Selected IPDL Case Studies

July 8, 2022

1 Authenticated-To-Secure Channel: IND-CPA Encryption

We assume a functionality that securely broadcasts a shared key to Alice and Bob. Alice wants to communicate q messages to Bob using an authenticated channel. The authenticated channel leaks each message to the adversary, and waits to receive an ok message back before delivering the in-flight message. To do so securely, Alice sends encryptions of her messages, which Bob decrypts using the shared key.

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 distribution $\mu : 1 \twoheadrightarrow \mathsf{key}$ on keys; and functions

$$\mathsf{enc} : \mathsf{msg} \times \mathsf{key} \twoheadrightarrow \mathsf{ctxt}$$
$$\mathsf{dec} : \mathsf{ctxt} \times \mathsf{key} \to \mathsf{msg}$$

The encode enc takes a message and a key and returns a distribution on ciphertexts; the decode function dec takes a ciphertext and a key and returns a message.

1.1 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 okay from the adversary, outputs the message:

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

1.2 The Real Protocol

The real-world protocol consists of Alice, Bob, the key-generating functionality, and the authenticated channel. The functionality samples a key from the key distribution:

- $\mathsf{Key} := \mathsf{samp}(\mu)$

Alice encodes 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 the adversary, and, upon receiving the okay from the adversary, forwards the ciphertext to Bob:

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

Bob decodes 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.

1.3 The Functional Assumptions

The *decryption-correctness* functional assumption states that encoding and decoding a single message yields the original message. That is, the protocol

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

rewrites strictly to

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

From this, it is easy to prove that encoding and decoding q messages yields the respective original messages. That is, the protocol

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

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

1.4 The Cryptographic Assumptions

The *indistinguishability under chosen plaintext attack (IND-CPA)* cryptographic assumption states that if the key is secret, encoding q arbitrary messages is computationally indistinguishable from encoding the chosen message q times. That is, the protocol

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

where Key is a hidden channel rewrites approximately to

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

1.5 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 channels $\mathsf{LeakMsgRcvd}^{\mathsf{id}}_{\mathsf{adv}}(-), \mathsf{OkCtxt}^{\mathsf{adv}}_{\mathsf{net}}(-)$ are inputs to the simulator and 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 the adversary. This is accomplished by generating a random key and encoding the chosen message:

- $\mathsf{Key} := \mathsf{samp}(\mu)$

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

Upon receiving the okay from the adversary for the generated ciphertext, the simulator gives the okay to the functionality to output the message:

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

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

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

1.6 Real ≈ Ideal + Simulator

Plugging in the simulator into the ideal functionality and hiding the internal communication yields the following:

- $\mathsf{Key} := \mathsf{samp}(\mu)$
- $\mathsf{LeakCtxt}^{\mathsf{net}}_{\mathsf{adv}}(i) := \_ \leftarrow \mathsf{LeakMsgRcvd}^{\mathsf{id}}_{\mathsf{adv}}(i);\ k \leftarrow \mathsf{Key};\ \mathsf{samp}(\mathsf{enc}\ (\mathsf{zeros}, k))$ for $0 \leqslant i < q$
- $\mathsf{OkMsg}^{\mathsf{adv}}_{\mathsf{id}}(i) := \mathsf{read}\ \mathsf{OkCtxt}^{\mathsf{adv}}_{\mathsf{net}}(i)$ for $0 \leqslant i < q$
- $\mathsf{LeakMsgRcvd}^{\mathsf{id}}_{\mathsf{adv}}(i) := m \leftarrow \mathsf{In}(i);\ \mathsf{ret}\ \checkmark$ for $0 \leqslant i < q$
- $\mathsf{Out}(i) := \_ \leftarrow \mathsf{OkMsg}^{\mathsf{adv}}_{\mathsf{id}}(i);\ \mathsf{read}\ \mathsf{In}(i)$ 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 the adversary can now be substituted away:

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

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

- $\mathsf{Key} := \mathsf{samp}(\ )\mu$
- $\mathsf{Send}(i) := m \leftarrow \mathsf{In}(i);\ k \leftarrow \mathsf{Key};\ \mathsf{samp}(\mathsf{enc}\ (m, k))$ for $0 \leqslant i < q$
- $\mathsf{LeakCtxt}^{\mathsf{net}}_{\mathsf{adv}}(i) := \mathsf{read}\ \mathsf{Send}(i)$ for $0 \leqslant i < q$
- $\mathsf{Recv}(i) := \_ \leftarrow \mathsf{OkCtxt}^{\mathsf{adv}}_{\mathsf{net}}(i);\ \mathsf{read}\ \mathsf{Send}(i)$ 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 conceptually separate the encryption and decryption actions from the message-passing in the real-world by introducing new internal channels $\mathsf{Enc}(-)$ and $\mathsf{Dec}(-)$:

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

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

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

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

The functional assumption of encyption-decryption correctness allows us to strictly rewrite the above protocol to the following one:

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

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

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

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

Plugging this into the original protocol yields the following:

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

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

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

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

Bob wishes to obtain exactly one of Alice's four messages, without revealing his choice. The "oblivious" part of the transfer refers to the fact that both Alice and Bob remain oblivious to the other one's secret - Alice never gets to learn which of the four messages Bob wanted, whereas Bob never finds out anything about the remaining three messages. We show that this 1-out-of-4 Oblivious Transfer (OT) construction can be realized from three idealized 1-out-of-2 OT instances, where, in each case, Bob only selects among two messages. Formally, we assume a type msg of messages; a distribution $\mu : 1 \twoheadrightarrow \mathsf{msg}$ on messages; and a bitwise xor function

$$\oplus : \mathsf{msg} \times \mathsf{msg} \to \mathsf{msg}$$

where we write $x \oplus y$ in place of $\oplus\,(x, y)$.

2 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 distribution $\mu : 1 \twoheadrightarrow \mathsf{msg}$ on messages; and a bitwise xor function

$$\oplus : \mathsf{msg} \times \mathsf{msg} \to \mathsf{msg}$$

where we write $x \oplus y$ in place of $\oplus\,(x, y)$.

2.1 The Ideal Functionality

The ideal functionality reads two messages m_0, m_1 from the sender, and one Boolean c from the receiver, and outputs message

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

Since the receiver is corrupted, the selected message and the receiver's choice are leaked to the adversary:

- $\mathsf{Out} := m_0 \leftarrow \mathsf{In}(0);\ m_1 \leftarrow \mathsf{In}(1);\ c \leftarrow \mathsf{Choice};\ \mathsf{if}\ c\ \mathsf{then\ ret}\ m_1\ \mathsf{else\ ret}\ m_0$
- $\mathsf{LeakChoice}^{\mathsf{id}}_{\mathsf{adv}} := \mathsf{read\ Choice}$
- $\mathsf{LeakOut}^{\mathsf{id}}_{\mathsf{adv}} := \mathsf{read\ Out}$

2.2 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:

- $\mathsf{Key}(0) := \mathsf{samp}(\mu)$
- $\mathsf{Key}(1) := \mathsf{samp}(\mu)$

Bob randomly decides on which key he wants and informs the adversary:

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

The OT functionality selects the corresponding key and sends it to Bob:

- $\mathsf{SharedKey} := f \leftarrow \mathsf{Flip};\ k_0 \leftarrow \mathsf{Key}(0);\ k_1 \leftarrow \mathsf{Key}(1);\ \mathsf{if}\ f\ \mathsf{then\ ret}\ k_1\ \mathsf{else\ ret}\ k_0$

Upon receiving the key, Bob leaks it to the adversary:

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

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

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

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

- $\mathsf{ChoiceEnc} := c \leftarrow \mathsf{Choice};\ f \leftarrow \mathsf{Flip};$
  $\mathsf{if}\ f\ \mathsf{then}\ \big(\mathsf{if}\ c\ \mathsf{then\ ret\ false\ else\ ret\ true}\big)\ \mathsf{else}\ \big(\mathsf{if}\ c\ \mathsf{then\ ret\ true\ else\ ret\ false}\big)$
- $\mathsf{LeakChoiceEnc}^{\mathsf{rec}}_{\mathsf{adv}} := \mathsf{read\ ChoiceEnc}$

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:

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

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

- $\mathsf{LeakMsgEnc}^{\mathsf{rec}}_{\mathsf{adv}}(0) := \mathsf{read\ MsgEnc}(0)$
- $\mathsf{LeakMsgEnc}^{\mathsf{rec}}_{\mathsf{adv}}(1) := \mathsf{read\ MsgEnc}(1)$

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:

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

Thus, we have the following code for Alice:

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

The code for Bob has the following form:

- $\mathsf{Flip} := \mathsf{samp}(\mathsf{flip})$
- $\mathsf{LeakFlip}^{\mathsf{rec}}_{\mathsf{adv}} := \mathsf{read\ Flip}$
- $\mathsf{LeakSharedKey}^{\mathsf{rec}}_{\mathsf{adv}} := \mathsf{read\ SharedKey}$
- $\mathsf{LeakChoice}^{\mathsf{rec}}_{\mathsf{adv}} := \mathsf{read\ Choice}$

- ChoiceEnc := $c \leftarrow$ Choice; $f \leftarrow$ Flip;
  if f then $\big($if c then ret false else ret true$\big)$ else $\big($if c then ret true else ret false$\big)$
- $\mathsf{LeakChoiceEnc}^{\mathsf{rec}}_{\mathsf{adv}}$:= read ChoiceEnc
- $\mathsf{LeakMsgEnc}^{\mathsf{rec}}_{\mathsf{adv}}(0)$:= read MsgEnc(0)
- $\mathsf{LeakMsgEnc}^{\mathsf{rec}}_{\mathsf{adv}}(1)$:= read MsgEnc(1)
- Out := $e_0 \leftarrow$ MsgEnc(0); $e_1 \leftarrow$ MsgEnc(1); $c \leftarrow$ Choice; $s \leftarrow$ SharedKey;
  if c then ret $e_1 \oplus s$ else ret $e_0 \oplus s$
- $\mathsf{LeakOut}^{\mathsf{rec}}_{\mathsf{adv}}$:= read Out

Finally, we recall the code for the OT functionality:

- SharedKey := $f \leftarrow$ Flip; $k_0 \leftarrow$ Key(0); $k_1 \leftarrow$ Key(1); if f then ret k_1 else ret k_0

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

2.3 The Functional Assumptions

We assume that the operation of bitwise xor with a fixed message is self-inverse: if $\mathsf{In}(0), \mathsf{In}(1), \mathsf{Out}$ are free channels of type msg, then the respective single-reaction protocols

- Out := $x \leftarrow$ read In(0); $y \leftarrow$ read In(1); ret $x \oplus (x \oplus y)$, and
- Out := $x \leftarrow$ read In(0); $y \leftarrow$ read In(1); ret y

are strictly equivalent, and so are the protocols

- Out := $x \leftarrow$ read In(0); $y \leftarrow$ read In(1); ret $(x \oplus y) \oplus y$, and
- Out := $x \leftarrow$ read In(0); $y \leftarrow$ read In(1); ret x.

We furthermore assume that the distribution μ on messages is invariant under the operation of xor-ing with a fixed message: if $\mathsf{In}, \mathsf{Out}$ are free channels of type msg, the protocols

- Out := $x \leftarrow$ read In; $y \leftarrow$ samp(μ); ret $x \oplus y$, and
- Out := $x \leftarrow$ read In; samp(μ)

are strictly equivalent.

2.4 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:

- Key(0) := samp(μ)
- Key(1) := samp(μ)
- Flip := samp(flip)
- $\mathsf{LeakFlip}^{\mathsf{rec}}_{\mathsf{adv}}$:= read Flip
- SharedKey := $f \leftarrow$ Flip; $k_0 \leftarrow$ Key(0); $k_1 \leftarrow$ Key(1); if f then ret k_1 else ret k_0
- $\mathsf{LeakSharedKey}^{\mathsf{rec}}_{\mathsf{adv}}$:= read SharedKey
- $\mathsf{LeakChoice}^{\mathsf{rec}}_{\mathsf{adv}}$:= read Choice

- ChoiceEnc := $c \leftarrow$ Choice; $f \leftarrow$ Flip;
  if f then $\big($if c then ret false else ret true$\big)$ else $\big($if c then ret true else ret false$\big)$
- $\mathsf{LeakChoiceEnc}^{\mathsf{rec}}_{\mathsf{adv}}$:= read ChoiceEnc
- MsgEnc(0) := $m_0 \leftarrow$ In(0); $m_1 \leftarrow$ In(1); $e \leftarrow$ ChoiceEnc; $k_0 \leftarrow$ Key(0); $k_1 \leftarrow$ Key(1);
  if e then ret $m_0 \oplus k_1$ else ret $m_0 \oplus k_0$
- MsgEnc(1) := $m_0 \leftarrow$ In(0); $m_1 \leftarrow$ In(1); $e \leftarrow$ ChoiceEnc; $k_0 \leftarrow$ Key(0); $k_1 \leftarrow$ Key(1);
  if e then ret $m_1 \oplus k_0$ else ret $m_1 \oplus k_1$
- $\mathsf{LeakMsgEnc}^{\mathsf{rec}}_{\mathsf{adv}}(0)$:= read MsgEnc(0)
- $\mathsf{LeakMsgEnc}^{\mathsf{rec}}_{\mathsf{adv}}(1)$:= read MsgEnc(1)
- Out := $e_0 \leftarrow$ MsgEnc(0); $e_1 \leftarrow$ MsgEnc(1); $c \leftarrow$ Choice; $s \leftarrow$ SharedKey;
  if c then ret $e_1 \oplus s$ else ret $e_0 \oplus s$
- $\mathsf{LeakOut}^{\mathsf{rec}}_{\mathsf{adv}}$:= read Out

Substituting the channel ChoiceEnc into MsgEnc(0) and MsgEnc(1) yields:

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

Substituting the channel SharedKey into Out yields:

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

Further substituting the channels MsgEnc(0) and MsgEnc(1) into Out yields:

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

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

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

After simplifying we get:

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

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

- Key(0) := samp(μ)
- Key(1) := samp(μ)
- Flip := samp(flip)
- $\mathsf{LeakFlip}^{\mathsf{rec}}_{\mathsf{adv}}$:= read Flip
- SharedKey := $f \leftarrow$ Flip; $k_0 \leftarrow$ Key(0); $k_1 \leftarrow$ Key(1); if f then ret k_1 else ret k_0

- $\mathsf{LeakSharedKey}^{\mathsf{rec}}_{\mathsf{adv}}$:= read SharedKey
- $\mathsf{LeakChoice}^{\mathsf{rec}}_{\mathsf{adv}}$:= read Choice
- ChoiceEnc := $c \leftarrow$ Choice; $f \leftarrow$ Flip;
  if f then $\big($if c then ret false else ret true$\big)$ else $\big($if c then ret true else ret false$\big)$
- $\mathsf{LeakChoiceEnc}^{\mathsf{rec}}_{\mathsf{adv}}$:= read ChoiceEnc
- MsgEnc(0) := $m_0 \leftarrow \mathsf{In}(0)$; $m_1 \leftarrow \mathsf{In}(1)$; $c \leftarrow$ Choice; $f \leftarrow$ Flip; $k_0 \leftarrow \mathsf{Key}(0)$; $k_1 \leftarrow \mathsf{Key}(1)$;
  if f then $\big($if c then ret $m_0 \oplus k_0$ else ret $m_0 \oplus k_1\big)$ else $\big($if c then ret $m_0 \oplus k_1$ else ret $m_0 \oplus k_0\big)$
- MsgEnc(1) := $m_0 \leftarrow \mathsf{In}(0)$; $m_1 \leftarrow \mathsf{In}(1)$; $c \leftarrow$ Choice; $f \leftarrow$ Flip; $k_0 \leftarrow \mathsf{Key}(0)$; $k_1 \leftarrow \mathsf{Key}(1)$;
  if f then $\big($if c then ret $m_1 \oplus k_1$ else ret $m_1 \oplus k_0\big)$ else $\big($if c then ret $m_1 \oplus k_0$ else ret $m_1 \oplus k_1\big)$
- $\mathsf{LeakMsgEnc}^{\mathsf{rec}}_{\mathsf{adv}}(0)$:= read MsgEnc(0)
- $\mathsf{LeakMsgEnc}^{\mathsf{rec}}_{\mathsf{adv}}(1)$:= read MsgEnc(1)
- Out := $m_0 \leftarrow \mathsf{In}(0)$; $m_1 \leftarrow \mathsf{In}(1)$; $c \leftarrow$ Choice; if c then ret m_1 else ret m_0
- $\mathsf{LeakOut}^{\mathsf{rec}}_{\mathsf{adv}}$:= read 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:

- Key(0) := samp(μ)
- Key(1) := samp(μ)
- Flip := samp(flip)
- $\mathsf{LeakFlip}^{\mathsf{rec}}_{\mathsf{adv}}$:= read Flip
- KeyPair := $f \leftarrow$ Flip; $k_0 \leftarrow \mathsf{Key}(0)$; $k_1 \leftarrow \mathsf{Key}(1)$; if f then ret (k_1, k_0) else ret (k_0, k_1)
- SharedKey := $k \leftarrow$ KeyPair; ret $(\mathsf{fst}_{\mathsf{msg}}\ k)$
- $\mathsf{LeakSharedKey}^{\mathsf{rec}}_{\mathsf{adv}}$:= read SharedKey
- $\mathsf{LeakChoice}^{\mathsf{rec}}_{\mathsf{adv}}$:= read Choice
- ChoiceEnc := $c \leftarrow$ Choice; $f \leftarrow$ Flip;
  if f then $\big($if c then ret false else ret true$\big)$ else $\big($if c then ret true else ret false$\big)$
- $\mathsf{LeakChoiceEnc}^{\mathsf{rec}}_{\mathsf{adv}}$:= read ChoiceEnc
- MsgEnc(0) := $m_0 \leftarrow \mathsf{In}(0)$; $m_1 \leftarrow \mathsf{In}(1)$; $c \leftarrow$ Choice; $k \leftarrow$ KeyPair;
  if c then ret $m_0 \oplus (\mathsf{snd}_{\mathsf{msg}}\ k)$ else ret $m_0 \oplus (\mathsf{fst}_{\mathsf{msg}}\ k)$
- MsgEnc(1) := $m_0 \leftarrow \mathsf{In}(0)$; $m_1 \leftarrow \mathsf{In}(1)$; $c \leftarrow$ Choice; $k \leftarrow$ KeyPair;
  if c then ret $m_1 \oplus (\mathsf{fst}_{\mathsf{msg}}\ k)$ else ret $m_1 \oplus (\mathsf{snd}_{\mathsf{msg}}\ k)$
- $\mathsf{LeakMsgEnc}^{\mathsf{rec}}_{\mathsf{adv}}(0)$:= read MsgEnc(0)
- $\mathsf{LeakMsgEnc}^{\mathsf{rec}}_{\mathsf{adv}}(1)$:= read MsgEnc(1)
- Out := $m_0 \leftarrow \mathsf{In}(0)$; $m_1 \leftarrow \mathsf{In}(1)$; $c \leftarrow$ Choice; if c then ret m_1 else ret m_0
- $\mathsf{LeakOut}^{\mathsf{rec}}_{\mathsf{adv}}$:= read Out

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

- $\mathsf{Flip} := \mathsf{samp}(\mathsf{flip})$
- $\mathsf{LeakFlip}^{\mathsf{rec}}_{\mathsf{adv}} := \text{read Flip}$
- $\mathsf{KeyPair} := f \leftarrow \mathsf{Flip};\ \text{if } f \text{ then } k_0 \leftarrow \mathsf{samp}(\mu);\ k_1 \leftarrow \mathsf{samp}(\mu);\ \text{ret } (k_1, k_0)$
  $\text{else } k_0 \leftarrow \mathsf{samp}(\mu);\ k_1 \leftarrow \mathsf{samp}(\mu);\ \text{ret } (k_0, k_1)$
- $\mathsf{SharedKey} := k \leftarrow \mathsf{KeyPair};\ \text{ret } (\mathsf{fst}_{\mathsf{msg}}\ k)$
- $\mathsf{LeakSharedKey}^{\mathsf{rec}}_{\mathsf{adv}} := \text{read SharedKey}$
- $\mathsf{LeakChoice}^{\mathsf{rec}}_{\mathsf{adv}} := \text{read Choice}$
- $\mathsf{ChoiceEnc} := c \leftarrow \mathsf{Choice};\ f \leftarrow \mathsf{Flip};$
  $\text{if } f \text{ then } \big(\text{if } c \text{ then ret false else ret true}\big) \text{ else } \big(\text{if } c \text{ then ret true else ret false}\big)$
- $\mathsf{LeakChoiceEnc}^{\mathsf{rec}}_{\mathsf{adv}} := \text{read ChoiceEnc}$
- $\mathsf{MsgEnc}(0) := m_0 \leftarrow \mathsf{In}(0);\ m_1 \leftarrow \mathsf{In}(1);\ c \leftarrow \mathsf{Choice};\ k \leftarrow \mathsf{KeyPair};$
  $\text{if } c \text{ then ret } m_0 \oplus (\mathsf{snd}_{\mathsf{msg}}\ k) \text{ else ret } m_0 \oplus (\mathsf{fst}_{\mathsf{msg}}\ k)$
- $\mathsf{MsgEnc}(1) := m_0 \leftarrow \mathsf{In}(0);\ m_1 \leftarrow \mathsf{In}(1);\ c \leftarrow \mathsf{Choice};\ k \leftarrow \mathsf{KeyPair};$
  $\text{if } c \text{ then ret } m_1 \oplus (\mathsf{fst}_{\mathsf{msg}}\ k) \text{ else ret } m_1 \oplus (\mathsf{snd}_{\mathsf{msg}}\ k)$
- $\mathsf{LeakMsgEnc}^{\mathsf{rec}}_{\mathsf{adv}}(0) := \text{read MsgEnc}(0)$
- $\mathsf{LeakMsgEnc}^{\mathsf{rec}}_{\mathsf{adv}}(1) := \text{read MsgEnc}(1)$
- $\mathsf{Out} := m_0 \leftarrow \mathsf{In}(0);\ m_1 \leftarrow \mathsf{In}(1);\ c \leftarrow \mathsf{Choice};\ \text{if } c \text{ then ret } m_1 \text{ else ret } m_0$
- $\mathsf{LeakOut}^{\mathsf{rec}}_{\mathsf{adv}} := \text{read Out}$

Since the samplings are interchangeable, we end up doing the same thing either way:

- $\mathsf{KeyPair} := f \leftarrow \mathsf{Flip};\ \text{if } f \text{ then } k_1 \leftarrow \mathsf{samp}(\mu);\ k_0 \leftarrow \mathsf{samp}(\mu);\ \text{ret } (k_1, k_0)$
  $\text{else } k_0 \leftarrow \mathsf{samp}(\mu);\ k_1 \leftarrow \mathsf{samp}(\mu);\ \text{ret } (k_0, k_1)$

So we may just as well not flip:

- $\mathsf{KeyPair} := k_0 \leftarrow \mathsf{samp}(\mu);\ k_1 \leftarrow \mathsf{samp}(\mu);\ \text{ret } (k_0, k_1)$

Unfolding the samplings gives us:

- $\mathsf{Key}(0) := \mathsf{samp}(\mu)$
- $\mathsf{Key}(1) := \mathsf{samp}(\mu)$
- $\mathsf{Flip} := \mathsf{samp}(\mathsf{flip})$
- $\mathsf{LeakFlip}^{\mathsf{rec}}_{\mathsf{adv}} := \text{read Flip}$
- $\mathsf{KeyPair} := k_0 \leftarrow \mathsf{Key}(0);\ k_1 \leftarrow \mathsf{Key}(1);\ \text{ret } (k_0, k_1)$
- $\mathsf{SharedKey} := k \leftarrow \mathsf{KeyPair};\ \text{ret } (\mathsf{fst}_{\mathsf{msg}}\ k)$
- $\mathsf{LeakSharedKey}^{\mathsf{rec}}_{\mathsf{adv}} := \text{read SharedKey}$
- $\mathsf{LeakChoice}^{\mathsf{rec}}_{\mathsf{adv}} := \text{read Choice}$

- ChoiceEnc := $c \leftarrow$ Choice; $f \leftarrow$ Flip;
  if f then $\big($if c then ret false else ret true$\big)$ else $\big($if c then ret true else ret false$\big)$
- LeakChoiceEnc$^{\text{rec}}_{\text{adv}}$:= read ChoiceEnc
- MsgEnc(0) := $m_0 \leftarrow$ In(0); $m_1 \leftarrow$ In(1); $c \leftarrow$ Choice; $k \leftarrow$ KeyPair;
  if c then ret $m_0 \oplus (\text{snd}_{\text{msg}}\ k)$ else ret $m_0 \oplus (\text{fst}_{\text{msg}}\ k)$
- MsgEnc(1) := $m_0 \leftarrow$ In(0); $m_1 \leftarrow$ In(1); $c \leftarrow$ Choice; $k \leftarrow$ KeyPair;
  if c then ret $m_1 \oplus (\text{fst}_{\text{msg}}\ k)$ else ret $m_1 \oplus (\text{snd}_{\text{msg}}\ k)$
- LeakMsgEnc$^{\text{rec}}_{\text{adv}}(0)$:= read MsgEnc(0)
- LeakMsgEnc$^{\text{rec}}_{\text{adv}}(1)$:= read MsgEnc(1)
- Out := $m_0 \leftarrow$ In(0); $m_1 \leftarrow$ In(1); $c \leftarrow$ Choice; if c then ret m_1 else ret m_0
- LeakOut$^{\text{rec}}_{\text{adv}}$:= read Out

The internal channel KeyPair can now be substituted away:

- Key(0) := samp(μ)
- Key(1) := samp(μ)
- Flip := samp(flip)
- LeakFlip$^{\text{rec}}_{\text{adv}}$:= read Flip
- SharedKey := read Key(0)
- LeakSharedKey$^{\text{rec}}_{\text{adv}}$:= read SharedKey
- LeakChoice$^{\text{rec}}_{\text{adv}}$:= read Choice
- ChoiceEnc := $c \leftarrow$ Choice; $f \leftarrow$ Flip;
  if f then $\big($if c then ret false else ret true$\big)$ else $\big($if c then ret true else ret false$\big)$
- LeakChoiceEnc$^{\text{rec}}_{\text{adv}}$:= read ChoiceEnc
- MsgEnc(0) := $m_0 \leftarrow$ In(0); $m_1 \leftarrow$ In(1); $c \leftarrow$ Choice; $k_0 \leftarrow$ Key(0); $k_1 \leftarrow$ Key(1);
  if c then ret $m_0 \oplus k_1$ else ret $m_0 \oplus k_0$
- MsgEnc(1) := $m_0 \leftarrow$ In(0); $m_1 \leftarrow$ In(1); $c \leftarrow$ Choice; $k_0 \leftarrow$ Key(0); $k_1 \leftarrow$ Key(1);
  if c then ret $m_1 \oplus k_0$ else ret $m_1 \oplus k_1$
- LeakMsgEnc$^{\text{rec}}_{\text{adv}}(0)$:= read MsgEnc(0)
- LeakMsgEnc$^{\text{rec}}_{\text{adv}}(1)$:= read MsgEnc(1)
- Out := $m_0 \leftarrow$ In(0); $m_1 \leftarrow$ In(1); $c \leftarrow$ Choice; if c then ret m_1 else ret m_0
- LeakOut$^{\text{rec}}_{\text{adv}}$:= read Out

The second key is now only referenced in the channels MsgEnc(0) and 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:

- Key(0) := samp(μ)
- Key(1) := samp(μ)

- Flip := samp(flip)
- $\mathsf{LeakFlip}^{\mathsf{rec}}_{\mathsf{adv}}$:= read Flip
- SharedKey := read Key(0)
- $\mathsf{LeakSharedKey}^{\mathsf{rec}}_{\mathsf{adv}}$:= read SharedKey
- $\mathsf{LeakChoice}^{\mathsf{rec}}_{\mathsf{adv}}$:= read Choice
- ChoiceEnc := $c \leftarrow$ Choice; $f \leftarrow$ Flip;
  if f then $\big($if c then ret false else ret true$\big)$ else $\big($if c then ret true else ret false$\big)$
- $\mathsf{LeakChoiceEnc}^{\mathsf{rec}}_{\mathsf{adv}}$:= read ChoiceEnc
- PrivateMsg := $m_0 \leftarrow$ In(0); $m_1 \leftarrow$ In(1); $c \leftarrow$ Choice; $k_1 \leftarrow$ Key(1);
  if c then ret $m_0 \oplus k_1$ else ret $m_1 \oplus k_1$
- MsgEnc(0) := $m_0 \leftarrow$ In(0); $m_1 \leftarrow$ In(1); $c \leftarrow$ Choice; $k_0 \leftarrow$ Key(0); $p \leftarrow$ PrivateMsg;
  if c then ret p else ret $m_0 \oplus k_0$
- MsgEnc(1) := $m_0 \leftarrow$ In(0); $m_1 \leftarrow$ In(1); $c \leftarrow$ Choice; $k_0 \leftarrow$ Key(0); $p \leftarrow$ PrivateMsg;
  if c then ret $m_1 \oplus k_0$ else ret p
- $\mathsf{LeakMsgEnc}^{\mathsf{rec}}_{\mathsf{adv}}(0)$:= read MsgEnc(0)
- $\mathsf{LeakMsgEnc}^{\mathsf{rec}}_{\mathsf{adv}}(1)$:= read MsgEnc(1)
- Out := $m_0 \leftarrow$ In(0); $m_1 \leftarrow$ In(1); $c \leftarrow$ Choice; if c then ret m_1 else ret m_0
- $\mathsf{LeakOut}^{\mathsf{rec}}_{\mathsf{adv}}$:= read Out

We can now fold the internal channel Key(1) into the channel PrivateMsg:

- Key(0) := samp(μ)
- Flip := samp(flip)
- $\mathsf{LeakFlip}^{\mathsf{rec}}_{\mathsf{adv}}$:= read Flip
- SharedKey := read Key(0)
- $\mathsf{LeakSharedKey}^{\mathsf{rec}}_{\mathsf{adv}}$:= read SharedKey
- $\mathsf{LeakChoice}^{\mathsf{rec}}_{\mathsf{adv}}$:= read Choice
- ChoiceEnc := $c \leftarrow$ Choice; $f \leftarrow$ Flip;
  if f then $\big($if c then ret false else ret true$\big)$ else $\big($if c then ret true else ret false$\big)$
- $\mathsf{LeakChoiceEnc}^{\mathsf{rec}}_{\mathsf{adv}}$:= read ChoiceEnc
- PrivateMsg := $m_0 \leftarrow$ In(0); $m_1 \leftarrow$ In(1); $c \leftarrow$ Choice;
  if c then $\big(k_1 \leftarrow$ samp(μ); ret $m_0 \oplus k_1\big)$ else $\big(k_1 \leftarrow$ samp(μ); ret $m_1 \oplus k_1\big)$
- MsgEnc(0) := $m_0 \leftarrow$ In(0); $m_1 \leftarrow$ In(1); $c \leftarrow$ Choice; $k_0 \leftarrow$ Key(0); $p \leftarrow$ PrivateMsg;
  if c then ret p else ret $m_0 \oplus k_0$
- MsgEnc(1) := $m_0 \leftarrow$ In(0); $m_1 \leftarrow$ In(1); $c \leftarrow$ Choice; $k_0 \leftarrow$ Key(0); $p \leftarrow$ PrivateMsg;
  if c then ret $m_1 \oplus k_0$ else ret p
- $\mathsf{LeakMsgEnc}^{\mathsf{rec}}_{\mathsf{adv}}(0)$:= read MsgEnc(0)

- $\mathsf{LeakMsgEnc}^{\mathsf{rec}}_{\mathsf{adv}}(1) := \text{read } \mathsf{MsgEnc}(1)$
- $\mathsf{Out} := m_0 \leftarrow \mathsf{In}(0);\ m_1 \leftarrow \mathsf{In}(1);\ c \leftarrow \mathsf{Choice};\ \text{if } c \text{ then ret } m_1 \text{ else ret } m_0$
- $\mathsf{LeakOut}^{\mathsf{rec}}_{\mathsf{adv}} := \text{read Out}$

Our functional assumptions imply that the single-reaction protocol

- $\mathsf{PrivateMsg} := m_0 \leftarrow \mathsf{In}(0);\ m_1 \leftarrow \mathsf{In}(1);\ c \leftarrow \mathsf{Choice};$
  $\text{if } c \text{ then } \big(k_1 \leftarrow \mathsf{samp}(\mu);\ \text{ret } m_0 \oplus k_1\big) \text{ else } \big(k_1 \leftarrow \mathsf{samp}(\mu);\ \text{ret } m_1 \oplus k_1\big)$

rewrites strictly to

- $\mathsf{PrivateMsg} := m_0 \leftarrow \mathsf{In}(0);\ m_1 \leftarrow \mathsf{In}(1);\ c \leftarrow \mathsf{Choice};\ \mathsf{samp}(\mu)$

Unfolding the sampling from PrivateMsg into the new(ish) internal channel $\mathsf{Key}(1)$ yields:

- $\mathsf{Key}(0) := \mathsf{samp}(\mu)$
- $\mathsf{Key}(1) := \mathsf{samp}(\mu)$
- $\mathsf{Flip} := \mathsf{samp}(\mathsf{flip})$
- $\mathsf{LeakFlip}^{\mathsf{rec}}_{\mathsf{adv}} := \text{read Flip}$
- $\mathsf{SharedKey} := \text{read } \mathsf{Key}(0)$
- $\mathsf{LeakSharedKey}^{\mathsf{rec}}_{\mathsf{adv}} := \text{read SharedKey}$
- $\mathsf{LeakChoice}^{\mathsf{rec}}_{\mathsf{adv}} := \text{read Choice}$
- $\mathsf{ChoiceEnc} := c \leftarrow \mathsf{Choice};\ f \leftarrow \mathsf{Flip};$
  $\text{if } f \text{ then } \big(\text{if } c \text{ then ret false else ret true}\big) \text{ else } \big(\text{if } c \text{ then ret true else ret false}\big)$
- $\mathsf{LeakChoiceEnc}^{\mathsf{rec}}_{\mathsf{adv}} := \text{read ChoiceEnc}$
- $\mathsf{PrivateMsg} := m_0 \leftarrow \mathsf{In}(0);\ m_1 \leftarrow \mathsf{In}(1);\ c \leftarrow \mathsf{Choice};\ \text{read } \mathsf{Key}(1)$
- $\mathsf{MsgEnc}(0) := m_0 \leftarrow \mathsf{In}(0);\ m_1 \leftarrow \mathsf{In}(1);\ c \leftarrow \mathsf{Choice};\ k_0 \leftarrow \mathsf{Key}(0);\ p \leftarrow \mathsf{PrivateMsg};$
  $\text{if } c \text{ then ret } p \text{ else ret } m_0 \oplus k_0$
- $\mathsf{MsgEnc}(1) := m_0 \leftarrow \mathsf{In}(0);\ m_1 \leftarrow \mathsf{In}(1);\ c \leftarrow \mathsf{Choice};\ k_0 \leftarrow \mathsf{Key}(0);\ p \leftarrow \mathsf{PrivateMsg};$
  $\text{if } c \text{ then ret } m_1 \oplus k_0 \text{ else ret } p$
- $\mathsf{LeakMsgEnc}^{\mathsf{rec}}_{\mathsf{adv}}(0) := \text{read } \mathsf{MsgEnc}(0)$
- $\mathsf{LeakMsgEnc}^{\mathsf{rec}}_{\mathsf{adv}}(1) := \text{read } \mathsf{MsgEnc}(1)$
- $\mathsf{Out} := m_0 \leftarrow \mathsf{In}(0);\ m_1 \leftarrow \mathsf{In}(1);\ c \leftarrow \mathsf{Choice};\ \text{if } c \text{ then ret } m_1 \text{ else ret } m_0$
- $\mathsf{LeakOut}^{\mathsf{rec}}_{\mathsf{adv}} := \text{read Out}$

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

- $\mathsf{Key}(0) := \mathsf{samp}(\mu)$
- $\mathsf{Key}(1) := \mathsf{samp}(\mu)$
- $\mathsf{Flip} := \mathsf{samp}(\mathsf{flip})$
- $\mathsf{LeakFlip}^{\mathsf{rec}}_{\mathsf{adv}} := \text{read Flip}$

- SharedKey := read Key(0)
- $\mathsf{LeakSharedKey}^{\mathsf{rec}}_{\mathsf{adv}}$:= read SharedKey
- $\mathsf{LeakChoice}^{\mathsf{rec}}_{\mathsf{adv}}$:= read Choice
- ChoiceEnc := $c \leftarrow$ Choice; $f \leftarrow$ Flip;
  if f then $\big($if c then ret false else ret true$\big)$ else $\big($if c then ret true else ret false$\big)$
- $\mathsf{LeakChoiceEnc}^{\mathsf{rec}}_{\mathsf{adv}}$:= read ChoiceEnc
- MsgEnc(0) := $m_0 \leftarrow$ In(0); $m_1 \leftarrow$ In(1); $c \leftarrow$ Choice; $k_0 \leftarrow$ Key(0); $k_1 \leftarrow$ Key(1);
  if c then ret k_1 else ret $m_0 \oplus k_0$
- MsgEnc(1) := $m_0 \leftarrow$ In(0); $m_1 \leftarrow$ In(1); $k_0 \leftarrow$ Key(0); $k_1 \leftarrow$ Key(1); $c \leftarrow$ Choice;
  if c then ret $m_1 \oplus k_0$ else ret k_1
- $\mathsf{LeakMsgEnc}^{\mathsf{rec}}_{\mathsf{adv}}(0)$:= read MsgEnc(0)
- $\mathsf{LeakMsgEnc}^{\mathsf{rec}}_{\mathsf{adv}}(1)$:= read MsgEnc(1)
- Out := $m_0 \leftarrow$ In(0); $m_1 \leftarrow$ In(1); $c \leftarrow$ Choice; if c then ret m_1 else ret m_0
- $\mathsf{LeakOut}^{\mathsf{rec}}_{\mathsf{adv}}$:= read Out

The channel Out can now be separated out as coming from the functionality, and the remainder of the protocol is turned into the simulator:

- Key(0) := samp(μ)
- Key(1) := samp(μ)
- Flip := samp(flip)
- $\mathsf{LeakFlip}^{\mathsf{rec}}_{\mathsf{adv}}$:= read Flip
- SharedKey := read Key(0)
- $\mathsf{LeakSharedKey}^{\mathsf{rec}}_{\mathsf{adv}}$:= read Key(0)
- $\mathsf{LeakChoice}^{\mathsf{rec}}_{\mathsf{adv}}$:= read $\mathsf{LeakChoice}^{\mathsf{id}}_{\mathsf{adv}}$
- ChoiceEnc := $c \leftarrow \mathsf{LeakChoice}^{\mathsf{id}}_{\mathsf{adv}}$; $f \leftarrow$ Flip;
  if f then $\big($if c then ret false else ret true$\big)$ else $\big($if c then ret true else ret false$\big)$
- $\mathsf{LeakChoiceEnc}^{\mathsf{rec}}_{\mathsf{adv}}$:= read ChoiceEnc
- MsgEnc(0) := $m \leftarrow \mathsf{LeakOut}^{\mathsf{id}}_{\mathsf{adv}}$; $c \leftarrow \mathsf{LeakChoice}^{\mathsf{id}}_{\mathsf{adv}}$; $k_0 \leftarrow$ Key(0); $k_1 \leftarrow$ Key(1);
  if c then ret k_1 else ret $m \oplus k_0$
- MsgEnc(1) := $m \leftarrow \mathsf{LeakOut}^{\mathsf{id}}_{\mathsf{adv}}$; $c \leftarrow \mathsf{LeakChoice}^{\mathsf{id}}_{\mathsf{adv}}$; $k_0 \leftarrow$ Key(0); $k_1 \leftarrow$ Key(1);
  if c then ret $m \oplus k_0$ else ret k_1
- $\mathsf{LeakMsgEnc}^{\mathsf{rec}}_{\mathsf{adv}}(0)$:= read MsgEnc(0)
- $\mathsf{LeakMsgEnc}^{\mathsf{rec}}_{\mathsf{adv}}(1)$:= read MsgEnc(1)
- $\mathsf{LeakOut}^{\mathsf{rec}}_{\mathsf{adv}}$:= read $\mathsf{LeakOut}^{\mathsf{id}}_{\mathsf{adv}}$

Plugging in the simulator into the ideal functionality and substituting away the internal channels $\mathsf{LeakChoice}^{\mathsf{id}}_{\mathsf{adv}}$ and $\mathsf{LeakOut}^{\mathsf{id}}_{\mathsf{adv}}$ that originally served as a line of communication for the adversary yields the final version of the real protocol, as desired.

2.5 The Simulator

For reference, we record here the simulator:

- $\mathsf{Key}(0) := \mathsf{samp}(\mu)$
- $\mathsf{Key}(1) := \mathsf{samp}(\mu)$
- $\mathsf{Flip} := \mathsf{samp}(\mathsf{flip})$
- $\mathsf{LeakFlip}^{\mathsf{rec}}_{\mathsf{adv}} := \text{read } \mathsf{Flip}$
- $\mathsf{SharedKey} := \text{read } \mathsf{Key}(0)$
- $\mathsf{LeakSharedKey}^{\mathsf{rec}}_{\mathsf{adv}} := \text{read } \mathsf{Key}(0)$
- $\mathsf{LeakChoice}^{\mathsf{rec}}_{\mathsf{adv}} := \text{read } \mathsf{LeakChoice}^{\mathsf{id}}_{\mathsf{adv}}$
- $\mathsf{ChoiceEnc} := c \leftarrow \mathsf{LeakChoice}^{\mathsf{id}}_{\mathsf{adv}};\ f \leftarrow \mathsf{Flip};$
  if f then $\big($if c then ret false else ret true$\big)$ else $\big($if c then ret true else ret false$\big)$
- $\mathsf{LeakChoiceEnc}^{\mathsf{rec}}_{\mathsf{adv}} := \text{read } \mathsf{ChoiceEnc}$
- $\mathsf{MsgEnc}(0) := m \leftarrow \mathsf{LeakOut}^{\mathsf{id}}_{\mathsf{adv}};\ c \leftarrow \mathsf{LeakChoice}^{\mathsf{id}}_{\mathsf{adv}};\ k_0 \leftarrow \mathsf{Key}(0);\ k_1 \leftarrow \mathsf{Key}(1);$
  if c then ret k_1 else ret $m \oplus k_0$
- $\mathsf{MsgEnc}(1) := m \leftarrow \mathsf{LeakOut}^{\mathsf{id}}_{\mathsf{adv}};\ c \leftarrow \mathsf{LeakChoice}^{\mathsf{id}}_{\mathsf{adv}};\ k_0 \leftarrow \mathsf{Key}(0);\ k_1 \leftarrow \mathsf{Key}(1);$
  if c then ret $m \oplus k_0$ else ret k_1
- $\mathsf{LeakMsgEnc}^{\mathsf{rec}}_{\mathsf{adv}}(0) := \text{read } \mathsf{MsgEnc}(0)$
- $\mathsf{LeakMsgEnc}^{\mathsf{rec}}_{\mathsf{adv}}(1) := \text{read } \mathsf{MsgEnc}(1)$
- $\mathsf{LeakOut}^{\mathsf{rec}}_{\mathsf{adv}} := \text{read } \mathsf{LeakOut}^{\mathsf{id}}_{\mathsf{adv}}$

3 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 corrupted. There are no functional or cryptographic assumptions, since we only work with Booleans. We write $x \oplus y$ as a shorthand for the reaction

$$\text{if } x \text{ then if } y \text{ then ret false else ret true else if } y \text{ then ret true else ret false}$$

3.1 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:

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

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

3.2 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:

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

To uniformly cover all cases, we assume channels $\mathsf{AdvCommit}(i)^{\mathsf{adv}}_{\mathsf{party}}$ 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:

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

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:

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

We again assume channels $\mathsf{AdvOpen}(i)^{\mathsf{adv}}_{\mathsf{party}}$ 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:

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

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

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

The output of each corrupted party i diverges:

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

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

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

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

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

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

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

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

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

3.3 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.

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

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

3.4 Real = Ideal + Simulator

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

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

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:

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

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

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

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:

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

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

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:

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

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

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

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

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

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 $\mathsf{SumCommit}(-)$:

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

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

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

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

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:

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

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

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

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

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

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

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:

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

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

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

The above can be simplified to

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

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:

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

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

The channel Flip now only occurs in the channel LastCommit, so we can extract the following subprotocol:

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

Here Flip is an internal channel, $\mathsf{SumCommit}(n+1)$ is an input, and LastCommit is an output. Folding Flip into LastCommit gives the following single-reaction protocol:

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

The uniformity of the distribution flip implies that the above rewrites to

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

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

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

Inserting this subprotocol back into the original protocol yields the following:

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

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

We can now substitute away the internal channel $\mathsf{LastCommit}$:

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

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

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

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

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