

Selected IPDL Case Studies

September 13, 2022

Abstract

We present here the full proofs of select IPDL case studies: Authenticated-To-Secure Channel: CPA Security in Section 1; Oblivious Transfer: 1-Out-Of-2 Pre-Processing in Section 2; and Multi-Party Coin Toss in Section 3.

1 Authenticated-To-Secure Channel: CPA Security

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 $\text{key}, \text{msg}, \text{ctxt}$ of keys, messages, and ciphertexts, respectively; a chosen message $\text{zeros} : 1 \rightarrow \text{msg}$; a distribution $\text{unif}_{\text{key}} : 1 \rightarrow \text{key}$ on keys; an encode algorithm

$$\text{enc} : \text{msg} \times \text{key} \rightarrow \text{ctxt}$$

that takes a message and a key, and returns a distribution on ciphertexts; and a decode algorithm

$$\text{dec} : \text{ctxt} \times \text{key} \rightarrow \text{msg}$$

that takes a ciphertext and a key, and returns a message.

1.1 The Assumptions

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

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

with input In and outputs $\text{Key}, \text{Enc}, \text{Dec}$ rewrites strictly to

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

The *indistinguishability under chosen plaintext attack (IND-CPA)* cryptographic assumption states that if the key is secret, encoding $q \in \mathbb{N}$ arbitrary messages is computationally indistinguishable from encoding the chosen message q times: in the channel context $\{\text{In}(i) : \text{msg}\}_i, \text{Key} : \text{key}, \{\text{Enc}(i) : \text{ctxt}\}_i$ where $i := 1, \dots, q$, the protocol

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

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

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

1.2 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 okay from the adversary, outputs the message:

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

1.3 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:

- $\text{Key} := \text{samp unif}_{\text{key}}$

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

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

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:

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

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

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

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

1.4 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 channels $\text{LeakMsgRcvd}_{\text{adv}}^{\text{id}}(-)$, $\text{OkCtxt}_{\text{net}}^{\text{adv}}(-)$ are inputs to the simulator and channels $\text{LeakCtxt}_{\text{adv}}^{\text{net}}(-)$, $\text{OkMsg}_{\text{id}}^{\text{adv}}(-)$ 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:

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

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

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

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

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

1.5 Real \approx Ideal + Simulator

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

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

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

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

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

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

We first substitute the hidden channels $\text{Recv}(-)$ away:

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

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

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

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

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

The assumption of encryption-decryption correctness applied q times allows us to strictly rewrite the above protocol to the following one:

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

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

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

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

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

Plugging this into the original protocol yields the following:

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

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

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

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

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 coin-flip distribution `flip : 1 → Bool`; a random distribution `unifmsg : 1 → msg` on messages; and a bitwise xor function

$$\oplus : \text{msg} \times \text{msg} \rightarrow \text{msg}$$

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

2.1 The Assumptions

At the expression level, we assume that the operation of bitwise xor with a fixed message is self-inverse: *i.e.*, we have the two axioms

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

At the reaction level, we assume that the coin flip is fair via the following axiom:

- $\cdot ; \cdot \vdash (f \leftarrow \text{samp flip}; \text{ if } f \text{ then ret false else ret true}) = \text{samp flip} : \emptyset \rightarrow \text{Bool}$.

Finally, we assume that the distribution `unifmsg` on messages is invariant under the operation of xor-ing with a fixed message (as is indeed the case when `unifmsg` is uniform):

- $\cdot ; x : \text{msg} \vdash (y \leftarrow \text{samp unif}_{\text{msg}}; \text{ ret } x \oplus y) = \text{samp unif}_{\text{msg}} : \emptyset \rightarrow \text{msg}$, and
- $\cdot ; y : \text{msg} \vdash (x \leftarrow \text{samp unif}_{\text{msg}}; \text{ ret } x \oplus y) = \text{samp unif}_{\text{msg}} : \emptyset \rightarrow \text{msg}$.

2.2 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:

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

Each of the inputs is accompanied by a corresponding leakage to the adversary, signaling that the input has been received but not its value:

- `Out := $m_0 \leftarrow \text{In}(0)$; $m_1 \leftarrow \text{In}(1)$; $c \leftarrow \text{Choice}$; if c then ret m_1 else ret m_0`
- `LeakMsgRcvdadvid(0) := $m_0 \leftarrow \text{In}(0)$; ret ✓`
- `LeakMsgRcvdadvid(1) := $m_1 \leftarrow \text{In}(1)$; ret ✓`
- `LeakMsgRcvdadvid := $c \leftarrow \text{Choice}$; ret ✓`

Additionally, since the receiver is corrupt, the selected message and the receiver's choice itself are also leaked to the adversary:

- `Out := $m_0 \leftarrow \text{In}(0)$; $m_1 \leftarrow \text{In}(1)$; $c \leftarrow \text{Choice}$; if c then ret m_1 else ret m_0`
- `LeakMsgRcvdadvid(0) := $m_0 \leftarrow \text{In}(0)$; ret ✓`
- `LeakMsgRcvdadvid(1) := $m_1 \leftarrow \text{In}(1)$; ret ✓`
- `LeakMsgRcvdadvid := $c \leftarrow \text{Choice}$; ret ✓`
- `LeakChoiceadvid := read Choice`
- `LeakOutadvid := read Out`

2.3 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:

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

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

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

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

- $\text{SharedKey} := k_0 \leftarrow \text{Key}(0); k_1 \leftarrow \text{Key}(1); f \leftarrow \text{Flip}; \text{if } f \text{ then ret } k_1 \text{ else ret } k_0$
- $\text{LeakMsgRcvd}_{\text{adv}}^{\text{ot}}(0) := k_0 \leftarrow \text{Key}(0); \text{ret } \checkmark$
- $\text{LeakMsgRcvd}_{\text{adv}}^{\text{ot}}(1) := k_1 \leftarrow \text{Key}(1); \text{ret } \checkmark$
- $\text{LeakMsgRcvd}_{\text{adv}}^{\text{ot}} := f \leftarrow \text{Flip}; \text{ret } \checkmark$
- $\text{LeakFlip}_{\text{adv}}^{\text{ot}} := \text{read Flip}$
- $\text{LeakSharedKey}_{\text{adv}}^{\text{ot}} := \text{read SharedKey}$

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

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

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

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

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:

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

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:

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

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

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

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:

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

Thus, we have the following code for Alice:

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

The code for Bob has the following form:

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

Finally, we recall the code for the OT functionality:

- $\text{SharedKey} := k_0 \leftarrow \text{Key}(0); k_1 \leftarrow \text{Key}(1); f \leftarrow \text{Flip}; \text{if } f \text{ then ret } k_1 \text{ else ret } k_0$
- $\text{LeakMsgRcvd}_{\text{adv}}^{\text{ot}}(0) := k_0 \leftarrow \text{Key}(0); \text{ret } \checkmark$
- $\text{LeakMsgRcvd}_{\text{adv}}^{\text{ot}}(1) := k_1 \leftarrow \text{Key}(1); \text{ret } \checkmark$
- $\text{LeakMsgRcvd}_{\text{adv}}^{\text{ot}} := f \leftarrow \text{Flip}; \text{ret } \checkmark$
- $\text{LeakFlip}_{\text{adv}}^{\text{ot}} := \text{read Flip}$
- $\text{LeakSharedKey}_{\text{adv}}^{\text{ot}} := \text{read SharedKey}$

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

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:

- $\text{Key}(0) := \text{samp unif}_{\text{msg}}$
- $\text{Key}(1) := \text{samp unif}_{\text{msg}}$
- $\text{Flip} := \text{samp flip}$
- $\text{LeakFlip}_{\text{adv}}^{\text{rec}} := \text{read Flip}$
- $\text{SharedKey} := k_0 \leftarrow \text{Key}(0); k_1 \leftarrow \text{Key}(1); f \leftarrow \text{Flip}; \text{if } f \text{ then ret } k_1 \text{ else ret } k_0$
- $\text{LeakMsgRcvd}_{\text{adv}}^{\text{ot}}(0) := k_0 \leftarrow \text{Key}(0); \text{ret } \checkmark$
- $\text{LeakMsgRcvd}_{\text{adv}}^{\text{ot}}(1) := k_1 \leftarrow \text{Key}(1); \text{ret } \checkmark$
- $\text{LeakMsgRcvd}_{\text{adv}}^{\text{ot}} := f \leftarrow \text{Flip}; \text{ret } \checkmark$
- $\text{LeakFlip}_{\text{adv}}^{\text{ot}} := \text{read Flip}$
- $\text{LeakSharedKey}_{\text{adv}}^{\text{ot}} := \text{read SharedKey}$
- $\text{LeakSharedKey}_{\text{adv}}^{\text{rec}} := \text{read SharedKey}$
- $\text{LeakChoice}_{\text{adv}}^{\text{rec}} := \text{read Choice}$
- $\text{ChoiceEnc} := f \leftarrow \text{Flip}; c \leftarrow \text{Choice};$
 $\text{if } c \text{ then (if } f \text{ then ret false else ret true) else (if } f \text{ then ret true else ret false)}$
- $\text{LeakChoiceEnc}_{\text{adv}}^{\text{rec}} := \text{read ChoiceEnc}$
- $\text{MsgEnc}(0) := m_0 \leftarrow \text{In}(0); m_1 \leftarrow \text{In}(1); k_0 \leftarrow \text{Key}(0); k_1 \leftarrow \text{Key}(1); e \leftarrow \text{ChoiceEnc};$
 $\text{if } e \text{ then ret } m_0 \oplus k_1 \text{ else ret } m_0 \oplus k_0$
- $\text{MsgEnc}(1) := m_0 \leftarrow \text{In}(0); m_1 \leftarrow \text{In}(1); k_0 \leftarrow \text{Key}(0); k_1 \leftarrow \text{Key}(1); e \leftarrow \text{ChoiceEnc};$
 $\text{if } e \text{ then ret } m_1 \oplus k_0 \text{ else ret } m_1 \oplus k_1$
- $\text{LeakMsgEnc}_{\text{adv}}^{\text{rec}}(0) := \text{read MsgEnc}(0)$
- $\text{LeakMsgEnc}_{\text{adv}}^{\text{rec}}(1) := \text{read MsgEnc}(1)$
- $\text{Out} := e_0 \leftarrow \text{MsgEnc}(0); e_1 \leftarrow \text{MsgEnc}(1); s \leftarrow \text{SharedKey}; c \leftarrow \text{Choice}; \text{if } c \text{ then ret } e_1 \oplus s \text{ else ret } e_0 \oplus s$
- $\text{LeakOut}_{\text{adv}}^{\text{rec}} := \text{read Out}$

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

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

Substituting the channel SharedKey into Out yields:

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

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

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

After simplifying we get:

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

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

- Key(0) := samp unif_{msg}
- Key(1) := samp unif_{msg}
- Flip := samp flip
- LeakFlip_{adv}^{rec} := read Flip
- SharedKey := $k_0 \leftarrow \text{Key}(0)$; $k_1 \leftarrow \text{Key}(1)$; $f \leftarrow \text{Flip}$; if f then ret k_1 else ret k_0
- LeakMsgRcvd_{adv}^{ot}(0) := $k_0 \leftarrow \text{Key}(0)$; ret ✓
- LeakMsgRcvd_{adv}^{ot}(1) := $k_1 \leftarrow \text{Key}(1)$; ret ✓
- LeakMsgRcvd_{adv}^{ot} := $f \leftarrow \text{Flip}$; ret ✓
- LeakFlip_{adv}^{ot} := read Flip
- LeakSharedKey_{adv}^{ot} := read SharedKey
- LeakSharedKey_{adv}^{rec} := read SharedKey
- LeakChoice_{adv}^{rec} := read Choice
- ChoiceEnc := $f \leftarrow \text{Flip}$; $c \leftarrow \text{Choice}$;
 if c then (if f then ret false else ret true) else (if f then ret true else ret false)
- LeakChoiceEnc_{adv}^{rec} := read ChoiceEnc
- MsgEnc(0) := $m_0 \leftarrow \text{In}(0)$; $m_1 \leftarrow \text{In}(1)$; $k_0 \leftarrow \text{Key}(0)$; $k_1 \leftarrow \text{Key}(1)$; $f \leftarrow \text{Flip}$; $c \leftarrow \text{Choice}$;
 if c then (if f then ret $m_0 \oplus k_0$ else ret $m_0 \oplus k_1$) else (if f then ret $m_0 \oplus k_1$ else ret $m_0 \oplus k_0$)
- MsgEnc(1) := $m_0 \leftarrow \text{In}(0)$; $m_1 \leftarrow \text{In}(1)$; $k_0 \leftarrow \text{Key}(0)$; $k_1 \leftarrow \text{Key}(1)$; $f \leftarrow \text{Flip}$; $c \leftarrow \text{Choice}$;
 if c then (if f then ret $m_1 \oplus k_1$ else ret $m_1 \oplus k_0$) else (if f then ret $m_1 \oplus k_0$ else ret $m_1 \oplus k_1$)
- LeakMsgEnc_{adv}^{rec}(0) := read MsgEnc(0)
- LeakMsgEnc_{adv}^{rec}(1) := read MsgEnc(1)
- Out := $m_0 \leftarrow \text{In}(0)$; $m_1 \leftarrow \text{In}(1)$; $c \leftarrow \text{Choice}$; if c then ret m_1 else ret m_0
- LeakOut_{adv}^{rec} := 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:

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

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

- $\text{Flip} := \text{samp flip}$
- $\text{LeakFlip}_{\text{adv}}^{\text{rec}} := \text{read Flip}$
- $\text{KeyPair} := k_0 \leftarrow \text{samp unif}_{\text{msg}}; k_1 \leftarrow \text{samp unif}_{\text{msg}}; f \leftarrow \text{Flip}; \text{if } f \text{ then ret } (k_1, k_0) \text{ else ret } (k_0, k_1)$
- $\text{SharedKey} := k \leftarrow \text{KeyPair}; \text{ret } (\text{fst } k)$
- $\text{LeakMsgRcvd}_{\text{adv}}^{\text{ot}}(0) := k_0 \leftarrow \text{Key}(0); \text{ret } \checkmark$
- $\text{LeakMsgRcvd}_{\text{adv}}^{\text{ot}}(1) := k_1 \leftarrow \text{Key}(1); \text{ret } \checkmark$
- $\text{LeakMsgRcvd}_{\text{adv}}^{\text{ot}} := f \leftarrow \text{Flip}; \text{ret } \checkmark$
- $\text{LeakFlip}_{\text{adv}}^{\text{ot}} := \text{read Flip}$

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

Rearranging the order of the samplings inside KeyPair yields the reaction

$$f \leftarrow \text{Flip}; k_0 \leftarrow \text{samp unif}_{\text{msg}}; k_1 \leftarrow \text{samp unif}_{\text{msg}}; \text{if } f \text{ then ret } (k_1, k_0) \text{ else ret } (k_0, k_1)$$

The samplings are interchangeable: the reaction snippet

$$k_0 \leftarrow \text{samp unif}_{\text{msg}}; k_1 \leftarrow \text{samp unif}_{\text{msg}}; \text{if } f \text{ then ret } (k_1, k_0) \text{ else ret } (k_0, k_1)$$

rewrites to

$$\begin{aligned} &\text{if } f \text{ then } k_0 \leftarrow \text{samp unif}_{\text{msg}}; k_1 \leftarrow \text{samp unif}_{\text{msg}}; \text{ret } (k_1, k_0) \\ &\text{else } k_0 \leftarrow \text{samp unif}_{\text{msg}}; k_1 \leftarrow \text{samp unif}_{\text{msg}}; \text{ret } (k_0, k_1) \end{aligned}$$

which in turn rewrites to

$$\begin{aligned} &\text{if } f \text{ then } k_1 \leftarrow \text{samp unif}_{\text{msg}}; k_0 \leftarrow \text{samp unif}_{\text{msg}}; \text{ret } (k_1, k_0) \\ &\text{else } k_0 \leftarrow \text{samp unif}_{\text{msg}}; k_1 \leftarrow \text{samp unif}_{\text{msg}}; \text{ret } (k_0, k_1) \end{aligned}$$

But this amounts to doing the same thing either way, so we may just as well not flip:

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

Unfolding the samplings back thus gives us:

- $\text{Key}(0) := \text{samp unif}_{\text{msg}}$
- $\text{Key}(1) := \text{samp unif}_{\text{msg}}$
- $\text{Flip} := \text{samp flip}$
- $\text{LeakFlip}_{\text{adv}}^{\text{rec}} := \text{read Flip}$
- $\text{KeyPair} := k_0 \leftarrow \text{Key}(0); k_1 \leftarrow \text{Key}(1); \text{ret } (k_0, k_1)$
- $\text{SharedKey} := k \leftarrow \text{KeyPair}; \text{ret } (\text{fst } k)$
- $\text{LeakMsgRcvd}_{\text{adv}}^{\text{ot}}(0) := k_0 \leftarrow \text{Key}(0); \text{ret } \checkmark$

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

The internal channel KeyPair can now be substituted away:

- $\text{Key}(0) := \text{samp unif}_{\text{msg}}$
- $\text{Key}(1) := \text{samp unif}_{\text{msg}}$
- $\text{Flip} := \text{samp flip}$
- $\text{LeakFlip}_{\text{adv}}^{\text{rec}} := \text{read Flip}$
- $\text{SharedKey} := \text{read Key}(0)$
- $\text{LeakMsgRcvd}_{\text{adv}}^{\text{ot}}(0) := k_0 \leftarrow \text{Key}(0); \text{ret } \checkmark$
- $\text{LeakMsgRcvd}_{\text{adv}}^{\text{ot}}(1) := k_1 \leftarrow \text{Key}(1); \text{ret } \checkmark$
- $\text{LeakMsgRcvd}_{\text{adv}}^{\text{ot}} := f \leftarrow \text{Flip}; \text{ret } \checkmark$
- $\text{LeakFlip}_{\text{adv}}^{\text{ot}} := \text{read Flip}$
- $\text{LeakSharedKey}_{\text{adv}}^{\text{ot}} := \text{read SharedKey}$
- $\text{LeakSharedKey}_{\text{adv}}^{\text{rec}} := \text{read SharedKey}$
- $\text{LeakChoice}_{\text{adv}}^{\text{rec}} := \text{read Choice}$
- $\text{ChoiceEnc} := f \leftarrow \text{Flip}; c \leftarrow \text{Choice};$
if c then (if f then ret false else ret true) else (if f then ret true else ret false)
- $\text{LeakChoiceEnc}_{\text{adv}}^{\text{rec}} := \text{read ChoiceEnc}$

- $\text{MsgEnc}(0) := m_0 \leftarrow \text{In}(0); m_1 \leftarrow \text{In}(1); c \leftarrow \text{Choice}; k_0 \leftarrow \text{Key}(0); k_1 \leftarrow \text{Key}(1);$
if c then $\text{ret } m_0 \oplus k_1$ else $\text{ret } m_0 \oplus k_0$
- $\text{MsgEnc}(1) := m_0 \leftarrow \text{In}(0); m_1 \leftarrow \text{In}(1); c \leftarrow \text{Choice}; k_0 \leftarrow \text{Key}(0); k_1 \leftarrow \text{Key}(1);$
if c then $\text{ret } m_1 \oplus k_0$ else $\text{ret } m_1 \oplus k_1$
- $\text{LeakMsgEnc}_{\text{adv}}^{\text{rec}}(0) := \text{read MsgEnc}(0)$
- $\text{LeakMsgEnc}_{\text{adv}}^{\text{rec}}(1) := \text{read MsgEnc}(1)$
- $\text{Out} := m_0 \leftarrow \text{In}(0); m_1 \leftarrow \text{In}(1); c \leftarrow \text{Choice};$ if c then $\text{ret } m_1$ else $\text{ret } m_0$
- $\text{LeakOut}_{\text{adv}}^{\text{rec}} := \text{read Out}$

The second key is now only referenced in the channels $\text{MsgEnc}(0)$ and $\text{MsgEnc}(1)$, 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 :

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

- $\text{LeakOut}_{\text{adv}}^{\text{rec}} := \text{read Out}$

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

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

Rearranging the order of the samplings inside PrivateMsg yields the reaction

$$m_0 \leftarrow \text{In}(0); m_1 \leftarrow \text{In}(1); c \leftarrow \text{Choice}; k_1 \leftarrow \text{samp unif}_{\text{msg}}; \text{if } c \text{ then ret } m_0 \oplus k_1 \text{ else ret } m_1 \oplus k_1$$

The reaction snippet

$$k_1 \leftarrow \text{samp unif}_{\text{msg}}; \text{if } c \text{ then ret } m_0 \oplus k_1 \text{ else ret } m_1 \oplus k_1$$

further rewrites to

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

Our functional assumptions assert that unif_{msg} is invariant under xor-ing with a fixed message, which yields:

$$\text{if } c \text{ then samp unif}_{\text{msg}} \text{ else samp unif}_{\text{msg}}$$

So we may just as well not branch:

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

Unfolding the sampling back gives us:

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

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

- $\text{Key}(0) := \text{samp unif}_{\text{msg}}$
- $\text{Key}(1) := \text{samp unif}_{\text{msg}}$
- $\text{Flip} := \text{samp flip}$
- $\text{LeakFlip}_{\text{adv}}^{\text{rec}} := \text{read Flip}$
- $\text{SharedKey} := \text{read Key}(0)$
- $\text{LeakMsgRcvd}_{\text{adv}}^{\text{ot}}(0) := k_0 \leftarrow \text{Key}(0); \text{ret } \checkmark$
- $\text{LeakMsgRcvd}_{\text{adv}}^{\text{ot}}(1) := k_1 \leftarrow \text{Key}(1); \text{ret } \checkmark$

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

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

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

- $\text{MsgEnc}(1) := m \leftarrow \text{LeakOut}_{\text{adv}}^{\text{id}}; c \leftarrow \text{LeakChoice}_{\text{adv}}^{\text{id}}; k_0 \leftarrow \text{Key}(0); k_1 \leftarrow \text{Key}(1);$
if c then $\text{ret } m \oplus k_0$ else $\text{ret } k_1$
- $\text{LeakMsgEnc}_{\text{adv}}^{\text{rec}}(0) := \text{read } \text{MsgEnc}(0)$
- $\text{LeakMsgEnc}_{\text{adv}}^{\text{rec}}(1) := \text{read } \text{MsgEnc}(1)$
- $\text{LeakOut}_{\text{adv}}^{\text{rec}} := \text{read } \text{LeakOut}_{\text{adv}}^{\text{id}}$

Plugging in the simulator into the ideal functionality and substituting away the internal channels $\text{LeakChoice}_{\text{adv}}^{\text{id}}$ and $\text{LeakOut}_{\text{adv}}^{\text{id}}$ 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:

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

3 Multi-Party Coin Toss

In this section we implement a protocol where $n + 2$ parties labeled $0, \dots, n + 1$ 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 $\text{flip} : 1 \rightarrow \text{Bool}$ and a Boolean sum function

$$\oplus : \text{Bool} \times \text{Bool} \rightarrow \text{Bool}$$

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

3.1 The Assumptions

At the expression level, we assume that the operation of Boolean sum with a fixed bit is self-inverse: *i.e.*, we have the two axioms

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

At the reaction 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):

- $\cdot ; x : \text{Bool} \vdash (y \leftarrow \text{samp flip}; \text{ret } x \oplus y) = \text{samp flip} : \emptyset \rightarrow \text{Bool}$, and
- $\cdot ; y : \text{Bool} \vdash (x \leftarrow \text{samp flip}; \text{ret } x \oplus y) = \text{samp flip} : \emptyset \rightarrow \text{Bool}$.

3.2 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:

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

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

3.3 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:

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

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:

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

To uniformly cover all cases, we assume channels $\text{AdvCommit}(i)_{\text{party}}^{\text{adv}}$ as inputs to the real protocol, for all $0 \leq i \leq n + 1$ 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:

- $\text{Committed}(i) := c_i \leftarrow \text{Commit}(i); \text{ret } \checkmark$

- $\text{LeakCommitted}(i)_{\text{adv}}^{\text{comm}} := \text{read Committed}(i)$

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

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

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

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

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

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

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

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

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

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

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

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

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

The output of each corrupted party i diverges:

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

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

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

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

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

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

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

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

3.4 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.

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

3.5 Real = Ideal + Simulator

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

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

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 $\text{AllCommitted}(-)$ that inductively keep a global track of commitment:

- $\text{Committed}(i) := c_i \leftarrow \text{Commit}(i); \text{ret } \checkmark \text{ for } 0 \leq i \leq n+1$

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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 $\text{SumCommit}(-)$:

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

By assumption, we can cancel out the Boolean sum:

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

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

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

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

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

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

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

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

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

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

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

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

The internal channel LastCommit can now be substituted away:

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

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

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

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