Flip}$

In the presence of this simplified definition, we can rewrite the channels $\mathsf{Out}(-)$ to the following:

- $\begin{cases} \mathsf{Out}(i) := \_ \leftarrow \mathsf{AllOpen}(n+2);\ \mathsf{read}\ \mathsf{SumCommit}(n+2) & \text{if } 0 \leqslant i \leqslant n+1 \text{ honest} \\ \mathsf{Out}(i) := \mathsf{read}\ \mathsf{Out}(i) & \text{otherwise} \end{cases}$

The original formulation of $\mathsf{SumCommit}(n+2)$ will be more convenient for our purposes, so we rewrite it back to end up with the following protocol:

- $\mathsf{Flip} := \mathsf{samp}\ \mathsf{flip}$
- $\begin{cases} \mathsf{Commit}(i) := \mathsf{samp}\ \mathsf{flip} & \text{if } 0 \leqslant i \leqslant n \text{ honest} \\ \mathsf{Commit}(i) := \mathsf{read}\ \mathsf{AdvCommit}^{\mathsf{adv}}_{\mathsf{party}(i)} & \text{otherwise} \end{cases}$
- $\mathsf{LastCommit} := x_{n+1} \leftarrow \mathsf{SumCommit}(n+1);\ f \leftarrow \mathsf{Flip};\ x_{n+1} \oplus f$
- $\begin{cases} \mathsf{SumCommit}(0) := \mathsf{ret}\ \mathsf{false} \\ \mathsf{SumCommit}(j+1) := x_j \leftarrow \mathsf{SumCommit}(j);\ c_j \leftarrow \mathsf{Commit}(j);\ x_j \oplus c_j & \text{for } 0 \leqslant j \leqslant n \end{cases}$
- $\mathsf{SumCommit}(n+2) := x_{n+1} \leftarrow \mathsf{SumCommit}(n+1);\ c_{n+1} \leftarrow \mathsf{LastCommit};\ x_{n+1} \oplus c_{n+1}$
- $\mathsf{Committed}(i) := c_i \leftarrow \mathsf{Commit}(i);\ \mathsf{ret}\ \checkmark \text{ for } 0 \leqslant i \leqslant n$
- $\mathsf{Committed}(n+1) := \mathsf{ret}\ \checkmark$
- $\mathsf{LeakCommitted}(i)^{\mathsf{comm}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{Committed}(i) \text{ for } 0 \leqslant i \leqslant n+1$
- $\begin{cases} \mathsf{Open}(i) := x_{n+2} \leftarrow \mathsf{SumCommit}(n+2);\ \mathsf{ret}\ \checkmark & \text{if } 0 \leqslant i \leqslant n+1 \text{ honest} \\ \mathsf{Open}(i) := \mathsf{read}\ \mathsf{AdvOpen}^{\mathsf{adv}}_{\mathsf{party}(i)} & \text{otherwise} \end{cases}$
- $\begin{cases} \mathsf{AllOpen}(0) := \mathsf{ret}\ \checkmark \\ \mathsf{AllOpen}(j+1) := \_ \leftarrow \mathsf{AllOpen}(j);\ \_ \leftarrow \mathsf{Open}(j);\ \mathsf{ret}\ \checkmark & \text{for } 0 \leqslant j \leqslant n+1 \end{cases}$
- $\mathsf{Opened}(i) := \_ \leftarrow \mathsf{Open}(i);\ \mathsf{read}\ \mathsf{Commit}(i) \text{ for } 0 \leqslant i \leqslant n+1$
- $\mathsf{LeakOpened}(i)^{\mathsf{comm}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{Opened}(i) \text{ for } 0 \leqslant i \leqslant n+1$
- $\begin{cases} \mathsf{Out}(i) := \_ \leftarrow \mathsf{AllOpen}(n+2);\ \mathsf{read}\ \mathsf{SumCommit}(n+2) & \text{if } 0 \leqslant i \leqslant n+1 \text{ honest} \\ \mathsf{Out}(i) := \mathsf{read}\ \mathsf{Out}(i) & \text{otherwise} \end{cases}$

The channel Flip now only occurs in the channel LastCommit, so we can fold it in:

- $\begin{cases} \mathsf{Commit}(i) := \mathsf{samp}\ \mathsf{flip} & \text{if } 0 \leqslant i \leqslant n \text{ honest} \\ \mathsf{Commit}(i) := \mathsf{read}\ \mathsf{AdvCommit}^{\mathsf{adv}}_{\mathsf{party}(i)} & \text{otherwise} \end{cases}$
- $\mathsf{LastCommit} := x_{n+1} \leftarrow \mathsf{SumCommit}(n+1);\ f \leftarrow \mathsf{samp}\ \mathsf{flip};\ x_{n+1} \oplus f$
- $\begin{cases} \mathsf{SumCommit}(0) := \mathsf{ret}\ \mathsf{false} \\ \mathsf{SumCommit}(j+1) := x_j \leftarrow \mathsf{SumCommit}(j);\ c_j \leftarrow \mathsf{Commit}(j);\ x_j \oplus c_j & \text{for } 0 \leqslant j \leqslant n \end{cases}$
- $\mathsf{SumCommit}(n+2) := x_{n+1} \leftarrow \mathsf{SumCommit}(n+1);\ c_{n+1} \leftarrow \mathsf{LastCommit};\ x_{n+1} \oplus c_{n+1}$
- $\mathsf{Committed}(i) := c_i \leftarrow \mathsf{Commit}(i);\ \mathsf{ret}\ \checkmark \text{ for } 0 \leqslant i \leqslant n$
- $\mathsf{Committed}(n+1) := \mathsf{ret}\ \checkmark$
- $\mathsf{LeakCommitted}(i)^{\mathsf{comm}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{Committed}(i) \text{ for } 0 \leqslant i \leqslant n+1$
- $\begin{cases} \mathsf{Open}(i) := x_{n+2} \leftarrow \mathsf{SumCommit}(n+2);\ \mathsf{ret}\ \checkmark & \text{if } 0 \leqslant i \leqslant n+1 \text{ honest} \\ \mathsf{Open}(i) := \mathsf{read}\ \mathsf{AdvOpen}^{\mathsf{adv}}_{\mathsf{party}(i)} & \text{otherwise} \end{cases}$

- $\begin{cases} \mathsf{AllOpen}(0) := \text{ret } ✓ \\ \mathsf{AllOpen}(j+1) := \_ \leftarrow \mathsf{AllOpen}(j);\ \_ \leftarrow \mathsf{Open}(j);\ \text{ret } ✓ & \text{for } 0 \leqslant j \leqslant n+1 \end{cases}$
- $\mathsf{Opened}(i) := \_ \leftarrow \mathsf{Open}(i);\ \text{read } \mathsf{Commit}(i) \text{ for } 0 \leqslant i \leqslant n+1$
- $\mathsf{LeakOpened}(i)^{\mathsf{comm}}_{\mathsf{adv}} := \text{read } \mathsf{Opened}(i) \text{ for } 0 \leqslant i \leqslant n+1$
- $\begin{cases} \mathsf{Out}(i) := \_ \leftarrow \mathsf{AllOpen}(n+2);\ \text{read } \mathsf{SumCommit}(n+2) & \text{if } 0 \leqslant i \leqslant n+1 \text{ honest} \\ \mathsf{Out}(i) := \text{read } \mathsf{Out}(i) & \text{otherwise} \end{cases}$

By assumption, the distribution flip is invariant under taking a Boolean sum with a fixed bit:

- $\mathsf{LastCommit} := x_{n+1} \leftarrow \mathsf{SumCommit}(n+1);\ \text{samp flip}$

We can unfold the sampling back into a new internal channel $\mathsf{Commit}(n+1)$:

- $\begin{cases} \mathsf{Commit}(i) := \text{samp flip} & \text{if } 0 \leqslant i \leqslant n+1 \text{ honest} \\ \mathsf{Commit}(i) := \text{read } \mathsf{AdvCommit}^{\mathsf{adv}}_{\mathsf{party}(i)} & \text{otherwise} \end{cases}$
- $\mathsf{LastCommit} := x_{n+1} \leftarrow \mathsf{SumCommit}(n);\ \text{read } \mathsf{Commit}(n+1)$
- $\begin{cases} \mathsf{SumCommit}(0) := \text{ret false} \\ \mathsf{SumCommit}(j+1) := x_j \leftarrow \mathsf{SumCommit}(j);\ c_j \leftarrow \mathsf{Commit}(j);\ x_j \oplus c_j & \text{for } 0 \leqslant j \leqslant n \end{cases}$
- $\mathsf{SumCommit}(n+2) := x_{n+1} \leftarrow \mathsf{SumCommit}(n+1);\ c_{n+1} \leftarrow \mathsf{LastCommit};\ x_{n+1} \oplus c_{n+1}$
- $\mathsf{Committed}(i) := c_i \leftarrow \mathsf{Commit}(i);\ \text{ret } ✓ \text{ for } 0 \leqslant i \leqslant n$
- $\mathsf{Committed}(n+1) := \text{ret } ✓$
- $\mathsf{LeakCommitted}(i)^{\mathsf{comm}}_{\mathsf{adv}} := \text{read } \mathsf{Committed}(i) \text{ for } 0 \leqslant i \leqslant n+1$
- $\begin{cases} \mathsf{Open}(i) := x_{n+2} \leftarrow \mathsf{SumCommit}(n+2);\ \text{ret } ✓ & \text{if } 0 \leqslant i \leqslant n+1 \text{ honest} \\ \mathsf{Open}(i) := \text{read } \mathsf{AdvOpen}^{\mathsf{adv}}_{\mathsf{party}(i)} & \text{otherwise} \end{cases}$
- $\begin{cases} \mathsf{AllOpen}(0) := \text{ret } ✓ \\ \mathsf{AllOpen}(j+1) := \_ \leftarrow \mathsf{AllOpen}(j);\ \_ \leftarrow \mathsf{Open}(j);\ \text{ret } ✓ & \text{for } 0 \leqslant j \leqslant n+1 \end{cases}$
- $\mathsf{Opened}(i) := \_ \leftarrow \mathsf{Open}(i);\ \text{read } \mathsf{Commit}(i) \text{ for } 0 \leqslant i \leqslant n+1$
- $\mathsf{LeakOpened}(i)^{\mathsf{comm}}_{\mathsf{adv}} := \text{read } \mathsf{Opened}(i) \text{ for } 0 \leqslant i \leqslant n+1$
- $\begin{cases} \mathsf{Out}(i) := \_ \leftarrow \mathsf{AllOpen}(n+2);\ \text{read } \mathsf{SumCommit}(n+2) & \text{if } 0 \leqslant i \leqslant n+1 \text{ honest} \\ \mathsf{Out}(i) := \text{read } \mathsf{Out}(i) & \text{otherwise} \end{cases}$

The internal channel LastCommit can now be substituted away:

- $\begin{cases} \mathsf{Commit}(i) := \text{samp flip} & \text{if } 0 \leqslant i \leqslant n+1 \text{ honest} \\ \mathsf{Commit}(i) := \text{read } \mathsf{AdvCommit}^{\mathsf{adv}}_{\mathsf{party}(i)} & \text{otherwise} \end{cases}$
- $\begin{cases} \mathsf{SumCommit}(0) := \text{ret false} \\ \mathsf{SumCommit}(j+1) := x_j \leftarrow \mathsf{SumCommit}(j);\ c_j \leftarrow \mathsf{Commit}(j);\ x_j \oplus c_j & \text{for } 0 \leqslant j \leqslant n+1 \end{cases}$
- $\mathsf{Committed}(i) := c_i \leftarrow \mathsf{Commit}(i);\ \text{ret } ✓ \text{ for } 0 \leqslant i \leqslant n$
- $\mathsf{Committed}(n+1) := \text{ret } ✓$
- $\mathsf{LeakCommitted}(i)^{\mathsf{comm}}_{\mathsf{adv}} := \text{read } \mathsf{Committed}(i) \text{ for } 0 \leqslant i \leqslant n+1$

- $\begin{cases} \mathsf{Open}(i) := x_{n+2} \leftarrow \mathsf{SumCommit}(n+2);\ \mathsf{ret}\ \checkmark & \text{if } 0 \leqslant i \leqslant n+1 \text{ honest} \\ \mathsf{Open}(i) := \mathsf{read}\ \mathsf{AdvOpen}^{\mathsf{adv}}_{\mathsf{party}(i)} & \text{otherwise} \end{cases}$
- $\begin{cases} \mathsf{AllOpen}(0) := \mathsf{ret}\ \checkmark \\ \mathsf{AllOpen}(j+1) := \_ \leftarrow \mathsf{AllOpen}(j);\ \_ \leftarrow \mathsf{Open}(j);\ \mathsf{ret}\ \checkmark & \text{for } 0 \leqslant j \leqslant n+1 \end{cases}$
- $\mathsf{Opened}(i) := \_ \leftarrow \mathsf{Open}(i);\ \mathsf{read}\ \mathsf{Commit}(i) \text{ for } 0 \leqslant i \leqslant n+1$
- $\mathsf{LeakOpened}(i)^{\mathsf{comm}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{Opened}(i) \text{ for } 0 \leqslant i \leqslant n+1$
- $\begin{cases} \mathsf{Out}(i) := \_ \leftarrow \mathsf{AllOpen}(n+2);\ \mathsf{read}\ \mathsf{SumCommit}(n+2) & \text{if } 0 \leqslant i \leqslant n+1 \text{ honest} \\ \mathsf{Out}(i) := \mathsf{read}\ \mathsf{Out}(i) & \text{otherwise} \end{cases}$

Finally, we rewrite the channel $\mathsf{Committed}(n+1)$ to include a gratuitous dependency on $\mathsf{Commit}(n+1)$:

- $\begin{cases} \mathsf{Commit}(i) := \mathsf{samp}\ \mathsf{flip} & \text{if } 0 \leqslant i \leqslant n+1 \text{ honest} \\ \mathsf{Commit}(i) := \mathsf{read}\ \mathsf{AdvCommit}^{\mathsf{adv}}_{\mathsf{party}(i)} & \text{otherwise} \end{cases}$
- $\begin{cases} \mathsf{SumCommit}(0) := \mathsf{ret}\ \mathsf{false} \\ \mathsf{SumCommit}(j+1) := x_j \leftarrow \mathsf{SumCommit}(j);\ c_j \leftarrow \mathsf{Commit}(j);\ x_j \oplus c_j & \text{for } 0 \leqslant j \leqslant n+1 \end{cases}$
- $\mathsf{Committed}(i) := c_i \leftarrow \mathsf{Commit}(i);\ \mathsf{ret}\ \checkmark \text{ for } 0 \leqslant i \leqslant n+1$
- $\mathsf{LeakCommitted}(i)^{\mathsf{comm}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{Committed}(i) \text{ for } 0 \leqslant i \leqslant n+1$
- $\begin{cases} \mathsf{Open}(i) := x_{n+2} \leftarrow \mathsf{SumCommit}(n+2);\ \mathsf{ret}\ \checkmark & \text{if } 0 \leqslant i \leqslant n+1 \text{ honest} \\ \mathsf{Open}(i) := \mathsf{read}\ \mathsf{AdvOpen}^{\mathsf{adv}}_{\mathsf{party}(i)} & \text{otherwise} \end{cases}$
- $\begin{cases} \mathsf{AllOpen}(0) := \mathsf{ret}\ \checkmark \\ \mathsf{AllOpen}(j+1) := \_ \leftarrow \mathsf{AllOpen}(j);\ \_ \leftarrow \mathsf{Open}(j);\ \mathsf{ret}\ \checkmark & \text{for } 0 \leqslant j \leqslant n+1 \end{cases}$
- $\mathsf{Opened}(i) := \_ \leftarrow \mathsf{Open}(i);\ \mathsf{read}\ \mathsf{Commit}(i) \text{ for } 0 \leqslant i \leqslant n+1$
- $\mathsf{LeakOpened}(i)^{\mathsf{comm}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{Opened}(i) \text{ for } 0 \leqslant i \leqslant n+1$
- $\begin{cases} \mathsf{Out}(i) := \_ \leftarrow \mathsf{AllOpen}(n+2);\ \mathsf{read}\ \mathsf{SumCommit}(n+2) & \text{if } 0 \leqslant i \leqslant n+1 \text{ honest} \\ \mathsf{Out}(i) := \mathsf{read}\ \mathsf{Out}(i) & \text{otherwise} \end{cases}$

But this is precisely the cleaned-up version of the real protocol.

9 Two-Party GMW Protocol

In the two-party GMW protocol, Alice and Bob jointly compute the value of a given Boolean circuit built out of *xor*-, *and*-, and *not* gates. The inputs to the circuit are divided between Alice and Bob, and neither party has access to the inputs of the other. For each gate, Alice and Bob maintain their respective *shares* of the actual value v computed by the gate, with Alice's share computed only from the information available to Alice, and analogously for Bob. The respective shares for Alice and Bob sum up to v. We prove the protocol secure against a semi-honest attacker in the case when Alice is corrupt and Bob is honest.

Formally, we assume a coin-flip distribution $\mathsf{flip} : 1 \twoheadrightarrow \mathsf{Bool}$; a Boolean sum function $\oplus : \mathsf{Bool} \times \mathsf{Bool} \to \mathsf{Bool}$, where we write $x \oplus y$ in place of $\oplus\ (x, y)$; a Boolean multiplication function $* : \mathsf{Bool} \times \mathsf{Bool} \to \mathsf{Bool}$, where we write $x * y$ in place of $*\ (x, y)$; and a Boolean negation function $\neg : \mathsf{Bool} \to \mathsf{Bool}$, where we write $\neg x$ in place of $\neg\ x$.

We represent Boolean circuits using the syntax below, where we assume an ambient (finite) set I of inputs. Starting from the empty circuit ϵ we add one gate at a time: an *input* gate allows us to plug into a specified input i; a *not* gate negates the value carried on wire k; an *xor* gate computes the Boolean sum of the two values carried on wires k and l; and an *and* gate does the same for Boolean product.

Inputs	$i \in I$		
Wires	$k, l \in \mathbb{N}$		
Circuits	C	$::=$	$\epsilon \mid C;\ \mathit{input\text{-}gate}(i) \mid C;\ \mathit{not\text{-}gate}(k) \mid C;\ \mathit{xor\text{-}gate}(k,l) \mid C;\ \mathit{and\text{-}gate}(k,l)$

A circuit C with $n \in \mathbb{N}$ wires is considered well-formed if each logical gate combines previously defined wires only:

$$\frac{}{\epsilon\ \mathsf{circuit}(0)} \qquad \frac{C\ \mathsf{circuit}(n)}{C;\ \mathit{input\text{-}gate}(i)\ \mathsf{circuit}(n+1)} \qquad \frac{C\ \mathsf{circuit}(n) \qquad k < n}{C;\ \mathit{not\text{-}gate}(k)\ \mathsf{circuit}(n+1)}$$

$$\frac{C\ \mathsf{circuit}(n) \qquad k < n \qquad l < n}{C;\ \mathit{xor\text{-}gate}(k,l)\ \mathsf{circuit}(n+1)} \qquad \frac{C\ \mathsf{circuit}(n) \qquad k < n \qquad l < n}{C;\ \mathit{and\text{-}gate}(k,l)\ \mathsf{circuit}(n+1)}$$

For our specific setup, we assume Alice has $N \geqslant 0$ inputs labeled $\{0, \ldots, N-1\}$, and Bob has $M \geqslant 0$ inputs labeled $\{0, \ldots, M-1\}$. The set of inputs to our ambient Boolean circuit C (with, let us say, K wires) is therefore $M + N$. We furthermore assume that a subset of the wires $\{0, \ldots, K-1\}$ is designated as outputs.

9.1 The Assumptions

At the expression level, we assume that the Boolean sum and product operations are commutative and associative:

- $x : \mathsf{Bool}, y : \mathsf{Bool} \vdash x \oplus y = y \oplus x : \mathsf{Bool}$,
- $x : \mathsf{Bool}, y : \mathsf{Bool} \vdash x * y = y * x : \mathsf{Bool}$,
- $x : \mathsf{Bool}, y : \mathsf{Bool}, z : \mathsf{Bool} \vdash (x \oplus y) \oplus z = x \oplus (y \oplus z) : \mathsf{Bool}$, and
- $x : \mathsf{Bool}, y : \mathsf{Bool}, z : \mathsf{Bool} \vdash (x * y) * z = x * (y * z) : \mathsf{Bool}$.

Furthermore, Boolean multiplication distributes over Boolean sum:

- $x : \mathsf{Bool}, y : \mathsf{Bool}, z : \mathsf{Bool} \vdash (x \oplus y) * z = (x * y) \oplus (y * z) : \mathsf{Bool}$.

Summing up a Boolean with itself yields false and summing up a Boolean with false yields the original Boolean:

- $x : \mathsf{Bool} \vdash x \oplus x = \mathsf{false} : \mathsf{Bool}$, and
- $x : \mathsf{Bool} \vdash x \oplus \mathsf{false} = x : \mathsf{Bool}$.

Negating a Boolean equals summing it up with true:

- $x : \mathsf{Bool} \vdash x \oplus \mathsf{true} = \neg x : \mathsf{Bool}$.

Finally, multiplying a Boolean with false or true yields false or the original Boolean, respectively:

- $x : \mathsf{Bool} \vdash x * \mathsf{false} = \mathsf{false} : \mathsf{Bool}$, and
- $x : \mathsf{Bool} \vdash x * \mathsf{true} = x : \mathsf{Bool}$.

At the distribution level, we assume that the distribution flip on Booleans is invariant under the operation of Boolean sum with a fixed Boolean (as is indeed the case when flip is uniform):

- $x : \mathsf{Bool} \vdash \big(y \leftarrow \mathsf{flip};\ 1[x \oplus y]\big) = \mathsf{flip} : \mathsf{Bool}$

9.2 The Ideal Protocol

The leakage from the ideal functionality includes the timing information for Bob's inputs plus the value of Alice's inputs (since she is semi-honest):

- $\mathsf{In}(\mathsf{A}, i)^{\mathsf{id}}_{\mathsf{adv}} := \mathsf{In}(\mathsf{A}, i)$ for $0 \leqslant i < N$
- $\mathsf{InRcvd}(\mathsf{B}, i)^{\mathsf{id}}_{\mathsf{adv}} := x \leftarrow \mathsf{In}(\mathsf{B}, i);\ \mathsf{ret}\ \checkmark$ for $0 \leqslant i < M$

In the inductive phase, the functionality computes the value carried by each wire $k < K$ of the ambient circuit by induction on the circuit:

- $\mathsf{Wires}(\epsilon, 0)$ is the protocol $\mathsf{0}$
- $\mathsf{Wires}\big(C;\ \mathit{input\text{-}gate}(i), K+1\big)$ is the composition of the protocol $\mathsf{Wires}(C, K)$ with the single-reaction protocol
  - $\begin{cases} \mathsf{Wire}(K) := \mathsf{read}\ \mathsf{In}(\mathsf{A}, i) & \text{if } i \text{ is an input of Alice} \\ \mathsf{Wire}(K) := \mathsf{read}\ \mathsf{In}(\mathsf{B}, i) & \text{if } i \text{ is an input of Bob} \end{cases}$
- $\mathsf{Wires}\big(C;\ \mathit{not\text{-}gate}(k), K+1\big)$ is the composition of the protocol $\mathsf{Wires}(C, K)$ with the single-reaction protocol
  - $\mathsf{Wire}(K) := x \leftarrow \mathsf{Wire}(k);\ \mathsf{ret}\ \neg x$
- $\mathsf{Wires}\big(C;\ \mathit{xor\text{-}gate}(k, l), K+1\big)$ is the composition of the protocol $\mathsf{Wires}(C, K)$ with the single-reaction protocol
  - $\mathsf{Wire}(K) := x \leftarrow \mathsf{Wire}(k);\ y \leftarrow \mathsf{Wire}(l);\ \mathsf{ret}\ x \oplus y$
- $\mathsf{Wires}\big(C;\ \mathit{and\text{-}gate}(k, l), K+1\big)$ is the composition of the protocol $\mathsf{Wires}(C, K)$ with the single-reaction protocol
  - $\mathsf{Wire}(K) := x \leftarrow \mathsf{Wire}(k);\ y \leftarrow \mathsf{Wire}(l);\ \mathsf{ret}\ x * y$

After performing the above computation, the ideal functionality outputs the computed value for each wire marked as an output, and leaks the outputs to the adversary on behalf of Alice:

- $\begin{cases} \mathsf{Out}(\mathsf{A}, k) := \mathsf{read}\ \mathsf{Wire}(k) & \text{if wire } 0 \leqslant k < K \text{ is output} \\ \mathsf{Out}(\mathsf{A}, k) := \mathsf{read}\ \mathsf{Out}(\mathsf{A}, k) & \text{otherwise} \end{cases}$
- $\begin{cases} \mathsf{Out}(\mathsf{B}, k) := \mathsf{read}\ \mathsf{Wire}(k) & \text{if wire } 0 \leqslant k < K \text{ is output} \\ \mathsf{Out}(\mathsf{B}, k) := \mathsf{read}\ \mathsf{Out}(\mathsf{B}, k) & \text{otherwise} \end{cases}$
- $\mathsf{Out}(\mathsf{A}, k)^{\mathsf{id}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{Out}(\mathsf{A}, k)$ for $0 \leqslant k < K$

Finally, the channels $\mathsf{Wire}(-)$ coming from the inductive protocol $\mathsf{Wires}(C, K)$ are designated as internal.

9.3 The Real Protocol

The real protocol consists of the two parties, plus an instance of an ideal 1-Out-Of-4 Oblivious Transfer (OT) functionality for each gate, with Alice the sender and Bob the receiver. The code for each party is separated into three parts: in the initial phase, each party computes and distributes everyone's shares for each of its inputs. In the inductive phase, each party computes their share of each wire by induction on the ambient circuit. At last, in the final phase, Alice and Bob send their shares of each output wire to each other and add them up to compute the result.

9.3.1 Alice: The Initial Phase

Alice generates her own shares randomly and sends Bob the sum of her share and the actual value of the input. Alice's share of Bob's input is sent to her by Bob. As Alice is semi-honest, she leaks the value of her inputs. After storing each share, Alice leaks it to the adversary. Furthermore, when sending Bob's share of her input to Bob, she simultaneously leaks it to the adversary, and after receiving her share of Bob's input from Bob, she forwards it along to the adversary.

- $\mathsf{In}(\mathsf{A}, i)^{\mathsf{A}}_{\mathsf{adv}} := \text{read } \mathsf{In}(\mathsf{A}, i) \text{ for } 0 \leqslant i < N$
- $\mathsf{InputShare}(\mathsf{A}, \mathsf{A}, i) := x \leftarrow \mathsf{In}(\mathsf{A}, i);\ \mathsf{samp\ flip} \text{ for } 0 \leqslant i < N$
- $\mathsf{InputShare}(\mathsf{A}, \mathsf{B}, i) := \text{read } \mathsf{SendInputShare}(\mathsf{A}, \mathsf{B}, i) \text{ for } 0 \leqslant i < M$
- $\mathsf{InputShare}(\mathsf{A}, \mathsf{A}, i)^{\mathsf{A}}_{\mathsf{adv}} := \text{read } \mathsf{InputShare}(\mathsf{A}, \mathsf{A}, i) \text{ for } 0 \leqslant i < N$
- $\mathsf{InputShare}(\mathsf{A}, \mathsf{B}, i)^{\mathsf{A}}_{\mathsf{adv}} := \text{read } \mathsf{InputShare}(\mathsf{A}, \mathsf{B}, i) \text{ for } 0 \leqslant i < M$
- $\mathsf{SendInputShare}(\mathsf{B}, \mathsf{A}, i) := x \leftarrow \mathsf{In}(\mathsf{A}, i);\ x_A \leftarrow \mathsf{InputShare}(\mathsf{A}, \mathsf{A}, i);\ \mathsf{ret}\ x \oplus x_A \text{ for } 0 \leqslant i < N$
- $\mathsf{SendInputShare}(\mathsf{B}, \mathsf{A}, i)^{\mathsf{A}}_{\mathsf{adv}} := \text{read } \mathsf{SendInputShare}(\mathsf{B}, \mathsf{A}, i) \text{ for } 0 \leqslant i < N$
- $\mathsf{SendInputShare}(\mathsf{A}, \mathsf{B}, i)^{\mathsf{A}}_{\mathsf{adv}} := \text{read } \mathsf{SendInputShare}(\mathsf{A}, \mathsf{B}, i) \text{ for } 0 \leqslant i < M$

9.3.2 Bob: The Initial Phase

Bob generates Alice's shares randomly, and sets his own share to be the sum of Alice's share and the actual value of the input. His share of Alice's input is sent to him by Alice. Since Bob is honest, the only leakage from him is the timing of his inputs.

- $\mathsf{InRcvd}(\mathsf{B}, i)^{\mathsf{B}}_{\mathsf{adv}} := x \leftarrow \mathsf{In}(\mathsf{B}, i);\ \mathsf{ret}\ \checkmark \text{ for } 0 \leqslant i < M$
- $\mathsf{InputShare}(\mathsf{B}, \mathsf{A}, i) := \text{read } \mathsf{SendInputShare}(\mathsf{B}, \mathsf{A}, i) \text{ for } 0 \leqslant i < N$
- $\mathsf{InputShare}(\mathsf{B}, \mathsf{B}, i) := x \leftarrow \mathsf{In}(\mathsf{B}, i);\ x_A \leftarrow \mathsf{SendInputShare}(\mathsf{A}, \mathsf{B}, i);\ \mathsf{ret}\ x \oplus x_A \text{ for } 0 \leqslant i < M$
- $\mathsf{SendInputShare}(\mathsf{A}, \mathsf{B}, i) := x \leftarrow \mathsf{In}(\mathsf{B}, i);\ \mathsf{samp\ flip} \text{ for } 0 \leqslant i < M$

9.3.3 Alice: The Inductive Phase

In the case of an *input* gate, Alice uses her corresponding input share from the initial stage. In the case of a *not* gate, she simply copies her share x_A of the incoming wire. If the gate is an *xor* gate, the resulting share is the sum $x_A \oplus y_A$ of the shares of the incoming two wires. The case of an *and* gate is the most complex. The sum of Alice's and Bob's respective shares must equal $(x_A \oplus x_B) * (y_A \oplus y_B)$, where x_A, y_A and x_B, y_B are the respective shares of Alice and Bob on the incoming two wires. We have

$$(x_A \oplus x_B) * (y_A \oplus y_B) = (x_A * y_A) \oplus (x_A * y_B) \oplus (x_B * y_A) \oplus (x_B * y_B)$$

and the quantity $(x_A * y_B) \oplus (x_B * y_A)$ cannot be directly computed by either Alice or Bob, as neither of them has access to the shares of the other. Instead, Alice and Bob engage in an idealized 1-Out-Of-4 OT exchange: there are four possible combinations of values that x_B, y_B can take, and Alice computes the value of $(x_A * y_B) \oplus (x_B * y_A)$ for each. This offers Bob four messages to choose from, and he selects the one corresponding to the actual values of x_B, y_B. A small caveat: in the exchange as described above, Bob would still be able to infer the value of Alice's shares in certain cases: *e.g.*, if $x_B = 0$ and $y_B = 1$, Bob gets the share x_A as the result of the exchange. To prevent this, Alice encodes her messages by xor-ing them with a random bit b that only she knows.

To stay consistent throughout the cases, we set up our protocol so that each gate induces the same set of outputs, even though some of these channels may not be relevant to the specific case in question. For example, at each gate Alice will construct a channel $\mathsf{OTBit}(\mathsf{A}, \mathsf{B}, -)$ to *potentially* store the aforementioned random bit b, in the case that the gate happens to be an *and* gate and she needs to engage in a 1-Out-Of-4 OT exchange with Bob. Similar remarks apply to all the OT channels, which are likewise only relevant for Boolean multiplication. Irrelevant channels will simply diverge, which makes them effectively nonexistent.

- $\mathsf{A}(\epsilon, 0)$ is the protocol $\mathsf{0}$
- $\mathsf{A}\big(C;\ \textit{input-gate}(i), K+1\big)$ is the composition of the protocol $\mathsf{A}(C, K)$ with the protocol we now describe. As mentioned before, Alice maintains a divergent Boolean channel $\mathsf{OTBit}(\mathsf{A}, \mathsf{B}, K)$, accompanied by the corresponding vacuous leakage:
  - $\mathsf{OTBit}(\mathsf{A}, \mathsf{B}, K) := \mathsf{read}\ \mathsf{OTBit}(\mathsf{A}, \mathsf{B}, K)$
  - $\mathsf{OTBit}(\mathsf{A}, \mathsf{B}, K)^{\mathsf{A}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{OTBit}(\mathsf{A}, \mathsf{B}, K)^{\mathsf{A}}_{\mathsf{adv}}$

  Alice's share is the input share as determined in the initial part of the protocol:

  - $\begin{cases} \mathsf{Share}(\mathsf{A}, K) := \mathsf{read}\ \mathsf{InputShare}(\mathsf{A}, \mathsf{A}, i) & \text{if } i \text{ is an input of Alice} \\ \mathsf{Share}(\mathsf{A}, K) := \mathsf{read}\ \mathsf{InputShare}(\mathsf{A}, \mathsf{B}, i) & \text{if } i \text{ is an input of Bob} \end{cases}$
  - $\mathsf{Share}(\mathsf{A}, K)^{\mathsf{A}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{Share}(\mathsf{A}, K)$

  The 1-Out-Of-4 OT exchange with Bob is vacuous:

  - $\mathsf{OTMsg}(\mathsf{A}, \mathsf{B}, K, 0) := \mathsf{read}\ \mathsf{OTMsg}(\mathsf{A}, \mathsf{B}, K, 0)$
  - $\mathsf{OTMsg}(\mathsf{A}, \mathsf{B}, K, 1) := \mathsf{read}\ \mathsf{OTMsg}(\mathsf{A}, \mathsf{B}, K, 1)$
  - $\mathsf{OTMsg}(\mathsf{A}, \mathsf{B}, K, 2) := \mathsf{read}\ \mathsf{OTMsg}(\mathsf{A}, \mathsf{B}, K, 2)$
  - $\mathsf{OTMsg}(\mathsf{A}, \mathsf{B}, K, 3) := \mathsf{read}\ \mathsf{OTMsg}(\mathsf{A}, \mathsf{B}, K, 3)$
- $\mathsf{A}\big(C;\ \textit{not-gate}(k), K+1\big)$ is the composition of the protocol $\mathsf{A}(C, K)$ with the following protocol, largely analogous to the previous case. Alice again maintains a divergent Boolean channel $\mathsf{OTBit}(\mathsf{A}, \mathsf{B}, K)$, accompanied by the corresponding vacuous leakage:
  - $\mathsf{OTBit}(\mathsf{A}, \mathsf{B}, K) := \mathsf{read}\ \mathsf{OTBit}(\mathsf{A}, \mathsf{B}, K)$
  - $\mathsf{OTBit}(\mathsf{A}, \mathsf{B}, K)^{\mathsf{A}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{OTBit}(\mathsf{A}, \mathsf{B}, K)^{\mathsf{A}}_{\mathsf{adv}}$

  Alice's share is the share on wire k:

  - $\mathsf{Share}(\mathsf{A}, K) := \mathsf{read}\ \mathsf{Share}(\mathsf{A}, k)$
  - $\mathsf{Share}(\mathsf{A}, K)^{\mathsf{A}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{Share}(\mathsf{A}, K)$

  As before, she engages in a vacuous 1-Out-Of-4 OT exchange with Bob:

  - $\mathsf{OTMsg}(\mathsf{A}, \mathsf{B}, K, 0) := \mathsf{read}\ \mathsf{OTMsg}(\mathsf{A}, \mathsf{B}, K, 0)$
  - $\mathsf{OTMsg}(\mathsf{A}, \mathsf{B}, K, 1) := \mathsf{read}\ \mathsf{OTMsg}(\mathsf{A}, \mathsf{B}, K, 1)$
  - $\mathsf{OTMsg}(\mathsf{A}, \mathsf{B}, K, 2) := \mathsf{read}\ \mathsf{OTMsg}(\mathsf{A}, \mathsf{B}, K, 2)$
  - $\mathsf{OTMsg}(\mathsf{A}, \mathsf{B}, K, 3) := \mathsf{read}\ \mathsf{OTMsg}(\mathsf{A}, \mathsf{B}, K, 3)$
- $\mathsf{A}\big(C;\ \textit{xor-gate}(k, l), K+1\big)$ is the composition of the protocol $\mathsf{A}(C, K)$ with the following protocol, analogous to the previous two cases. As before, Alice maintains a divergent Boolean channel $\mathsf{OTBit}(\mathsf{A}, \mathsf{B}, K)$, accompanied by the corresponding vacuous leakage:
  - $\mathsf{OTBit}(\mathsf{A}, \mathsf{B}, K) := \mathsf{read}\ \mathsf{OTBit}(\mathsf{A}, \mathsf{B}, K)$
  - $\mathsf{OTBit}(\mathsf{A}, \mathsf{B}, K)^{\mathsf{A}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{OTBit}(\mathsf{A}, \mathsf{B}, K)^{\mathsf{A}}_{\mathsf{adv}}$

  Alice's share is the sum of shares on wires k and l:

  - $\mathsf{Share}(\mathsf{A}, K) := x_A \leftarrow \mathsf{Share}(\mathsf{A}, k);\ y_A \leftarrow \mathsf{Share}(\mathsf{A}, l);\ \mathsf{ret}\ x_A \oplus y_A$
  - $\mathsf{Share}(\mathsf{A}, K)^{\mathsf{A}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{Share}(\mathsf{A}, K)$

  As before, Alice engages in a vacuous 1-Out-Of-4 OT exchange with Bob:

  - $\mathsf{OTMsg}(\mathsf{A}, \mathsf{B}, K, 0) := \mathsf{read}\ \mathsf{OTMsg}(\mathsf{A}, \mathsf{B}, K, 0)$

- $\mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,1) := \mathsf{read}\ \mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,1)$
- $\mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,2) := \mathsf{read}\ \mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,2)$
- $\mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,3) := \mathsf{read}\ \mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,3)$

- $\mathsf{A}\big(C;\ \textit{and-gate}(k,l),K+1\big)$ is the composition of the protocol $\mathsf{A}(C,K)$ with the protocol we now describe. First, Alice uniformly generates a random bit for the OT exchange with Bob:

  - $\mathsf{OTBit}(\mathsf{A},\mathsf{B},K) := x_A \leftarrow \mathsf{Share}(\mathsf{A},k);\ y_A \leftarrow \mathsf{Share}(\mathsf{A},l);\ \mathsf{samp\ flip}$
  - $\mathsf{OTBit}(\mathsf{A},\mathsf{B},K)^{\mathsf{A}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{OTBit}(\mathsf{A},\mathsf{B},K)$

  Her share is the encoded product of shares on wires k and l, where the encoding is the summation with the above random bit:

  - $\mathsf{Share}(\mathsf{A},K) := b_A \leftarrow \mathsf{OTBit}(\mathsf{A},\mathsf{B},K);\ x_A \leftarrow \mathsf{Share}(\mathsf{A},k);\ y_A \leftarrow \mathsf{Share}(\mathsf{A},l);\ \mathsf{ret}\ (x_A * y_A) \oplus b_A$
  - $\mathsf{Share}(\mathsf{A},K)^{\mathsf{A}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{Share}(\mathsf{A},K)$

  Alice's 1-Out-Of-4 OT exchange with Bob is now proper:

  - $\mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,0) := b_A \leftarrow \mathsf{OTBit}(\mathsf{A},\mathsf{B},K);\ x_A \leftarrow \mathsf{Share}(\mathsf{A},k);\ y_A \leftarrow \mathsf{Share}(\mathsf{A},l);\ \mathsf{ret}\ b_A$
  - $\mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,1) := b_A \leftarrow \mathsf{OTBit}(\mathsf{A},\mathsf{B},K);\ x_A \leftarrow \mathsf{Share}(\mathsf{A},k);\ y_A \leftarrow \mathsf{Share}(\mathsf{A},l);\ \mathsf{ret}\ b_A \oplus x_A$
  - $\mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,2) := b_A \leftarrow \mathsf{OTBit}(\mathsf{A},\mathsf{B},K);\ x_A \leftarrow \mathsf{Share}(\mathsf{A},k);\ y_A \leftarrow \mathsf{Share}(\mathsf{A},l);\ \mathsf{ret}\ b_A \oplus y_A$
  - $\mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,3) := b_A \leftarrow \mathsf{OTBit}(\mathsf{A},\mathsf{B},K);\ x_A \leftarrow \mathsf{Share}(\mathsf{A},k);\ y_A \leftarrow \mathsf{Share}(\mathsf{A},l);\ \mathsf{ret}\ b_A \oplus x_A \oplus y_A$

9.3.4 Bob: The Inductive Phase

In the case of an *input* gate, Bob uses his corresponding input share from the initial stage. In the case of a *not* gate, the resulting share is a negation of the share x_B of the incoming wire. If the gate is an *xor* gate, the resulting share is the sum $x_B \oplus y_B$ of the shares of the incoming two wires. Finally, in the case of an *and* gate, Bob engages in an idealized 1-Out-Of-4 exchange with Alice as described in the previous section. To compute his share, he adds the result of the OT exchange to the product $x_B * y_B$ of the shares of the incoming two wires.

- $\mathsf{B}(\epsilon,0)$ is the protocol $\mathsf{0}$
- $\mathsf{B}\big(C;\ \textit{input-gate}(i),K+1\big)$ is the composition of the protocol $\mathsf{B}(C,K)$ with the protocol

  - $\mathsf{OTBit}(\mathsf{B},\mathsf{A},K) := \mathsf{read}\ \mathsf{OTBit}(\mathsf{B},\mathsf{A},K)$
  - $\begin{cases} \mathsf{Share}(\mathsf{B},K) := \mathsf{read}\ \mathsf{InputShare}(\mathsf{B},\mathsf{A},i) & \text{if } i \text{ is an input of Alice} \\ \mathsf{Share}(\mathsf{B},K) := \mathsf{read}\ \mathsf{InputShare}(\mathsf{B},\mathsf{B},i) & \text{if } i \text{ is an input of Bob} \end{cases}$
  - $\mathsf{OTChoice}(\mathsf{B},\mathsf{A},K,0) := \mathsf{read}\ \mathsf{OTChoice}(\mathsf{B},\mathsf{A},K,0)$
  - $\mathsf{OTChoice}(\mathsf{B},\mathsf{A},K,1) := \mathsf{read}\ \mathsf{OTChoice}(\mathsf{B},\mathsf{A},K,1)$

  As for Alice, we include the requisite vacuous OT channels to stay consistent throughout the cases.

- $\mathsf{B}\big(C;\ \textit{not-gate}(k),K+1\big)$ the composition of the protocol $\mathsf{B}(C,K)$ with the protocol

  - $\mathsf{OTBit}(\mathsf{B},\mathsf{A},K) := \mathsf{read}\ \mathsf{OTBit}(\mathsf{B},\mathsf{A},K)$
  - $\mathsf{Share}(\mathsf{B},K) := x_B \leftarrow \mathsf{Share}(\mathsf{B},k);\ \mathsf{ret}\ \neg x_B$
  - $\mathsf{OTChoice}(\mathsf{B},\mathsf{A},K,0) := \mathsf{read}\ \mathsf{OTChoice}(\mathsf{B},\mathsf{A},K,0)$
  - $\mathsf{OTChoice}(\mathsf{B},\mathsf{A},K,1) := \mathsf{read}\ \mathsf{OTChoice}(\mathsf{B},\mathsf{A},K,1)$

- $\mathsf{B}\big(C;\ \textit{xor-gate}(k,l),K+1\big)$ is the composition of the protocol $\mathsf{B}(C,K)$ with the protocol

  - $\mathsf{OTBit}(\mathsf{B},\mathsf{A},K) := \mathsf{read}\ \mathsf{OTBit}(\mathsf{B},\mathsf{A},K)$

  - $\mathsf{Share}(\mathsf{B}, K) := x_B \leftarrow \mathsf{Share}(\mathsf{B}, k);\ y_B \leftarrow \mathsf{Share}(\mathsf{B}, l);\ \mathsf{ret}\ x_B \oplus y_B$
  - $\mathsf{OTChoice}(\mathsf{B}, \mathsf{A}, K, 0) := \mathsf{read}\ \mathsf{OTChoice}(\mathsf{B}, \mathsf{A}, K, 0)$
  - $\mathsf{OTChoice}(\mathsf{B}, \mathsf{A}, K, 1) := \mathsf{read}\ \mathsf{OTChoice}(\mathsf{B}, \mathsf{A}, K, 1)$
- $\mathsf{B}\big(C;\ \textit{and-gate}(k, l), K+1\big)$ is the composition of the protocol $\mathsf{B}(C, K)$ with the protocol
  - $\mathsf{OTBit}(\mathsf{B}, \mathsf{A}, K) := \mathsf{read}\ \mathsf{OTOut}(\mathsf{A}, \mathsf{B}, K)$
  - $\mathsf{Share}(\mathsf{B}, K) := b_B \leftarrow \mathsf{OTBit}(\mathsf{B}, \mathsf{A}, K);\ x_B \leftarrow \mathsf{Share}(\mathsf{B}, k);\ y_B \leftarrow \mathsf{Share}(\mathsf{B}, l);\ \mathsf{ret}\ b_B \oplus (x_B * y_B)$
  - $\mathsf{OTChoice}(\mathsf{B}, \mathsf{A}, K, 0) := \mathsf{read}\ \mathsf{Share}(\mathsf{B}, k)$
  - $\mathsf{OTChoice}(\mathsf{B}, \mathsf{A}, K, 1) := \mathsf{read}\ \mathsf{Share}(\mathsf{B}, l)$

9.3.5 Alice: The Final Phase

For each output wire, Alice sends her share to Bob while simultaneously leaking it to the adversary. Each share received from Bob is likewise forwarded to the adversary. Finally, Alice computes the output by summing up her and Bob's respective shares, and leaks the result to the adversary.

- $\begin{cases} \mathsf{SendFinalShare}(\mathsf{A}, k) := \mathsf{read}\ \mathsf{Share}(\mathsf{A}, k) & \text{if wire } 0 \leqslant k < K \text{ is output} \\ \mathsf{SendFinalShare}(\mathsf{A}, k) := \mathsf{read}\ \mathsf{SendFinalShare}(\mathsf{A}, k) & \text{otherwise} \end{cases}$
- $\mathsf{SendFinalShare}(\mathsf{A}, k)^{\mathsf{A}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{SendFinalShare}(\mathsf{A}, k)$ for $0 \leqslant k < K$
- $\mathsf{SendFinalShare}(\mathsf{B}, k)^{\mathsf{A}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{SendFinalShare}(\mathsf{B}, k)$ for $0 \leqslant k < K$
- $\mathsf{Out}(\mathsf{A}, k) := x_A \leftarrow \mathsf{SendFinalShare}(\mathsf{A}, k);\ x_B \leftarrow \mathsf{SendFinalShare}(\mathsf{B}, k);\ \mathsf{ret}\ x_A \oplus x_B$ for $0 \leqslant k < K$
- $\mathsf{Out}(\mathsf{A}, k)^{\mathsf{A}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{Out}(\mathsf{A}, k)$ for $0 \leqslant k < K$

9.3.6 Bob: The Final Phase

For each output wire, Bob sends his share to Alice and computes the output by summing up his and Alice's respective shares.

- $\begin{cases} \mathsf{SendFinalShare}(\mathsf{B}, k) := \mathsf{read}\ \mathsf{Share}(\mathsf{B}, k) & \text{if wire } 0 \leqslant k < K \text{ is output} \\ \mathsf{SendFinalShare}(\mathsf{B}, k) := \mathsf{read}\ \mathsf{SendFinalShare}(\mathsf{B}, k) & \text{otherwise} \end{cases}$
- $\mathsf{Out}(\mathsf{B}, k) := x_A \leftarrow \mathsf{SendFinalShare}(\mathsf{A}, k);\ x_B \leftarrow \mathsf{SendFinalShare}(\mathsf{B}, k);\ \mathsf{ret}\ x_A \oplus x_B$ for $0 \leqslant k < K$

9.3.7 1-Out-Of-4 Oblivious Transfer Functionality

For each wire $0 \leqslant k < K$ we have a separate idealized 1-Out-Of-4 OT functionality $\mathsf{1OutOf4OT}(k)$, which we now describe. The functionality starts by selecting the correct message:

- $\mathsf{OTOut}(\mathsf{A}, \mathsf{B}, k) := m_0 \leftarrow \mathsf{OTMsg}(\mathsf{A}, \mathsf{B}, k, 0);\ m_1 \leftarrow \mathsf{OTMsg}(\mathsf{A}, \mathsf{B}, k, 1);\ m_2 \leftarrow \mathsf{OTMsg}(\mathsf{A}, \mathsf{B}, k, 2);$
  $m_3 \leftarrow \mathsf{OTMsg}(\mathsf{A}, \mathsf{B}, k, 3);\ c_0 \leftarrow \mathsf{OTChoice}(\mathsf{B}, \mathsf{A}, K, 0);\ c_1 \leftarrow \mathsf{OTChoice}(\mathsf{B}, \mathsf{A}, K, 1);$
  $\mathsf{if}\ c_0\ \mathsf{then}\ \big(\mathsf{if}\ c_1\ \mathsf{then}\ \mathsf{ret}\ m_3\ \mathsf{else}\ \mathsf{ret}\ m_2\big)\ \mathsf{else}\ \big(\mathsf{if}\ c_1\ \mathsf{then}\ \mathsf{ret}\ m_1\ \mathsf{else}\ \mathsf{ret}\ m_0\big)$

Since Alice is semi-honest, the functionality leaks the value of all messages received from Alice:

- $\mathsf{OTMsg}(\mathsf{A}, \mathsf{B}, k, 0)^{\mathsf{ot}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{OTMsg}(\mathsf{A}, \mathsf{B}, k, 0)$
- $\mathsf{OTMsg}(\mathsf{A}, \mathsf{B}, k, 1)^{\mathsf{ot}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{OTMsg}(\mathsf{A}, \mathsf{B}, k, 1)$
- $\mathsf{OTMsg}(\mathsf{A}, \mathsf{B}, k, 2)^{\mathsf{ot}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{OTMsg}(\mathsf{A}, \mathsf{B}, k, 2)$
- $\mathsf{OTMsg}(\mathsf{A}, \mathsf{B}, k, 3)^{\mathsf{ot}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{OTMsg}(\mathsf{A}, \mathsf{B}, k, 3)$

Since Bob is honest, only the timing information for his inputs is leaked:

- $\mathsf{OTChoiceRcvd}(\mathsf{B}, \mathsf{A}, k, 0)^{\mathsf{ot}}_{\mathsf{adv}} := c_0 \leftarrow \mathsf{OTChoice}(\mathsf{B}, \mathsf{A}, k, 0);\ \mathsf{ret}\ \checkmark$
- $\mathsf{OTChoiceRcvd}(\mathsf{B}, \mathsf{A}, k, 1)^{\mathsf{ot}}_{\mathsf{adv}} := c_1 \leftarrow \mathsf{OTChoice}(\mathsf{B}, \mathsf{A}, k, 1);\ \mathsf{ret}\ \checkmark$

9.3.8 The Real Protocol

The complete code for Alice arises as the composition of Alice's initial, inductive, and final stages, followed by the hiding of the communication internal to Alice - namely, the channels

- $\mathsf{InputShare}(\mathsf{A}, \mathsf{A}, i)$ for $0 \leqslant i < N$,
- $\mathsf{InputShare}(\mathsf{A}, \mathsf{B}, i)$ for $0 \leqslant i < M$,
- $\mathsf{OTBit}(\mathsf{A}, \mathsf{B}, k)$ for $0 \leqslant k < K$, and
- $\mathsf{Share}(\mathsf{A}, k)$ for $0 \leqslant k < K$.

Analogously, the complete code for Bob arises as the composition of Bob's initial, inductive, and final stages, followed by the hiding of the communication internal to Bob - namely, the channels

- $\mathsf{InputShare}(\mathsf{B}, \mathsf{A}, i)$ for $0 \leqslant i < N$,
- $\mathsf{InputShare}(\mathsf{B}, \mathsf{B}, i)$ for $0 \leqslant i < M$,
- $\mathsf{OTBit}(\mathsf{B}, \mathsf{A}, k)$ for $0 \leqslant k < K$, and
- $\mathsf{Share}(\mathsf{B}, k)$ for $0 \leqslant k < K$.

Finally, the real protocol is a composition of the two parties plus K copies of the OT functionality,

- $\mathsf{1OutOf4OT}(k)$ for $0 \leqslant k < K$,

all followed by the hiding of the internal communication among the two parties and the functionality: the channels

- $\mathsf{SendInputShare}(\mathsf{A}, \mathsf{B}, i)$ for $0 \leqslant i < M$,
- $\mathsf{SendInputShare}(\mathsf{B}, \mathsf{A}, i)$ for $0 \leqslant i < N$,
- $\mathsf{OTMsg}(\mathsf{A}, \mathsf{B}, k, 0)$ for $0 \leqslant k < K$,
- $\mathsf{OTMsg}(\mathsf{A}, \mathsf{B}, k, 1)$ for $0 \leqslant k < K$,
- $\mathsf{OTMsg}(\mathsf{A}, \mathsf{B}, k, 2)$ for $0 \leqslant k < K$,
- $\mathsf{OTMsg}(\mathsf{A}, \mathsf{B}, k, 3)$ for $0 \leqslant k < K$,
- $\mathsf{OTChoice}(\mathsf{B}, \mathsf{A}, k, 0)$ for $0 \leqslant k < K$,
- $\mathsf{OTChoice}(\mathsf{B}, \mathsf{A}, k, 1)$ for $0 \leqslant k < K$,
- $\mathsf{OTOut}(\mathsf{B}, \mathsf{A}, k)$ for $0 \leqslant k < K$,
- $\mathsf{SendFinalShare}(\mathsf{A}, k)$ for $0 \leqslant k < K$, and
- $\mathsf{SendFinalShare}(\mathsf{B}, k)$ $0 \leqslant k < K$.

9.4 Real = Ideal + Simulator

Our goal is to keep simplifying the real protocol until it becomes clear how to extract out a suitable simulator. We first restructure the entire protocol as a composition of an initial part, an inductive part, and a final part, followed by the hiding of the channels

- $\mathsf{InputShare}(\mathsf{A}, \mathsf{A}, i)$ for $0 \leqslant i < N$,
- $\mathsf{InputShare}(\mathsf{A}, \mathsf{B}, i)$ for $0 \leqslant i < M$,
- $\mathsf{InputShare}(\mathsf{B}, \mathsf{A}, i)$ for $0 \leqslant i < N$,

- $\mathsf{InputShare}(\mathsf{B}, \mathsf{B}, i)$ for $0 \leqslant i < M$,
- $\mathsf{OTBit}(\mathsf{A}, \mathsf{B}, k)$ for $0 \leqslant k < K$,
- $\mathsf{OTBit}(\mathsf{B}, \mathsf{A}, k)$ for $0 \leqslant k < K$,
- $\mathsf{Share}(\mathsf{A}, k)$ for $0 \leqslant k < K$, and
- $\mathsf{Share}(\mathsf{B}, k)$ for $0 \leqslant k < K$.

The initial part arises by composing together the initial parts for Alice and Bob, and declaring the channels $\mathsf{SendInputShare}(\mathsf{B}, \mathsf{A}, -)$ and $\mathsf{SendInputShare}(\mathsf{A}, \mathsf{B}, -)$ as internal:

- $\mathsf{In}(\mathsf{A}, i)^{\mathsf{A}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{In}(\mathsf{A}, i)$ for $0 \leqslant i < N$
- $\mathsf{InRcvd}(\mathsf{B}, i)^{\mathsf{B}}_{\mathsf{adv}} := x \leftarrow \mathsf{In}(\mathsf{B}, i);\ \mathsf{ret}\ \checkmark$ for $0 \leqslant i < M$
- $\mathsf{InputShare}(\mathsf{A}, \mathsf{A}, i) := x \leftarrow \mathsf{In}(\mathsf{A}, i);\ \mathsf{samp\ flip}$ for $0 \leqslant i < N$
- $\mathsf{InputShare}(\mathsf{A}, \mathsf{B}, i) := \mathsf{read}\ \mathsf{SendInputShare}(\mathsf{A}, \mathsf{B}, i)$ for $0 \leqslant i < M$
- $\mathsf{InputShare}(\mathsf{B}, \mathsf{A}, i) := \mathsf{read}\ \mathsf{SendInputShare}(\mathsf{B}, \mathsf{A}, i)$ for $0 \leqslant i < N$
- $\mathsf{InputShare}(\mathsf{B}, \mathsf{B}, i) := x_A \leftarrow \mathsf{SendInputShare}(\mathsf{A}, \mathsf{B}, i);\ x \leftarrow \mathsf{In}(\mathsf{B}, i);\ \mathsf{ret}\ x_A \oplus x$ for $0 \leqslant i < M$
- $\mathsf{InputShare}(\mathsf{A}, \mathsf{A}, i)^{\mathsf{A}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{InputShare}(\mathsf{A}, \mathsf{A}, i)$ for $0 \leqslant i < N$
- $\mathsf{InputShare}(\mathsf{A}, \mathsf{B}, i)^{\mathsf{A}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{InputShare}(\mathsf{A}, \mathsf{B}, i)$ for $0 \leqslant i < M$
- $\mathsf{SendInputShare}(\mathsf{B}, \mathsf{A}, i) := x_A \leftarrow \mathsf{InputShare}(\mathsf{A}, \mathsf{A}, i);\ x \leftarrow \mathsf{In}(\mathsf{A}, i);\ \mathsf{ret}\ x_A \oplus x$ for $0 \leqslant i < N$
- $\mathsf{SendInputShare}(\mathsf{A}, \mathsf{B}, i) := x \leftarrow \mathsf{In}(\mathsf{B}, i);\ \mathsf{samp\ flip}$ for $0 \leqslant i < M$
- $\mathsf{SendInputShare}(\mathsf{B}, \mathsf{A}, i)^{\mathsf{A}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{SendInputShare}(\mathsf{B}, \mathsf{A}, i)$ for $0 \leqslant i < N$
- $\mathsf{SendInputShare}(\mathsf{A}, \mathsf{B}, i)^{\mathsf{A}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{SendInputShare}(\mathsf{A}, \mathsf{B}, i)$ for $0 \leqslant i < M$

The inductive part of the real protocol arises by composing together the inductive parts for Alice and Bob plus the K copies of the OT functionality, and declaring the communication with the OT functionality as internal:

- $\mathsf{Real}(\epsilon, 0)$ is the protocol $\mathsf{0}$
- $\mathsf{Real}\big(C;\ \mathit{input\text{-}gate}(i), K+1\big)$ is the composition of the protocol $\mathsf{Real}(C, K)$ with the protocol
  - $\mathsf{OTBit}(\mathsf{A}, \mathsf{B}, K) := \mathsf{read}\ \mathsf{OTBit}(\mathsf{A}, \mathsf{B}, K)$
  - $\mathsf{OTBit}(\mathsf{B}, \mathsf{A}, K) := \mathsf{read}\ \mathsf{OTBit}(\mathsf{B}, \mathsf{A}, K)$
  - $\begin{cases} \mathsf{Share}(\mathsf{A}, K) := \mathsf{read}\ \mathsf{InputShare}(\mathsf{A}, \mathsf{A}, i) & \text{if } i \text{ is an input of Alice} \\ \mathsf{Share}(\mathsf{A}, K) := \mathsf{read}\ \mathsf{InputShare}(\mathsf{A}, \mathsf{B}, i) & \text{if } i \text{ is an input of Bob} \end{cases}$
  - $\begin{cases} \mathsf{Share}(\mathsf{B}, K) := \mathsf{read}\ \mathsf{InputShare}(\mathsf{B}, \mathsf{A}, i) & \text{if } i \text{ is an input of Alice} \\ \mathsf{Share}(\mathsf{B}, K) := \mathsf{read}\ \mathsf{InputShare}(\mathsf{B}, \mathsf{B}, i) & \text{if } i \text{ is an input of Bob} \end{cases}$
  - $\mathsf{OTMsg}(\mathsf{A}, \mathsf{B}, K, 0) := \mathsf{read}\ \mathsf{OTMsg}(\mathsf{A}, \mathsf{B}, K, 0)$
  - $\mathsf{OTMsg}(\mathsf{A}, \mathsf{B}, K, 1) := \mathsf{read}\ \mathsf{OTMsg}(\mathsf{A}, \mathsf{B}, K, 1)$
  - $\mathsf{OTMsg}(\mathsf{A}, \mathsf{B}, K, 2) := \mathsf{read}\ \mathsf{OTMsg}(\mathsf{A}, \mathsf{B}, K, 2)$
  - $\mathsf{OTMsg}(\mathsf{A}, \mathsf{B}, K, 3) := \mathsf{read}\ \mathsf{OTMsg}(\mathsf{A}, \mathsf{B}, K, 3)$
  - $\mathsf{OTChoice}(\mathsf{B}, \mathsf{A}, K, 0) := \mathsf{read}\ \mathsf{OTChoice}(\mathsf{B}, \mathsf{A}, K, 0)$

- $\mathsf{OTChoice}(\mathsf{B},\mathsf{A},K,1) := \mathsf{read}\ \mathsf{OTChoice}(\mathsf{B},\mathsf{A},K,1)$
- $\mathsf{OTOut}(\mathsf{B},\mathsf{A},K) := m_0 \leftarrow \mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,0);\ m_1 \leftarrow \mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,1);\ m_2 \leftarrow \mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,2);$ $m_3 \leftarrow \mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,3);\ c_0 \leftarrow \mathsf{OTChoice}(\mathsf{B},\mathsf{A},K,0);\ c_1 \leftarrow \mathsf{OTChoice}(\mathsf{B},\mathsf{A},K,1);$ $\mathsf{if}\ c_0\ \mathsf{then}\ \big(\mathsf{if}\ c_1\ \mathsf{then\ ret}\ m_3\ \mathsf{else\ ret}\ m_2\big)\ \mathsf{else}\ \big(\mathsf{if}\ c_1\ \mathsf{then\ ret}\ m_1\ \mathsf{else\ ret}\ m_0\big)$
- $\mathsf{OTBit}(\mathsf{A},\mathsf{B},K)^{\mathsf{A}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{OTBit}(\mathsf{A},\mathsf{B},K)^{\mathsf{A}}_{\mathsf{adv}}$
- $\mathsf{Share}(\mathsf{A},K)^{\mathsf{A}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{Share}(\mathsf{A},K)$
- $\mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,0)^{\mathsf{ot}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,0)$
- $\mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,1)^{\mathsf{ot}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,1)$
- $\mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,2)^{\mathsf{ot}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,2)$
- $\mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,3)^{\mathsf{ot}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,3)$
- $\mathsf{OTChoiceRcvd}(\mathsf{B},\mathsf{A},K,0)^{\mathsf{ot}}_{\mathsf{adv}} := c_0 \leftarrow \mathsf{OTChoice}(\mathsf{B},\mathsf{A},K,0);\ \mathsf{ret}\ \checkmark$
- $\mathsf{OTChoiceRcvd}(\mathsf{B},\mathsf{A},K,1)^{\mathsf{ot}}_{\mathsf{adv}} := c_1 \leftarrow \mathsf{OTChoice}(\mathsf{B},\mathsf{A},K,1);\ \mathsf{ret}\ \checkmark$

- $\mathsf{Real}\big(C;\ \textit{not-gate}(k), K+1\big)$ is the composition of the protocol $\mathsf{Real}(C,K)$ with the protocol
  - $\mathsf{OTBit}(\mathsf{A},\mathsf{B},K) := \mathsf{read}\ \mathsf{OTBit}(\mathsf{A},\mathsf{B},K)$
  - $\mathsf{OTBit}(\mathsf{B},\mathsf{A},K) := \mathsf{read}\ \mathsf{OTBit}(\mathsf{B},\mathsf{A},K)$
  - $\mathsf{Share}(\mathsf{A},K) := \mathsf{read}\ \mathsf{Share}(\mathsf{A},k)$
  - $\mathsf{Share}(\mathsf{B},K) := x_B \leftarrow \mathsf{Share}(\mathsf{B},k);\ \mathsf{ret}\ \neg x_B$
  - $\mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,0) := \mathsf{read}\ \mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,0)$
  - $\mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,1) := \mathsf{read}\ \mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,1)$
  - $\mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,2) := \mathsf{read}\ \mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,2)$
  - $\mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,3) := \mathsf{read}\ \mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,3)$
  - $\mathsf{OTChoice}(\mathsf{B},\mathsf{A},K,0) := \mathsf{read}\ \mathsf{OTChoice}(\mathsf{B},\mathsf{A},K,0)$
  - $\mathsf{OTChoice}(\mathsf{B},\mathsf{A},K,1) := \mathsf{read}\ \mathsf{OTChoice}(\mathsf{B},\mathsf{A},K,1)$
  - $\mathsf{OTOut}(\mathsf{B},\mathsf{A},K) := m_0 \leftarrow \mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,0);\ m_1 \leftarrow \mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,1);\ m_2 \leftarrow \mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,2);$ $m_3 \leftarrow \mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,3);\ c_0 \leftarrow \mathsf{OTChoice}(\mathsf{B},\mathsf{A},K,0);\ c_1 \leftarrow \mathsf{OTChoice}(\mathsf{B},\mathsf{A},K,1);$ $\mathsf{if}\ c_0\ \mathsf{then}\ \big(\mathsf{if}\ c_1\ \mathsf{then\ ret}\ m_3\ \mathsf{else\ ret}\ m_2\big)\ \mathsf{else}\ \big(\mathsf{if}\ c_1\ \mathsf{then\ ret}\ m_1\ \mathsf{else\ ret}\ m_0\big)$
  - $\mathsf{OTBit}(\mathsf{A},\mathsf{B},K)^{\mathsf{A}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{OTBit}(\mathsf{A},\mathsf{B},K)^{\mathsf{A}}_{\mathsf{adv}}$
  - $\mathsf{Share}(\mathsf{A},K)^{\mathsf{A}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{Share}(\mathsf{A},K)$
  - $\mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,0)^{\mathsf{ot}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,0)$
  - $\mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,1)^{\mathsf{ot}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,1)$
  - $\mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,2)^{\mathsf{ot}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,2)$
  - $\mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,3)^{\mathsf{ot}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,3)$
  - $\mathsf{OTChoiceRcvd}(\mathsf{B},\mathsf{A},K,0)^{\mathsf{ot}}_{\mathsf{adv}} := c_0 \leftarrow \mathsf{OTChoice}(\mathsf{B},\mathsf{A},K,0);\ \mathsf{ret}\ \checkmark$
  - $\mathsf{OTChoiceRcvd}(\mathsf{B},\mathsf{A},K,1)^{\mathsf{ot}}_{\mathsf{adv}} := c_1 \leftarrow \mathsf{OTChoice}(\mathsf{B},\mathsf{A},K,1);\ \mathsf{ret}\ \checkmark$
- $\mathsf{Real}\big(C;\ \textit{xor-gate}(k,l), K+1\big)$ is the composition of the protocol $\mathsf{Real}(C,K)$ with the protocol
  - $\mathsf{OTBit}(\mathsf{A},\mathsf{B},K) := \mathsf{read}\ \mathsf{OTBit}(\mathsf{A},\mathsf{B},K)$
  - $\mathsf{OTBit}(\mathsf{B},\mathsf{A},K) := \mathsf{read}\ \mathsf{OTBit}(\mathsf{B},\mathsf{A},K)$
  - $\mathsf{Share}(\mathsf{A},K) := x_A \leftarrow \mathsf{Share}(\mathsf{A},k);\ y_A \leftarrow \mathsf{Share}(\mathsf{A},l);\ \mathsf{ret}\ x_A \oplus y_A$
  - $\mathsf{Share}(\mathsf{B},K) := x_B \leftarrow \mathsf{Share}(\mathsf{B},k);\ y_B \leftarrow \mathsf{Share}(\mathsf{B},l);\ \mathsf{ret}\ x_B \oplus y_B$

- $\mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,0) := \text{read } \mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,0)$
- $\mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,1) := \text{read } \mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,1)$
- $\mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,2) := \text{read } \mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,2)$
- $\mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,3) := \text{read } \mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,3)$
- $\mathsf{OTChoice}(\mathsf{B},\mathsf{A},K,0) := \text{read } \mathsf{OTChoice}(\mathsf{B},\mathsf{A},K,0)$
- $\mathsf{OTChoice}(\mathsf{B},\mathsf{A},K,1) := \text{read } \mathsf{OTChoice}(\mathsf{B},\mathsf{A},K,1)$
- $\mathsf{OTOut}(\mathsf{B},\mathsf{A},K) := m_0 \leftarrow \mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,0);\ m_1 \leftarrow \mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,1);\ m_2 \leftarrow \mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,2);$ $m_3 \leftarrow \mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,3);\ c_0 \leftarrow \mathsf{OTChoice}(\mathsf{B},\mathsf{A},K,0);\ c_1 \leftarrow \mathsf{OTChoice}(\mathsf{B},\mathsf{A},K,1);$ $\text{if } c_0 \text{ then } \big(\text{if } c_1 \text{ then ret } m_3 \text{ else ret } m_2\big) \text{ else } \big(\text{if } c_1 \text{ then ret } m_1 \text{ else ret } m_0\big)$
- $\mathsf{OTBit}(\mathsf{A},\mathsf{B},K)^{\mathsf{A}}_{\mathsf{adv}} := \text{read } \mathsf{OTBit}(\mathsf{A},\mathsf{B},K)^{\mathsf{A}}_{\mathsf{adv}}$
- $\mathsf{Share}(\mathsf{A},K)^{\mathsf{A}}_{\mathsf{adv}} := \text{read } \mathsf{Share}(\mathsf{A},K)$
- $\mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,0)^{\mathsf{ot}}_{\mathsf{adv}} := \text{read } \mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,0)$
- $\mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,1)^{\mathsf{ot}}_{\mathsf{adv}} := \text{read } \mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,1)$
- $\mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,2)^{\mathsf{ot}}_{\mathsf{adv}} := \text{read } \mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,2)$
- $\mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,3)^{\mathsf{ot}}_{\mathsf{adv}} := \text{read } \mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,3)$
- $\mathsf{OTChoiceRcvd}(\mathsf{B},\mathsf{A},K,0)^{\mathsf{ot}}_{\mathsf{adv}} := c_0 \leftarrow \mathsf{OTChoice}(\mathsf{B},\mathsf{A},K,0);\ \text{ret } \checkmark$
- $\mathsf{OTChoiceRcvd}(\mathsf{B},\mathsf{A},K,1)^{\mathsf{ot}}_{\mathsf{adv}} := c_1 \leftarrow \mathsf{OTChoice}(\mathsf{B},\mathsf{A},K,1);\ \text{ret } \checkmark$

- $\mathsf{Real}\big(C;\ \mathit{and\text{-}gate}(k,l), K+1\big)$ is the composition of the protocol $\mathsf{Real}(C,K)$ with the protocol
  - $\mathsf{OTBit}(\mathsf{A},\mathsf{B},K) := x_A \leftarrow \mathsf{Share}(\mathsf{A},k);\ y_A \leftarrow \mathsf{Share}(\mathsf{A},l);\ \text{samp flip}$
  - $\mathsf{OTBit}(\mathsf{B},\mathsf{A},K) := \mathsf{OTOut}(\mathsf{A},\mathsf{B},K)$
  - $\mathsf{Share}(\mathsf{A},K) := b_A \leftarrow \mathsf{OTBit}(\mathsf{A},\mathsf{B},K);\ x_A \leftarrow \mathsf{Share}(\mathsf{A},k);\ y_A \leftarrow \mathsf{Share}(\mathsf{A},l);\ \text{ret } (x_A * y_A) \oplus b_A$
  - $\mathsf{Share}(\mathsf{B},K) := b_B \leftarrow \mathsf{OTBit}(\mathsf{B},\mathsf{A},K);\ x_B \leftarrow \mathsf{Share}(\mathsf{B},k);\ y_B \leftarrow \mathsf{Share}(\mathsf{B},l);\ \text{ret } b_B \oplus (x_B * y_B)$
  - $\mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,0) := b_A \leftarrow \mathsf{OTBit}(\mathsf{A},\mathsf{B},K);\ x_A \leftarrow \mathsf{Share}(\mathsf{A},k);\ y_A \leftarrow \mathsf{Share}(\mathsf{A},l);\ \text{ret } b_A$
  - $\mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,1) := b_A \leftarrow \mathsf{OTBit}(\mathsf{A},\mathsf{B},K);\ x_A \leftarrow \mathsf{Share}(\mathsf{A},k);\ y_A \leftarrow \mathsf{Share}(\mathsf{A},l);\ \text{ret } b_A \oplus x_A$
  - $\mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,2) := b_A \leftarrow \mathsf{OTBit}(\mathsf{A},\mathsf{B},K);\ x_A \leftarrow \mathsf{Share}(\mathsf{A},k);\ y_A \leftarrow \mathsf{Share}(\mathsf{A},l);\ \text{ret } b_A \oplus y_A$
  - $\mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,3) := b_A \leftarrow \mathsf{OTBit}(\mathsf{A},\mathsf{B},K);\ x_A \leftarrow \mathsf{Share}(\mathsf{A},k);\ y_A \leftarrow \mathsf{Share}(\mathsf{A},l);\ \text{ret } b_A \oplus x_A \oplus y_A$
  - $\mathsf{OTChoice}(\mathsf{B},\mathsf{A},K,0) := \text{read } \mathsf{Share}(\mathsf{B},k)$
  - $\mathsf{OTChoice}(\mathsf{B},\mathsf{A},K,1) := \text{read } \mathsf{Share}(\mathsf{B},l)$
  - $\mathsf{OTOut}(\mathsf{B},\mathsf{A},K) := m_0 \leftarrow \mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,0);\ m_1 \leftarrow \mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,1);\ m_2 \leftarrow \mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,2);$ $m_3 \leftarrow \mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,3);\ c_0 \leftarrow \mathsf{OTChoice}(\mathsf{B},\mathsf{A},K,0);\ c_1 \leftarrow \mathsf{OTChoice}(\mathsf{B},\mathsf{A},K,1);$ $\text{if } c_0 \text{ then } \big(\text{if } c_1 \text{ then ret } m_3 \text{ else ret } m_2\big) \text{ else } \big(\text{if } c_1 \text{ then ret } m_1 \text{ else ret } m_0\big)$
  - $\mathsf{OTBit}(\mathsf{A},\mathsf{B},K)^{\mathsf{A}}_{\mathsf{adv}} := \text{read } \mathsf{OTBit}(\mathsf{A},\mathsf{B},K)^{\mathsf{A}}_{\mathsf{adv}}$
  - $\mathsf{Share}(\mathsf{A},K)^{\mathsf{A}}_{\mathsf{adv}} := \text{read } \mathsf{Share}(\mathsf{A},K)$
  - $\mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,0)^{\mathsf{ot}}_{\mathsf{adv}} := \text{read } \mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,0)$
  - $\mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,1)^{\mathsf{ot}}_{\mathsf{adv}} := \text{read } \mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,1)$
  - $\mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,2)^{\mathsf{ot}}_{\mathsf{adv}} := \text{read } \mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,2)$
  - $\mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,3)^{\mathsf{ot}}_{\mathsf{adv}} := \text{read } \mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,3)$
  - $\mathsf{OTChoiceRcvd}(\mathsf{B},\mathsf{A},K,0)^{\mathsf{ot}}_{\mathsf{adv}} := c_0 \leftarrow \mathsf{OTChoice}(\mathsf{B},\mathsf{A},K,0);\ \text{ret } \checkmark$
  - $\mathsf{OTChoiceRcvd}(\mathsf{B},\mathsf{A},K,1)^{\mathsf{ot}}_{\mathsf{adv}} := c_1 \leftarrow \mathsf{OTChoice}(\mathsf{B},\mathsf{A},K,1);\ \text{ret } \checkmark$

At last, the final part of the real protocol arises by composing together the final parts for Alice and Bob, and declaring the channels $\mathsf{SendFinalShare}(\mathsf{A}, -)$ and $\mathsf{SendFinalShare}(\mathsf{B}, -)$ as internal:

- $\begin{cases} \mathsf{SendFinalShare}(\mathsf{A}, k) := \mathsf{read}\ \mathsf{Share}(\mathsf{A}, k) & \text{if wire } 0 \leqslant k < K \text{ is output} \\ \mathsf{SendFinalShare}(\mathsf{A}, k) := \mathsf{read}\ \mathsf{SendFinalShare}(\mathsf{A}, k) & \text{otherwise} \end{cases}$
- $\begin{cases} \mathsf{SendFinalShare}(\mathsf{B}, k) := \mathsf{read}\ \mathsf{Share}(\mathsf{B}, k) & \text{if wire } 0 \leqslant k < K \text{ is output} \\ \mathsf{SendFinalShare}(\mathsf{B}, k) := \mathsf{read}\ \mathsf{SendFinalShare}(\mathsf{B}, k) & \text{otherwise} \end{cases}$
- $\mathsf{SendFinalShare}(\mathsf{A}, k)^{\mathsf{A}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{SendFinalShare}(\mathsf{A}, k)$ for $0 \leqslant k < K$
- $\mathsf{SendFinalShare}(\mathsf{B}, k)^{\mathsf{A}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{SendFinalShare}(\mathsf{B}, k)$ for $0 \leqslant k < K$
- $\mathsf{Out}(\mathsf{A}, k) := x_A \leftarrow \mathsf{SendFinalShare}(\mathsf{A}, k);\ x_B \leftarrow \mathsf{SendFinalShare}(\mathsf{B}, k);\ \mathsf{ret}\ x_A \oplus x_B$ for $0 \leqslant k < K$
- $\mathsf{Out}(\mathsf{B}, k) := x_A \leftarrow \mathsf{SendFinalShare}(\mathsf{A}, k);\ x_B \leftarrow \mathsf{SendFinalShare}(\mathsf{B}, k);\ \mathsf{ret}\ x_A \oplus x_B$ for $0 \leqslant k < K$
- $\mathsf{Out}(\mathsf{A}, k)^{\mathsf{A}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{Out}(\mathsf{A}, k)$ for $0 \leqslant k < K$

9.4.1 Simplifying The Real Protocol: The Initial Phase

Our goal here is to eliminate the internal channels $\mathsf{SendInputShare}(\mathsf{B}, \mathsf{A}, -)$ and $\mathsf{SendInputShare}(\mathsf{A}, \mathsf{B}, -)$. We first replace any mention of these in channels $\mathsf{InputShare}(\mathsf{B}, \mathsf{B}, i)$, $\mathsf{SendInputShare}(\mathsf{B}, \mathsf{A}, i)^{\mathsf{A}}_{\mathsf{adv}}$, $\mathsf{SendInputShare}(\mathsf{A}, \mathsf{B}, i)^{\mathsf{A}}_{\mathsf{adv}}$ by the corresponding channels $\mathsf{InputShare}(\mathsf{B}, \mathsf{A}, -)$ and $\mathsf{InputShare}(\mathsf{A}, \mathsf{B}, -)$:

- $\mathsf{In}(\mathsf{A}, i)^{\mathsf{A}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{In}(\mathsf{A}, i)$ for $0 \leqslant i < N$
- $\mathsf{InRcvd}(\mathsf{B}, i)^{\mathsf{B}}_{\mathsf{adv}} := x \leftarrow \mathsf{In}(\mathsf{B}, i);\ \mathsf{ret}\ \checkmark$ for $0 \leqslant i < M$
- $\mathsf{InputShare}(\mathsf{A}, \mathsf{A}, i) := x \leftarrow \mathsf{In}(\mathsf{A}, i);\ \mathsf{samp\ flip}$ for $0 \leqslant i < N$
- $\mathsf{InputShare}(\mathsf{A}, \mathsf{B}, i) := \mathsf{read}\ \mathsf{SendInputShare}(\mathsf{A}, \mathsf{B}, i)$ for $0 \leqslant i < M$
- $\mathsf{InputShare}(\mathsf{B}, \mathsf{A}, i) := \mathsf{read}\ \mathsf{SendInputShare}(\mathsf{B}, \mathsf{A}, i)$ for $0 \leqslant i < N$
- $\mathsf{InputShare}(\mathsf{B}, \mathsf{B}, i) := x_A \leftarrow \mathsf{InputShare}(\mathsf{A}, \mathsf{B}, i);\ x \leftarrow \mathsf{In}(\mathsf{B}, i);\ \mathsf{ret}\ x_A \oplus x$ for $0 \leqslant i < M$
- $\mathsf{InputShare}(\mathsf{A}, \mathsf{A}, i)^{\mathsf{A}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{InputShare}(\mathsf{A}, \mathsf{A}, i)$ for $0 \leqslant i < N$
- $\mathsf{InputShare}(\mathsf{A}, \mathsf{B}, i)^{\mathsf{A}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{InputShare}(\mathsf{A}, \mathsf{B}, i)$ for $0 \leqslant i < M$
- $\mathsf{SendInputShare}(\mathsf{B}, \mathsf{A}, i) := x_A \leftarrow \mathsf{InputShare}(\mathsf{A}, \mathsf{A}, i);\ x \leftarrow \mathsf{In}(\mathsf{A}, i);\ \mathsf{ret}\ x_A \oplus x$ for $0 \leqslant i < N$
- $\mathsf{SendInputShare}(\mathsf{A}, \mathsf{B}, i) := x \leftarrow \mathsf{In}(\mathsf{B}, i);\ \mathsf{samp\ flip}$ for $0 \leqslant i < M$
- $\mathsf{SendInputShare}(\mathsf{B}, \mathsf{A}, i)^{\mathsf{A}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{InputShare}(\mathsf{B}, \mathsf{A}, i)$ for $0 \leqslant i < N$
- $\mathsf{SendInputShare}(\mathsf{A}, \mathsf{B}, i)^{\mathsf{A}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{InputShare}(\mathsf{A}, \mathsf{B}, i)$ for $0 \leqslant i < M$

Now the only place where the channels $\mathsf{SendInputShare}(\mathsf{B}, \mathsf{A}, -)$ and $\mathsf{SendInputShare}(\mathsf{A}, \mathsf{B}, -)$ show up is in the very channels $\mathsf{InputShare}(\mathsf{B}, \mathsf{A}, -)$ and $\mathsf{InputShare}(\mathsf{A}, \mathsf{B}, -)$, respectively. So we can fold them in:

- $\mathsf{In}(\mathsf{A}, i)^{\mathsf{A}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{In}(\mathsf{A}, i)$ for $0 \leqslant i < N$
- $\mathsf{InRcvd}(\mathsf{B}, i)^{\mathsf{B}}_{\mathsf{adv}} := x \leftarrow \mathsf{In}(\mathsf{B}, i);\ \mathsf{ret}\ \checkmark$ for $0 \leqslant i < M$
- $\mathsf{InputShare}(\mathsf{A}, \mathsf{A}, i) := x \leftarrow \mathsf{In}(\mathsf{A}, i);\ \mathsf{samp\ flip}$ for $0 \leqslant i < N$
- $\mathsf{InputShare}(\mathsf{A}, \mathsf{B}, i) := x \leftarrow \mathsf{In}(\mathsf{B}, i);\ \mathsf{samp\ flip}$ for $0 \leqslant i < M$

- $\mathsf{InputShare}(\mathsf{B},\mathsf{A},i) := x_A \leftarrow \mathsf{InputShare}(\mathsf{A},\mathsf{A},i);\ x \leftarrow \mathsf{In}(\mathsf{A},i);\ \mathsf{ret}\ x_A \oplus x \text{ for } 0 \leqslant i < N$
- $\mathsf{InputShare}(\mathsf{B},\mathsf{B},i) := x_A \leftarrow \mathsf{InputShare}(\mathsf{A},\mathsf{B},i);\ x \leftarrow \mathsf{In}(\mathsf{B},i);\ \mathsf{ret}\ x_A \oplus x \text{ for } 0 \leqslant i < M$
- $\mathsf{InputShare}(\mathsf{A},\mathsf{A},i)^{\mathsf{A}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{InputShare}(\mathsf{A},\mathsf{A},i) \text{ for } 0 \leqslant i < N$
- $\mathsf{InputShare}(\mathsf{A},\mathsf{B},i)^{\mathsf{A}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{InputShare}(\mathsf{A},\mathsf{B},i) \text{ for } 0 \leqslant i < M$
- $\mathsf{SendInputShare}(\mathsf{B},\mathsf{A},i)^{\mathsf{A}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{InputShare}(\mathsf{B},\mathsf{A},i) \text{ for } 0 \leqslant i < N$
- $\mathsf{SendInputShare}(\mathsf{A},\mathsf{B},i)^{\mathsf{A}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{InputShare}(\mathsf{A},\mathsf{B},i) \text{ for } 0 \leqslant i < M$

9.4.2 Simplifying The Real Protocol: The Inductive Stage

Our next order of business is to eliminate all channels interacting with the OT functionality. In the case of *input-*, *not-*, and *xor* gates, the OT channels are vacuous and only appear in the corresponding leakage channels. The leakage channels themselves are therefore vacuous: in the presence of

- $\mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,i) := \mathsf{read}\ \mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,i)$,

the two channel definitions

- $\mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,i)^{\mathsf{ot}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,i)$, and
- $\mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,i)^{\mathsf{ot}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,i)^{\mathsf{ot}}_{\mathsf{adv}}$

are equivalent. Similarly, in the presence of

- $\mathsf{OTChoice}(\mathsf{B},\mathsf{A},K,j) := \mathsf{read}\ \mathsf{OTChoice}(\mathsf{B},\mathsf{A},K,j)$,

the two channel definitions

- $\mathsf{OTChoiceRcvd}(\mathsf{B},\mathsf{A},K,j)^{\mathsf{ot}}_{\mathsf{adv}} := c_j \leftarrow \mathsf{OTChoice}(\mathsf{B},\mathsf{A},K,j);\ \mathsf{ret}\ \checkmark$, and
- $\mathsf{OTChoiceRcvd}(\mathsf{B},\mathsf{A},K,j)^{\mathsf{ot}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{OTChoiceRcvd}(\mathsf{B},\mathsf{A},K,j)^{\mathsf{ot}}_{\mathsf{adv}}$

are equivalent. In the case of an *and* gate, we start by eliminating any mention of the OT channels from the leakage channels. By substituting the channels $\mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,-)$ into the channels $\mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,-)^{\mathsf{ot}}_{\mathsf{adv}}$, we obtain

- $\mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,0)^{\mathsf{ot}}_{\mathsf{adv}} := b_A \leftarrow \mathsf{OTBit}(\mathsf{A},\mathsf{B},K);\ x_A \leftarrow \mathsf{Share}(\mathsf{A},k);\ y_A \leftarrow \mathsf{Share}(\mathsf{A},l);\ \mathsf{ret}\ b_A$
- $\mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,1)^{\mathsf{ot}}_{\mathsf{adv}} := b_A \leftarrow \mathsf{OTBit}(\mathsf{A},\mathsf{B},K);\ x_A \leftarrow \mathsf{Share}(\mathsf{A},k);\ y_A \leftarrow \mathsf{Share}(\mathsf{A},l);\ \mathsf{ret}\ b_A \oplus x_A$
- $\mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,2)^{\mathsf{ot}}_{\mathsf{adv}} := b_A \leftarrow \mathsf{OTBit}(\mathsf{A},\mathsf{B},K);\ x_A \leftarrow \mathsf{Share}(\mathsf{A},k);\ y_A \leftarrow \mathsf{Share}(\mathsf{A},l);\ \mathsf{ret}\ b_A \oplus y_A$
- $\mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,3)^{\mathsf{ot}}_{\mathsf{adv}} := b_A \leftarrow \mathsf{OTBit}(\mathsf{A},\mathsf{B},K);\ x_A \leftarrow \mathsf{Share}(\mathsf{A},k);\ y_A \leftarrow \mathsf{Share}(\mathsf{A},l);\ \mathsf{ret}\ b_A \oplus x_A \oplus y_A$

By substituting the channels $\mathsf{OTChoice}(\mathsf{B},\mathsf{A},K,-)$ into the channels $\mathsf{OTChoiceRcvd}(\mathsf{B},\mathsf{A},K,-)^{\mathsf{ot}}_{\mathsf{adv}}$, we get

- $\mathsf{OTChoiceRcvd}(\mathsf{B},\mathsf{A},K,0)^{\mathsf{ot}}_{\mathsf{adv}} := x_B \leftarrow \mathsf{Share}(\mathsf{B},k);\ \mathsf{ret}\ \checkmark$
- $\mathsf{OTChoiceRcvd}(\mathsf{B},\mathsf{A},K,1)^{\mathsf{ot}}_{\mathsf{adv}} := y_B \leftarrow \mathsf{Share}(\mathsf{B},l);\ \mathsf{ret}\ \checkmark$

Substituting the channels $\mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,-)$ and $\mathsf{OTChoice}(\mathsf{B},\mathsf{A},K,-)$ into $\mathsf{OTOut}(\mathsf{B},\mathsf{A},K)$ yields

- $\mathsf{OTOut}(\mathsf{B},\mathsf{A},K) := b_A \leftarrow \mathsf{OTBit}(\mathsf{A},\mathsf{B},K);\ x_A \leftarrow \mathsf{Share}(\mathsf{A},k);\ y_A \leftarrow \mathsf{Share}(\mathsf{A},l);\ x_B \leftarrow \mathsf{Share}(\mathsf{B},k);\ y_B \leftarrow \mathsf{Share}(\mathsf{B},l);\ \mathsf{if}\ x_B\ \mathsf{then}\ \big(\mathsf{if}\ y_B\ \mathsf{then}\ \mathsf{ret}\ b_A \oplus x_A \oplus y_A\ \mathsf{else}\ \mathsf{ret}\ b_A \oplus y_A\big)\ \mathsf{else}\ \big(\mathsf{if}\ y_B\ \mathsf{then}\ \mathsf{ret}\ b_A \oplus x_A\ \mathsf{else}\ \mathsf{ret}\ b_A\big)$

The above is just a fancy way of saying the following:

- $\mathsf{OTOut}(\mathsf{B},\mathsf{A},K) := b_A \leftarrow \mathsf{OTBit}(\mathsf{A},\mathsf{B},K);\ x_A \leftarrow \mathsf{Share}(\mathsf{A},k);\ y_A \leftarrow \mathsf{Share}(\mathsf{A},l);\ x_B \leftarrow \mathsf{Share}(\mathsf{B},k);\ y_B \leftarrow \mathsf{Share}(\mathsf{B},l);\ \mathsf{ret}\ b_A \oplus (x_A * y_B) \oplus (x_B * y_A)$

Substituting this new definition of $\mathsf{OTOut}(K)$ into the channel $\mathsf{OTBit}(\mathsf{B},\mathsf{A},K)$ yields

- $\mathsf{OTBit}(\mathsf{B},\mathsf{A},K) := b_A \leftarrow \mathsf{OTBit}(\mathsf{A},\mathsf{B},K);\ x_A \leftarrow \mathsf{Share}(\mathsf{A},k);\ y_A \leftarrow \mathsf{Share}(\mathsf{A},l);\ x_B \leftarrow \mathsf{Share}(\mathsf{B},k);\ y_B \leftarrow \mathsf{Share}(\mathsf{B},l);\ \mathsf{ret}\ b_A \oplus (x_A * y_B) \oplus (x_B * y_A)$

Summarizing, we can rewrite the inductive part of the real protocol as follows:

- $\mathsf{Real}(\epsilon,0)$ is the protocol 0
- $\mathsf{Real}\big(C;\ \textit{input-gate}(i),K+1\big)$ is the composition of the protocol $\mathsf{Real}(C,K)$ with the protocol
  - $\mathsf{OTBit}(\mathsf{A},\mathsf{B},K) := \mathsf{read}\ \mathsf{OTBit}(\mathsf{A},\mathsf{B},K)$
  - $\mathsf{OTBit}(\mathsf{B},\mathsf{A},K) := \mathsf{read}\ \mathsf{OTBit}(\mathsf{B},\mathsf{A},K)$
  - $\begin{cases}\mathsf{Share}(\mathsf{A},K) := \mathsf{read}\ \mathsf{InputShare}(\mathsf{A},\mathsf{A},i) & \text{if } i \text{ is an input of Alice}\\ \mathsf{Share}(\mathsf{A},K) := \mathsf{read}\ \mathsf{InputShare}(\mathsf{A},\mathsf{B},i) & \text{if } i \text{ is an input of Bob}\end{cases}$
  - $\begin{cases}\mathsf{Share}(\mathsf{B},K) := \mathsf{read}\ \mathsf{InputShare}(\mathsf{B},\mathsf{A},i) & \text{if } i \text{ is an input of Alice}\\ \mathsf{Share}(\mathsf{B},K) := \mathsf{read}\ \mathsf{InputShare}(\mathsf{B},\mathsf{B},i) & \text{if } i \text{ is an input of Bob}\end{cases}$
  - $\mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,0) := \mathsf{read}\ \mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,0)$
  - $\mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,1) := \mathsf{read}\ \mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,1)$
  - $\mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,2) := \mathsf{read}\ \mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,2)$
  - $\mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,3) := \mathsf{read}\ \mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,3)$
  - $\mathsf{OTChoice}(\mathsf{B},\mathsf{A},K,0) := \mathsf{read}\ \mathsf{OTChoice}(\mathsf{B},\mathsf{A},K,0)$
  - $\mathsf{OTChoice}(\mathsf{B},\mathsf{A},K,1) := \mathsf{read}\ \mathsf{OTChoice}(\mathsf{B},\mathsf{A},K,1)$
  - $\mathsf{OTOut}(\mathsf{B},\mathsf{A},K) := m_0 \leftarrow \mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,0);\ m_1 \leftarrow \mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,1);\ m_2 \leftarrow \mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,2);\ m_3 \leftarrow \mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,3);\ c_0 \leftarrow \mathsf{OTChoice}(\mathsf{B},\mathsf{A},K,0);\ c_1 \leftarrow \mathsf{OTChoice}(\mathsf{B},\mathsf{A},K,1);$ $\mathsf{if}\ c_0\ \mathsf{then}\ \big(\mathsf{if}\ c_1\ \mathsf{then}\ \mathsf{ret}\ m_3\ \mathsf{else}\ \mathsf{ret}\ m_2\big)\ \mathsf{else}\ \big(\mathsf{if}\ c_1\ \mathsf{then}\ \mathsf{ret}\ m_1\ \mathsf{else}\ \mathsf{ret}\ m_0\big)$
  - $\mathsf{OTBit}(\mathsf{A},\mathsf{B},K)^{\mathsf{A}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{OTBit}(\mathsf{A},\mathsf{B},K)^{\mathsf{A}}_{\mathsf{adv}}$
  - $\mathsf{Share}(\mathsf{A},K)^{\mathsf{A}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{Share}(\mathsf{A},K)$
  - $\mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,0)^{\mathsf{ot}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,0)^{\mathsf{ot}}_{\mathsf{adv}}$
  - $\mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,1)^{\mathsf{ot}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,1)^{\mathsf{ot}}_{\mathsf{adv}}$
  - $\mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,2)^{\mathsf{ot}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,2)^{\mathsf{ot}}_{\mathsf{adv}}$
  - $\mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,3)^{\mathsf{ot}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,3)^{\mathsf{ot}}_{\mathsf{adv}}$
  - $\mathsf{OTChoiceRcvd}(\mathsf{B},\mathsf{A},K,0)^{\mathsf{ot}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{OTChoiceRcvd}(\mathsf{B},\mathsf{A},K,0)^{\mathsf{ot}}_{\mathsf{adv}}$
  - $\mathsf{OTChoiceRcvd}(\mathsf{B},\mathsf{A},K,1)^{\mathsf{ot}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{OTChoiceRcvd}(\mathsf{B},\mathsf{A},K,1)^{\mathsf{ot}}_{\mathsf{adv}}$
- $\mathsf{Real}\big(C;\ \textit{not-gate}(k),K+1\big)$ is the composition of the protocol $\mathsf{Real}(C,K)$ with the protocol
  - $\mathsf{OTBit}(\mathsf{A},\mathsf{B},K) := \mathsf{read}\ \mathsf{OTBit}(\mathsf{A},\mathsf{B},K)$
  - $\mathsf{OTBit}(\mathsf{B},\mathsf{A},K) := \mathsf{read}\ \mathsf{OTBit}(\mathsf{B},\mathsf{A},K)$
  - $\mathsf{Share}(\mathsf{A},K) := \mathsf{read}\ \mathsf{Share}(\mathsf{A},k)$
  - $\mathsf{Share}(\mathsf{B},K) := x_B \leftarrow \mathsf{Share}(\mathsf{B},k);\ \mathsf{ret}\ \neg x_B$
  - $\mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,0) := \mathsf{read}\ \mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,0)$
  - $\mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,1) := \mathsf{read}\ \mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,1)$
  - $\mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,2) := \mathsf{read}\ \mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,2)$
  - $\mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,3) := \mathsf{read}\ \mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,3)$

- $\mathsf{OTChoice}(\mathsf{B},\mathsf{A},K,0) := \text{read } \mathsf{OTChoice}(\mathsf{B},\mathsf{A},K,0)$
- $\mathsf{OTChoice}(\mathsf{B},\mathsf{A},K,1) := \text{read } \mathsf{OTChoice}(\mathsf{B},\mathsf{A},K,1)$
- $\mathsf{OTOut}(\mathsf{B},\mathsf{A},K) := m_0 \leftarrow \mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,0);\ m_1 \leftarrow \mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,1);\ m_2 \leftarrow \mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,2);$ $m_3 \leftarrow \mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,3);\ c_0 \leftarrow \mathsf{OTChoice}(\mathsf{B},\mathsf{A},K,0);\ c_1 \leftarrow \mathsf{OTChoice}(\mathsf{B},\mathsf{A},K,1);$ $\mathsf{if}\ c_0\ \mathsf{then}\ \big(\mathsf{if}\ c_1\ \mathsf{then}\ \mathsf{ret}\ m_3\ \mathsf{else}\ \mathsf{ret}\ m_2\big)\ \mathsf{else}\ \big(\mathsf{if}\ c_1\ \mathsf{then}\ \mathsf{ret}\ m_1\ \mathsf{else}\ \mathsf{ret}\ m_0\big)$
- $\mathsf{OTBit}(\mathsf{A},\mathsf{B},K)^{\mathsf{A}}_{\mathsf{adv}} := \text{read } \mathsf{OTBit}(\mathsf{A},\mathsf{B},K)^{\mathsf{A}}_{\mathsf{adv}}$
- $\mathsf{Share}(\mathsf{A},K)^{\mathsf{A}}_{\mathsf{adv}} := \text{read } \mathsf{Share}(\mathsf{A},K)$
- $\mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,0)^{\mathsf{ot}}_{\mathsf{adv}} := \text{read } \mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,0)^{\mathsf{ot}}_{\mathsf{adv}}$
- $\mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,1)^{\mathsf{ot}}_{\mathsf{adv}} := \text{read } \mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,1)^{\mathsf{ot}}_{\mathsf{adv}}$
- $\mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,2)^{\mathsf{ot}}_{\mathsf{adv}} := \text{read } \mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,2)^{\mathsf{ot}}_{\mathsf{adv}}$
- $\mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,3)^{\mathsf{ot}}_{\mathsf{adv}} := \text{read } \mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,3)^{\mathsf{ot}}_{\mathsf{adv}}$
- $\mathsf{OTChoiceRcvd}(\mathsf{B},\mathsf{A},K,0)^{\mathsf{ot}}_{\mathsf{adv}} := \text{read } \mathsf{OTChoiceRcvd}(\mathsf{B},\mathsf{A},K,0)^{\mathsf{ot}}_{\mathsf{adv}}$
- $\mathsf{OTChoiceRcvd}(\mathsf{B},\mathsf{A},K,1)^{\mathsf{ot}}_{\mathsf{adv}} := \text{read } \mathsf{OTChoiceRcvd}(\mathsf{B},\mathsf{A},K,1)^{\mathsf{ot}}_{\mathsf{adv}}$

- $\mathsf{Real}\big(C;\ \textit{xor-gate}(k,l),K+1\big)$ is the composition of the protocol $\mathsf{Real}(C,K)$ with the protocol
  - $\mathsf{OTBit}(\mathsf{A},\mathsf{B},K) := \text{read } \mathsf{OTBit}(\mathsf{A},\mathsf{B},K)$
  - $\mathsf{OTBit}(\mathsf{B},\mathsf{A},K) := \text{read } \mathsf{OTBit}(\mathsf{B},\mathsf{A},K)$
  - $\mathsf{Share}(\mathsf{A},K) := x_A \leftarrow \mathsf{Share}(\mathsf{A},k);\ y_A \leftarrow \mathsf{Share}(\mathsf{A},l);\ \mathsf{ret}\ x_A \oplus y_A$
  - $\mathsf{Share}(\mathsf{B},K) := x_B \leftarrow \mathsf{Share}(\mathsf{B},k);\ y_B \leftarrow \mathsf{Share}(\mathsf{B},l);\ \mathsf{ret}\ x_B \oplus y_B$
  - $\mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,0) := \text{read } \mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,0)$
  - $\mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,1) := \text{read } \mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,1)$
  - $\mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,2) := \text{read } \mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,2)$
  - $\mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,3) := \text{read } \mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,3)$
  - $\mathsf{OTChoice}(\mathsf{B},\mathsf{A},K,0) := \text{read } \mathsf{OTChoice}(\mathsf{B},\mathsf{A},K,0)$
  - $\mathsf{OTChoice}(\mathsf{B},\mathsf{A},K,1) := \text{read } \mathsf{OTChoice}(\mathsf{B},\mathsf{A},K,1)$
  - $\mathsf{OTOut}(\mathsf{B},\mathsf{A},K) := m_0 \leftarrow \mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,0);\ m_1 \leftarrow \mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,1);\ m_2 \leftarrow \mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,2);$ $m_3 \leftarrow \mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,3);\ c_0 \leftarrow \mathsf{OTChoice}(\mathsf{B},\mathsf{A},K,0);\ c_1 \leftarrow \mathsf{OTChoice}(\mathsf{B},\mathsf{A},K,1);$ $\mathsf{if}\ c_0\ \mathsf{then}\ \big(\mathsf{if}\ c_1\ \mathsf{then}\ \mathsf{ret}\ m_3\ \mathsf{else}\ \mathsf{ret}\ m_2\big)\ \mathsf{else}\ \big(\mathsf{if}\ c_1\ \mathsf{then}\ \mathsf{ret}\ m_1\ \mathsf{else}\ \mathsf{ret}\ m_0\big)$
  - $\mathsf{OTBit}(\mathsf{A},\mathsf{B},K)^{\mathsf{A}}_{\mathsf{adv}} := \text{read } \mathsf{OTBit}(\mathsf{A},\mathsf{B},K)^{\mathsf{A}}_{\mathsf{adv}}$
  - $\mathsf{Share}(\mathsf{A},K)^{\mathsf{A}}_{\mathsf{adv}} := \text{read } \mathsf{Share}(\mathsf{A},K)$
  - $\mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,0)^{\mathsf{ot}}_{\mathsf{adv}} := \text{read } \mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,0)^{\mathsf{ot}}_{\mathsf{adv}}$
  - $\mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,1)^{\mathsf{ot}}_{\mathsf{adv}} := \text{read } \mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,1)^{\mathsf{ot}}_{\mathsf{adv}}$
  - $\mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,2)^{\mathsf{ot}}_{\mathsf{adv}} := \text{read } \mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,2)^{\mathsf{ot}}_{\mathsf{adv}}$
  - $\mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,3)^{\mathsf{ot}}_{\mathsf{adv}} := \text{read } \mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,3)^{\mathsf{ot}}_{\mathsf{adv}}$
  - $\mathsf{OTChoiceRcvd}(\mathsf{B},\mathsf{A},K,0)^{\mathsf{ot}}_{\mathsf{adv}} := \text{read } \mathsf{OTChoiceRcvd}(\mathsf{B},\mathsf{A},K,0)^{\mathsf{ot}}_{\mathsf{adv}}$
  - $\mathsf{OTChoiceRcvd}(\mathsf{B},\mathsf{A},K,1)^{\mathsf{ot}}_{\mathsf{adv}} := \text{read } \mathsf{OTChoiceRcvd}(\mathsf{B},\mathsf{A},K,1)^{\mathsf{ot}}_{\mathsf{adv}}$
- $\mathsf{Real}\big(C;\ \textit{and-gate}(k,l),K+1\big)$ is the composition of the protocol $\mathsf{Real}(C,K)$ with the protocol
  - $\mathsf{OTBit}(\mathsf{A},\mathsf{B},K) := x_A \leftarrow \mathsf{Share}(\mathsf{A},k);\ y_A \leftarrow \mathsf{Share}(\mathsf{A},l);\ \mathsf{samp\ flip}$
  - $\mathsf{OTBit}(\mathsf{B},\mathsf{A},K) := b_A \leftarrow \mathsf{OTBit}(\mathsf{A},\mathsf{B},K);\ x_A \leftarrow \mathsf{Share}(\mathsf{A},k);\ y_A \leftarrow \mathsf{Share}(\mathsf{A},l);\ x_B \leftarrow \mathsf{Share}(\mathsf{B},k);$ $y_B \leftarrow \mathsf{Share}(\mathsf{B},l);\mathsf{ret}\ b_A \oplus (x_A * y_B) \oplus (x_B * y_A)$

- $\mathsf{Share}(\mathsf{A}, K) := b_A \leftarrow \mathsf{OTBit}(\mathsf{A}, \mathsf{B}, K);\ x_A \leftarrow \mathsf{Share}(\mathsf{A}, k);\ y_A \leftarrow \mathsf{Share}(\mathsf{A}, l);\ \mathsf{ret}\ (x_A * y_A) \oplus b_A$
- $\mathsf{Share}(\mathsf{B}, K) := b_B \leftarrow \mathsf{OTBit}(\mathsf{B}, \mathsf{A}, K);\ x_B \leftarrow \mathsf{Share}(\mathsf{B}, k);\ y_B \leftarrow \mathsf{Share}(\mathsf{B}, l);\ \mathsf{ret}\ b_B \oplus (x_B * y_B)$
- $\mathsf{OTMsg}(\mathsf{A}, \mathsf{B}, K, 0) := b_A \leftarrow \mathsf{OTBit}(\mathsf{A}, \mathsf{B}, K);\ x_A \leftarrow \mathsf{Share}(\mathsf{A}, k);\ y_A \leftarrow \mathsf{Share}(\mathsf{A}, l);\ \mathsf{ret}\ b_A$
- $\mathsf{OTMsg}(\mathsf{A}, \mathsf{B}, K, 1) := b_A \leftarrow \mathsf{OTBit}(\mathsf{A}, \mathsf{B}, K);\ x_A \leftarrow \mathsf{Share}(\mathsf{A}, k);\ y_A \leftarrow \mathsf{Share}(\mathsf{A}, l);\ \mathsf{ret}\ b_A \oplus x_A$
- $\mathsf{OTMsg}(\mathsf{A}, \mathsf{B}, K, 2) := b_A \leftarrow \mathsf{OTBit}(\mathsf{A}, \mathsf{B}, K);\ x_A \leftarrow \mathsf{Share}(\mathsf{A}, k);\ y_A \leftarrow \mathsf{Share}(\mathsf{A}, l);\ \mathsf{ret}\ b_A \oplus y_A$
- $\mathsf{OTMsg}(\mathsf{A}, \mathsf{B}, K, 3) := b_A \leftarrow \mathsf{OTBit}(\mathsf{A}, \mathsf{B}, K);\ x_A \leftarrow \mathsf{Share}(\mathsf{A}, k);\ y_A \leftarrow \mathsf{Share}(\mathsf{A}, l);\ \mathsf{ret}\ b_A \oplus x_A \oplus y_A$
- $\mathsf{OTChoice}(\mathsf{B}, \mathsf{A}, K, 0) := \mathsf{read}\ \mathsf{Share}(\mathsf{B}, k)$
- $\mathsf{OTChoice}(\mathsf{B}, \mathsf{A}, K, 1) := \mathsf{read}\ \mathsf{Share}(\mathsf{B}, l)$
- $\mathsf{OTOut}(\mathsf{B}, \mathsf{A}, K) := b_A \leftarrow \mathsf{OTBit}(\mathsf{A}, \mathsf{B}, K);\ x_A \leftarrow \mathsf{Share}(\mathsf{A}, k);\ y_A \leftarrow \mathsf{Share}(\mathsf{A}, l);\ x_B \leftarrow \mathsf{Share}(\mathsf{B}, k);\ y_B \leftarrow \mathsf{Share}(\mathsf{B}, l);\ \mathsf{ret}\ b_A \oplus (x_A * y_B) \oplus (x_B * y_A)$
- $\mathsf{OTBit}(\mathsf{A}, \mathsf{B}, K)^{\mathsf{A}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{OTBit}(\mathsf{A}, \mathsf{B}, K)$
- $\mathsf{Share}(\mathsf{A}, K)^{\mathsf{A}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{Share}(\mathsf{A}, K)$
- $\mathsf{OTMsg}(\mathsf{A}, \mathsf{B}, K, 0)^{\mathsf{ot}}_{\mathsf{adv}} := b_A \leftarrow \mathsf{OTBit}(\mathsf{A}, \mathsf{B}, K);\ x_A \leftarrow \mathsf{Share}(\mathsf{A}, k);\ y_A \leftarrow \mathsf{Share}(\mathsf{A}, l);\ \mathsf{ret}\ b_A$
- $\mathsf{OTMsg}(\mathsf{A}, \mathsf{B}, K, 1)^{\mathsf{ot}}_{\mathsf{adv}} := b_A \leftarrow \mathsf{OTBit}(\mathsf{A}, \mathsf{B}, K);\ x_A \leftarrow \mathsf{Share}(\mathsf{A}, k);\ y_A \leftarrow \mathsf{Share}(\mathsf{A}, l);\ \mathsf{ret}\ b_A \oplus x_A$
- $\mathsf{OTMsg}(\mathsf{A}, \mathsf{B}, K, 2)^{\mathsf{ot}}_{\mathsf{adv}} := b_A \leftarrow \mathsf{OTBit}(\mathsf{A}, \mathsf{B}, K);\ x_A \leftarrow \mathsf{Share}(\mathsf{A}, k);\ y_A \leftarrow \mathsf{Share}(\mathsf{A}, l);\ \mathsf{ret}\ b_A \oplus y_A$
- $\mathsf{OTMsg}(\mathsf{A}, \mathsf{B}, K, 3)^{\mathsf{ot}}_{\mathsf{adv}} := b_A \leftarrow \mathsf{OTBit}(\mathsf{A}, \mathsf{B}, K);\ x_A \leftarrow \mathsf{Share}(\mathsf{A}, k);\ y_A \leftarrow \mathsf{Share}(\mathsf{A}, l);\ \mathsf{ret}\ b_A \oplus x_A \oplus y_A$
- $\mathsf{OTChoiceRcvd}(\mathsf{B}, \mathsf{A}, K, 0)^{\mathsf{ot}}_{\mathsf{adv}} := x_B \leftarrow \mathsf{Share}(\mathsf{B}, k);\ \mathsf{ret}\ \checkmark$
- $\mathsf{OTChoiceRcvd}(\mathsf{B}, \mathsf{A}, K, 1)^{\mathsf{ot}}_{\mathsf{adv}} := y_B \leftarrow \mathsf{Share}(\mathsf{B}, l);\ \mathsf{ret}\ \checkmark$

We can now split the real part of the protocol into four parts, each defined by induction on the circuit. The first part $\mathsf{Alice}(C, K)$ defines the channels that can be seen as comprising Alice's part of the computation, $\mathsf{OTBit}(\mathsf{A}, \mathsf{B}, -)$ and $\mathsf{Share}(\mathsf{A}, -)$:

- $\mathsf{Alice}(\epsilon, 0)$ is the protocol $\mathsf{0}$
- $\mathsf{Alice}\big(C;\ \textit{input-gate}(i), K+1\big)$ is the composition of the protocol $\mathsf{Alice}(C, K)$ with the protocol
  - $\mathsf{OTBit}(\mathsf{A}, \mathsf{B}, K) := \mathsf{read}\ \mathsf{OTBit}(\mathsf{A}, \mathsf{B}, K)$
  - $\begin{cases} \mathsf{Share}(\mathsf{A}, K) := \mathsf{read}\ \mathsf{InputShare}(\mathsf{A}, \mathsf{A}, i) & \text{if } i \text{ is an input of Alice} \\ \mathsf{Share}(\mathsf{A}, K) := \mathsf{read}\ \mathsf{InputShare}(\mathsf{A}, \mathsf{B}, i) & \text{if } i \text{ is an input of Bob} \end{cases}$
- $\mathsf{Alice}\big(C;\ \textit{not-gate}(k), K+1\big)$ is the composition of the protocol $\mathsf{Alice}(C, K)$ with the protocol
  - $\mathsf{OTBit}(\mathsf{A}, \mathsf{B}, K) := \mathsf{read}\ \mathsf{OTBit}(\mathsf{A}, \mathsf{B}, K)$
  - $\mathsf{Share}(\mathsf{A}, K) := \mathsf{read}\ \mathsf{Share}(\mathsf{A}, k)$
- $\mathsf{Alice}\big(C;\ \textit{xor-gate}(k, l), K+1\big)$ is the composition of the protocol $\mathsf{Alice}(C, K)$ with the protocol
  - $\mathsf{OTBit}(\mathsf{A}, \mathsf{B}, K) := \mathsf{read}\ \mathsf{OTBit}(\mathsf{A}, \mathsf{B}, K)$
  - $\mathsf{Share}(\mathsf{A}, K) := x_A \leftarrow \mathsf{Share}(\mathsf{A}, k);\ y_A \leftarrow \mathsf{Share}(\mathsf{A}, l);\ \mathsf{ret}\ x_A \oplus y_A$
- $\mathsf{Alice}\big(C;\ \textit{and-gate}(k, l), K+1\big)$ is the composition of the protocol $\mathsf{Alice}(C, K)$ with the protocol
  - $\mathsf{OTBit}(\mathsf{A}, \mathsf{B}, K) := x_A \leftarrow \mathsf{Share}(\mathsf{A}, k);\ y_A \leftarrow \mathsf{Share}(\mathsf{A}, l);\ \mathsf{samp\ flip}$
  - $\mathsf{Share}(\mathsf{A}, K) := b_A \leftarrow \mathsf{OTBit}(\mathsf{A}, \mathsf{B}, K);\ x_A \leftarrow \mathsf{Share}(\mathsf{A}, k);\ y_A \leftarrow \mathsf{Share}(\mathsf{A}, l);\ \mathsf{ret}\ (x_A * y_A) \oplus b_A$

Analogously, the part $\mathsf{Bob}(C, K)$ defines the channels that can be seen as comprising Bob's part of the computation, $\mathsf{OTBit}(\mathsf{B}, \mathsf{A}, -)$ and $\mathsf{Share}(\mathsf{B}, -)$:

- $\mathsf{Bob}(\epsilon, 0)$ is the protocol $\mathsf{0}$
- $\mathsf{Bob}\big(C;\ \textit{input-gate}(i), K+1\big)$ is the composition of the protocol $\mathsf{Bob}(C, K)$ with the protocol
  - $\mathsf{OTBit}(\mathsf{B}, \mathsf{A}, K) := \mathsf{read}\ \mathsf{OTBit}(\mathsf{B}, \mathsf{A}, K)$
  - $\begin{cases} \mathsf{Share}(\mathsf{B}, K) := \mathsf{read}\ \mathsf{InputShare}(\mathsf{B}, \mathsf{A}, i) & \text{if } i \text{ is an input of Alice} \\ \mathsf{Share}(\mathsf{B}, K) := \mathsf{read}\ \mathsf{InputShare}(\mathsf{B}, \mathsf{B}, i) & \text{if } i \text{ is an input of Bob} \end{cases}$
- $\mathsf{Bob}\big(C;\ \textit{not-gate}(k), K+1\big)$ is the composition of the protocol $\mathsf{Bob}(C, K)$ with the protocol
  - $\mathsf{OTBit}(\mathsf{B}, \mathsf{A}, K) := \mathsf{read}\ \mathsf{OTBit}(\mathsf{B}, \mathsf{A}, K)$
  - $\mathsf{Share}(\mathsf{B}, K) := x_B \leftarrow \mathsf{Share}(\mathsf{B}, k);\ \mathsf{ret}\ \neg x_B$
- $\mathsf{Bob}\big(C;\ \textit{xor-gate}(k, l), K+1\big)$ is the composition of the protocol $\mathsf{Bob}(C, K)$ with the protocol
  - $\mathsf{OTBit}(\mathsf{B}, \mathsf{A}, K) := \mathsf{read}\ \mathsf{OTBit}(\mathsf{B}, \mathsf{A}, K)$
  - $\mathsf{Share}(\mathsf{B}, K) := x_B \leftarrow \mathsf{Share}(\mathsf{B}, k);\ y_B \leftarrow \mathsf{Share}(\mathsf{B}, l);\ \mathsf{ret}\ x_B \oplus y_B$
- $\mathsf{Bob}\big(C;\ \textit{and-gate}(k, l), K+1\big)$ is the composition of the protocol $\mathsf{Bob}(C, K)$ with the protocol
  - $\mathsf{OTBit}(\mathsf{B}, \mathsf{A}, K) := b_A \leftarrow \mathsf{OTBit}(\mathsf{A}, \mathsf{B}, K);\ x_A \leftarrow \mathsf{Share}(\mathsf{A}, k);\ y_A \leftarrow \mathsf{Share}(\mathsf{A}, l);\ x_B \leftarrow \mathsf{Share}(\mathsf{B}, k);$ $y_B \leftarrow \mathsf{Share}(\mathsf{B}, l);\ \mathsf{ret}\ b_A \oplus (x_A * y_B) \oplus (x_B * y_A)$
  - $\mathsf{Share}(\mathsf{B}, K) := b_B \leftarrow \mathsf{OTBit}(\mathsf{B}, \mathsf{A}, K);\ x_B \leftarrow \mathsf{Share}(\mathsf{B}, k);\ y_B \leftarrow \mathsf{Share}(\mathsf{B}, l);\ \mathsf{ret}\ b_B \oplus (x_B * y_B)$

The third part $\mathsf{Adv}(C, K)$ defines all the leakage channels intended for the adversary:

- $\mathsf{Adv}(\epsilon, 0)$ is the protocol $\mathsf{0}$
- $\mathsf{Adv}\big(C;\ \textit{input-gate}(i), K+1\big)$ is the composition of the protocol $\mathsf{Adv}(C, K)$ with the protocol
  - $\mathsf{OTBit}(\mathsf{A}, \mathsf{B}, K)^{\mathsf{A}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{OTBit}(\mathsf{A}, \mathsf{B}, K)^{\mathsf{A}}_{\mathsf{adv}}$
  - $\mathsf{Share}(\mathsf{A}, K)^{\mathsf{A}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{Share}(\mathsf{A}, K)$
  - $\mathsf{OTMsg}(\mathsf{A}, \mathsf{B}, K, 0)^{\mathsf{ot}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{OTMsg}(\mathsf{A}, \mathsf{B}, K, 0)^{\mathsf{ot}}_{\mathsf{adv}}$
  - $\mathsf{OTMsg}(\mathsf{A}, \mathsf{B}, K, 1)^{\mathsf{ot}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{OTMsg}(\mathsf{A}, \mathsf{B}, K, 1)^{\mathsf{ot}}_{\mathsf{adv}}$
  - $\mathsf{OTMsg}(\mathsf{A}, \mathsf{B}, K, 2)^{\mathsf{ot}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{OTMsg}(\mathsf{A}, \mathsf{B}, K, 2)^{\mathsf{ot}}_{\mathsf{adv}}$
  - $\mathsf{OTMsg}(\mathsf{A}, \mathsf{B}, K, 3)^{\mathsf{ot}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{OTMsg}(\mathsf{A}, \mathsf{B}, K, 3)^{\mathsf{ot}}_{\mathsf{adv}}$
  - $\mathsf{OTChoiceRcvd}(\mathsf{B}, \mathsf{A}, k, 0)^{\mathsf{ot}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{OTChoiceRcvd}(\mathsf{B}, \mathsf{A}, k, 0)^{\mathsf{ot}}_{\mathsf{adv}}$
  - $\mathsf{OTChoiceRcvd}(\mathsf{B}, \mathsf{A}, k, 1)^{\mathsf{ot}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{OTChoiceRcvd}(\mathsf{B}, \mathsf{A}, k, 1)^{\mathsf{ot}}_{\mathsf{adv}}$
- $\mathsf{Adv}\big(C;\ \textit{not-gate}(k), K+1\big)$ is the composition of the protocol $\mathsf{Adv}(C, K)$ with the protocol
  - $\mathsf{OTBit}(\mathsf{A}, \mathsf{B}, K)^{\mathsf{A}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{OTBit}(\mathsf{A}, \mathsf{B}, K)^{\mathsf{A}}_{\mathsf{adv}}$
  - $\mathsf{Share}(\mathsf{A}, K)^{\mathsf{A}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{Share}(\mathsf{A}, K)$
  - $\mathsf{OTMsg}(\mathsf{A}, \mathsf{B}, K, 0)^{\mathsf{ot}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{OTMsg}(\mathsf{A}, \mathsf{B}, K, 0)^{\mathsf{ot}}_{\mathsf{adv}}$
  - $\mathsf{OTMsg}(\mathsf{A}, \mathsf{B}, K, 1)^{\mathsf{ot}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{OTMsg}(\mathsf{A}, \mathsf{B}, K, 1)^{\mathsf{ot}}_{\mathsf{adv}}$
  - $\mathsf{OTMsg}(\mathsf{A}, \mathsf{B}, K, 2)^{\mathsf{ot}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{OTMsg}(\mathsf{A}, \mathsf{B}, K, 2)^{\mathsf{ot}}_{\mathsf{adv}}$
  - $\mathsf{OTMsg}(\mathsf{A}, \mathsf{B}, K, 3)^{\mathsf{ot}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{OTMsg}(\mathsf{A}, \mathsf{B}, K, 3)^{\mathsf{ot}}_{\mathsf{adv}}$
  - $\mathsf{OTChoiceRcvd}(\mathsf{B}, \mathsf{A}, k, 0)^{\mathsf{ot}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{OTChoiceRcvd}(\mathsf{B}, \mathsf{A}, k, 0)^{\mathsf{ot}}_{\mathsf{adv}}$
  - $\mathsf{OTChoiceRcvd}(\mathsf{B}, \mathsf{A}, k, 1)^{\mathsf{ot}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{OTChoiceRcvd}(\mathsf{B}, \mathsf{A}, k, 1)^{\mathsf{ot}}_{\mathsf{adv}}$
- $\mathsf{Adv}\big(C;\ \textit{xor-gate}(k, l), K+1\big)$ is the composition of the protocol $\mathsf{Adv}(C, K)$ with the protocol

  - $\mathsf{OTBit}(\mathsf{A},\mathsf{B},K)^{\mathsf{A}}_{\mathsf{adv}} := \text{read } \mathsf{OTBit}(\mathsf{A},\mathsf{B},K)^{\mathsf{A}}_{\mathsf{adv}}$
  - $\mathsf{Share}(\mathsf{A},K)^{\mathsf{A}}_{\mathsf{adv}} := \text{read } \mathsf{Share}(\mathsf{A},K)$
  - $\mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,0)^{\mathsf{ot}}_{\mathsf{adv}} := \text{read } \mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,0)^{\mathsf{ot}}_{\mathsf{adv}}$
  - $\mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,1)^{\mathsf{ot}}_{\mathsf{adv}} := \text{read } \mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,1)^{\mathsf{ot}}_{\mathsf{adv}}$
  - $\mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,2)^{\mathsf{ot}}_{\mathsf{adv}} := \text{read } \mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,2)^{\mathsf{ot}}_{\mathsf{adv}}$
  - $\mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,3)^{\mathsf{ot}}_{\mathsf{adv}} := \text{read } \mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,3)^{\mathsf{ot}}_{\mathsf{adv}}$
  - $\mathsf{OTChoiceRcvd}(\mathsf{B},\mathsf{A},k,0)^{\mathsf{ot}}_{\mathsf{adv}} := \text{read } \mathsf{OTChoiceRcvd}(\mathsf{B},\mathsf{A},k,0)^{\mathsf{ot}}_{\mathsf{adv}}$
  - $\mathsf{OTChoiceRcvd}(\mathsf{B},\mathsf{A},k,1)^{\mathsf{ot}}_{\mathsf{adv}} := \text{read } \mathsf{OTChoiceRcvd}(\mathsf{B},\mathsf{A},k,1)^{\mathsf{ot}}_{\mathsf{adv}}$
- $\mathsf{Adv}\big(C;\ \textit{and-gate}(k,l), K+1\big)$ is the composition of the protocol $\mathsf{Adv}(C,K)$ with the protocol
  - $\mathsf{OTBit}(\mathsf{A},\mathsf{B},K)^{\mathsf{A}}_{\mathsf{adv}} := \text{read } \mathsf{OTBit}(\mathsf{A},\mathsf{B},K)$
  - $\mathsf{Share}(\mathsf{A},K)^{\mathsf{A}}_{\mathsf{adv}} := \text{read } \mathsf{Share}(\mathsf{A},K)$
  - $\mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,0)^{\mathsf{ot}}_{\mathsf{adv}} := b_A \leftarrow \mathsf{OTBit}(\mathsf{A},\mathsf{B},K);\ x_A \leftarrow \mathsf{Share}(\mathsf{A},k);\ y_A \leftarrow \mathsf{Share}(\mathsf{A},l);\ \text{ret } b_A$
  - $\mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,1)^{\mathsf{ot}}_{\mathsf{adv}} := b_A \leftarrow \mathsf{OTBit}(\mathsf{A},\mathsf{B},K);\ x_A \leftarrow \mathsf{Share}(\mathsf{A},k);\ y_A \leftarrow \mathsf{Share}(\mathsf{A},l);\ \text{ret } b_A \oplus x_A$
  - $\mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,2)^{\mathsf{ot}}_{\mathsf{adv}} := b_A \leftarrow \mathsf{OTBit}(\mathsf{A},\mathsf{B},K);\ x_A \leftarrow \mathsf{Share}(\mathsf{A},k);\ y_A \leftarrow \mathsf{Share}(\mathsf{A},l);\ \text{ret } b_A \oplus y_A$
  - $\mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,3)^{\mathsf{ot}}_{\mathsf{adv}} := b_A \leftarrow \mathsf{OTBit}(\mathsf{A},\mathsf{B},K);\ x_A \leftarrow \mathsf{Share}(\mathsf{A},k);\ y_A \leftarrow \mathsf{Share}(\mathsf{A},l);\ \text{ret } b_A \oplus x_A \oplus y_A$
  - $\mathsf{OTChoiceRcvd}(\mathsf{B},\mathsf{A},k,0)^{\mathsf{ot}}_{\mathsf{adv}} := x_B \leftarrow \mathsf{Share}(\mathsf{B},k);\ \text{ret } \checkmark$
  - $\mathsf{OTChoiceRcvd}(\mathsf{B},\mathsf{A},k,1)^{\mathsf{ot}}_{\mathsf{adv}} := y_B \leftarrow \mathsf{Share}(\mathsf{B},l);\ \text{ret } \checkmark$

Finally, part $\mathsf{1OutOf4OT}(C,K)$ defines the channels intended for communication with the OT functionality:

- $\mathsf{1OutOf4OT}(\epsilon,0)$ is the protocol $\mathsf{0}$
- $\mathsf{1OutOf4OT}\big(C;\ \textit{input-gate}(i), K+1\big)$ is the composition of the protocol $\mathsf{1OutOf4OT}(C,K)$ with the protocol
  - $\mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,0) := \text{read } \mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,0)$
  - $\mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,1) := \text{read } \mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,1)$
  - $\mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,2) := \text{read } \mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,2)$
  - $\mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,3) := \text{read } \mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,3)$
  - $\mathsf{OTChoice}(\mathsf{B},\mathsf{A},K,0) := \text{read } \mathsf{OTChoice}(\mathsf{B},\mathsf{A},K,0)$
  - $\mathsf{OTChoice}(\mathsf{B},\mathsf{A},K,1) := \text{read } \mathsf{OTChoice}(\mathsf{B},\mathsf{A},K,1)$
  - $\mathsf{OTOut}(\mathsf{B},\mathsf{A},K) := m_0 \leftarrow \mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,0);\ m_1 \leftarrow \mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,1);\ m_2 \leftarrow \mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,2);$ $m_3 \leftarrow \mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,3);\ c_0 \leftarrow \mathsf{OTChoice}(\mathsf{B},\mathsf{A},K,0);\ c_1 \leftarrow \mathsf{OTChoice}(\mathsf{B},\mathsf{A},K,1);$ $\text{if } c_0 \text{ then } \big(\text{if } c_1 \text{ then ret } m_3 \text{ else ret } m_2\big) \text{ else } \big(\text{if } c_1 \text{ then ret } m_1 \text{ else ret } m_0\big)$
- $\mathsf{1OutOf4OT}\big(C;\ \textit{not-gate}(k), K+1\big)$ is the composition of the protocol $\mathsf{1OutOf4OT}(C,K)$ with the protocol
  - $\mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,0) := \text{read } \mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,0)$
  - $\mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,1) := \text{read } \mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,1)$
  - $\mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,2) := \text{read } \mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,2)$
  - $\mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,3) := \text{read } \mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,3)$
  - $\mathsf{OTChoice}(\mathsf{B},\mathsf{A},K,0) := \text{read } \mathsf{OTChoice}(\mathsf{B},\mathsf{A},K,0)$
  - $\mathsf{OTChoice}(\mathsf{B},\mathsf{A},K,1) := \text{read } \mathsf{OTChoice}(\mathsf{B},\mathsf{A},K,1)$
  - $\mathsf{OTOut}(\mathsf{B},\mathsf{A},K) := m_0 \leftarrow \mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,0);\ m_1 \leftarrow \mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,1);\ m_2 \leftarrow \mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,2);$ $m_3 \leftarrow \mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,3);\ c_0 \leftarrow \mathsf{OTChoice}(\mathsf{B},\mathsf{A},K,0);\ c_1 \leftarrow \mathsf{OTChoice}(\mathsf{B},\mathsf{A},K,1);$ $\text{if } c_0 \text{ then } \big(\text{if } c_1 \text{ then ret } m_3 \text{ else ret } m_2\big) \text{ else } \big(\text{if } c_1 \text{ then ret } m_1 \text{ else ret } m_0\big)$

- $\mathsf{1OutOf4OT}\big(C;\ \textit{xor-gate}(k,l), K+1\big)$ is the composition of the protocol $\mathsf{1OutOf4OT}(C,K)$ with the protocol
  - $\mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,0) := \mathsf{read}\ \mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,0)$
  - $\mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,1) := \mathsf{read}\ \mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,1)$
  - $\mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,2) := \mathsf{read}\ \mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,2)$
  - $\mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,3) := \mathsf{read}\ \mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,3)$
  - $\mathsf{OTChoice}(\mathsf{B},\mathsf{A},K,0) := \mathsf{read}\ \mathsf{OTChoice}(\mathsf{B},\mathsf{A},K,0)$
  - $\mathsf{OTChoice}(\mathsf{B},\mathsf{A},K,1) := \mathsf{read}\ \mathsf{OTChoice}(\mathsf{B},\mathsf{A},K,1)$
  - $\mathsf{OTOut}(\mathsf{B},\mathsf{A},K) := m_0 \leftarrow \mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,0);\ m_1 \leftarrow \mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,1);\ m_2 \leftarrow \mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,2);$ $m_3 \leftarrow \mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,3);\ c_0 \leftarrow \mathsf{OTChoice}(\mathsf{B},\mathsf{A},K,0);\ c_1 \leftarrow \mathsf{OTChoice}(\mathsf{B},\mathsf{A},K,1);$ $\mathsf{if}\ c_0\ \mathsf{then}\ \big(\mathsf{if}\ c_1\ \mathsf{then}\ \mathsf{ret}\ m_3\ \mathsf{else}\ \mathsf{ret}\ m_2\big)\ \mathsf{else}\ \big(\mathsf{if}\ c_1\ \mathsf{then}\ \mathsf{ret}\ m_1\ \mathsf{else}\ \mathsf{ret}\ m_0\big)$
- $\mathsf{1OutOf4OT}\big(C;\ \textit{and-gate}(k,l), K+1\big)$ is the composition of the protocol $\mathsf{1OutOf4OT}(C,K)$ with the protocol
  - $\mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,0) := b_A \leftarrow \mathsf{OTBit}(\mathsf{A},\mathsf{B},K);\ x_A \leftarrow \mathsf{Share}(\mathsf{A},k);\ y_A \leftarrow \mathsf{Share}(\mathsf{A},l);\ \mathsf{ret}\ b_A$
  - $\mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,1) := b_A \leftarrow \mathsf{OTBit}(\mathsf{A},\mathsf{B},K);\ x_A \leftarrow \mathsf{Share}(\mathsf{A},k);\ y_A \leftarrow \mathsf{Share}(\mathsf{A},l);\ \mathsf{ret}\ b_A \oplus x_A$
  - $\mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,2) := b_A \leftarrow \mathsf{OTBit}(\mathsf{A},\mathsf{B},K);\ x_A \leftarrow \mathsf{Share}(\mathsf{A},k);\ y_A \leftarrow \mathsf{Share}(\mathsf{A},l);\ \mathsf{ret}\ b_A \oplus y_A$
  - $\mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,3) := b_A \leftarrow \mathsf{OTBit}(\mathsf{A},\mathsf{B},K);\ x_A \leftarrow \mathsf{Share}(\mathsf{A},k);\ y_A \leftarrow \mathsf{Share}(\mathsf{A},l);\ \mathsf{ret}\ b_A \oplus x_A \oplus y_A$
  - $\mathsf{OTChoice}(\mathsf{B},\mathsf{A},K,0) := \mathsf{read}\ \mathsf{Share}(\mathsf{B},k)$
  - $\mathsf{OTChoice}(\mathsf{B},\mathsf{A},K,1) := \mathsf{read}\ \mathsf{Share}(\mathsf{B},l)$
  - $\mathsf{OTOut}(\mathsf{B},\mathsf{A},K) := b_A \leftarrow \mathsf{OTBit}(\mathsf{A},\mathsf{B},K);\ x_A \leftarrow \mathsf{Share}(\mathsf{A},k);\ y_A \leftarrow \mathsf{Share}(\mathsf{A},l);\ x_B \leftarrow \mathsf{Share}(\mathsf{B},k);$ $y_B \leftarrow \mathsf{Share}(\mathsf{B},l);\ \mathsf{ret}\ b_A \oplus (x_A * y_B) \oplus (x_B * y_A)$

We notice that after the earlier simplification, none of the channels defined by $\mathsf{1OutOf4OT}(C,K)$ are utilized anywhere outside of $\mathsf{1OutOf4OT}(C,K)$, and as such we may discard this protocol fragment entirely.

9.4.3 Simplifying The Real Protocol: The Final Stage

Our goal here is to eliminate the internal channels $\mathsf{SendFinalShare}(\mathsf{A},-)$ and $\mathsf{SendFinalShare}(\mathsf{B},-)$. If k is an output wire, this is a simple substitution. Otherwise, the channels $\mathsf{SendFinalShare}(\mathsf{A},k)$ and $\mathsf{SendFinalShare}(\mathsf{B},k)$ diverge, and so do channels $\mathsf{SendFinalShare}(\mathsf{A},k)^{\mathsf{A}}_{\mathsf{adv}}$, $\mathsf{SendFinalShare}(\mathsf{B},k)^{\mathsf{A}}_{\mathsf{adv}}$ and $\mathsf{Out}(\mathsf{A},k)$, $\mathsf{Out}(\mathsf{B},k)$:

- $\begin{cases} \mathsf{SendFinalShare}(\mathsf{A},k)^{\mathsf{A}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{Share}(\mathsf{A},k) & \text{if wire } 0 \leqslant k < K \text{ is output} \\ \mathsf{SendFinalShare}(\mathsf{A},k)^{\mathsf{A}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{SendFinalShare}(\mathsf{A},k)^{\mathsf{A}}_{\mathsf{adv}} & \text{otherwise} \end{cases}$
- $\begin{cases} \mathsf{SendFinalShare}(\mathsf{B},k)^{\mathsf{A}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{Share}(\mathsf{B},k) & \text{if wire } 0 \leqslant k < K \text{ is output} \\ \mathsf{SendFinalShare}(\mathsf{B},k)^{\mathsf{A}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{SendFinalShare}(\mathsf{B},k)^{\mathsf{A}}_{\mathsf{adv}} & \text{otherwise} \end{cases}$
- $\begin{cases} \mathsf{Out}(\mathsf{A},k) := x_A \leftarrow \mathsf{Share}(\mathsf{A},k);\ x_B \leftarrow \mathsf{Share}(\mathsf{B},k);\ \mathsf{ret}\ x_A \oplus x_B & \text{if wire } 0 \leqslant k < K \text{ is output} \\ \mathsf{Out}(\mathsf{A},k) := \mathsf{read}\ \mathsf{Out}(\mathsf{A},k) & \text{otherwise} \end{cases}$
- $\begin{cases} \mathsf{Out}(\mathsf{B},k) := x_A \leftarrow \mathsf{Share}(\mathsf{A},k);\ x_B \leftarrow \mathsf{Share}(\mathsf{B},k);\ \mathsf{ret}\ x_A \oplus x_B & \text{if wire } 0 \leqslant k < K \text{ is output} \\ \mathsf{Out}(\mathsf{B},k) := \mathsf{read}\ \mathsf{Out}(\mathsf{B},k) & \text{otherwise} \end{cases}$
- $\mathsf{Out}(\mathsf{A},k)^{\mathsf{A}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{Out}(\mathsf{A},k)$ for $0 \leqslant k < K$

9.4.4 The Timing of Shares

Since the simulator does not have access to the value of Bob's inputs, any dependency on these needs to be eliminated. And since the value of Bob's shares depends on the value of his inputs, the shares themselves need to be eliminated. Upon examining the inductive part of the real protocol, we see that the only place where we depend on Bob's shares is in $\mathsf{Adv}(C, K)$, when we leak the timing of Bob's messages to the OT functionality in the case of an *and* gate, *i.e.*, in the channels $\mathsf{OTChoiceRcvd}(\mathsf{B}, \mathsf{A}, -, -)^{\mathsf{ot}}_{\mathsf{adv}}$. But even in this case, the actual *value* of Bob's shares is immaterial - it is only the timing information that matters. To this end, we introduce new internal channels

- $\mathsf{InOk}(\mathsf{A}, i) := x \leftarrow \mathsf{In}(\mathsf{A}, i);\ \mathsf{ret}\ \checkmark$ for $0 \leqslant i < N$
- $\mathsf{InOk}(\mathsf{B}, i) := x \leftarrow \mathsf{In}(\mathsf{B}, i);\ \mathsf{ret}\ \checkmark$ for $0 \leqslant i < M$
- $\mathsf{InputShareOk}(\mathsf{A}, \mathsf{A}, i) := x_A \leftarrow \mathsf{InputShare}(\mathsf{A}, \mathsf{A}, i);\ \mathsf{ret}\ \checkmark$ for $0 \leqslant i < N$
- $\mathsf{InputShareOk}(\mathsf{A}, \mathsf{B}, i) := x_A \leftarrow \mathsf{InputShare}(\mathsf{A}, \mathsf{B}, i);\ \mathsf{ret}\ \checkmark$ for $0 \leqslant i < M$
- $\mathsf{InputShareOk}(\mathsf{B}, \mathsf{A}, i) := x_B \leftarrow \mathsf{InputShare}(\mathsf{B}, \mathsf{A}, i);\ \mathsf{ret}\ \checkmark$ for $0 \leqslant i < N$
- $\mathsf{InputShareOk}(\mathsf{B}, \mathsf{B}, i) := x_B \leftarrow \mathsf{InputShare}(\mathsf{B}, \mathsf{B}, i);\ \mathsf{ret}\ \checkmark$ for $0 \leqslant i < M$
- $\mathsf{OTBitOk}(\mathsf{A}, \mathsf{B}, k) := b_A \leftarrow \mathsf{OTBitOk}(\mathsf{A}, \mathsf{B}, k);\ \mathsf{ret}\ \checkmark$ for $0 \leqslant k < K$
- $\mathsf{OTBitOk}(\mathsf{B}, \mathsf{A}, k) := b_B \leftarrow \mathsf{OTBitOk}(\mathsf{B}, \mathsf{A}, k);\ \mathsf{ret}\ \checkmark$ for $0 \leqslant k < K$
- $\mathsf{ShareOk}(\mathsf{A}, k) := x_A \leftarrow \mathsf{Share}(\mathsf{A}, k);\ \mathsf{ret}\ \checkmark$ for $0 \leqslant k < K$
- $\mathsf{ShareOk}(\mathsf{B}, k) := x_B \leftarrow \mathsf{Share}(\mathsf{B}, k);\ \mathsf{ret}\ \checkmark$ for $0 \leqslant k < K$

to record the relevant timing information. In the presence of these channels, we may rewrite the protocol $\mathsf{Adv}(C, K)$ so that no explicit reference to the value of Bob's shares occurs:

- $\mathsf{Adv}(\epsilon, 0)$ is the protocol $\mathsf{0}$
- $\mathsf{Adv}\big(C;\ \textit{input-gate}(i), K+1\big)$ is the composition of the protocol $\mathsf{Adv}(C, K)$ with the protocol
  - $\mathsf{OTBit}(\mathsf{A}, \mathsf{B}, K)^{\mathsf{A}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{OTBit}(\mathsf{A}, \mathsf{B}, K)^{\mathsf{A}}_{\mathsf{adv}}$
  - $\mathsf{Share}(\mathsf{A}, K)^{\mathsf{A}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{Share}(\mathsf{A}, K)$
  - $\mathsf{OTMsg}(\mathsf{A}, \mathsf{B}, K, 0)^{\mathsf{ot}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{OTMsg}(\mathsf{A}, \mathsf{B}, K, 0)^{\mathsf{ot}}_{\mathsf{adv}}$
  - $\mathsf{OTMsg}(\mathsf{A}, \mathsf{B}, K, 1)^{\mathsf{ot}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{OTMsg}(\mathsf{A}, \mathsf{B}, K, 1)^{\mathsf{ot}}_{\mathsf{adv}}$
  - $\mathsf{OTMsg}(\mathsf{A}, \mathsf{B}, K, 2)^{\mathsf{ot}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{OTMsg}(\mathsf{A}, \mathsf{B}, K, 2)^{\mathsf{ot}}_{\mathsf{adv}}$
  - $\mathsf{OTMsg}(\mathsf{A}, \mathsf{B}, K, 3)^{\mathsf{ot}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{OTMsg}(\mathsf{A}, \mathsf{B}, K, 3)^{\mathsf{ot}}_{\mathsf{adv}}$
  - $\mathsf{OTChoiceRcvd}(\mathsf{B}, \mathsf{A}, K, 0)^{\mathsf{ot}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{OTChoiceRcvd}(\mathsf{B}, \mathsf{A}, K, 0)^{\mathsf{ot}}_{\mathsf{adv}}$
  - $\mathsf{OTChoiceRcvd}(\mathsf{B}, \mathsf{A}, K, 1)^{\mathsf{ot}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{OTChoiceRcvd}(\mathsf{B}, \mathsf{A}, K, 1)^{\mathsf{ot}}_{\mathsf{adv}}$
- $\mathsf{Adv}\big(C;\ \textit{not-gate}(k), K+1\big)$ is the composition of the protocol $\mathsf{Adv}(C, K)$ with the protocol
  - $\mathsf{OTBit}(\mathsf{A}, \mathsf{B}, K)^{\mathsf{A}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{OTBit}(\mathsf{A}, \mathsf{B}, K)^{\mathsf{A}}_{\mathsf{adv}}$
  - $\mathsf{Share}(\mathsf{A}, K)^{\mathsf{A}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{Share}(\mathsf{A}, K)$
  - $\mathsf{OTMsg}(\mathsf{A}, \mathsf{B}, K, 0)^{\mathsf{ot}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{OTMsg}(\mathsf{A}, \mathsf{B}, K, 0)^{\mathsf{ot}}_{\mathsf{adv}}$
  - $\mathsf{OTMsg}(\mathsf{A}, \mathsf{B}, K, 1)^{\mathsf{ot}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{OTMsg}(\mathsf{A}, \mathsf{B}, K, 1)^{\mathsf{ot}}_{\mathsf{adv}}$
  - $\mathsf{OTMsg}(\mathsf{A}, \mathsf{B}, K, 2)^{\mathsf{ot}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{OTMsg}(\mathsf{A}, \mathsf{B}, K, 2)^{\mathsf{ot}}_{\mathsf{adv}}$
  - $\mathsf{OTMsg}(\mathsf{A}, \mathsf{B}, K, 3)^{\mathsf{ot}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{OTMsg}(\mathsf{A}, \mathsf{B}, K, 3)^{\mathsf{ot}}_{\mathsf{adv}}$

  - $\mathsf{OTChoiceRcvd}(\mathsf{B},\mathsf{A},K,0)^{\mathsf{ot}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{OTChoiceRcvd}(\mathsf{B},\mathsf{A},K,0)^{\mathsf{ot}}_{\mathsf{adv}}$
  - $\mathsf{OTChoiceRcvd}(\mathsf{B},\mathsf{A},K,1)^{\mathsf{ot}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{OTChoiceRcvd}(\mathsf{B},\mathsf{A},K,1)^{\mathsf{ot}}_{\mathsf{adv}}$
- $\mathsf{Adv}\big(C;\ \textit{xor-gate}(k,l),K+1\big)$ is the composition of the protocol $\mathsf{Adv}(C,K)$ with the protocol
  - $\mathsf{OTBit}(\mathsf{A},\mathsf{B},K)^{\mathsf{A}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{OTBit}(\mathsf{A},\mathsf{B},K)^{\mathsf{A}}_{\mathsf{adv}}$
  - $\mathsf{Share}(\mathsf{A},K)^{\mathsf{A}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{Share}(\mathsf{A},K)$
  - $\mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,0)^{\mathsf{ot}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,0)^{\mathsf{ot}}_{\mathsf{adv}}$
  - $\mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,1)^{\mathsf{ot}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,1)^{\mathsf{ot}}_{\mathsf{adv}}$
  - $\mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,2)^{\mathsf{ot}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,2)^{\mathsf{ot}}_{\mathsf{adv}}$
  - $\mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,3)^{\mathsf{ot}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,3)^{\mathsf{ot}}_{\mathsf{adv}}$
  - $\mathsf{OTChoiceRcvd}(\mathsf{B},\mathsf{A},K,0)^{\mathsf{ot}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{OTChoiceRcvd}(\mathsf{B},\mathsf{A},K,0)^{\mathsf{ot}}_{\mathsf{adv}}$
  - $\mathsf{OTChoiceRcvd}(\mathsf{B},\mathsf{A},K,1)^{\mathsf{ot}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{OTChoiceRcvd}(\mathsf{B},\mathsf{A},K,1)^{\mathsf{ot}}_{\mathsf{adv}}$
- $\mathsf{Adv}\big(C;\ \textit{and-gate}(k,l),K+1\big)$ is the composition of the protocol $\mathsf{Adv}(C,K)$ with the protocol
  - $\mathsf{OTBit}(\mathsf{A},\mathsf{B},K)^{\mathsf{A}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{OTBit}(\mathsf{A},\mathsf{B},K)$
  - $\mathsf{Share}(\mathsf{A},K)^{\mathsf{A}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{Share}(\mathsf{A},K)$
  - $\mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,0)^{\mathsf{ot}}_{\mathsf{adv}} := b_A \leftarrow \mathsf{OTBit}(\mathsf{A},\mathsf{B},K);\ x_A \leftarrow \mathsf{Share}(\mathsf{A},k);\ y_A \leftarrow \mathsf{Share}(\mathsf{A},l);\ \mathsf{ret}\ b_A$
  - $\mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,1)^{\mathsf{ot}}_{\mathsf{adv}} := b_A \leftarrow \mathsf{OTBit}(\mathsf{A},\mathsf{B},K);\ x_A \leftarrow \mathsf{Share}(\mathsf{A},k);\ y_A \leftarrow \mathsf{Share}(\mathsf{A},l);\ \mathsf{ret}\ b_A \oplus x_A$
  - $\mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,2)^{\mathsf{ot}}_{\mathsf{adv}} := b_A \leftarrow \mathsf{OTBit}(\mathsf{A},\mathsf{B},K);\ x_A \leftarrow \mathsf{Share}(\mathsf{A},k);\ y_A \leftarrow \mathsf{Share}(\mathsf{A},l);\ \mathsf{ret}\ b_A \oplus y_A$
  - $\mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,3)^{\mathsf{ot}}_{\mathsf{adv}} := b_A \leftarrow \mathsf{OTBit}(\mathsf{A},\mathsf{B},K);\ x_A \leftarrow \mathsf{Share}(\mathsf{A},k);\ y_A \leftarrow \mathsf{Share}(\mathsf{A},l);\ \mathsf{ret}\ b_A \oplus x_A \oplus y_A$
  - $\mathsf{OTChoiceRcvd}(\mathsf{B},\mathsf{A},K,0)^{\mathsf{ot}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{ShareOk}(\mathsf{B},k)$
  - $\mathsf{OTChoiceRcvd}(\mathsf{B},\mathsf{A},K,1)^{\mathsf{ot}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{ShareOk}(\mathsf{B},l)$

By design, Alice's and Bob's respective shares of each wire add up to the actual value carried by the wire. If we knew the latter, let's say by inductively computing the circuit the same way the ideal functionality does, we could get rid of Bob's shares entirely by replacing them with the sum of Alice's shares and the corresponding values. For this to work, however, we first need to arrange the timing so that Bob's computation of each wire happens after Alice's. To this end, we start by amending Bob's shares with a gratuitous dependency on timing:

- $\mathsf{Bob}(\epsilon,0)$ is the protocol $\mathsf{0}$
- $\mathsf{Bob}\big(C;\ \textit{input-gate}(i),K+1\big)$ is the composition of the protocol $\mathsf{Bob}(C,K)$ with the protocol
  - $\mathsf{OTBit}(\mathsf{B},\mathsf{A},K) := \mathsf{read}\ \mathsf{OTBit}(\mathsf{B},\mathsf{A},K)$
  - $\begin{cases}\mathsf{Share}(\mathsf{B},K) := \_ \leftarrow \mathsf{InputShareOk}(\mathsf{B},\mathsf{A},i);\ \mathsf{read}\ \mathsf{InputShare}(\mathsf{B},\mathsf{A},i) & \text{if } i \text{ is an input of Alice}\\ \mathsf{Share}(\mathsf{B},K) := \_ \leftarrow \mathsf{InputShareOk}(\mathsf{B},\mathsf{B},i);\ \mathsf{read}\ \mathsf{InputShare}(\mathsf{B},\mathsf{B},i) & \text{if } i \text{ is an input of Bob}\end{cases}$
- $\mathsf{Bob}\big(C;\ \textit{not-gate}(k),K+1\big)$ is the composition of the protocol $\mathsf{Bob}(C,K)$ with the protocol
  - $\mathsf{OTBit}(\mathsf{B},\mathsf{A},K) := \mathsf{read}\ \mathsf{OTBit}(\mathsf{B},\mathsf{A},K)$
  - $\mathsf{Share}(\mathsf{B},K) := \_ \leftarrow \mathsf{ShareOk}(\mathsf{B},k);\ x_B \leftarrow \mathsf{Share}(\mathsf{B},k);\ \mathsf{ret}\ \neg x_B$
- $\mathsf{Bob}\big(C;\ \textit{xor-gate}(k,l),K+1\big)$ is the composition of the protocol $\mathsf{Bob}(C,K)$ with the protocol
  - $\mathsf{OTBit}(\mathsf{B},\mathsf{A},K) := \mathsf{read}\ \mathsf{OTBit}(\mathsf{B},\mathsf{A},K)$
  - $\mathsf{Share}(\mathsf{B},K) := \_ \leftarrow \mathsf{ShareOk}(\mathsf{B},k);\ x_B \leftarrow \mathsf{Share}(\mathsf{B},k);\ \_ \leftarrow \mathsf{ShareOk}(\mathsf{B},l);\ y_B \leftarrow \mathsf{Share}(\mathsf{B},l);\ \mathsf{ret}\ x_B \oplus y_B$
- $\mathsf{Bob}\big(C;\ \textit{and-gate}(k,l),K+1\big)$ is the composition of the protocol $\mathsf{Bob}(C,K)$ with the protocol

- $\mathsf{OTBit}(\mathsf{B},\mathsf{A},K) := b_A \leftarrow \mathsf{OTBit}(\mathsf{A},\mathsf{B},K);\ x_A \leftarrow \mathsf{Share}(\mathsf{A},k);\ y_A \leftarrow \mathsf{Share}(\mathsf{A},l);\ x_B \leftarrow \mathsf{Share}(\mathsf{B},k);\ y_B \leftarrow \mathsf{Share}(\mathsf{B},l);\ \mathsf{ret}\ b_A \oplus (x_A * y_B) \oplus (x_B * y_A)$
- $\mathsf{Share}(\mathsf{B},K) := \_ \leftarrow \mathsf{OTBitOk}(\mathsf{B},\mathsf{A},K);\ b_B \leftarrow \mathsf{OTBit}(\mathsf{B},\mathsf{A},K);\ \_ \leftarrow \mathsf{ShareOk}(\mathsf{B},k);\ x_B \leftarrow \mathsf{Share}(\mathsf{B},k);\ \_ \leftarrow \mathsf{ShareOk}(\mathsf{B},l);\ y_B \leftarrow \mathsf{Share}(\mathsf{B},l);\ \mathsf{ret}\ b_B \oplus (x_B * y_B)$

Clearly, we can make the timing of the shares independent of their actual value; this was the point of introducing the timing of shares in the first place. We can define the channels

- $\mathsf{OTBitOk}(\mathsf{B},\mathsf{A},k) := b_B \leftarrow \mathsf{OTBit}(\mathsf{B},\mathsf{A},k);\ \mathsf{ret}\ \checkmark$ for $0 \leqslant k < K$
- $\mathsf{ShareOk}(\mathsf{B},k) := x_B \leftarrow \mathsf{Share}(\mathsf{B},k);\ \mathsf{ret}\ \checkmark$ for $0 \leqslant k < K$

equivalently by induction:

- $\mathsf{BobOk}(\epsilon,0)$ is the protocol $\mathsf{0}$
- $\mathsf{BobOk}\big(C;\ \textit{input-gate}(i),K+1\big)$ is the composition of the protocol $\mathsf{BobOk}(C,K)$ with the protocol
  - $\mathsf{OTBitOk}(\mathsf{B},\mathsf{A},K) := \mathsf{read}\ \mathsf{OTBitOk}(\mathsf{B},\mathsf{A},K)$
  - $\begin{cases} \mathsf{ShareOk}(\mathsf{B},K) := \mathsf{read}\ \mathsf{InputShareOk}(\mathsf{B},\mathsf{A},i) & \text{if } i \text{ is an input of Alice} \\ \mathsf{ShareOk}(\mathsf{B},K) := \mathsf{read}\ \mathsf{InputShareOk}(\mathsf{B},\mathsf{B},i) & \text{if } i \text{ is an input of Bob} \end{cases}$
- $\mathsf{BobOk}\big(C;\ \textit{not-gate}(k),K+1\big)$ is the composition of the protocol $\mathsf{BobOk}(C,K)$ with the protocol
  - $\mathsf{OTBitOk}(\mathsf{B},\mathsf{A},K) := \mathsf{read}\ \mathsf{OTBitOk}(\mathsf{B},\mathsf{A},K)$
  - $\mathsf{ShareOk}(\mathsf{B},K) := \mathsf{read}\ \mathsf{ShareOk}(\mathsf{B},k)$
- $\mathsf{BobOk}\big(C;\ \textit{xor-gate}(k,l),K+1\big)$ is the composition of the protocol $\mathsf{BobOk}(C,K)$ with the protocol
  - $\mathsf{OTBitOk}(\mathsf{B},\mathsf{A},K) := \mathsf{read}\ \mathsf{OTBitOk}(\mathsf{B},\mathsf{A},K)$
  - $\mathsf{ShareOk}(\mathsf{B},K) := \_ \leftarrow \mathsf{ShareOk}(\mathsf{B},k);\ \_ \leftarrow \mathsf{ShareOk}(\mathsf{B},l);\ \mathsf{ret}\ \checkmark$
- $\mathsf{BobOk}\big(C;\ \textit{and-gate}(k,l),K+1\big)$ is the composition of the protocol $\mathsf{BobOk}(C,K)$ with the protocol
  - $\mathsf{OTBitOk}(\mathsf{B},\mathsf{A},K) := \_ \leftarrow \mathsf{OTBitOk}(\mathsf{A},\mathsf{B},K);\ \_ \leftarrow \mathsf{ShareOk}(\mathsf{A},k);\ \_ \leftarrow \mathsf{ShareOk}(\mathsf{A},l);\ \_ \leftarrow \mathsf{ShareOk}(\mathsf{B},k);\ \_ \leftarrow \mathsf{ShareOk}(\mathsf{B},l);\ \mathsf{ret}\ \checkmark$
  - $\mathsf{ShareOk}(\mathsf{B},K) := \_ \leftarrow \mathsf{OTBitOk}(\mathsf{B},\mathsf{A},K);\ \_ \leftarrow \mathsf{ShareOk}(\mathsf{B},k);\ \_ \leftarrow \mathsf{ShareOk}(\mathsf{B},l);\ \mathsf{ret}\ \checkmark$

We can likewise express the channels

- $\mathsf{InputShareOk}(\mathsf{B},\mathsf{A},i) := x_B \leftarrow \mathsf{InputShare}(\mathsf{B},\mathsf{A},i);\ \mathsf{ret}\ \checkmark$ for $0 \leqslant i < N$
- $\mathsf{InputShareOk}(\mathsf{B},\mathsf{B},i) := x_B \leftarrow \mathsf{InputShare}(\mathsf{B},\mathsf{B},i);\ \mathsf{ret}\ \checkmark$ for $0 \leqslant i < M$

explicitly as

- $\mathsf{InputShareOk}(\mathsf{B},\mathsf{A},i) := \_ \leftarrow \mathsf{InputShareOk}(\mathsf{A},\mathsf{A},i);\ \_ \leftarrow \mathsf{InOk}(\mathsf{A},i);\ \mathsf{ret}\ \checkmark$ for $0 \leqslant i < N$
- $\mathsf{InputShareOk}(\mathsf{B},\mathsf{B},i) := \_ \leftarrow \mathsf{InputShareOk}(\mathsf{A},\mathsf{B},i);\ \_ \leftarrow \mathsf{InOk}(\mathsf{B},i);\ \mathsf{ret}\ \checkmark$ for $0 \leqslant i < M$

We now carry out the same procedure for Alice. We characterize the channels

- $\mathsf{OTBitOk}(\mathsf{A},\mathsf{B},k) := b_A \leftarrow \mathsf{OTBitOk}(\mathsf{A},\mathsf{B},k);\ \mathsf{ret}\ \checkmark$ for $0 \leqslant k < K$
- $\mathsf{ShareOk}(\mathsf{A},k) := x_A \leftarrow \mathsf{Share}(\mathsf{A},k);\ \mathsf{ret}\ \checkmark$ for $0 \leqslant k < K$

by induction as follows:

- $\mathsf{AliceOk}(\epsilon,0)$ is the protocol $\mathsf{0}$

- $\mathsf{AliceOk}\big(C;\ \textit{input-gate}(i), K+1\big)$ is the composition of the protocol $\mathsf{AliceOk}(C, K)$ with the protocol
  - $\mathsf{OTBitOk}(\mathsf{A}, \mathsf{B}, K) := \mathsf{read}\ \mathsf{OTBitOk}(\mathsf{A}, \mathsf{B}, K)$
  - $\begin{cases} \mathsf{ShareOk}(\mathsf{A}, K) := \mathsf{read}\ \mathsf{InputShareOk}(\mathsf{A}, \mathsf{A}, i) & \text{if } i \text{ is an input of Alice} \\ \mathsf{ShareOk}(\mathsf{A}, K) := \mathsf{read}\ \mathsf{InputShareOk}(\mathsf{A}, \mathsf{B}, i) & \text{if } i \text{ is an input of Bob} \end{cases}$
- $\mathsf{AliceOk}\big(C;\ \textit{not-gate}(k), K+1\big)$ is the composition of the protocol $\mathsf{AliceOk}(C, K)$ with the protocol
  - $\mathsf{OTBitOk}(\mathsf{A}, \mathsf{B}, K) := \mathsf{read}\ \mathsf{OTBitOk}(\mathsf{A}, \mathsf{B}, K)$
  - $\mathsf{ShareOk}(\mathsf{A}, K) := \mathsf{read}\ \mathsf{ShareOk}(\mathsf{A}, k)$
- $\mathsf{AliceOk}\big(C;\ \textit{xor-gate}(k, l), K+1\big)$ is the composition of the protocol $\mathsf{AliceOk}(C, K)$ with the protocol
  - $\mathsf{OTBitOk}(\mathsf{A}, \mathsf{B}, K) := \mathsf{read}\ \mathsf{OTBitOk}(\mathsf{A}, \mathsf{B}, K)$
  - $\mathsf{ShareOk}(\mathsf{A}, K) := \_ \leftarrow \mathsf{ShareOk}(\mathsf{A}, k);\ \_ \leftarrow \mathsf{ShareOk}(\mathsf{A}, l);\ \mathsf{ret}\ \checkmark$
- $\mathsf{AliceOk}\big(C;\ \textit{and-gate}(k, l), K+1\big)$ is the composition of the protocol $\mathsf{AliceOk}(C, K)$ with the protocol
  - $\mathsf{OTBitOk}(\mathsf{A}, \mathsf{B}, K) := \_ \leftarrow \mathsf{ShareOk}(\mathsf{A}, k);\ \_ \leftarrow \mathsf{ShareOk}(\mathsf{A}, l);\ \mathsf{ret}\ \checkmark$
  - $\mathsf{ShareOk}(\mathsf{A}, K) := \_ \leftarrow \mathsf{OTBitOk}(\mathsf{A}, \mathsf{B}, K);\ \_ \leftarrow \mathsf{ShareOk}(\mathsf{A}, k);\ \_ \leftarrow \mathsf{ShareOk}(\mathsf{A}, l);\ \mathsf{ret}\ \checkmark$

We can likewise express the channels

- $\mathsf{InputShareOk}(\mathsf{A}, \mathsf{A}, i) := x_A \leftarrow \mathsf{InputShare}(\mathsf{A}, \mathsf{A}, i);\ \mathsf{ret}\ \checkmark$ for $0 \leqslant i < N$
- $\mathsf{InputShareOk}(\mathsf{A}, \mathsf{B}, i) := x_A \leftarrow \mathsf{InputShare}(\mathsf{A}, \mathsf{B}, i);\ \mathsf{ret}\ \checkmark$ for $0 \leqslant i < M$

explicitly as

- $\mathsf{InputShareOk}(\mathsf{A}, \mathsf{A}, i) := \mathsf{read}\ \mathsf{InOk}(\mathsf{A}, i)$ for $0 \leqslant i < N$
- $\mathsf{InputShareOk}(\mathsf{A}, \mathsf{B}, i) := \mathsf{read}\ \mathsf{InOk}(\mathsf{B}, i)$ for $0 \leqslant i < M$

The timing of Bob's and Alice's shares is now easily seen to be the same: specifically, we can write the channels

- $\mathsf{InputShareOk}(\mathsf{B}, \mathsf{A}, i) := \_ \leftarrow \mathsf{InputShareOk}(\mathsf{A}, \mathsf{A}, i);\ \_ \leftarrow \mathsf{InOk}(\mathsf{A}, i);\ \mathsf{ret}\ \checkmark$ for $0 \leqslant i < N$
- $\mathsf{InputShareOk}(\mathsf{B}, \mathsf{B}, i) := \_ \leftarrow \mathsf{InputShareOk}(\mathsf{A}, \mathsf{B}, i);\ \_ \leftarrow \mathsf{InOk}(\mathsf{B}, i);\ \mathsf{ret}\ \checkmark$ for $0 \leqslant i < M$

equivalently as

- $\mathsf{InputShareOk}(\mathsf{B}, \mathsf{A}, i) := \mathsf{read}\ \mathsf{InputShareOk}(\mathsf{A}, \mathsf{A}, i)$ for $0 \leqslant i < N$
- $\mathsf{InputShareOk}(\mathsf{B}, \mathsf{B}, i) := \mathsf{read}\ \mathsf{InputShareOk}(\mathsf{A}, \mathsf{B}, i)$ for $0 \leqslant i < M$

and express the inductively defined protocol fragment $\mathsf{BobOk}(C, K)$ simply as

- $\mathsf{OTBitOk}(\mathsf{B}, \mathsf{A}, k) := \mathsf{read}\ \mathsf{OTBitOk}(\mathsf{A}, \mathsf{B}, k)$ for $0 \leqslant k < K$
- $\mathsf{ShareOk}(\mathsf{B}, k) := \mathsf{read}\ \mathsf{ShareOk}(\mathsf{A}, k)$ for $0 \leqslant k < K$

The closed-form expression for the timing of Alice's shares is more useful, so we revert the channels

- $\mathsf{InputShareOk}(\mathsf{A}, \mathsf{A}, i) := \mathsf{read}\ \mathsf{InOk}(\mathsf{A}, i)$ for $0 \leqslant i < N$
- $\mathsf{InputShareOk}(\mathsf{A}, \mathsf{B}, i) := \mathsf{read}\ \mathsf{InOk}(\mathsf{B}, i)$ for $0 \leqslant i < M$

back to

- $\mathsf{InputShareOk}(\mathsf{A}, \mathsf{A}, i) := x_A \leftarrow \mathsf{InputShare}(\mathsf{A}, \mathsf{A}, i);\ \mathsf{ret}\ \checkmark$ for $0 \leqslant i < N$
- $\mathsf{InputShareOk}(\mathsf{A}, \mathsf{B}, i) := x_A \leftarrow \mathsf{InputShare}(\mathsf{A}, \mathsf{B}, i);\ \mathsf{ret}\ \checkmark$ for $0 \leqslant i < M$

and revert the inductively defined protocol fragment $\mathsf{AliceOk}(C, K)$ back to

- $\mathsf{OTBitOk}(\mathsf{A}, \mathsf{B}, k) := b_A \leftarrow \mathsf{OTBit}(\mathsf{A}, \mathsf{B}, k);\ \mathsf{ret}\ \checkmark$ for $0 \leqslant k < K$
- $\mathsf{ShareOk}(\mathsf{A}, k) := x_A \leftarrow \mathsf{Share}(\mathsf{A}, k);\ \mathsf{ret}\ \checkmark$

In the presence of these latest definitions for timing, we can rewrite the timing dependency in the protocol $\mathsf{Bob}(C, K)$ as follows:

- $\mathsf{Bob}(\epsilon, 0)$ is the protocol $\mathsf{0}$
- $\mathsf{Bob}\big(C;\ \mathit{input\text{-}gate}(i), K+1\big)$ is the composition of the protocol $\mathsf{Bob}(C, K)$ with the protocol
  - $\mathsf{OTBit}(\mathsf{B}, \mathsf{A}, K) := \mathsf{read}\ \mathsf{OTBit}(\mathsf{B}, \mathsf{A}, K)$
  - $\begin{cases} \mathsf{Share}(\mathsf{B}, K) := x_A \leftarrow \mathsf{InputShare}(\mathsf{A}, \mathsf{A}, i);\ \mathsf{read}\ \mathsf{InputShare}(\mathsf{B}, \mathsf{A}, i) & \text{if } i \text{ is an input of Alice} \\ \mathsf{Share}(\mathsf{B}, K) := x_A \leftarrow \mathsf{InputShare}(\mathsf{A}, \mathsf{B}, i);\ \mathsf{read}\ \mathsf{InputShare}(\mathsf{B}, \mathsf{B}, i) & \text{if } i \text{ is an input of Bob} \end{cases}$
- $\mathsf{Bob}\big(C;\ \mathit{not\text{-}gate}(k), K+1\big)$ is the composition of the protocol $\mathsf{Bob}(C, K)$ with the protocol
  - $\mathsf{OTBit}(\mathsf{B}, \mathsf{A}, K) := \mathsf{read}\ \mathsf{OTBit}(\mathsf{B}, \mathsf{A}, K)$
  - $\mathsf{Share}(\mathsf{B}, K) := x_A \leftarrow \mathsf{Share}(\mathsf{A}, k);\ x_B \leftarrow \mathsf{Share}(\mathsf{B}, k);\ \mathsf{ret}\ \neg x_B$
- $\mathsf{Bob}\big(C;\ \mathit{xor\text{-}gate}(k, l), K+1\big)$ is the composition of the protocol $\mathsf{Bob}(C, K)$ with the protocol
  - $\mathsf{OTBit}(\mathsf{B}, \mathsf{A}, K) := \mathsf{read}\ \mathsf{OTBit}(\mathsf{B}, \mathsf{A}, K)$
  - $\mathsf{Share}(\mathsf{B}, K) := x_A \leftarrow \mathsf{Share}(\mathsf{A}, k);\ x_B \leftarrow \mathsf{Share}(\mathsf{B}, k);\ y_A \leftarrow \mathsf{Share}(\mathsf{A}, l);\ y_B \leftarrow \mathsf{Share}(\mathsf{B}, l);\ \mathsf{ret}\ x_B \oplus y_B$
- $\mathsf{Bob}\big(C;\ \mathit{and\text{-}gate}(k, l), K+1\big)$ is the composition of the protocol $\mathsf{Bob}(C, K)$ with the protocol
  - $\mathsf{OTBit}(\mathsf{B}, \mathsf{A}, K) := b_A \leftarrow \mathsf{OTBit}(\mathsf{A}, \mathsf{B}, K);\ x_A \leftarrow \mathsf{Share}(\mathsf{A}, k);\ y_A \leftarrow \mathsf{Share}(\mathsf{A}, l);\ x_B \leftarrow \mathsf{Share}(\mathsf{B}, k);\ y_B \leftarrow \mathsf{Share}(\mathsf{B}, l);\ \mathsf{ret}\ b_A \oplus (x_A * y_B) \oplus (x_B * y_A)$
  - $\mathsf{Share}(\mathsf{B}, K) := b_A \leftarrow \mathsf{OTBit}(\mathsf{A}, \mathsf{B}, K);\ b_B \leftarrow \mathsf{OTBit}(\mathsf{B}, \mathsf{A}, K);\ x_A \leftarrow \mathsf{Share}(\mathsf{A}, k);\ x_B \leftarrow \mathsf{Share}(\mathsf{B}, k);\ y_B \leftarrow \mathsf{Share}(\mathsf{A}, l);\ y_B \leftarrow \mathsf{Share}(\mathsf{B}, l);\ \mathsf{ret}\ b_B \oplus (x_B * y_B)$

Finally, we can eliminate all references to timing channels from the protocol fragment $\mathsf{Adv}(C, K)$, specifically from the channels $\mathsf{OTChoiceRcvd}(\mathsf{B}, \mathsf{A}, -, -)^{\mathsf{ot}}_{\mathsf{adv}}$:

- $\mathsf{Adv}(\epsilon, 0)$ is the protocol $\mathsf{0}$
- $\mathsf{Adv}\big(C;\ \mathit{input\text{-}gate}(i), K+1\big)$ is the composition of the protocol $\mathsf{Adv}(C, K)$ with the protocol
  - $\mathsf{OTBit}(\mathsf{A}, \mathsf{B}, K)^{\mathsf{A}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{OTBit}(\mathsf{A}, \mathsf{B}, K)^{\mathsf{A}}_{\mathsf{adv}}$
  - $\mathsf{Share}(\mathsf{A}, K)^{\mathsf{A}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{Share}(\mathsf{A}, K)$
  - $\mathsf{OTMsg}(\mathsf{A}, \mathsf{B}, K, 0)^{\mathsf{ot}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{OTMsg}(\mathsf{A}, \mathsf{B}, K, 0)^{\mathsf{ot}}_{\mathsf{adv}}$
  - $\mathsf{OTMsg}(\mathsf{A}, \mathsf{B}, K, 1)^{\mathsf{ot}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{OTMsg}(\mathsf{A}, \mathsf{B}, K, 1)^{\mathsf{ot}}_{\mathsf{adv}}$
  - $\mathsf{OTMsg}(\mathsf{A}, \mathsf{B}, K, 2)^{\mathsf{ot}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{OTMsg}(\mathsf{A}, \mathsf{B}, K, 2)^{\mathsf{ot}}_{\mathsf{adv}}$
  - $\mathsf{OTMsg}(\mathsf{A}, \mathsf{B}, K, 3)^{\mathsf{ot}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{OTMsg}(\mathsf{A}, \mathsf{B}, K, 3)^{\mathsf{ot}}_{\mathsf{adv}}$
  - $\mathsf{OTChoiceRcvd}(\mathsf{B}, \mathsf{A}, K, 0)^{\mathsf{ot}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{OTChoiceRcvd}(\mathsf{B}, \mathsf{A}, K, 0)^{\mathsf{ot}}_{\mathsf{adv}}$
  - $\mathsf{OTChoiceRcvd}(\mathsf{B}, \mathsf{A}, K, 1)^{\mathsf{ot}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{OTChoiceRcvd}(\mathsf{B}, \mathsf{A}, K, 1)^{\mathsf{ot}}_{\mathsf{adv}}$
- $\mathsf{Adv}\big(C;\ \mathit{not\text{-}gate}(k), K+1\big)$ is the composition of the protocol $\mathsf{Adv}(C, K)$ with the protocol
  - $\mathsf{OTBit}(\mathsf{A}, \mathsf{B}, K)^{\mathsf{A}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{OTBit}(\mathsf{A}, \mathsf{B}, K)^{\mathsf{A}}_{\mathsf{adv}}$
  - $\mathsf{Share}(\mathsf{A}, K)^{\mathsf{A}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{Share}(\mathsf{A}, K)$

  - $\mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,0)^{\mathsf{ot}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,0)^{\mathsf{ot}}_{\mathsf{adv}}$
  - $\mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,1)^{\mathsf{ot}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,1)^{\mathsf{ot}}_{\mathsf{adv}}$
  - $\mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,2)^{\mathsf{ot}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,2)^{\mathsf{ot}}_{\mathsf{adv}}$
  - $\mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,3)^{\mathsf{ot}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,3)^{\mathsf{ot}}_{\mathsf{adv}}$
  - $\mathsf{OTChoiceRcvd}(\mathsf{B},\mathsf{A},K,0)^{\mathsf{ot}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{OTChoiceRcvd}(\mathsf{B},\mathsf{A},K,0)^{\mathsf{ot}}_{\mathsf{adv}}$
  - $\mathsf{OTChoiceRcvd}(\mathsf{B},\mathsf{A},K,1)^{\mathsf{ot}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{OTChoiceRcvd}(\mathsf{B},\mathsf{A},K,1)^{\mathsf{ot}}_{\mathsf{adv}}$
- $\mathsf{Adv}\big(C;\ \textit{xor-gate}(k,l),K+1\big)$ is the composition of the protocol $\mathsf{Adv}(C,K)$ with the protocol
  - $\mathsf{OTBit}(\mathsf{A},\mathsf{B},K)^{\mathsf{A}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{OTBit}(\mathsf{A},\mathsf{B},K)^{\mathsf{A}}_{\mathsf{adv}}$
  - $\mathsf{Share}(\mathsf{A},K)^{\mathsf{A}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{Share}(\mathsf{A},K)$
  - $\mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,0)^{\mathsf{ot}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,0)^{\mathsf{ot}}_{\mathsf{adv}}$
  - $\mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,1)^{\mathsf{ot}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,1)^{\mathsf{ot}}_{\mathsf{adv}}$
  - $\mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,2)^{\mathsf{ot}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,2)^{\mathsf{ot}}_{\mathsf{adv}}$
  - $\mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,3)^{\mathsf{ot}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,3)^{\mathsf{ot}}_{\mathsf{adv}}$
  - $\mathsf{OTChoiceRcvd}(\mathsf{B},\mathsf{A},K,0)^{\mathsf{ot}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{OTChoiceRcvd}(\mathsf{B},\mathsf{A},K,0)^{\mathsf{ot}}_{\mathsf{adv}}$
  - $\mathsf{OTChoiceRcvd}(\mathsf{B},\mathsf{A},K,1)^{\mathsf{ot}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{OTChoiceRcvd}(\mathsf{B},\mathsf{A},K,1)^{\mathsf{ot}}_{\mathsf{adv}}$
- $\mathsf{Adv}\big(C;\ \textit{and-gate}(k,l),K+1\big)$ is the composition of the protocol $\mathsf{Adv}(C,K)$ with the protocol
  - $\mathsf{OTBit}(\mathsf{A},\mathsf{B},K)^{\mathsf{A}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{OTBit}(\mathsf{A},\mathsf{B},K)$
  - $\mathsf{Share}(\mathsf{A},K)^{\mathsf{A}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{Share}(\mathsf{A},K)$
  - $\mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,0)^{\mathsf{ot}}_{\mathsf{adv}} := b_A \leftarrow \mathsf{OTBit}(\mathsf{A},\mathsf{B},K);\ x_A \leftarrow \mathsf{Share}(\mathsf{A},k);\ y_A \leftarrow \mathsf{Share}(\mathsf{A},l);\ \mathsf{ret}\ b_A$
  - $\mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,1)^{\mathsf{ot}}_{\mathsf{adv}} := b_A \leftarrow \mathsf{OTBit}(\mathsf{A},\mathsf{B},K);\ x_A \leftarrow \mathsf{Share}(\mathsf{A},k);\ y_A \leftarrow \mathsf{Share}(\mathsf{A},l);\ \mathsf{ret}\ b_A \oplus x_A$
  - $\mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,2)^{\mathsf{ot}}_{\mathsf{adv}} := b_A \leftarrow \mathsf{OTBit}(\mathsf{A},\mathsf{B},K);\ x_A \leftarrow \mathsf{Share}(\mathsf{A},k);\ y_A \leftarrow \mathsf{Share}(\mathsf{A},l);\ \mathsf{ret}\ b_A \oplus y_A$
  - $\mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,3)^{\mathsf{ot}}_{\mathsf{adv}} := b_A \leftarrow \mathsf{OTBit}(\mathsf{A},\mathsf{B},K);\ x_A \leftarrow \mathsf{Share}(\mathsf{A},k);\ y_A \leftarrow \mathsf{Share}(\mathsf{A},l);\ \mathsf{ret}\ b_A \oplus x_A \oplus y_A$
  - $\mathsf{OTChoiceRcvd}(\mathsf{B},\mathsf{A},K,0)^{\mathsf{ot}}_{\mathsf{adv}} := \_ \leftarrow \mathsf{Share}(\mathsf{A},k);\ \mathsf{ret}\ \checkmark$
  - $\mathsf{OTChoiceRcvd}(\mathsf{B},\mathsf{A},K,1)^{\mathsf{ot}}_{\mathsf{adv}} := \_ \leftarrow \mathsf{Share}(\mathsf{A},l);\ \mathsf{ret}\ \checkmark$

At this point the timing channels are unused and we can safely discard them.

9.4.5 The Sum Of Shares

As mentioned before, Alice's and Bob's respective shares of each wire sum up to the actual value carried by the wire. To make this invariant explicit, we add new internal channels

- $\mathsf{Wire}(k) := x_A \leftarrow \mathsf{Share}(\mathsf{A},k);\ x_B \leftarrow \mathsf{Share}(\mathsf{B},k);\ \mathsf{ret}\ x_A \oplus x_B$ for $0 \leqslant k < K$

that record the sum of shares. In the presence of the above, we can rewrite the final part of the real protocol as

- $\begin{cases} \mathsf{SendFinalShare}(\mathsf{A},k)^{\mathsf{A}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{Share}(\mathsf{A},k) & \text{if wire } 0 \leqslant k < K \text{ is output} \\ \mathsf{SendFinalShare}(\mathsf{A},k)^{\mathsf{A}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{SendFinalShare}(\mathsf{A},k)^{\mathsf{A}}_{\mathsf{adv}} & \text{otherwise} \end{cases}$
- $\begin{cases} \mathsf{SendFinalShare}(\mathsf{B},k)^{\mathsf{A}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{Share}(\mathsf{B},k) & \text{if wire } 0 \leqslant k < K \text{ is output} \\ \mathsf{SendFinalShare}(\mathsf{B},k)^{\mathsf{A}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{SendFinalShare}(\mathsf{B},k)^{\mathsf{A}}_{\mathsf{adv}} & \text{otherwise} \end{cases}$
- $\begin{cases} \mathsf{Out}(\mathsf{A},k) := \mathsf{read}\ \mathsf{Wire}(k) & \text{if wire } 0 \leqslant k < K \text{ is output} \\ \mathsf{Out}(\mathsf{A},k) := \mathsf{read}\ \mathsf{Out}(\mathsf{A},k) & \text{otherwise} \end{cases}$

- $\begin{cases} \mathsf{Out}(\mathsf{B}, k) := \text{read } \mathsf{Wire}(k) & \text{if wire } 0 \leqslant k < K \text{ is output} \\ \mathsf{Out}(\mathsf{B}, k) := \text{read } \mathsf{Out}(\mathsf{B}, k) & \text{otherwise} \end{cases}$
- $\mathsf{Out}(\mathsf{A}, k)^{\mathsf{A}}_{\mathsf{adv}} := \text{read } \mathsf{Out}(\mathsf{A}, k) \text{ for } 0 \leqslant k < K$

Right now, Bob's shares are computed inductively and the channels $\mathsf{Wire}(-)$ have a closed form. By induction on circuits, we can flip this around: we characterize the channels

- $\mathsf{Wire}(k) := x_A \leftarrow \mathsf{Share}(\mathsf{A}, k);\ x_B \leftarrow \mathsf{Share}(\mathsf{B}, k);\ \text{ret } x_A \oplus x_B \text{ for } 0 \leqslant k < K$

inductively as the protocol $\mathsf{Wires}(C, K)$ that's part of the ideal functionality. At the same time, we split the protocol $\mathsf{B}(C, K)$ into a closed-form part

- $\mathsf{Share}(\mathsf{B}, k) := x_A \leftarrow \mathsf{Share}(\mathsf{A}, k);\ x \leftarrow \mathsf{Wire}(k);\ \text{ret } x_A \oplus x \text{ for } 0 \leqslant k < K$

and an inductive part $\mathsf{BobOTBit}(C, K)$:

- $\mathsf{BobOTBit}(\epsilon, 0)$ is the protocol $\mathsf{0}$
- $\mathsf{BobOTBit}\big(C;\ \mathit{input\text{-}gate}(i), K+1\big)$ is the composition of the protocol $\mathsf{BobOTBit}(C, K)$ with the single-reaction protocol
  - $\mathsf{OTBit}(\mathsf{B}, \mathsf{A}, K) := \text{read } \mathsf{OTBit}(\mathsf{B}, \mathsf{A}, K)$
- $\mathsf{BobOTBit}\big(C;\ \mathit{not\text{-}gate}(k), K+1\big)$ is the composition of the protocol $\mathsf{BobOTBit}(C, K)$ with the single-reaction protocol
  - $\mathsf{OTBit}(\mathsf{B}, \mathsf{A}, K) := \text{read } \mathsf{OTBit}(\mathsf{B}, \mathsf{A}, K)$
- $\mathsf{BobOTBit}\big(C;\ \mathit{xor\text{-}gate}(k, l), K+1\big)$ is the composition of the protocol $\mathsf{BobOTBit}(C, K)$ with the single-reaction protocol
  - $\mathsf{OTBit}(\mathsf{B}, \mathsf{A}, K) := \text{read } \mathsf{OTBit}(\mathsf{B}, \mathsf{A}, K)$
- $\mathsf{BobOTBit}\big(C;\ \mathit{and\text{-}gate}(k, l), K+1\big)$ is the composition of the protocol $\mathsf{BobOTBit}(C, K)$ with the single-reaction protocol
  - $\mathsf{OTBit}(\mathsf{B}, \mathsf{A}, K) := b_A \leftarrow \mathsf{OTBit}(\mathsf{A}, \mathsf{B}, K);\ x_A \leftarrow \mathsf{Share}(\mathsf{A}, k);\ y_A \leftarrow \mathsf{Share}(\mathsf{A}, l);\ x_B \leftarrow \mathsf{Share}(\mathsf{B}, k);$ $y_B \leftarrow \mathsf{Share}(\mathsf{B}, l);\ \text{ret } b_A \oplus (x_A * y_B) \oplus (x_B * y_A)$

To see why this works, we consider each gate in turn.

- In the case of an *input* gate, let us assume i is an input of Alice; the case for Bob is entirely analogous. We start by substituting the channels $\mathsf{Share}(\mathsf{A}, K)$ and $\mathsf{Share}(\mathsf{B}, K)$ into the channel
  - $\mathsf{Wire}(K) := x_A \leftarrow \mathsf{Share}(\mathsf{A}, K);\ x_B \leftarrow \mathsf{Share}(\mathsf{B}, K);\ \text{ret } x_A \oplus x_B$, yielding
  - $\mathsf{Wire}(K) := x_A \leftarrow \mathsf{InputShare}(\mathsf{A}, \mathsf{A}, i);\ x_B \leftarrow \mathsf{InputShare}(\mathsf{B}, \mathsf{A}, i);\ \text{ret } x_A \oplus x_B$

  Substituting this new definition of $\mathsf{Wire}(K)$ along with the channel $\mathsf{Share}(\mathsf{A}, K)$ into the channel
  - $\mathsf{Share}(\mathsf{B}, K) := x_A \leftarrow \mathsf{Share}(\mathsf{A}, K);\ x \leftarrow \mathsf{Wire}(K);\ \text{ret } x_A \oplus x$ yields
  - $\mathsf{Share}(\mathsf{B}, K) := x_A \leftarrow \mathsf{InputShare}(\mathsf{A}, \mathsf{A}, i);\ x_B \leftarrow \mathsf{InputShare}(\mathsf{B}, \mathsf{A}, i);\ \text{ret } x_A \oplus (x_A \oplus x_B)$

  After canceling out the two applications of $\oplus$ and simplifying, we get
  - $\mathsf{Share}(\mathsf{B}, K) := x_A \leftarrow \mathsf{InputShare}(\mathsf{A}, \mathsf{A}, i);\ \text{read } \mathsf{InputShare}(\mathsf{B}, \mathsf{A}, i)$

  which is precisely the desired inductive formulation of $\mathsf{Share}(\mathsf{B}, K)$.

  We continue to work on the channel $\mathsf{Wire}(K)$. A further substitution of the channel $\mathsf{InputShare}(\mathsf{B}, \mathsf{A}, i)$ into the channel $\mathsf{Wire}(K)$ yields

– $\mathsf{Wire}(K) := x_A \leftarrow \mathsf{InputShare}(\mathsf{A}, \mathsf{A}, i);\ x \leftarrow \mathsf{In}(\mathsf{A}, i);\ \mathsf{ret}\ x_A \oplus (x_A \oplus x)$

After canceling out the two applications of $\oplus$ and simplifying, we get

– $\mathsf{Wire}(K) := x_A \leftarrow \mathsf{InputShare}(\mathsf{A}, \mathsf{A}, i);\ \mathsf{read}\ \mathsf{In}(\mathsf{A}, i)$

Dropping the gratuitous dependency on the channel $\mathsf{InputShare}(\mathsf{A}, \mathsf{A}, i)$ yields

– $\mathsf{Wire}(K) := \mathsf{read}\ \mathsf{In}(\mathsf{A}, i)$

which is precisely the desired inductive formulation of $\mathsf{Wire}(K)$.

- In the case of a *not* gate, we substitute the channels $\mathsf{Share}(\mathsf{A}, K)$ and $\mathsf{Share}(\mathsf{B}, K)$ into the channel
  – $\mathsf{Wire}(K) := x_A \leftarrow \mathsf{Share}(\mathsf{A}, K);\ x_B \leftarrow \mathsf{Share}(\mathsf{B}, K);\ \mathsf{ret}\ x_A \oplus x_B$ to obtain
  – $\mathsf{Wire}(K) := x_A \leftarrow \mathsf{Share}(\mathsf{A}, k);\ x_B \leftarrow \mathsf{Share}(\mathsf{B}, k);\ \mathsf{ret}\ x_A \oplus (\neg x_B)$

  Substituting this new definition of $\mathsf{Wire}(K)$ along with the channel $\mathsf{Share}(\mathsf{A}, K)$ into the channel
  – $\mathsf{Share}(\mathsf{B}, K) := x_A \leftarrow \mathsf{Share}(\mathsf{A}, K);\ x \leftarrow \mathsf{Wire}(K);\ \mathsf{ret}\ x_A \oplus x$ yields
  – $\mathsf{Share}(\mathsf{B}, K) := x_A \leftarrow \mathsf{Share}(\mathsf{A}, k);\ x_B \leftarrow \mathsf{Share}(\mathsf{B}, k);\ \mathsf{ret}\ x_A \oplus (x_A \oplus (\neg x_B))$

  After canceling out the two applications of $\oplus$, we get
  – $\mathsf{Share}(\mathsf{B}, K) := x_A \leftarrow \mathsf{Share}(\mathsf{A}, k);\ x_B \leftarrow \mathsf{Share}(\mathsf{B}, k);\ \mathsf{ret}\ \neg x_B$

  which is precisely the desired inductive formulation of $\mathsf{Share}(\mathsf{B}, K)$.
  We continue to work on the channel $\mathsf{Wire}(K)$. The negation can be brought to the top level:
  – $\mathsf{Wire}(K) := x_A \leftarrow \mathsf{Share}(\mathsf{A}, k);\ x_B \leftarrow \mathsf{Share}(\mathsf{B}, k);\ \mathsf{ret}\ \neg(x_A \oplus x_B)$

  This is precisely what we get when we substitute the channel $\mathsf{Wire}(k)$ into the channel
  – $\mathsf{Wire}(K) := x \leftarrow \mathsf{Wire}(k);\ \mathsf{ret}\ \neg x$

  using our inductive hypothesis, thereby yielding the desired inductive formulation of $\mathsf{Wire}(K)$.
- In the case of an *xor* gate, we substitute the channels $\mathsf{Share}(\mathsf{A}, K)$ and $\mathsf{Share}(\mathsf{B}, K)$ into the channel
  – $\mathsf{Wire}(K) := x_A \leftarrow \mathsf{Share}(\mathsf{A}, K);\ x_B \leftarrow \mathsf{Share}(\mathsf{B}, K);\ \mathsf{ret}\ x_A \oplus x_B$ to obtain
  – $\mathsf{Wire}(K) := x_A \leftarrow \mathsf{Share}(\mathsf{A}, k);\ y_A \leftarrow \mathsf{Share}(\mathsf{A}, l);\ x_B \leftarrow \mathsf{Share}(\mathsf{B}, k);\ y_B \leftarrow \mathsf{Share}(\mathsf{B}, l);$
  $\mathsf{ret}\ (x_A \oplus y_A) \oplus (x_B \oplus y_B)$

  Substituting this new definition of $\mathsf{Wire}(K)$ along with the channel $\mathsf{Share}(\mathsf{A}, K)$ into the channel
  – $\mathsf{Share}(\mathsf{B}, K) := x_A \leftarrow \mathsf{Share}(\mathsf{A}, K);\ x \leftarrow \mathsf{Wire}(K);\ \mathsf{ret}\ x_A \oplus x$ yields
  – $\mathsf{Share}(\mathsf{B}, K) := x_A \leftarrow \mathsf{Share}(\mathsf{A}, k);\ y_A \leftarrow \mathsf{Share}(\mathsf{A}, l);\ x_B \leftarrow \mathsf{Share}(\mathsf{B}, k);\ y_B \leftarrow \mathsf{Share}(\mathsf{B}, l);$
  $\mathsf{ret}\ (x_A \oplus y_A) \oplus (x_A \oplus y_A) \oplus (x_B \oplus y_B)$

  After canceling out the two top-level applications of $\oplus$, we get
  – $\mathsf{Share}(\mathsf{B}, K) := x_A \leftarrow \mathsf{Share}(\mathsf{A}, k);\ y_A \leftarrow \mathsf{Share}(\mathsf{A}, l);\ x_B \leftarrow \mathsf{Share}(\mathsf{B}, k);\ y_B \leftarrow \mathsf{Share}(\mathsf{B}, l);\ \mathsf{ret}\ x_B \oplus y_B$

  After rearranging, the above becomes
  – $\mathsf{Share}(\mathsf{B}, K) := x_A \leftarrow \mathsf{Share}(\mathsf{A}, k);\ x_B \leftarrow \mathsf{Share}(\mathsf{B}, k);\ y_A \leftarrow \mathsf{Share}(\mathsf{A}, l);\ y_B \leftarrow \mathsf{Share}(\mathsf{B}, l);\ \mathsf{ret}\ x_B \oplus y_B$

  which is precisely the desired inductive formulation of $\mathsf{Share}(\mathsf{B}, K)$.
  We continue to work on the channel $\mathsf{Wire}(K)$. Since $\oplus$ is commutative, we can write the channel Wire as

- $\mathsf{Wire}(K) := x_A \leftarrow \mathsf{Share}(\mathsf{A}, k);\ y_A \leftarrow \mathsf{Share}(\mathsf{A}, l);\ x_B \leftarrow \mathsf{Share}(\mathsf{B}, k);\ y_B \leftarrow \mathsf{Share}(\mathsf{B}, l);$ $\mathsf{ret}\ (x_A \oplus x_B) \oplus (y_A \oplus y_B)$

After rearranging, the above becomes

- $\mathsf{Wire}(K) := x_A \leftarrow \mathsf{Share}(\mathsf{A}, k);\ x_B \leftarrow \mathsf{Share}(\mathsf{B}, k);\ y_A \leftarrow \mathsf{Share}(\mathsf{A}, l);\ y_B \leftarrow \mathsf{Share}(\mathsf{B}, l);$ $\mathsf{ret}\ (x_A \oplus x_B) \oplus (y_A \oplus y_B)$

This is precisely what we get when we substitute the channels $\mathsf{Wire}(k)$ and $\mathsf{Wire}(l)$ into the channel

- $\mathsf{Wire}(K) := x \leftarrow \mathsf{Wire}(k);\ y \leftarrow \mathsf{Wire}(l);\ \mathsf{ret}\ x \oplus y$

using our inductive hypothesis twice, thereby yielding the desired inductive formulation of $\mathsf{Wire}(K)$.

- In the case of an *and* gate, we again start by substituting the channels $\mathsf{Share}(\mathsf{A}, K)$ and $\mathsf{Share}(\mathsf{B}, K)$ into the channel
  - $\mathsf{Wire}(K) := x_A \leftarrow \mathsf{Share}(\mathsf{A}, K);\ x_B \leftarrow \mathsf{Share}(\mathsf{B}, K);\ \mathsf{ret}\ x_A \oplus x_B$ to obtain
  - $\mathsf{Wire}(K) := b_A \leftarrow \mathsf{OTBit}(\mathsf{A}, \mathsf{B}, K);\ x_A \leftarrow \mathsf{Share}(\mathsf{A}, k);\ y_A \leftarrow \mathsf{Share}(\mathsf{A}, l);\ b_B \leftarrow \mathsf{OTBit}(\mathsf{B}, \mathsf{A}, K);$ $x_B \leftarrow \mathsf{Share}(\mathsf{B}, k);\ y_B \leftarrow \mathsf{Share}(\mathsf{B}, l);\ \mathsf{ret}\ \big((x_A * y_A) \oplus b_A\big) \oplus \big(b_B \oplus (x_B * y_B)\big)$

  Substituting this new definition of $\mathsf{Wire}(K)$ along with the channel $\mathsf{Share}(\mathsf{A}, K)$ into the channel
  - $\mathsf{Share}(\mathsf{B}, K) := x_A \leftarrow \mathsf{Share}(\mathsf{A}, K);\ x \leftarrow \mathsf{Wire}(K);\ \mathsf{ret}\ x_A \oplus x$ yields
  - $\mathsf{Share}(\mathsf{B}, K) := b_A \leftarrow \mathsf{OTBit}(\mathsf{A}, \mathsf{B}, K);\ x_A \leftarrow \mathsf{Share}(\mathsf{A}, k);\ y_A \leftarrow \mathsf{Share}(\mathsf{A}, l);\ b_B \leftarrow \mathsf{OTBit}(\mathsf{B}, \mathsf{A}, K);$ $x_B \leftarrow \mathsf{Share}(\mathsf{B}, k);\ y_B \leftarrow \mathsf{Share}(\mathsf{B}, l);\ \mathsf{ret}\ \big((x_A * y_A) \oplus b_A\big) \oplus \big((x_A * y_A) \oplus b_A\big) \oplus \big(b_B \oplus (x_B * y_B)\big)$

  After canceling out the two top-level applications of $\oplus$ we get
  - $\mathsf{Share}(\mathsf{B}, K) := b_A \leftarrow \mathsf{OTBit}(\mathsf{A}, \mathsf{B}, K);\ x_A \leftarrow \mathsf{Share}(\mathsf{A}, k);\ y_A \leftarrow \mathsf{Share}(\mathsf{A}, l);\ b_B \leftarrow \mathsf{OTBit}(\mathsf{B}, \mathsf{A}, K);$ $x_B \leftarrow \mathsf{Share}(\mathsf{B}, k);\ y_B \leftarrow \mathsf{Share}(\mathsf{B}, l);\ \mathsf{ret}\ b_B \oplus (x_B * y_B)$

  After rearranging, the above becomes
  - $\mathsf{Share}(\mathsf{B}, K) := b_A \leftarrow \mathsf{OTBit}(\mathsf{A}, \mathsf{B}, K);\ b_B \leftarrow \mathsf{OTBit}(\mathsf{B}, \mathsf{A}, K);\ x_A \leftarrow \mathsf{Share}(\mathsf{A}, k);\ x_B \leftarrow \mathsf{Share}(\mathsf{B}, k);$ $y_A \leftarrow \mathsf{Share}(\mathsf{A}, l);\ y_B \leftarrow \mathsf{Share}(\mathsf{B}, l);\ \mathsf{ret}\ b_B \oplus (x_B * y_B)$

  which is precisely the desired inductive formulation of $\mathsf{Share}(\mathsf{B}, K)$.

  We continue to work on the channel $\mathsf{Wire}(K)$. A further substitution of the channel $\mathsf{OTBit}(\mathsf{B}, \mathsf{A}, K)$ yields
  - $\mathsf{Wire}(K) := b_A \leftarrow \mathsf{OTBit}(\mathsf{A}, \mathsf{B}, K);\ x_A \leftarrow \mathsf{Share}(\mathsf{A}, k);\ y_A \leftarrow \mathsf{Share}(\mathsf{A}, l);\ x_B \leftarrow \mathsf{Share}(\mathsf{B}, k);$ $y_B \leftarrow \mathsf{Share}(\mathsf{B}, l);\ \mathsf{ret}\ \big((x_A * y_A) \oplus b_A\big) \oplus \big(b_A \oplus (x_A * y_B) \oplus (x_B * y_A) \oplus (x_B * y_B)\big)$

  or, reassociated,
  - $\mathsf{Wire}(K) := b_A \leftarrow \mathsf{OTBit}(\mathsf{A}, \mathsf{B}, K);\ x_A \leftarrow \mathsf{Share}(\mathsf{A}, k);\ y_A \leftarrow \mathsf{Share}(\mathsf{A}, l);\ x_B \leftarrow \mathsf{Share}(\mathsf{B}, k);$ $y_B \leftarrow \mathsf{Share}(\mathsf{B}, l);\ \mathsf{ret}\ (x_A * y_A) \oplus \big(b_A \oplus b_A \oplus (x_A * y_B) \oplus (x_B * y_A)\big) \oplus (x_B * y_B)$

  After canceling out the two applications of $\oplus$ with b_A, we get
  - $\mathsf{Wire}(K) := b_A \leftarrow \mathsf{OTBit}(\mathsf{A}, \mathsf{B}, K);\ x_A \leftarrow \mathsf{Share}(\mathsf{A}, k);\ y_A \leftarrow \mathsf{Share}(\mathsf{A}, l);\ x_B \leftarrow \mathsf{Share}(\mathsf{B}, k);$ $y_B \leftarrow \mathsf{Share}(\mathsf{B}, l);\ \mathsf{ret}\ (x_A * y_A) \oplus (x_A * y_B) \oplus (x_B * y_A) \oplus (x_B * y_B)$

  Dopping the gratuitous dependency on $\mathsf{OTBit}(\mathsf{A}, \mathsf{B}, K)$ yields
  - $\mathsf{Wire}(K) := x_A \leftarrow \mathsf{Share}(\mathsf{A}, k);\ y_A \leftarrow \mathsf{Share}(\mathsf{A}, l);\ x_B \leftarrow \mathsf{Share}(\mathsf{B}, k);\ y_B \leftarrow \mathsf{Share}(\mathsf{B}, l);$ $\mathsf{ret}\ (x_A * y_A) \oplus (x_A * y_B) \oplus (x_B * y_A) \oplus (x_B * y_B)$

  Applying distributivity yields

– $\mathsf{Wire}(K) := x_A \leftarrow \mathsf{Share}(\mathsf{A}, k);\ y_A \leftarrow \mathsf{Share}(\mathsf{A}, l);\ x_B \leftarrow \mathsf{Share}(\mathsf{B}, k);\ y_B \leftarrow \mathsf{Share}(\mathsf{B}, l);$
$\mathsf{ret}\ (x_A \oplus x_B) * (y_A \oplus y_B)$

After rearranging, the above becomes

– $\mathsf{Wire}(K) := x_A \leftarrow \mathsf{Share}(\mathsf{A}, k);\ x_B \leftarrow \mathsf{Share}(\mathsf{B}, k);\ y_A \leftarrow \mathsf{Share}(\mathsf{A}, l);\ y_B \leftarrow \mathsf{Share}(\mathsf{B}, l);$
$\mathsf{ret}\ (x_A \oplus x_B) * (y_A \oplus y_B)$

This is precisely what we get when we substitute the channels $\mathsf{Wire}(k)$ and $\mathsf{Wire}(l)$ into the channel

– $\mathsf{Wire}(K) := x \leftarrow \mathsf{Wire}(k);\ y \leftarrow \mathsf{Wire}(l);\ \mathsf{ret}\ x * y$

using our inductive hypothesis twice, thereby yielding the desired inductive formulation of $\mathsf{Wire}(K)$.

Since we have a closed form for computing Bob's shares, we can discard the protocol fragment $\mathsf{BobOTBit}(C, K)$ entirely. Moreover, at this point the only places where we refer to Bob's side of the computation is when leaking Bob's initial and final shares, *i.e.*, in the channels $\mathsf{SendInputShare}(\mathsf{B}, \mathsf{A}, -)^{\mathsf{A}}_{\mathsf{adv}}$ and $\mathsf{SendFinalShare}(\mathsf{B}, -)^{\mathsf{A}}_{\mathsf{adv}}$. This can be easily remedied by performing the appropriate substitutions. Substituting the channels $\mathsf{InputShare}(\mathsf{B}, \mathsf{A}, -)$ into the channels $\mathsf{SendInputShare}(\mathsf{B}, \mathsf{A}, -)^{\mathsf{A}}_{\mathsf{adv}}$ yields the following version of the initial part of the real protocol:

- $\mathsf{In}(\mathsf{A}, i)^{\mathsf{A}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{In}(\mathsf{A}, i)\ \text{for}\ 0 \leqslant i < N$
- $\mathsf{InRcvd}(\mathsf{B}, i)^{\mathsf{B}}_{\mathsf{adv}} := x \leftarrow \mathsf{In}(\mathsf{B}, i);\ \mathsf{ret}\ \checkmark\ \text{for}\ 0 \leqslant i < M$
- $\mathsf{InputShare}(\mathsf{A}, \mathsf{A}, i) := x \leftarrow \mathsf{In}(\mathsf{A}, i);\ \mathsf{samp\ flip}\ \text{for}\ 0 \leqslant i < N$
- $\mathsf{InputShare}(\mathsf{A}, \mathsf{B}, i) := x \leftarrow \mathsf{In}(\mathsf{B}, i);\ \mathsf{samp\ flip}\ \text{for}\ 0 \leqslant i < M$
- $\mathsf{InputShare}(\mathsf{B}, \mathsf{A}, i) := x_A \leftarrow \mathsf{InputShare}(\mathsf{A}, \mathsf{A}, i);\ x \leftarrow \mathsf{In}(\mathsf{A}, i);\ \mathsf{ret}\ x_A \oplus x\ \text{for}\ 0 \leqslant i < N$
- $\mathsf{InputShare}(\mathsf{B}, \mathsf{B}, i) := x_A \leftarrow \mathsf{InputShare}(\mathsf{A}, \mathsf{B}, i);\ x \leftarrow \mathsf{In}(\mathsf{B}, i);\ \mathsf{ret}\ x_A \oplus x\ \text{for}\ 0 \leqslant i < M$
- $\mathsf{InputShare}(\mathsf{A}, \mathsf{A}, i)^{\mathsf{A}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{InputShare}(\mathsf{A}, \mathsf{A}, i)\ \text{for}\ 0 \leqslant i < N$
- $\mathsf{InputShare}(\mathsf{A}, \mathsf{B}, i)^{\mathsf{A}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{InputShare}(\mathsf{A}, \mathsf{B}, i)\ \text{for}\ 0 \leqslant i < M$
- $\mathsf{SendInputShare}(\mathsf{B}, \mathsf{A}, i)^{\mathsf{A}}_{\mathsf{adv}} := x_A \leftarrow \mathsf{InputShare}(\mathsf{A}, \mathsf{A}, i);\ x \leftarrow \mathsf{In}(\mathsf{A}, i);\ \mathsf{ret}\ x_A \oplus x\ \text{for}\ 0 \leqslant i < N$
- $\mathsf{SendInputShare}(\mathsf{A}, \mathsf{B}, i)^{\mathsf{A}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{InputShare}(\mathsf{A}, \mathsf{B}, i)\ \text{for}\ 0 \leqslant i < M$

Similarly, substituting the channels

- $\mathsf{Share}(\mathsf{B}, k) := x_A \leftarrow \mathsf{Share}(\mathsf{A}, k);\ x \leftarrow \mathsf{Wire}(k);\ \mathsf{ret}\ x_A \oplus x\ \text{for}\ 0 \leqslant k < K$

into the channels $\mathsf{SendFinalShare}(\mathsf{B}, -)^{\mathsf{A}}_{\mathsf{adv}}$ yields the following version of the final part of the real protocol:

- $\begin{cases} \mathsf{SendFinalShare}(\mathsf{A}, k)^{\mathsf{A}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{Share}(\mathsf{A}, k) & \text{if wire } 0 \leqslant k < K \text{ is output} \\ \mathsf{SendFinalShare}(\mathsf{A}, k)^{\mathsf{A}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{SendFinalShare}(\mathsf{A}, k)^{\mathsf{A}}_{\mathsf{adv}} & \text{otherwise} \end{cases}$
- $\begin{cases} \mathsf{SendFinalShare}(\mathsf{B}, k)^{\mathsf{A}}_{\mathsf{adv}} := x_A \leftarrow \mathsf{Share}(\mathsf{A}, k);\ x \leftarrow \mathsf{Wire}(k);\ \mathsf{ret}\ x_A \oplus x & \text{if wire } 0 \leqslant k < K \text{ is output} \\ \mathsf{SendFinalShare}(\mathsf{B}, k)^{\mathsf{A}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{SendFinalShare}(\mathsf{B}, k)^{\mathsf{A}}_{\mathsf{adv}} & \text{otherwise} \end{cases}$
- $\begin{cases} \mathsf{Out}(\mathsf{A}, k) := \mathsf{read}\ \mathsf{Wire}(k) & \text{if wire } 0 \leqslant k < K \text{ is output} \\ \mathsf{Out}(\mathsf{A}, k) := \mathsf{read}\ \mathsf{Out}(\mathsf{A}, k) & \text{otherwise} \end{cases}$
- $\begin{cases} \mathsf{Out}(\mathsf{B}, k) := \mathsf{read}\ \mathsf{Wire}(k) & \text{if wire } 0 \leqslant k < K \text{ is output} \\ \mathsf{Out}(\mathsf{B}, k) := \mathsf{read}\ \mathsf{Out}(\mathsf{B}, k) & \text{otherwise} \end{cases}$
- $\mathsf{Out}(\mathsf{A}, k)^{\mathsf{A}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{Out}(\mathsf{A}, k)\ \text{for}\ 0 \leqslant k < K$

At this point we can discard all the channels coming from Bob's side:

- $\mathsf{InputShare}(\mathsf{B}, \mathsf{A}, i)$ for $0 \leqslant i < N$,
- $\mathsf{InputShare}(\mathsf{B}, \mathsf{B}, i)$ for $0 \leqslant i < M$, and
- $\mathsf{Share}(\mathsf{B}, k)$ for $0 \leqslant k < K$.

This yields the final version of the real protocol.

9.5 The Simulator

The protocol $\mathsf{Wire}(C, K)$ and the channels

- $\begin{cases} \mathsf{Out}(\mathsf{A}, k) := \mathsf{read}\ \mathsf{Wire}(k) & \text{if wire } 0 \leqslant k < K \text{ is output} \\ \mathsf{Out}(\mathsf{A}, k) := \mathsf{read}\ \mathsf{Out}(\mathsf{A}, k) & \text{otherwise} \end{cases}$
- $\begin{cases} \mathsf{Out}(\mathsf{B}, k) := \mathsf{read}\ \mathsf{Wire}(k) & \text{if wire } 0 \leqslant k < K \text{ is output} \\ \mathsf{Out}(\mathsf{B}, k) := \mathsf{read}\ \mathsf{Out}(\mathsf{B}, k) & \text{otherwise} \end{cases}$

can now be separated out as coming from the functionality, and the remainder of the protocol is turned into a simulator. The simulator arises as a composition of an initial part, an inductive part, and a final part – all described below – followed by the hiding of the channels

- $\mathsf{InputShare}(\mathsf{A}, \mathsf{A}, i)$ for $0 \leqslant i < N$,
- $\mathsf{InputShare}(\mathsf{A}, \mathsf{B}, i)$ for $0 \leqslant i < M$,
- $\mathsf{InputShare}(\mathsf{B}, \mathsf{A}, i)$ for $0 \leqslant i < N$,
- $\mathsf{InputShare}(\mathsf{B}, \mathsf{B}, i)$ for $0 \leqslant i < M$,
- $\mathsf{OTBit}(\mathsf{A}, \mathsf{B}, k)$ for $0 \leqslant k < K$,
- $\mathsf{OTBit}(\mathsf{B}, \mathsf{A}, k)$ for $0 \leqslant k < K$,
- $\mathsf{Share}(\mathsf{A}, k)$ for $0 \leqslant k < K$, and
- $\mathsf{Share}(\mathsf{B}, k)$ for $0 \leqslant k < K$.

Plugging the simulator into the ideal functionality and substituting away the channels $\mathsf{In}(\mathsf{A}, -)^{\mathsf{id}}_{\mathsf{adv}}$, $\mathsf{InRcvd}(\mathsf{B}, -)^{\mathsf{id}}_{\mathsf{adv}}$, and $\mathsf{Out}(\mathsf{A}, -)^{\mathsf{id}}_{\mathsf{adv}}$ that originally served as a line of communication for the adversary yields the final version of the real protocol, as desired.

9.5.1 The Simulator: The Initial Phase

The code for the initial part of the simulator is as follows:

- $\mathsf{In}(\mathsf{A}, i)^{\mathsf{A}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{In}(\mathsf{A}, i)^{\mathsf{id}}_{\mathsf{adv}}$ for $0 \leqslant i < N$
- $\mathsf{InRcvd}(\mathsf{B}, i)^{\mathsf{B}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{InRcvd}(\mathsf{B}, i)^{\mathsf{id}}_{\mathsf{adv}}$ for $0 \leqslant i < M$
- $\mathsf{InputShare}(\mathsf{A}, \mathsf{A}, i) := x \leftarrow \mathsf{In}(\mathsf{A}, i)^{\mathsf{id}}_{\mathsf{adv}};\ \mathsf{samp\ flip}$ for $0 \leqslant i < N$
- $\mathsf{InputShare}(\mathsf{A}, \mathsf{B}, i) := \_ \leftarrow \mathsf{InRcvd}(\mathsf{B}, i)^{\mathsf{id}}_{\mathsf{adv}};\ \mathsf{samp\ flip}$ for $0 \leqslant i < M$
- $\mathsf{InputShare}(\mathsf{A}, \mathsf{A}, i)^{\mathsf{A}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{InputShare}(\mathsf{A}, \mathsf{A}, i)$ for $0 \leqslant i < N$
- $\mathsf{InputShare}(\mathsf{A}, \mathsf{B}, i)^{\mathsf{A}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{InputShare}(\mathsf{A}, \mathsf{B}, i)$ for $0 \leqslant i < M$
- $\mathsf{SendInputShare}(\mathsf{B}, \mathsf{A}, i)^{\mathsf{A}}_{\mathsf{adv}} := x_A \leftarrow \mathsf{InputShare}(\mathsf{A}, \mathsf{A}, i);\ x \leftarrow \mathsf{In}(\mathsf{A}, i)^{\mathsf{id}}_{\mathsf{adv}};\ \mathsf{ret}\ x_A \oplus x$ for $0 \leqslant i < N$
- $\mathsf{SendInputShare}(\mathsf{A}, \mathsf{B}, i)^{\mathsf{A}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{InputShare}(\mathsf{A}, \mathsf{B}, i)$ for $0 \leqslant i < M$

9.5.2 The Simulator: The Inductive Phase

The inductive part of the simulator $\mathsf{Sim}(C, K)$ is the composition of the three protocols $\mathsf{Alice}(C, K)$, $\mathsf{Wires}(C, K)$, and $\mathsf{Adv}(C, K)$:

- $\mathsf{Sim}(\epsilon, 0)$ is the protocol $\mathsf{0}$
- $\mathsf{Sim}\big(C;\ \textit{input-gate}(i), K+1\big)$ is the composition of the protocol $\mathsf{Sim}(C, K)$ with the protocol
  - $\mathsf{OTBit}(\mathsf{A}, \mathsf{B}, K) := \mathsf{read}\ \mathsf{OTBit}(\mathsf{A}, \mathsf{B}, K)$
  - $\begin{cases} \mathsf{Share}(\mathsf{A}, K) := \mathsf{read}\ \mathsf{InputShare}(\mathsf{A}, \mathsf{A}, i) & \text{if } i \text{ is an input of Alice} \\ \mathsf{Share}(\mathsf{A}, K) := \mathsf{read}\ \mathsf{InputShare}(\mathsf{A}, \mathsf{B}, i) & \text{if } i \text{ is an input of Bob} \end{cases}$
  - $\begin{cases} \mathsf{Wire}(K) := \mathsf{read}\ \mathsf{In}(\mathsf{A}, i) & \text{if } i \text{ is an input of Alice} \\ \mathsf{Wire}(K) := \mathsf{read}\ \mathsf{In}(\mathsf{B}, i) & \text{if } i \text{ is an input of Bob} \end{cases}$
  - $\mathsf{OTBit}(\mathsf{A}, \mathsf{B}, K)^{\mathsf{A}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{OTBit}(\mathsf{A}, \mathsf{B}, K)^{\mathsf{A}}_{\mathsf{adv}}$
  - $\mathsf{Share}(\mathsf{A}, K)^{\mathsf{A}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{Share}(\mathsf{A}, K)$
  - $\mathsf{OTMsg}(\mathsf{A}, \mathsf{B}, K, 0)^{\mathsf{ot}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{OTMsg}(\mathsf{A}, \mathsf{B}, K, 0)^{\mathsf{ot}}_{\mathsf{adv}}$
  - $\mathsf{OTMsg}(\mathsf{A}, \mathsf{B}, K, 1)^{\mathsf{ot}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{OTMsg}(\mathsf{A}, \mathsf{B}, K, 1)^{\mathsf{ot}}_{\mathsf{adv}}$
  - $\mathsf{OTMsg}(\mathsf{A}, \mathsf{B}, K, 2)^{\mathsf{ot}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{OTMsg}(\mathsf{A}, \mathsf{B}, K, 2)^{\mathsf{ot}}_{\mathsf{adv}}$
  - $\mathsf{OTMsg}(\mathsf{A}, \mathsf{B}, K, 3)^{\mathsf{ot}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{OTMsg}(\mathsf{A}, \mathsf{B}, K, 3)^{\mathsf{ot}}_{\mathsf{adv}}$
  - $\mathsf{OTChoiceRcvd}(\mathsf{B}, \mathsf{A}, K, 0)^{\mathsf{ot}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{OTChoiceRcvd}(\mathsf{B}, \mathsf{A}, K, 0)^{\mathsf{ot}}_{\mathsf{adv}}$
  - $\mathsf{OTChoiceRcvd}(\mathsf{B}, \mathsf{A}, K, 1)^{\mathsf{ot}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{OTChoiceRcvd}(\mathsf{B}, \mathsf{A}, K, 1)^{\mathsf{ot}}_{\mathsf{adv}}$
- $\mathsf{Sim}\big(C;\ \textit{not-gate}(k), K+1\big)$ is the composition of the protocol $\mathsf{Sim}(C, K)$ with the protocol
  - $\mathsf{OTBit}(\mathsf{A}, \mathsf{B}, K) := \mathsf{read}\ \mathsf{OTBit}(\mathsf{A}, \mathsf{B}, K)$
  - $\mathsf{Share}(\mathsf{A}, K) := \mathsf{read}\ \mathsf{Share}(\mathsf{A}, k)$
  - $\mathsf{Wire}(K) := x \leftarrow \mathsf{Wire}(k);\ \mathsf{ret}\ \neg x$
  - $\mathsf{OTBit}(\mathsf{A}, \mathsf{B}, K)^{\mathsf{A}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{OTBit}(\mathsf{A}, \mathsf{B}, K)^{\mathsf{A}}_{\mathsf{adv}}$
  - $\mathsf{Share}(\mathsf{A}, K)^{\mathsf{A}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{Share}(\mathsf{A}, K)$
  - $\mathsf{OTMsg}(\mathsf{A}, \mathsf{B}, K, 0)^{\mathsf{ot}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{OTMsg}(\mathsf{A}, \mathsf{B}, K, 0)^{\mathsf{ot}}_{\mathsf{adv}}$
  - $\mathsf{OTMsg}(\mathsf{A}, \mathsf{B}, K, 1)^{\mathsf{ot}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{OTMsg}(\mathsf{A}, \mathsf{B}, K, 1)^{\mathsf{ot}}_{\mathsf{adv}}$
  - $\mathsf{OTMsg}(\mathsf{A}, \mathsf{B}, K, 2)^{\mathsf{ot}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{OTMsg}(\mathsf{A}, \mathsf{B}, K, 2)^{\mathsf{ot}}_{\mathsf{adv}}$
  - $\mathsf{OTMsg}(\mathsf{A}, \mathsf{B}, K, 3)^{\mathsf{ot}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{OTMsg}(\mathsf{A}, \mathsf{B}, K, 3)^{\mathsf{ot}}_{\mathsf{adv}}$
  - $\mathsf{OTChoiceRcvd}(\mathsf{B}, \mathsf{A}, K, 0)^{\mathsf{ot}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{OTChoiceRcvd}(\mathsf{B}, \mathsf{A}, K, 0)^{\mathsf{ot}}_{\mathsf{adv}}$
  - $\mathsf{OTChoiceRcvd}(\mathsf{B}, \mathsf{A}, K, 1)^{\mathsf{ot}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{OTChoiceRcvd}(\mathsf{B}, \mathsf{A}, K, 1)^{\mathsf{ot}}_{\mathsf{adv}}$
- $\mathsf{Sim}\big(C;\ \textit{xor-gate}(k, l), K+1\big)$ is the composition of the protocol $\mathsf{Sim}(C, K)$ with the protocol
  - $\mathsf{OTBit}(\mathsf{A}, \mathsf{B}, K) := \mathsf{read}\ \mathsf{OTBit}(\mathsf{A}, \mathsf{B}, K)$
  - $\mathsf{Share}(\mathsf{A}, K) := x_A \leftarrow \mathsf{Share}(\mathsf{A}, k);\ y_A \leftarrow \mathsf{Share}(\mathsf{A}, l);\ \mathsf{ret}\ x_A \oplus y_A$
  - $\mathsf{Wire}(K) := x \leftarrow \mathsf{Wire}(k);\ y \leftarrow \mathsf{Wire}(l);\ \mathsf{ret}\ x \oplus y$
  - $\mathsf{OTBit}(\mathsf{A}, \mathsf{B}, K)^{\mathsf{A}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{OTBit}(\mathsf{A}, \mathsf{B}, K)^{\mathsf{A}}_{\mathsf{adv}}$
  - $\mathsf{Share}(\mathsf{A}, K)^{\mathsf{A}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{Share}(\mathsf{A}, K)$
  - $\mathsf{OTMsg}(\mathsf{A}, \mathsf{B}, K, 0)^{\mathsf{ot}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{OTMsg}(\mathsf{A}, \mathsf{B}, K, 0)^{\mathsf{ot}}_{\mathsf{adv}}$

- $\mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,1)^{\mathsf{ot}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,1)^{\mathsf{ot}}_{\mathsf{adv}}$
- $\mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,2)^{\mathsf{ot}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,2)^{\mathsf{ot}}_{\mathsf{adv}}$
- $\mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,3)^{\mathsf{ot}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,3)^{\mathsf{ot}}_{\mathsf{adv}}$
- $\mathsf{OTChoiceRcvd}(\mathsf{B},\mathsf{A},K,0)^{\mathsf{ot}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{OTChoiceRcvd}(\mathsf{B},\mathsf{A},K,0)^{\mathsf{ot}}_{\mathsf{adv}}$
- $\mathsf{OTChoiceRcvd}(\mathsf{B},\mathsf{A},K,1)^{\mathsf{ot}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{OTChoiceRcvd}(\mathsf{B},\mathsf{A},K,1)^{\mathsf{ot}}_{\mathsf{adv}}$

- $\mathsf{Sim}\big(C;\ \textit{and-gate}(k,l), K+1\big)$ is the composition of the protocol $\mathsf{Sim}(C,K)$ with the protocol
  - $\mathsf{OTBit}(\mathsf{A},\mathsf{B},K) := x_A \leftarrow \mathsf{Share}(\mathsf{A},k);\ y_A \leftarrow \mathsf{Share}(\mathsf{A},l);\ \mathsf{samp\ flip}$
  - $\mathsf{Share}(\mathsf{A},K) := b_A \leftarrow \mathsf{OTBit}(\mathsf{A},\mathsf{B},K);\ x_A \leftarrow \mathsf{Share}(\mathsf{A},k);\ y_A \leftarrow \mathsf{Share}(\mathsf{A},l);\ \mathsf{ret}\ (x_A * y_A) \oplus b_A$
  - $\mathsf{Wire}(K) := x \leftarrow \mathsf{Wire}(k);\ y \leftarrow \mathsf{Wire}(l);\ \mathsf{ret}\ x * y$
  - $\mathsf{OTBit}(\mathsf{A},\mathsf{B},K)^{\mathsf{A}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{OTBit}(\mathsf{A},\mathsf{B},K)$
  - $\mathsf{Share}(\mathsf{A},K)^{\mathsf{A}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{Share}(\mathsf{A},K)$
  - $\mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,0)^{\mathsf{ot}}_{\mathsf{adv}} := b_A \leftarrow \mathsf{OTBit}(\mathsf{A},\mathsf{B},K);\ x_A \leftarrow \mathsf{Share}(\mathsf{A},k);\ y_A \leftarrow \mathsf{Share}(\mathsf{A},l);\ \mathsf{ret}\ b_A$
  - $\mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,1)^{\mathsf{ot}}_{\mathsf{adv}} := b_A \leftarrow \mathsf{OTBit}(\mathsf{A},\mathsf{B},K);\ x_A \leftarrow \mathsf{Share}(\mathsf{A},k);\ y_A \leftarrow \mathsf{Share}(\mathsf{A},l);\ \mathsf{ret}\ b_A \oplus x_A$
  - $\mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,2)^{\mathsf{ot}}_{\mathsf{adv}} := b_A \leftarrow \mathsf{OTBit}(\mathsf{A},\mathsf{B},K);\ x_A \leftarrow \mathsf{Share}(\mathsf{A},k);\ y_A \leftarrow \mathsf{Share}(\mathsf{A},l);\ \mathsf{ret}\ b_A \oplus y_A$
  - $\mathsf{OTMsg}(\mathsf{A},\mathsf{B},K,3)^{\mathsf{ot}}_{\mathsf{adv}} := b_A \leftarrow \mathsf{OTBit}(\mathsf{A},\mathsf{B},K);\ x_A \leftarrow \mathsf{Share}(\mathsf{A},k);\ y_A \leftarrow \mathsf{Share}(\mathsf{A},l);\ \mathsf{ret}\ b_A \oplus x_A \oplus y_A$
  - $\mathsf{OTChoiceRcvd}(\mathsf{B},\mathsf{A},K,0)^{\mathsf{ot}}_{\mathsf{adv}} := \_ \leftarrow \mathsf{Share}(\mathsf{A},k);\ \mathsf{ret}\ \checkmark$
  - $\mathsf{OTChoiceRcvd}(\mathsf{B},\mathsf{A},K,1)^{\mathsf{ot}}_{\mathsf{adv}} := \_ \leftarrow \mathsf{Share}(\mathsf{A},l);\ \mathsf{ret}\ \checkmark$

9.5.3 The Simulator: The Final Phase

The code for the final part of the simulator is as follows:

- $\begin{cases} \mathsf{SendFinalShare}(\mathsf{A},k)^{\mathsf{A}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{Share}(\mathsf{A},k) & \text{if wire } 0 \leqslant k < K \text{ is output} \\ \mathsf{SendFinalShare}(\mathsf{A},k)^{\mathsf{A}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{SendFinalShare}(\mathsf{A},k)^{\mathsf{A}}_{\mathsf{adv}} & \text{otherwise} \end{cases}$
- $\begin{cases} \mathsf{SendFinalShare}(\mathsf{B},k)^{\mathsf{A}}_{\mathsf{adv}} := x_A \leftarrow \mathsf{Share}(\mathsf{A},k); x \leftarrow \mathsf{Out}(\mathsf{A},k)^{\mathsf{id}}_{\mathsf{adv}};\ \mathsf{ret}\ x_A \oplus x & \text{if wire } 0 \leqslant k < K \text{ is output} \\ \mathsf{SendFinalShare}(\mathsf{B},k)^{\mathsf{A}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{SendFinalShare}(\mathsf{B},k)^{\mathsf{A}}_{\mathsf{adv}} & \text{otherwise} \end{cases}$
- $\mathsf{Out}(\mathsf{A},k)^{\mathsf{A}}_{\mathsf{adv}} := \mathsf{read}\ \mathsf{Out}(\mathsf{A},k)^{\mathsf{id}}_{\mathsf{adv}}$ for $0 \leqslant k < K$