Towards Modular Foundations for Composable Security

Lúcás Críostóir Meier
lucas@cronokirby.com

February 1, 2023

Abstract

We do things with UC security.

1 Introduction

[Mei22]

Definition 1.1 (Adversaries). An adversary is a cool thing.

Theorem 1.1 (Cool Beans). Woah mama

And that's what matters.

■

Lemma 1.2. Woah mama again!

Corollary 1.3. Woah mama again!

Γ^0

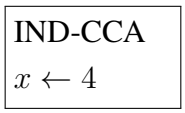
$x \leftarrow 3$
if $x + 2$:
  $y \xleftarrow{\$} \mathbb{F}_q$
$m \Rightarrow \langle \mathcal{P}_i, \mathcal{P}_j \rangle$ $y \leftarrow 4$
$m \Leftarrow \langle \text{OT}, \mathcal{P}_i \rangle$ $x \leftarrow 3$

Foo(x, y):
  Bar(x, y)

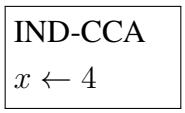
Game 1.1: Some Game

1.1 Relevance of Time Travel

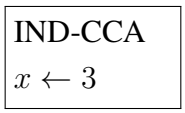
stuff



Protocol 1.2: Some Protocol



Protocol 1.3: Some Protocol



Functionality 1.4: Encryption

2 State-Separable Proofs

3 Systems

3.1 Asynchronous Packages

While the intuition of yield statements is simple, defining them precisely is a bit more tricky.

Definition 3.1 (Yield Statements). We define the semantics of **yield** by compiling functions with such statements to functions without them.

Note that we don't define the semantics for functions which still contain references to oracles. Like before, we can delay the definition of semantics until all of the pseudo-code has been inlined.

A first small change is to make it so that the function accepts one argument, a binary string, and all yield points also accept binary strings as continuation. Like with plain packages, we can implement richer types on top by adding additional checks to the well-formedness of binary strings, aborting otherwise.

The next step is to make it so that all the local variables of the function F are present in the global state. So, if a local variable v is present, then every use of v is replaced with a use of the global variable $F.v$ in the package. This allows the state of the function to be saved across yields.

The next step is transforming all the control flow of a function to use **ifgoto**, rather than structured programming constructs like **while** or **if**. The function is

broken into lines, each of which contains a single statement. Each line is given a number, starting at 0. The execution of a function F involves a special variable $\texttt{pc}$, representing the current line being executing. Excluding **yield** and **return** a single line statement has one of the forms:

$$\langle \texttt{var} \rangle \leftarrow \langle \texttt{expr} \rangle$$
$$\langle \texttt{var} \rangle \xleftarrow{\$} \langle \texttt{dist} \rangle$$

which have well defined semantics already. Additionally, after these statements, we set $\texttt{pc} \leftarrow \texttt{pc} + 1$.

The semantics of **ifgoto** $\langle \texttt{expr} \rangle i$ is:

$$\texttt{pc} \leftarrow \textbf{ if } \langle \texttt{expr} \rangle \textbf{ then } i \textbf{ else } \texttt{pc} + 1$$

This gives us a conditional jump, and by using `true` as the condition, we get a standard unconditional jump.

This allows us to define **if** and **while** statements in the natural way.

Finally, we need to augment functions to handle **yield** and **return** statements. To handle this, each function F also has an associated variable $F.\texttt{pc}$, which stores the program counter for the function. This is different than the local $\texttt{pc}$ which is while the function is execution. $F.\texttt{pc}$ is simply used to remember the program counter after a yield statement.

The function now starts with:

$$\textbf{ifgoto } \texttt{true } F.\text{pc}$$

This has the effect of resuming execution at the saved program counter.

Furthermore, the input variable x to F is replaced with a special variable `input`, which holds the input supplied to the function. At the start of the function body, we add:

$$0 : F.x \leftarrow \texttt{input}$$

to capture the fact that the original input variable needs to get assigned to the input to the function.

The semantics of $F.m \leftarrow$ **yield** v are:

$$\begin{aligned}(i-1) &: F.\text{pc} \leftarrow i + 1 \\ i &: \textbf{return } (\texttt{yield}, v) \\ (i+1) &: F.m \leftarrow \texttt{input}\end{aligned}$$

The semantics of **return** v become:

$$\begin{aligned}&F.\text{pc} \leftarrow 0 \\ &\textbf{return } (\texttt{return}, v)\end{aligned}$$

The main difference is that we annotate the return value to be different than yield statements, but otherwise the semantics are the same.

□

Note that while calling a function which can yield will notify the caller as to whether or not the return value was *yielded* or *returned*, syntactically the caller often ignores this, simply doing $x \leftarrow F(\ldots)$, meaning that they simply use return value x, discarding the tag.

Syntax 3.2 (Empty Yields). In many cases, no value is yielded, or returned back, which we can write as:

$$\textbf{yield}$$

which is shorthand for:

$$\bullet \leftarrow \textbf{yield } \bullet$$

i.e. just yielding a dummy value and ignoring the result.

□

Unless otherwise specified, we only consider empty yields from now on.

We define these semantics via the **await** statement.

Syntax 3.3 (Await Statements). We define the semantics of $v \leftarrow$ **await** $F(\ldots)$ in a straightforward way:

$$\begin{aligned}
&(\text{tag}, v) \leftarrow (\texttt{yield}, \bot)\\
&\textbf{while } \text{tag} = \texttt{yield}:\\
&\quad \textbf{if } v \neq \bot:\\
&\quad\quad \textbf{yield}\\
&\quad (\text{tag}, v) \leftarrow F(\ldots)
\end{aligned}$$

In other words, we keep calling the function until it actually returns its final value, but we do yield to our caller whenever our function yield, but we do yield to our caller whenever our function yields.

□

Sometimes we want to await several values at once, returning the first one which completes. To that end, we define the **select** statement.

Syntax 3.4 (Select Statements). Select statements generalize await statements in that they allow waiting for multiple events concurrently.

More formally, we define:

select :
$v_1 \leftarrow$ **await** $F_1(\ldots)$:
$\langle \text{body}_1 \rangle$
$\vdots$
$v_n \leftarrow$ **await** $F_n(\ldots)$:
$\langle \text{body}_n \rangle$

As follows:

$(\text{tag}_i, v_i) \leftarrow (\texttt{yield}, \bot)$
$i \leftarrow 0$
while $\nexists i.\ \text{tag}_i \neq \texttt{yield}$:
if $i \geq n$:
$i \leftarrow 0$
yield
$i \leftarrow i + 1$
$(\text{tag}_i, v_i) \leftarrow F_i(\ldots)$
$\langle \text{body}_i \rangle$

Note that the order in which we call the functions is completely deterministic, and fair. It's also important that we yield, like with await statements, but we only do so after having pinged each of our underlying functions at least once. This is so that if one of the function is immediately ready, we never yield.

□

Definition 3.5 (Asynchronous Packages). An *asynchronous* package P is a package which uses the additional syntax from Definition 3.1 and Syntax 3.3, 3.4.

□

Note that our syntax sugar definitions means that whenever one of the constructs such as yield and what not are used, they are immediately replaced with their underlying semantics. Thus, an asynchronous package *literally* is a package which does not use any of those syntactical constructs.

In/Out are well defined elaborate. Naturally, the definitions of $\circ$ and $\otimes$ for packages also generalize directly to asynchronous packages.

3.2 Channels and System Composition

Definition 3.6 (Systems). A *system* is a package which uses channels.

We denote by InChan(S) the set of channels the system receives on, and OutChan(S) the set of channels the system sends on, and define

$$\text{Chan}(S) := \text{OutChan}(S) \cup \text{InChan}(S)$$

Additionally we require that $\text{OutChan}(S) \cap \text{InChan}(S) = \emptyset$

□

We also define shorthands $\text{Chan}(A, B, \ldots) = \text{Chan}(A) \cup \text{Chan}(B) \cup \ldots$. expand.

Definition 3.7. We can compile systems to not use channels. We denote by $\text{NoChan}(S)$ the package corresponding to a system S, with the use of channels replaced with function calls.

Channels define three new syntactic constructions, for sending and receiving along a channel, along with testing how many messages are in a channel. We replace these with function calls as follows:

Sending, with $m \Rightarrow P$ becomes:

$$\text{Channels.Send}_P(m)$$

Testing, with $n \leftarrow$ **test** P becomes

$$n \leftarrow \text{Channels.Test}_P()$$

Receiving, with $m \Leftarrow P$ becomes:

$$m \leftarrow \textbf{await } \text{Channels.Recv}_P()$$

Receiving is an asynchronous function, because the channel might not have any available messages for us.

These function calls are parameterized by the channel, meaning that that we have a separate function for each channel.

□

One consequence of this definition with separate functions for each channel is that $\text{Channels}(S) \otimes \text{Channels}(R) = \text{Channels}(S \cup R)$.

Armed with the syntax sugar for channels, and the Channels game, we can convert a system S into a package via:

$$\text{SysPack}(S) := \text{NoChan}(S) \circ (\text{Channels}(\text{Chan}(S)) \otimes 1(\text{In}(S)))$$

This package will have the same input and output functions as the system S, but with the usage of channels replaced with actual semantics.

This allows us to lift our standard equality relations on packages onto *systems*.

```
Channels({A_1, ..., A_n})

q[A_i] ← FifoQueue.New()

Send_{A_i}(m):
  q[A_i].Push(m)

Test_{A_i}():
  return q[A_i].Length()

Recv_{A_i}():
  while q[A_i].IsEmpty()
    yield
  q[A_i].Next()
```

Game 3.1: Channels

Definition 3.8. Given systems A, B, we say that they have the same *shape* if

- $\text{In}(A) = \text{In}(B)$,
- $\text{Out}(A) = \text{Out}(B)$,
- $\text{InChan}(A) = \text{InChan}(B)$,
- $\text{OutChan}(A) = \text{OutChan}(B)$.

□

Definition 3.9 (Literal System Equality). Given systems A, B with the same shape, we say that they are *literally* equal, written $A \equiv B$ if

$$\text{NoChan}(A) = \text{NoChan}(B)$$

□

Definition 3.10 (System Tensoring). Given two systems, A and B, with $\text{Out}(A) \cap \text{Out}(B) = \emptyset$, we can define their tensor product $A * B$, which is any system $A * B$ satisfying:

- $\text{NoChan}(A * B) = \text{NoChan}(A) \otimes \text{NoChan}(B)$,
- $\text{InChan}(A * B) = \text{InChan}(A) \cup \text{InChan}(B)$,
- $\text{OutChan}(A * B) = \text{OutChan}(A) \cup \text{OutChan}(B)$,
- $\text{In}(A * B) = \text{In}(A) \cup \text{In}(B)$.

□

Note that combining the definition above with the definition of SysPack means that:

$$\text{SysPack}(A * B) = \begin{pmatrix} \text{NoChan}(A) \\ \otimes \\ \text{NoChan}(B) \end{pmatrix} \circ \begin{pmatrix} \text{Channels}(\text{Chan}(A) \cup \text{Chan}(B)) \\ \otimes \\ 1(\text{In}(A) \cup \text{In}(B)) \end{pmatrix}$$

This implies the following lemma.

Lemma 3.1. System tensoring is associative, i.e. $A * (B * C) \equiv (A * B) * C$.
Proof: This follows directly from the associativity of $\otimes$ for packages and $\cup$.

■

Lemma 3.2. System tensoring is commutative, i.e. $A * B \equiv B * A$

Proof: This follows from the commutativity of $\otimes$ and $\cup$.

■

Definition 3.11 (Overlapping Systems). Two systems A and B overlap if $\text{Chan}(A) \cap \text{Chan}(B) \neq \emptyset$.

In the case of non-overlapping systems, we write $A \otimes B$ instead of $A * B$, insisting on the fact that they don't communicate.

Definition 3.12 (System Composition). Given two systems, A and B, we can define their (horizontal) composition $A \circ B$ as any system, provided a few constraints hold:

- A and B do not overlap ($\text{Chan}(A) \cap \text{Chan}(B) = \emptyset$)
- $\text{In}(A) \subseteq \text{Out}(B)$

With these in place, we define the composition as any system $A \circ B$ such that:

- $\text{NoChan}(A \circ B) = \text{NoChan}(A) \circ \begin{pmatrix} \text{NoChan}(B) \\ \otimes \\ 1(\text{Channels}(\text{Chan}(A))) \end{pmatrix}$,
- $\text{InChan}(A \circ B) = \text{InChan}(A) \cup \text{InChan}(B)$,
- $\text{OutChan}(A \circ B) = \text{OutChan}(A) \cup \text{OutChan}(B)$,
- $\text{In}(A \circ B) = \text{In}(B)$.

□

Lemma 3.3. System composition is associative, i.e. $A \circ (B \circ C) \equiv (A \circ B) \circ C$.

Proof: This follows from the associativity of $\circ$ for *packages*.

■

Lemma 3.4 (Interchange Lemma). Given systems A, B, C, D such that $A \circ B$ and $C \circ D$ are well defined, $A * C$ and $B * D$ are well defined, and neither A nor C overlap with B or D, i.e. the following relation holds:

$$\begin{pmatrix} A \\ * \\ C \end{pmatrix} \circ \begin{pmatrix} B \\ * \\ D \end{pmatrix} \equiv \begin{matrix} (A \circ B) \\ * \\ (C \circ D) \end{matrix}$$

Proof: InChan, OutChan, and In are equal for both of these systems, by associativity of $\cup$. We now look at NoChan. Starting with the right hand side, we get:

$$\text{NoChan} \begin{pmatrix} (A \circ B) \\ * \\ (C \circ D) \end{pmatrix} = \begin{pmatrix} \text{NoChan}(A \circ B) \\ \otimes \\ \text{NoChan}(C \circ D) \end{pmatrix} = \begin{pmatrix} \text{NoChan}(A) \circ \begin{pmatrix} \text{NoChan}(B) \\ \otimes \\ 1(\text{Channels}(\text{Chan}(A))) \end{pmatrix} \\ \otimes \\ \text{NoChan}(C) \circ \begin{pmatrix} \text{NoChan}(D) \\ \otimes \\ 1(\text{Channels}(\text{Chan}(C))) \end{pmatrix} \end{pmatrix}$$

Next, apply the interchange lemma for packages, to get:

$$\begin{pmatrix} \text{NoChan}(A) \\ \otimes \\ \text{NoChan}(C) \end{pmatrix} \circ \begin{pmatrix} \text{NoChan}(B) \\ \otimes \\ 1(\text{Channels}(\text{Chan}(A))) \\ \otimes \\ \text{NoChan}(D) \\ \otimes \\ 1(\text{Channels}(\text{Chan}(C))) \end{pmatrix}$$

Then, observe that:

$$\text{Channels}(S_1 \cup S_2) = \text{Channels}(S_1) \otimes \text{Channels}(S_2)$$

We can use this, along with the commutativity of $\otimes$ to get:

$$\begin{pmatrix} \text{NoChan}(A) \\ \otimes \\ \text{NoChan}(C) \end{pmatrix} \circ \begin{pmatrix} \text{NoChan}(B) \\ \otimes \\ \text{NoChan}(D) \\ \otimes \\ 1(\text{Channels}(\text{Chan}(A * C))) \end{pmatrix}$$

Which is just:

$$\text{NoChan} \left(\begin{pmatrix} A \\ * \\ C \end{pmatrix} \circ \begin{pmatrix} B \\ * \\ D \end{pmatrix} \right)$$

■

Definition 3.13 (System Games). Analogously to games, we define a *system game* as a system S with $\text{In}(S) = \emptyset$.

□

Definition 3.14 (System Equality). We say that two systems A, B with the same shape are equal, written $A = B$, if:

$$\text{SysPack}(A) = \text{SysPack}(B)$$

□

Definition 3.15 (System Indistinguishability). We say that two systems A, B with the same shape are indistinguishable up to ϵ, written $A \overset{\epsilon}{\approx} B$, if:

$$\text{SysPack}(A) \overset{\epsilon}{\approx} \text{SysPack}(B)$$

□

Lemma 3.5 (Transitivity of System Equality). Given systems A, B, C, we have:

1. $A \equiv B, B \equiv C \implies A \equiv C$,
2. $A = B, B = C \implies A = C$,
3. $A \overset{\epsilon_1}{\approx} B, B \overset{\epsilon_2}{\approx} C \implies A \overset{\epsilon_1+\epsilon_2}{\approx} C$.

provided these expressions are well-defined.

Proof: This follows immediately from the fact that equality and Indistinguishability for *packages* satisfy similar relations, and the notions for systems are defined in terms of the package $\text{SysPack}(\ldots)$.

■

Lemma 3.6 (Composition Compatability). Given systems A, B, B', we have:

1. $B = B' \implies A \circ B = A \circ B'$,
2. $B \overset{\epsilon}{\approx} B' \implies A \circ B \overset{\epsilon}{\approx} A \circ B'$.

provided these expressions are well-defined.

Proof: We prove that

$$\text{SysPack}(A \circ B) = \text{SysPack}(A) \circ \text{SysPack}(B)$$

which then clearly implies this lemma by application of the similar properties for packages.

We start with:

$$\text{SysPack}(A \circ B) = \text{NoChan}(A) \circ \begin{pmatrix} \text{NoChan}(B) \\ \otimes \\ 1(\text{Channels}(\text{Chan}(A))) \end{pmatrix} \circ \begin{pmatrix} \text{Channels}(\text{Chan}(A) \cup \text{Chan}(B)) \\ \otimes \\ 1(\text{In}(B)) \end{pmatrix}$$

We then use the fact that $\text{Channels}(S \cup R) = \text{Channels}(S) \otimes \text{Channels}(R)$, and the interchange lemma, to get:

$$\text{NoChan}(A) \circ \begin{pmatrix} \text{NoChan}(B) \\ \otimes \\ \text{Channels}(\text{Chan}(A)) \end{pmatrix} \circ \begin{pmatrix} \text{Channels}(\text{Chan}(B)) \\ \otimes \\ 1(\text{In}(B)) \end{pmatrix}$$

Apply interchange once more, to get:

$$\text{NoChan}(A) \circ \begin{pmatrix} 1(\text{In}(A)) \\ \otimes \\ \text{Channels}(\text{Chan}(A)) \end{pmatrix} \circ \text{NoChan}(B) \circ \begin{pmatrix} \text{Channels}(\text{Chan}(B)) \\ \otimes \\ 1(\text{In}(B)) \end{pmatrix}$$

Which is none other than:

$$\text{SysPack}(A) \circ \text{SysPack}(B)$$

concluding our proof.

■

Lemma 3.7 (Strict Tensoring Compatability). Given systems A, B, B', we have:

1. $B = B' \implies A \otimes B = A \otimes B'$,
2. $B \overset{\epsilon}{\approx} B' \implies A \otimes B \overset{\epsilon}{\approx} A \otimes B'$.

provided these expressions are well-defined.

Proof: Similar to Lemma 3.6, we start by proving:

$$\text{SysPack}(A \otimes B) = \text{SysPack}(A) \otimes \text{SysPack}(B)$$

which then entails our theorem through similar properties for packages.

Our starting point is:

$$\text{SysPack}(A \otimes B) = \begin{pmatrix} \text{NoChan}(A) \\ \otimes \\ \text{NoChan}(B) \end{pmatrix} \circ \begin{pmatrix} \text{Channels}(\text{Chan}(A) \cup \text{Chan}(B)) \\ \otimes \\ 1(\text{In}(A), \text{In}(B)) \end{pmatrix}$$

We can write this as:

$$\begin{pmatrix}\text{NoChan}(A)\\ \otimes \\ \text{NoChan}(B)\end{pmatrix} \circ \begin{pmatrix}\text{Channels}(\text{Chan}(A))\\ \otimes \\ 1(\text{In}(A))\\ \otimes \\ \text{Channels}(\text{Chan}(B))\\ \otimes \\ 1(\text{In}(B))\end{pmatrix}$$

Crucially, we can use the fact that A and B do not overlap, in order to apply the interchange lemma, giving us:

$$\begin{matrix}\text{NoChan}(A) \circ \begin{pmatrix}\text{Channels}(\text{Chan}(A))\\ \otimes \\ 1(\text{In}(A))\end{pmatrix}\\ \otimes \\ \text{NoChan}(B) \circ \begin{pmatrix}\text{Channels}(\text{Chan}(B))\\ \otimes \\ 1(\text{In}(B))\end{pmatrix}\end{matrix}$$

Which is none other than:

$$\text{SysPack}(A) \otimes \text{SysPack}(B)$$

concluding our proof.

■

4 Protocols and Composition

Definition 4.1 (Protocols). A *protocol* $\mathcal{P}$ consists of:

- Systems $P_1, \ldots, P_n$, called *players*
- An asynchronous package F, called the *ideal functionality*
- A set Leakage $\subseteq \text{Out}(F)$, called the leakage

Furthermore, we also impose requirements on the channels and functions these elements use.

First, we require that the player systems are jointly closed, with no extra channels that aren't connected to other players:

$$\bigcup_{i\in[n]} \text{OutChan}(P_i) = \bigcup_{i\in[n]} \text{InChan}(P_i)$$

Second, we require that the functions the systems depend on are disjoint, outside of the ideal functionality:

$$\forall i, j \in [n]. \quad \text{In}(P_i) \cap \text{In}(P_j) \subseteq \text{Out}(F)$$

Third, we require that the functions the systems export on are disjoint:

$$\forall i, j \in [n]. \quad \text{Out}(P_i) \cap \text{Out}(P_j) = \emptyset$$

We can also define a few convenient notations related to the interface of a base protocol.

Let $\text{Out}_i(\mathcal{P}) := \text{Out}(P_i)$, and let $\text{In}_i(\mathcal{P}) := \text{In}(P_i)/\text{Out}(F)$. We then define $\text{Out}(\mathcal{P}) := \bigcup_{i\in[n]} \text{Out}_i(\mathcal{P})$ and $\text{In}(\mathcal{P}) := \bigcup_{i\in[n]} \text{In}_i(\mathcal{P})$. Let $\text{IdealIn}_i(\mathcal{P}) := \text{In}(P_i) \cap \text{Out}(F)$.

Finally, we define

$$\text{IdealIn}(\mathcal{P}) := \text{In}(F)$$

□

Definition 4.2 (Closed Protocol). We say that a protocol $\mathcal{P}$ is *closed* if $\text{In}(\mathcal{P}) = \emptyset$ and $\text{IdealIn}(\mathcal{P}) = \emptyset$.

□

Definition 4.3 (Literal Equality). Given two protocols $\mathcal{P}$ and $\mathcal{Q}$, we say that they are *literally equal*, written as $\mathcal{P} \equiv \mathcal{Q}$ when:

- $\mathcal{P}.n = \mathcal{Q}.n$
- There exists a permuation $\pi : [n] \leftrightarrow [n]$ such that $\forall i \in [n].\ \mathcal{P}.P_i \equiv \mathcal{Q}.P_{\pi(i)}$
- $\mathcal{P}.F = \mathcal{Q}.F$
- $\mathcal{P}.\text{Leakage} = \mathcal{Q}.\text{Leakage}$

□

Definition 4.4 (Vertical Composition). Given an protocol $\mathcal{P}$ and a package G, satisfying $\text{IdealIn}(\mathcal{P}) \subseteq \text{Out}(G)$, we can define the protocol $\mathcal{P} \circ G$.

$\mathcal{P}\circ G$ has the same players and leakage as $\mathcal{P}$, but its ideal functionality F becomes $F \circ G$.

□

Claim 4.1 (Vertical Composition is Associative). For any protocol $\mathcal{P}$, and packages G, H, such that their composition is well defined, we have

$$\mathcal{P} \circ (G \circ H) = (\mathcal{P} \circ G) \circ H$$

Proof: This follows from the definition of vertical composition and the associativity of $\circ$ for packages. ■

Definition 4.5 (Horizontal Composition). Given two protocols $\mathcal{P}, \mathcal{Q}$, we can define the protocol $\mathcal{P} \lhd \mathcal{Q}$, provided a few requirements hold.

First, we need: $\text{In}(\mathcal{P}) \subseteq \text{Out}(\mathcal{Q})$. We also require that the functions exposed by a player in $\mathcal{Q}$ are used by *exactly* one player in $\mathcal{P}$. We express this as:

$$\forall i \in [\mathcal{Q}.n].\ \exists! j \in [\mathcal{P}.n]. \quad \text{In}_j \cap \text{Out}_i \neq \emptyset$$

Second, we require that the players share no channels between the two protocols. In other words $\text{Chan}(\mathcal{P}.P_i) \cap \text{Chan}(\mathcal{Q}.P_j) = \emptyset$, for all P_i, P_j.

Third, we require that the ideal functionalities of one protocol aren't used in the other.

$$\text{Out}(\mathcal{P}.F) \cap \text{In}(\mathcal{Q}) = \emptyset$$
$$\text{Out}(\mathcal{Q}.F) \cap \text{In}(\mathcal{P}) = \emptyset$$

Finally, we require that the ideal functionalities do not overlap, in the sense that $\text{Out}(\mathcal{P}.F) \cap \text{Out}(\mathcal{Q}.F) = \emptyset$

Our first condition has an interesting consequence: every player $\mathcal{Q}.P_j$ has its functions used by exactly one player $\mathcal{P}.P_i$. In that case, we say that $\mathcal{P}.P_i$ *uses* $\mathcal{Q}.P_j$.

With this in hand, we can define $\mathcal{P} \lhd \mathcal{Q}$.

The players will consist of:

$$\mathcal{P}.P_i \circ \left(\begin{array}{c} \Large * \\ {\scriptstyle \mathcal{Q}.P_j \text{ used by } \mathcal{P}.P_i} \\ \otimes \\ 1(\text{IdealIn}_i) \end{array} \right) \begin{array}{l} \\ \mathcal{Q}.P_j \\ \\ \\ \end{array}$$

And, because of our assumption, each player in $\mathcal{Q}$ appears somewhere in this equation.

The ideal functionality is $\mathcal{P}.F \otimes \mathcal{Q}.F$, and the leakage is $\mathcal{P}.\text{Leakage} \cup \mathcal{Q}.\text{Leakage}$.

We can also easily show that this definition is well defined, satisfying the required properties of an protocol. Because of the definition of the players, we see that:

$$\bigcup_{i \in [(\mathcal{P} \lhd \mathcal{Q}).n]} \text{OutChan}((\mathcal{P} \lhd \mathcal{Q}).P_i) = \left(\bigcup_{i \in [\mathcal{P}.n]} \text{OutChan}(\mathcal{P}.P_i) \right) \cup \left(\bigcup_{i \in [\mathcal{Q}.n]} \text{OutChan}(\mathcal{Q}.P_i) \right)$$

since $\text{OutChan}(A \circ B) = \text{OutChan}(A \otimes B) = \text{OutChan}(A, B)$. A similar reasoning applies to InChan, allowing us to conclude that:

$$\bigcup_{i \in [(\mathcal{P} \lhd \mathcal{Q}).n]} \text{OutChan}((\mathcal{P} \lhd \mathcal{Q}).P_i) = \bigcup_{i \in [(\mathcal{P} \lhd \mathcal{Q}).n]} \text{InChan}((\mathcal{P} \lhd \mathcal{Q}).P_i)$$

as required.

By definition, the dependencies In of each player in $\mathcal{P} \lhd \mathcal{Q}$ are the union of several players in $\mathcal{Q}$, and the ideal dependencies of players in $\mathcal{P}$, both of these are required to be disjoint, so disjointness property continues to hold.

Finally, since each player is of the form $\mathcal{P}.P_i \circ \ldots$, the condition on Out_i is also satisfied in $\mathcal{P} \lhd \mathcal{Q}$, since $\mathcal{P}$ does.

□

Lemma 4.2. Horizontal composition is associative, i.e. $\mathcal{P} \lhd (\mathcal{Q} \lhd \mathcal{R}) \equiv (\mathcal{P} \lhd \mathcal{Q}) \lhd \mathcal{R}$ for all protocols $\mathcal{P}, \mathcal{Q}, \mathcal{R}$ where this expression is well defined.

Proof: For the ideal functionalities, it's clear that by the associativity of $\otimes$ for systems, the resulting functionality is the same in both cases.

The trickier part of the proof is showing that the resulting players are identical.

It's convenient to define a relation for the players in $\mathcal{R}$ that get used in $\mathcal{P}$ via the players in $\mathcal{Q}$. To that end, we say that $\mathcal{P}.P_i$ *uses* $\mathcal{R}.P_j$ if there exists $\mathcal{Q}.P_k$ such that $\mathcal{P}.P_i$ uses $\mathcal{Q}.P_k$, and $\mathcal{Q}.P_k$ uses $\mathcal{R}.P_j$.

The players of $\mathcal{P} \lhd (\mathcal{Q} \lhd \mathcal{R})$ are of the form:

$$\mathcal{P}.P_i \circ \left(\begin{array}{c} \displaystyle\mathop{*}_{\mathcal{Q}.P_j \text{ used by } \mathcal{P}.P_i} \mathcal{Q}.P_j \circ \left(\begin{array}{c} \displaystyle\mathop{*}_{\mathcal{R}.P_k \text{ used by } \mathcal{Q}.P_j} \mathcal{R}.P_k \\ \otimes \\ 1(\mathcal{Q}.\text{IdealIn}_j) \end{array} \right) \\ \otimes \\ 1(\mathcal{P}.\text{IdealIn}_i) \end{array} \right)$$

While those in $(\mathcal{P} \lhd \mathcal{Q})\mathcal{R}$ are of the form:

$$\left(\mathcal{P}.P_i \circ \left(\begin{array}{c} \displaystyle\mathop{*}_{\mathcal{Q}.P_j \text{ used by } \mathcal{P}.P_i} \mathcal{Q}.P_j \\ \otimes \\ 1(\mathcal{P}.\text{IdealIn}_i) \end{array} \right) \right) \circ \left(\begin{array}{c} \displaystyle\mathop{*}_{\mathcal{R}.P_k \text{ used by } \mathcal{P}.P_i} \mathcal{R}.P_k \\ \otimes \\ 1(\mathcal{Q}.\text{IdealIn}_j) \end{array} \right)$$

Now, we can apply the associativity of $\circ$ for systems, and also group the $\mathcal{R}.P_k$ players based on which $\mathcal{Q}.P_j$ uses them:

$$\mathcal{P}.P_i \circ \left(\begin{array}{c} \displaystyle\mathop{*}_{\mathcal{Q}.P_j \text{ used by } \mathcal{P}.P_i} \mathcal{Q}.P_j \\ \otimes \\ 1(\mathcal{P}.\text{IdealIn}_i) \end{array} \right) \circ \left(\mathop{*}_{\mathcal{Q}.P_j} \left(\begin{array}{c} \displaystyle\mathop{*}_{\mathcal{R}.P_k \text{ used by } \mathcal{Q}.P_j} \mathcal{R}.P_k \\ \otimes \\ 1(\mathcal{Q}.\text{IdealIn}_j) \end{array} \right) \right)$$

Now, the conditions are satisfied for applying the interchange lemma (Lemma 3.4), giving us:

$$\mathcal{P}.P_i \circ \left(\begin{array}{c} \underset{\mathcal{Q}.P_j \text{ used by } \mathcal{P}.P_i}{*} \mathcal{Q}.P_j \circ \left(\begin{array}{c} \underset{\mathcal{R}.P_k \text{ used by } \mathcal{Q}.P_j}{*} \mathcal{R}.P_k \\ \otimes \\ 1(\mathcal{Q}.\text{IdealIn}_j) \end{array} \right) \\ \otimes \\ 1(\mathcal{P}.\text{IdealIn}_i) \end{array} \right)$$

Which is non other than the players in $\mathcal{P} \lhd (\mathcal{Q} \lhd \mathcal{R})$.

■

Definition 4.6 (Concurrent Composition). Given two protocols $\mathcal{P}, \mathcal{Q}$, we can define their concurrent composition—or tensor product— $\mathcal{P} \otimes \mathcal{Q}$, provided a few requirements hold. We require that:

1. $\text{In}(\mathcal{P}) \cap \text{In}(\mathcal{Q}) = \emptyset$.
2. $\text{Out}(\mathcal{P}) \cap \text{Out}(\mathcal{Q}) = \emptyset$.
3. $\text{Out}(\mathcal{P}.F) \cap \text{Out}(\mathcal{Q}.F) = \emptyset$ *or* $\mathcal{P}.F = \mathcal{Q}.F$.
4. $\text{Leakage}(\mathcal{P}) \cap \text{In}(\mathcal{Q}) = \emptyset = \text{Leakage}(\mathcal{Q}) \cap \text{In}(\mathcal{P})$

The players of $\mathcal{P} \otimes \mathcal{Q}$ consist of all the players in $\mathcal{P}$ and $\mathcal{Q}$. The ideal functionality is $\mathcal{P}.F \otimes \mathcal{Q}.F$, unless $\mathcal{P}.F = \mathcal{Q}.F$, in which case the ideal functionality is simply $\mathcal{P}.F$. In either case, the leakage is $\mathcal{P}.\text{Leakage} \cup \mathcal{Q}.\text{Leakage}$. This use of $\otimes$ is well defined by assumption.

The resulting protocol is also clearly well defined.

The jointly closed property holds because we've simply taken the union of both player sets.

Since $\text{In}(\mathcal{P}) \cap \text{In}(\mathcal{Q}) = \emptyset$, it also holds that for every P_i, P_j in $\mathcal{P} \otimes \mathcal{Q}$, we have $\text{In}(P_i) \cap \text{In}(P_j) = \emptyset$, since each player comes from either $\mathcal{P}$ or $\mathcal{Q}$.

Finally, $\text{Out}(\mathcal{P}) \cap \text{Out}(\mathcal{Q}) = \emptyset$, we have that $\text{Out}(P_i) \cap \text{Out}(P_j) = \emptyset$, by the same reasoning.

□

The reason why we allow for $F = G$ is so that you can have like the same 1

Lemma 4.3. Concurrent composition is associative and commutative. I.e. $\mathcal{P} \otimes (\mathcal{Q} \otimes \mathcal{R}) \equiv (\mathcal{P} \otimes \mathcal{Q}) \otimes \mathcal{R}$, and $\mathcal{P} \otimes \mathcal{Q} \equiv \mathcal{Q} \otimes \mathcal{P}$ for all protocols $\mathcal{P}, \mathcal{Q}, \mathcal{R}$ where these expressions are well defined.

Proof:

By the definition of $\equiv$, all that matter is the *set* of players, and not their order. Because $\cup$ is associative, and so is $\otimes$ for systems, we conclude that concurrent composition is associative as well, since the resulting set of players and ideal functionality are the same in both cases.

Similarly, since $\cup$ and $\otimes$ (for systems) are commutative, we conclude that concurrenty composition is commutative.

■

4.1 Corruption and Simulation

Definition 4.7 ("Honest" Corruption). Given a system P, we define the "honest" corruption of P

$$\text{Corrupt}_H(P) := P$$

This is clearly equality preserving, by tautology.

□

Definition 4.8 (Semi-Honest Corruption). Given a system P, we can define the semi-honest corruption $\text{Corrupt}_{\text{SH}}(P)$.

This is a transformation of of P, providing access to its "view". More formally, $\text{Corrupt}_{\text{SH}}(P)$ is a system which works the same as P, but with an additional public variable log, which contains several sub logs:

1. $\text{log}.A_i$ for each sending channel A_i,
2. $\text{log}.B_i$ for each receiving channel B_i,
3. $\text{log}.F$ for each input function F.
4. $\text{log}.G$ for each output function G.

Each of these sub logs is initialized with $\text{log}.\bullet \leftarrow$ FifoQueue.New(). Additionally, $\text{Corrupt}_{\text{SH}}(P)$ modifies P by pushing events to these logs at different points in time. These events are:

- $(\texttt{call}, (x_1, \ldots, x_n))$ to $\text{log}.F$ when a function call $F(x_1, \ldots, x_n)$ happens.
- $(\texttt{ret}, y)$ to $\text{log}.F$ when the function F returns a value y.
- $(\texttt{input}, (x_1, \ldots, x_n))$ to $\text{log}.G$ when the function G is called with $(x_i, \ldots)$ as input.
- m to $\text{log}.A$ when a value m is sent on channel A.
- m to $\text{log}.B$ when a value m is received on channel B.

This transformation is also equality respecting. First, note that if $P \equiv P'$ as systems, then then $\text{NoChan}(P) = \text{NoChan}(P')$, and so their logs will be the same.

□

Definition 4.9 (Malicious Corruption). Given a system P with:

$$\begin{aligned}\text{In}(P) &= \{F_1, \ldots, F_n\}\\ \text{OutChan}(P) &= \{A_1, \ldots, A_m\}\\ \text{InChan}(P) &= \{B_1, \ldots, B_l\}\end{aligned}$$

we define the malicious corruption $\text{Corrupt}_M(P)$ as the following game:

$\text{Corrupt}_M(P)$

$\underline{\text{Call}_{F_i}((x_1, \ldots, x_n)):}$
return $F_i(x_1, \ldots, x_n)$

$\underline{\text{Send}_{A_i}(m):}$
$m \Rightarrow A_i$

$\underline{\text{Test}_{B_i}():}$
return test B_i

$\underline{\text{Recv}_{B_i}():}$
return $m \Leftarrow B_i$

In other words, malicious corruption provides access to the functions and channels used by P, but no more than that.

This is also equality preserving, since $\text{Corrupt}_M(P)$ depends only on the channels used by P and the functions called by P, all of which are the same for any $P' \equiv P$.

□

Lemma 4.4 (Simulating Corruptions). We can simulate corruptions using strong forms of corruption. In particular, there exists systems S_{SH} and S_{H} such that for all systems P, we have:

$$\begin{aligned}\text{Corrupt}_{\text{SH}}(P) &= S_{\text{SH}} \circ \text{Corrupt}_M(P)\\ \text{Corrupt}_{\text{H}}(P) &= S_{\text{H}} \circ \text{Corrupt}_{\text{SH}}(P)\end{aligned}$$

Proof: For the simulation of honest corruption, we can simply ignore the additional log variable, and set $S_{\text{H}} := 1(\text{Out}(P))$.

For semi-honest corruption, S_{SH} is formed by first transforming $\text{Corrupt}_{\text{SH}}(P)$, replacing:

- every function call with $\text{Call}_{F_i}(\ldots)$,
- every sending of a message m on A with $\text{Send}_A(m)$,
- every length test of B with $\text{Test}_B()$,
- every reception of a message on B with $\text{Recv}_B()$.

The result is clearly a perfect emulation of semi-honest corruption using malicious corruption.

■

Sometimes, it's useful to be able to talk about corruptions in general, in which case we write $\text{Corrupt}_\kappa(P)$, for $\kappa \in \{\text{H}, \text{SH}, \text{M}\}$.

Definition 4.10 (Corruption Models). Given a protocol $\mathcal{P}$ with players $P_1, \ldots, P_n$, a *corruption model* C is a function $C : [\mathcal{P}.n] \to \{\text{H}, \text{SH}, \text{M}\}$. This provides a corruption C_i associated with each player P_i. We can then define $\text{Corrupt}_C(P_i) := \text{Corrupt}_{C_i}(P_i)$.

Corruption models have a natural partial order associated with them. We have:

$$\text{H} < \text{SH} < \text{M}$$

and then we say that $C \geq C'$ if $\forall i \in [n]. \quad C_i \geq C'_i$.

A *class of corruptions* $\mathcal{C}$ is simply a set of corruption models.

□

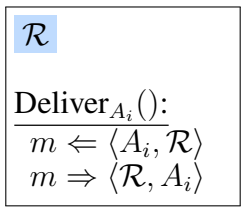
Some common classes are:

- The class of malicious corruptions, where all but one player is malicious.
- The class of malicious corruptions, where all but one player is semi-honest.

Definition 4.11 (Instantiation). Given a protocol $\mathcal{P}$ with $\text{In}(\mathcal{P}) = \emptyset$, and a corruption model C, we can define an *instantiation* $\text{Inst}_C(\mathcal{P})$, which is a system defining the semantics of the protocol.

First, we need to define a transformation of systems to use a *router* $\mathcal{R}$, which will be a special system allowing an adversary to control the order of delivery of messages.

Let $\{A_1, \ldots, A_n\} = \text{Chan}(P_1, \ldots, P_n)$. We then define $\mathcal{R}$ as the syten:



Next, we define a transformation Routed(S) of a system, which makes communication pass via the router:

- Whenever S sends m via A, Routed(S) sends m via $\langle A, \mathcal{R} \rangle$.
- Whenever S receives m via B, Routed(S) recieves m via $\langle \mathcal{R}, B \rangle$.

With this in hand, we define:

$$\text{Inst}_C(\mathcal{P}) := \left(\begin{array}{c} \circledast_{i \in [n]} \text{Routed}(\text{Corrupt}_C(P_i)) \\ * \\ \mathcal{R} \\ \otimes \\ 1(\text{Leakage}) \end{array} \right) \circ F$$

□

Lemma 4.5 (Properties of Routed). For any systems A, B, we have:

$$\text{Routed}(A \circ B) = \text{Routed}(A) \circ \text{Routed}(B)$$
$$\text{Routed}(A * B) = \text{Routed}(A) * \text{Routed}(B)$$
$$\text{Routed}(A \otimes B) = \text{Routed}(A) \otimes \text{Routed}(B)$$

(provided these expressions are well defined)

Proof: The Routed transformation simply renames each sending and receiving channel in a system. In all the cases above, even $A * B$, all of the channels present in A and B are present in the composition, and so all of these equations hold.

■

Definition 4.12 (Compatible Corruptions). Given protocols $\mathcal{P}$, $\mathcal{Q}$, and a corruption model C for $\mathcal{Q}$, we can define a notion of a *compatible* corruption model C' for $\mathcal{P} \otimes \mathcal{Q}$ or $\mathcal{P} \circ \mathcal{Q}$, provided these expressions are well defined.

A corruption model C' for $\mathcal{P} \otimes \mathcal{Q}$. is compatible with C when every corruption of a player in $\mathcal{Q}$ is $\geq$ that of the corresponding corruption in C.

We say that a corruption model C' for $\mathcal{P} \circ \mathcal{Q}$ is compatible with a corruption model C for $\mathcal{Q}$ if for every $\mathcal{Q}.P_j$ used by $\mathcal{P}.P_i$, the corruption level of $\mathcal{Q}.P_j$ in $\mathcal{C}'$ is $\geq$ the corruption level of $\mathcal{P}.P_i$ in $\mathcal{C}$.

Furthermore, we say that C' is *strictly* compatible with C if the above property holds with $=$, and not just $\geq$.

This extends to corruption *classes* as well. A corruption class $\mathcal{C}'$ is (strictly) compatible with a class $\mathcal{C}$, if every $C' \in \mathcal{C}'$ is (strictly) compatible with some $C \in \mathcal{C}$.

□

Theorem 4.6 (Concurrent Breakdown). Given protocols $\mathcal{P}$, $\mathcal{Q}$, and a corruption model C for $\mathcal{Q}$, then for any corruption model C' for $\mathcal{P} \otimes \mathcal{Q}$ compatible with C, we have:

$$\text{Inst}_{C'}(\mathcal{P} \otimes \mathcal{Q}) = \text{Inst}_{C'}(\mathcal{P}) \otimes \text{Inst}_{C}(\mathcal{Q})$$

Proof: If we unroll $\text{Inst}_{C'}(\mathcal{P} \otimes \mathcal{Q})$, we get:

$$\begin{pmatrix} \mathcal{R} \\ * \\ \left(*_{i \in [\mathcal{P}.n]} \text{Routed}(\text{Corrupt}_{C'}(\mathcal{P}.P_i))\right) \\ * \\ \left(*_{i \in [\mathcal{Q}.n]} \text{Routed}(\text{Corrupt}_{C'}(\mathcal{Q}.P_i))\right) \\ \otimes \\ 1(\mathcal{P}.\text{Leakage}, \mathcal{Q}.\text{Leakage}) \end{pmatrix} \circ \begin{pmatrix} \mathcal{P}.F \\ \otimes \\ \mathcal{Q}.F \end{pmatrix}$$

We can apply a few observations here:

1. Since $\mathcal{C}'$ is compatible with $\mathcal{C}$, then $\mathcal{Q}.P_i$ follows a corruption from $\mathcal{C}$.
2. $\mathcal{R}$ can be written as $\mathcal{R}_{\mathcal{P}} \otimes \mathcal{R}_{\mathcal{Q}}$, with one system using channels in $\mathcal{P}$, and the other using channels in $\mathcal{Q}$.
3. Since protocols are closed, we can use $\otimes$ between the players in $\mathcal{P}$ and $\mathcal{Q}$, since they never send messages to each other.

This results in the following:

$$\begin{pmatrix} \mathcal{R}_{\mathcal{P}} * \left(*_{i \in [\mathcal{P}.n]} \text{Routed}(\text{Corrupt}_{C'}(\mathcal{P}.P_i))\right) \otimes 1(\mathcal{P}.\text{Leakage}) \\ \otimes \\ \mathcal{R}_{\mathcal{Q}} * \left(*_{i \in [\mathcal{Q}.n]} \text{Routed}(\text{Corrupt}_{C}(\mathcal{Q}.P_i))\right) \otimes 1(\mathcal{Q}.\text{Leakage}) \end{pmatrix} \circ \begin{pmatrix} \mathcal{P}.F \\ \otimes \\ \mathcal{Q}.F \end{pmatrix}$$

From here, we apply Lemma 3.4 (interchange), to get:

$$\begin{matrix} \text{Inst}_{C'}(\mathcal{P}) \\ \otimes \\ \text{Inst}_{C}(\mathcal{Q}) \end{matrix}$$

■

Theorem 4.7 (Horizontal Breakdown). Given protocols $\mathcal{P}$, $\mathcal{Q}$, and a corruption model C for $\mathcal{Q}$, then for any compatible corruption model C' for $\mathcal{P} \lhd \mathcal{Q}$, there exists systems $S_1, \ldots, S_{\mathcal{Q}.n}$ and a set $L_{\mathcal{Q}}$ such that:

$$\text{Inst}_{C'}(\mathcal{P} \lhd \mathcal{Q}) = 1(O) \circ \begin{pmatrix} \circledast_{i \in [\mathcal{P}.n]} \text{Routed}(\text{Corrupt}'_{C'}(\mathcal{P}.P_i)) \\ * \\ \mathcal{R}_{\mathcal{P}} \\ \otimes \\ 1(\text{Leakage}, L_{\mathcal{Q}}) \end{pmatrix} \circ \begin{pmatrix} \mathcal{P}.F \\ \otimes \\ 1(\text{Out}(\mathcal{R}_q)) \\ \otimes \\ 1(\mathcal{Q}.\text{Leakage}) \\ \otimes \\ \bigotimes_{i \in [\mathcal{Q}.n]} S_i \end{pmatrix} \circ \begin{pmatrix} \text{Inst}_C(\mathcal{Q}) \\ \otimes \\ 1(\text{In}(\mathcal{P}.F)) \end{pmatrix}$$

where $O := \text{Out}(\text{Inst}_{C'}(\mathcal{P} \lhd \mathcal{Q}))$, $\mathcal{R}_{\mathcal{P}} \circ \mathcal{R}_{\mathcal{Q}} = \mathcal{R}$ are a decomposition of the router $\mathcal{R}$ for $\mathcal{P} \lhd \mathcal{Q}$, and $\text{Corrupt}'_{C'}(\ldots)$ is the same as $\text{Corrupt}_{C'}$, except that malicious corruption contains no Call_{F_i} functions, for $F_i \notin \text{Out}(\mathcal{P}.F)$

Furthermore, if the models are *strictly* compatible, then $S_j = 1(\text{Out}(\text{Routed}(\text{Corrupt}_C(\mathcal{Q}.P_i))))$.

Proof: We start by unrolling $\text{Inst}_{C'}(\mathcal{P} \lhd \mathcal{Q})$, to get:

$$\text{Inst}_C(\mathcal{P} \lhd \mathcal{Q}) = \begin{pmatrix} \circledast_{i \in [\mathcal{P}.n]} \text{Routed} \left(\text{Corrupt}_{C'} \left(\mathcal{P}.P_i \circ \begin{pmatrix} \circledast_{\mathcal{Q}.P_j \text{ used by } \mathcal{P}.P_i} \mathcal{Q}.P_j \\ \otimes \\ 1(\text{IdealIn}_i) \end{pmatrix} \right) \right) \\ * \\ \mathcal{R} \\ \otimes \\ 1(\text{Leakage}) \end{pmatrix} \circ \begin{pmatrix} \mathcal{P}.F \\ \otimes \\ \mathcal{Q}.F \end{pmatrix}$$

Our strategy will be to progressively build up an equivalent system to this one, starting with Corrupt_C, then Routed, etc.

First, some observations about $\text{Corrupt}_\kappa(P \circ (1(I) \otimes Q_1 * \cdots * Q_m))$, where $I \cap \text{In}(Q_1, \ldots) = \emptyset$.

In the case of malicious corruption, we have:

$$\text{Corrupt}_M(P \circ (1(I) \otimes Q_1 * \cdots)) = 1(O) \circ \begin{pmatrix} \text{Corrupt}'_M(P) \\ \otimes \\ 1(\text{Out}(\text{Corrupt}_M(Q_1)), \ldots) \end{pmatrix} \circ \begin{pmatrix} 1(I) \\ \otimes \\ \text{Corrupt}_M(Q_1) \\ * \\ \cdots \end{pmatrix}$$

for $O = \text{Out}(\text{Corrupt}_M(P \circ (Q_1 * \cdots)))$. This holds by definition, since corruption $P \circ (Q_1 * \cdots)$ precisely allows sending messages on behalf of P or any Q_i, as well as calling the input functions to the Q_i systems. Since we can't call the functions that P uses, we use $\text{Corrupt}'_M$, which modifies malicious corruption to only contain Send_{A_i}, Test_{B_i}, Recv_{B_i}, and Call_{F_i} for $F_i \in$

I. In particular the Call$_\bullet$ functions are omitted for the functions provided by $Q_1, \ldots, Q_m$. We can write this expression more concisely, using $1(L^M)$ for $L^M = \text{Out}(\text{Corrupt}_M(Q_1)) \cup \cdots$.

Next, we look at semi-honest corruption. Because the logs are divided into independent sub logs, we can write:

$$\text{Corrupt}_{\text{SH}}(P \circ (1(I) \otimes Q_1 * \cdots)) = 1(O) \circ \begin{pmatrix} \text{Corrupt}_{\text{SH}}(P) \\ \otimes \\ 1(\{Q_1.\text{log}, \ldots\}) \end{pmatrix} \circ \begin{pmatrix} 1(I) \\ \otimes \\ \text{Corrupt}_{\text{SH}}(Q_1) \\ * \\ \cdots \end{pmatrix}$$

where $O = \text{Out}(\text{Corrupt}_{\text{SH}}(P \circ (Q_1 * \cdots)))$

And for honest corruption, we have

$$\text{Corrupt}_{\text{H}}(P \circ (1(I) \otimes Q_1 * \cdots)) = P \circ (1(I) \otimes Q_1 * \cdots)$$

Now, the compatibility condition of C' relative to C does not guarantee that if $\mathcal{P}.P_i$ uses $\mathcal{Q}.P_j$, then $\mathcal{Q}.P_j$ has the same level of corruption: it only guarantees a level of corruption at least as strong. By Lemma 4.10, we can simulate a weaker form of corruption using a stronger form, via some simulator system S, depending on the levels of corruption.

Using these simulators, we get, slightly different results based on the level of corruption.

When $C'_i = \text{M}$:

$$\text{Corrupt}_{C'}((\mathcal{P} \triangleleft \mathcal{Q}).P_i) = 1(O_i) \circ \begin{pmatrix} \text{Corrupt}'_{C'}(\mathcal{P}.P_i) \\ \otimes \\ 1(L_i) \end{pmatrix} \circ \begin{pmatrix} \ast_{\mathcal{Q}.P_j \text{ used by } \mathcal{P}.P_i} \text{Corrupt}_C(\mathcal{Q}.P_j) \\ \otimes \\ 1(\text{IdealIn}_i) \end{pmatrix}$$

with $O_i = \text{Out}(\text{Corrupt}_{C'}(\mathcal{P} \triangleleft \mathcal{Q}).P_i)$, $L_i = \bigcup_{\mathcal{Q}.P_j \text{ used by } \mathcal{P}.P_i} \text{Out}(\text{Corrupt}_M(\mathcal{Q}.P_j))$. No simulation is needed, since the compatibility of C' with C guarantees that all of the players used by $\mathcal{P}.P_i$ are maliciously corrupted.

When $C'_i = \text{SH}$:

$$\text{Corrupt}_{C'}((\mathcal{P} \triangleleft \mathcal{Q}).P_i) = 1(O_i) \circ \begin{pmatrix} \text{Corrupt}_{\text{C}'}(P) \\ \otimes \\ 1(L_i) \end{pmatrix} \circ \begin{pmatrix} \ast_{\mathcal{Q}.P_j \text{ used by } \mathcal{P}.P_i} S_j \circ \text{Corrupt}_C(\mathcal{Q}.P_j) \\ \otimes \\ 1(\text{IdealIn}_i) \end{pmatrix}$$

with $O_i = \text{Out}(\text{Corrupt}_{C'}(\mathcal{P} \triangleleft \mathcal{Q}).P_i)$, $L_i = \{\mathcal{Q}.P_j.\text{log} \mid \mathcal{Q}.P_j \text{ used by } \mathcal{P}.P_i\}$, and S_j depending on the level of corruption for $\mathcal{Q}.P_j$ in C:

- $S_j = S_{\text{SH}}$ if $C_j = \text{M}$
- $S_j = 1$ if $C_j = \text{SH}$

When $C'_i = \text{H}$:

$$\text{Corrupt}_{C'}((\mathcal{P} \triangleleft \mathcal{Q}).P_i) = \text{Corrupt}_{\text{C'}}(P) \circ \begin{pmatrix} \ast_{\mathcal{Q}.P_j \text{ used by } \mathcal{P}.P_i} S_j \circ \text{Corrupt}_C(\mathcal{Q}.P_j) \\ \otimes \\ 1(\text{IdealIn}_i) \end{pmatrix}$$

with S_j depending on the level of corruption for $\mathcal{Q}.P_j$ in C:

- $S_j = S_{\text{H}} \circ S_{\text{SH}}$ if $C_j = \text{M}$
- $S_j = S_{\text{H}}$ if $C_j = \text{SH}$
- $S_j = 1$ if $C_j = \text{H}$

We can unify these three cases, writing:

$$\text{Corrupt}'_{C'}((\mathcal{P} \triangleleft \mathcal{Q}).P_i) = 1(O_i) \circ \begin{pmatrix} \text{Corrupt}_{\text{C'}}(P) \\ \otimes \\ 1(L_i) \end{pmatrix} \circ \begin{pmatrix} \ast_{\mathcal{Q}.P_j \text{ used by } \mathcal{P}.P_i} S_j \circ \text{Corrupt}_C(\mathcal{Q}.P_j) \\ \otimes \\ 1(\text{IdealIn}_i) \end{pmatrix}$$

with O_i and L_i depending on the corruption level of $\mathcal{P}.P_i$, and S_j depending on the corruption levels of both $\mathcal{P}.P_i$ and $\mathcal{Q}.P_j$.

By the properties of Routed (Lemma 4.5), we have:

$$\text{Routed}(\text{Corrupt}'_{C'}((\mathcal{P} \triangleleft \mathcal{Q}).P_i)) =$$

$$1(O_i) \circ \begin{pmatrix} \text{Routed}(\text{Corrupt}'_{\text{C'}}(P)) \\ \otimes \\ 1(L_i) \end{pmatrix} \circ \begin{pmatrix} \ast_{\mathcal{Q}.P_j \text{ used by } \mathcal{P}.P_i} S_j \circ \text{Routed}(\text{Corrupt}_C(\mathcal{Q}.P_j)) \\ \otimes \\ 1(\text{IdealIn}_i) \end{pmatrix}$$

Next, we need to add the router $\mathcal{R}$. We note that since $\mathcal{P}$ and $\mathcal{Q}$ have separate channels, we can write $\mathcal{R} = \mathcal{R}_{\mathcal{P}} \circ \mathcal{R}_{\mathcal{Q}}$, where the latter contains only the channels in $\mathcal{Q}$, and the former contains the channels in $\mathcal{P}$, and provides access to those in $\mathcal{Q}$ via its function dependencies. Combing this with the interchange lemma, we get:

$$\mathcal{R} \ast \ast_{i \in [\mathcal{P}.n]} \text{Routed}(\text{Corrupt}'_{C'}((\mathcal{P} \triangleleft \mathcal{Q}).P_i)) \ast \mathcal{R} =$$

$$1(\text{Out}(\mathcal{R}), O_1, \ldots, O_{\mathcal{P}.n}) \circ \begin{pmatrix} \text{Routed}(\text{Corrupt}_{\text{C'}}(P)) \\ \ast \\ \mathcal{R}_{\mathcal{P}} \\ \otimes \\ 1(L_1, \ldots, L_{\mathcal{P}.n}) \end{pmatrix} \circ \begin{pmatrix} \ast_{j \in [\mathcal{Q}.n]} S_j \circ \text{Routed}(\text{Corrupt}_C(\mathcal{Q}.P_j)) \\ \ast \\ \mathcal{R}_{\mathcal{Q}} \\ \otimes \\ 1(\text{Out}(F)) \end{pmatrix}$$

All that remains is to add the ideal functionalities, giving us, after application of the interchange lemma:

$$\text{Inst}_{C'}(\mathcal{P} \lhd \mathcal{Q}) =$$
$$1(O) \circ \begin{pmatrix} \text{Routed}(\text{Corrupt}'_{C'}(P)) \\ * \\ \mathcal{R}_{\mathcal{P}} \\ \otimes \\ 1(\text{Leakage}, L_{\mathcal{Q}}) \end{pmatrix} \circ \begin{pmatrix} *_{j \in [\mathcal{Q}.n]}\, S_j \circ \text{Routed}(\text{Corrupt}_C(\mathcal{Q}.P_j)) \\ * \\ \mathcal{R}_{\mathcal{Q}} \\ \otimes \\ 1(\text{Leakage}, \text{Out}(F)) \end{pmatrix} \circ \begin{pmatrix} \mathcal{P}.F \\ \otimes \\ \mathcal{Q}.F \end{pmatrix}$$

with $O := \text{Out}(\text{Inst}_{C'}(\mathcal{P} \lhd \mathcal{Q}))$, and $L_{\mathcal{Q}} := \bigcup_{i \in [\mathcal{P}.n]} L_i$.

Now, because $\mathcal{Q}$ does not use any of the functions in $\mathcal{P}.F$, and because each simulator S_j does not use any channels, we can rewrite this as:

$$1(O) \circ \begin{pmatrix} \text{Routed}(\text{Corrupt}'_{C'}(P)) \\ * \\ \mathcal{R}_{\mathcal{P}} \\ \otimes \\ 1(\text{Leakage}, L_{\mathcal{Q}}) \end{pmatrix} \circ \begin{pmatrix} \mathcal{P}.F \\ \otimes \\ 1(\text{Out}(\mathcal{R}_{\mathcal{Q}})) \\ \otimes \\ 1(\mathcal{Q}.\text{Leakage}) \\ \otimes \\ \bigotimes_{j \in [\mathcal{Q}.n]} S_j \end{pmatrix} \circ \left(\begin{pmatrix} *_{j \in [\mathcal{Q}.n]}\, \text{Routed}(\text{Corrupt}_C(\mathcal{Q}.P_j)) \\ * \\ \mathcal{R}_{\mathcal{Q}} \\ \otimes \\ 1(\mathcal{Q}.\text{Leakage}) \\ \otimes \\ 1(\text{In}(\mathcal{P}.F)) \end{pmatrix} \circ \mathcal{Q}.F \right)$$

We can then notice that the right hand side of this equation is simply $\text{Inst}_C(\mathcal{Q})$, concluding our proof.

■

4.2 Equality and Simulation

Definition 4.13 (Shape). We say that two protocols $\mathcal{P}, \mathcal{Q}$ have the same *shape* if there exists a protocol $\mathcal{Q}' \equiv \mathcal{Q}$ such that:

- $\mathcal{P}.n = \mathcal{Q}'.n$,
- $\forall i \in [n]. \quad \text{In}(\mathcal{P}.P_i) = \text{In}(\mathcal{Q}'.Q_i)$,
- $\forall i \in [n]. \quad \text{Out}(\mathcal{P}.P_i) = \text{Out}(\mathcal{Q}'.Q_i)$,
- $\text{Leakage}(\mathcal{P}) = \text{Leakage}(\mathcal{Q}')$,
- $\text{IdealIn}(\mathcal{P}) = \text{IdealIn}(\mathcal{Q}')$.

□

Definition 4.14 (Semantic Equality). We say that two closed protocols $\mathcal{P}$ and $\mathcal{Q}$, with the same shape, are equal under a class of corruptions $\mathcal{C}$, written as $\mathcal{P} =_{\mathcal{C}} \mathcal{Q}$, when we have:

$$\forall C \in \mathcal{C}. \quad \text{Inst}_C(\mathcal{P}) = \text{Inst}_C(\mathcal{Q}')$$

as systems, with $\mathcal{Q}' \equiv \mathcal{Q}$ as per Definition 4.13.

□

Definition 4.15 (Indistinguishability). We say that two closed protocols $\mathcal{P}$ and $\mathcal{Q}$, with the same shape, are *indistinguishable* up to ϵ under a class of corruptions $\mathcal{C}$, written as $\mathcal{P} \overset{\epsilon}{\approx}_{\mathcal{C}} \mathcal{Q}$, when we have:

$$\forall C \in \mathcal{C}. \quad \text{Inst}_C(\mathcal{P}) \overset{\epsilon}{\approx} \text{Inst}_C(\mathcal{Q}')$$

as systems, with $\mathcal{Q}' \equiv \mathcal{Q}$ as per Definition 4.13.

□

Definition 4.16 (Simulated Instantiation). A simulator S for a closed protocol $\mathcal{P}$ under a corruption model C is a system satisfying:

- $\text{InChan}(S), \text{OutChan}(S) = \emptyset$,
- $\text{In}(S) = \text{Leakage} \cup \left(\bigcup_{C_i=\text{M}} \text{Out}(\text{Corrupt}_{\text{M}}(P_i))\right) \cup \left(\bigcup_{C_i=\text{SH}} P_i.\text{log}\right)$,
- $\text{Out}(S) = \text{In}(S)$,

Given such a simulator, we can define the simulated instantiation of $\mathcal{P}$ under C with S as:

$$\text{SimInst}_{S,C}(\mathcal{P}) := \begin{pmatrix} S \\ \otimes \\ 1(\text{Out}(\text{Inst}_C(\mathcal{P}))/\text{Out}(S)) \end{pmatrix} \circ \text{Inst}_C(\mathcal{P})$$

□

Definition 4.17 (Simulatability). Given closed protocols $\mathcal{P}, \mathcal{Q}$ with the same shape, we say that $\mathcal{P}$ is *simulatable* up to ϵ by $\mathcal{Q}$ under a class of corruptions $\mathcal{C}$, written as $\mathcal{P} \overset{\epsilon}{\rightsquigarrow}_{\mathcal{C}} \mathcal{Q}$, when:

$$\forall C \in \mathcal{C}. \exists S. \quad \text{Inst}_C(\mathcal{P}) \overset{\epsilon}{\approx} \text{SimInst}_{S,C}(\mathcal{Q}')$$

as systems, with $\mathcal{Q}' \equiv \mathcal{Q}$ as per Definition 4.13.

□

Theorem 4.8 (Equality Hierarchy). For any corruption class $\mathcal{C}$, we have:

1. $\mathcal{P} \equiv \mathcal{Q} \implies \mathcal{P} =_{\mathcal{C}} \mathcal{Q}$.
2. $\mathcal{P} =_{\mathcal{C}} \mathcal{Q} \implies \mathcal{P} \overset{0}{\approx}_{\mathcal{C}} \mathcal{Q}$.
3. $\mathcal{P} \overset{\epsilon}{\approx}_{\mathcal{C}} \mathcal{Q} \implies \mathcal{P} \overset{\epsilon}{\rightsquigarrow}_{\mathcal{C}} \mathcal{Q}$.

Proof:

1. For any $C \in \mathcal{C}$, Corrupt_C and Routed are equality respecting, so we have:

$$\forall i \in [n]. \quad \text{Routed}(\text{Corrupt}_C(\mathcal{P}.P_i)) = \text{Routed}(\text{Corrupt}_C(\mathcal{Q}.P_i))$$

Furthermore, the equality of players between $\mathcal{P}$ and $\mathcal{Q}$ makes $\mathcal{P}.\mathcal{R} = \mathcal{Q}.\mathcal{R}$.

And then, the fact that $\mathcal{P}.F = \mathcal{Q}.F$ forces Leakage to be the same as well.

Finally, since $\circ, *, \otimes$ are respect $\equiv$, we can clearly see that $\text{Inst}_C(\mathcal{P}) = \text{Inst}_C(\mathcal{Q})$, since all the sub-components are literally equal.

2. For any systems A, B, we have $A = B \implies A \overset{0}{\approx} B$. Applying this to $\text{Inst}_C(\mathcal{P})$ and $\text{Inst}_C(\mathcal{Q})$ gives us our result.

3. It suffices to define a simulator S such that $\text{SimInst}_{S,C}(\mathcal{Q}) = \text{Inst}_C(\mathcal{Q})$, which will then show our result. We can simply take $S = 1(\ldots)$ for the right set.

■

Theorem 4.9 (Transitivity of Equality). For any closed protocols $\mathcal{L}, \mathcal{P}, \mathcal{Q}$ with the same shape, and any class of corruptions $\mathcal{C}$, we have:

1. $\mathcal{L} =_{\mathcal{C}} \mathcal{P}, \mathcal{P} =_{\mathcal{C}} \mathcal{Q} \implies \mathcal{L} =_{\mathcal{C}} \mathcal{Q}$,
2. $\mathcal{L} \overset{\epsilon_1}{\approx}_{\mathcal{C}} \mathcal{P}, \mathcal{P} \overset{\epsilon_2}{\approx}_{\mathcal{C}} \mathcal{Q} \implies \mathcal{L} \overset{\epsilon_1+\epsilon_2}{\approx}_{\mathcal{C}} \mathcal{Q}$,
3. $\mathcal{L} \overset{\epsilon_1}{\rightsquigarrow}_{\mathcal{C}} \mathcal{P}, \mathcal{P} \overset{\epsilon_2}{\rightsquigarrow}_{\mathcal{C}} \mathcal{Q} \implies \mathcal{L} \overset{\epsilon_1+\epsilon_2}{\rightsquigarrow}_{\mathcal{C}} \mathcal{Q}$.

Proof: The first two parts follow directly from Lemma 3.5 (transitivity for system equality). Indeed, we just look at $\text{Inst}_C(\mathcal{L})$, $\text{Inst}_C(\mathcal{P})$, and $\text{Inst}_C(\mathcal{Q})$ as systems, for any corruption model C.

For part 3, by assumption we have, for any $C \in \mathcal{C}$:

- $\text{Inst}_C(\mathcal{L}) \overset{\epsilon_1}{\approx} \begin{pmatrix} S_1 \\ \otimes \\ 1(O) \end{pmatrix} \text{Inst}_C(\mathcal{P})$,
- $\text{Inst}_C(\mathcal{P}) \overset{\epsilon_1}{\approx} \begin{pmatrix} S_2 \\ \otimes \\ 1(O) \end{pmatrix} \text{Inst}_C(\mathcal{Q})$.

This means that:

$$\text{Inst}_C(\mathcal{L}) \overset{\epsilon_1+\epsilon_2}{\approx} \begin{pmatrix} S_1 \\ \otimes \\ 1(O) \end{pmatrix} \circ \begin{pmatrix} S_2 \\ \otimes \\ 1(O) \end{pmatrix} \circ \text{Inst}_C(\mathcal{Q})$$

applying the properties we have for systems.

Then, we can apply interchange to write this as:

$$\begin{pmatrix} S_1 \circ S_2 \\ \otimes \\ 1(O) \end{pmatrix} \circ \text{Inst}_C(\mathcal{Q})$$

which concludes our proof, since $S_1 \circ S_2$ will be a valid simulator.

■

Theorem 4.10 (Malicious Completeness). Let $\mathcal{P}$ and $\mathcal{Q}$ closed protocols with the same shape. Given any class of corruptions $\mathcal{C}$, let $\mathcal{C}'$ be a related class, containing models in $\mathcal{C}$ with some malicious corruptions replaced with semi-honest corruptions. We then have:

1. $\mathcal{P} =_C \mathcal{Q} \implies \mathcal{P} =_{C'} \mathcal{Q}$,
2. $\mathcal{P} \overset{\epsilon}{\approx}_C \mathcal{Q} \implies \mathcal{P} \overset{\epsilon}{\approx}_{C'} \mathcal{Q}$,

Proof: Lemma (simulating corruptions) is the crux of our proof. It implies that there existts a system S_{SH} such that

$$\text{Corrupt}_{\text{SH}}(P) = S_{\text{SH}} \circ \text{Corrupt}_M(P)$$

As a consequence, for any $C' \in \mathcal{C}'$ and the $C \in \mathcal{C}$ it's related to, there exists a *simulator* S_{SH} such that:

$$\text{Inst}_{C'}(\mathcal{P}) = \begin{pmatrix} S_{\text{SH}} \\ \otimes \\ 1(O) \end{pmatrix} \circ \text{Inst}_C(\mathcal{P})$$

which simulates all of the semi-honest corruptions in C' from the malicious ones in C.

This immediately implies parts 1 and 2, by the fact that $\circ$ for systems respects equality and indistinguishability.

■

Theorem 4.11 (Vertical Composition Theorem). For any protocol $\mathcal{P}$ and game G, such that $\mathcal{P} \circ G$ is well defined and closed, and for any corruption class $\mathcal{C}$, we have:

1. $G = G' \implies \mathcal{P} \circ G =_{\mathcal{C}} \mathcal{P} \circ G'$
2. $G \overset{\epsilon}{\approx} G' \implies \mathcal{P} \circ G \overset{\epsilon}{\approx}_{\mathcal{C}} \mathcal{P} \circ G'$

Proof: We start by noting that $\text{Inst}_C(\mathcal{P} \circ G) = A \circ F \circ G$, for some system A. Part 1 follows immediately from this, since $\circ$ is equality respecting.

Part 2 follows by applying Lemma **??**, which entails that for any system S, we have $S \circ G \overset{\epsilon}{\approx} S \circ G'$.

■

Theorem 4.12 (Concurrent Composition Theorem). Let $\mathcal{P}, \mathcal{Q}$ be protocols, with $\mathcal{P} \otimes \mathcal{Q}$ well defined and closed. For any compatible corruption classes $\mathcal{C}, \mathcal{C}'$ it holds that:

1. $\mathcal{Q} =_{\mathcal{C}} \mathcal{Q}' \implies \mathcal{P} \otimes \mathcal{Q} =_{\mathcal{C}'} \mathcal{P} \otimes \mathcal{Q}'$
2. $\mathcal{Q} \overset{\epsilon}{\approx}_{\mathcal{C}} \mathcal{Q}' \implies \mathcal{P} \otimes \mathcal{Q} \overset{\epsilon}{\approx}_{\mathcal{C}'} \mathcal{P} \otimes \mathcal{Q}'$
3. $\mathcal{Q} \overset{\epsilon}{\rightsquigarrow}_{\mathcal{C}} \mathcal{Q}' \implies \mathcal{P} \otimes \mathcal{Q} \overset{\epsilon}{\rightsquigarrow}_{\mathcal{C}'} \mathcal{P} \otimes \mathcal{Q}'$

Proof: Theorem 4.6 (concurrent breakdown) will be essential to our proof. This implies that $\forall C \in \mathcal{C}$, then for any compatible $C' \in \mathcal{C}'$ we have:

$$\text{Inst}_{C'}(\mathcal{P} \otimes \mathcal{Q}) = \text{Inst}_{C'}(\mathcal{P}) \otimes \text{Inst}_{C}(\mathcal{Q})$$

1. Since $\mathcal{Q} =_{\mathcal{C}} \mathcal{Q}'$, we have $\forall C \in \mathcal{C}$. $\text{Inst}_C(\mathcal{Q}) = \text{Inst}_C(\mathcal{Q}')$. Now, consider any $C' \in \mathcal{C}'$. By our assumption that $\mathcal{C}'$ is compatible with $\mathcal{C}$, there exists a $C \in \mathcal{C}$ that C' is compatible with. Using concurrent breakdown, we then have:

$$\text{Inst}_{C'}(\mathcal{P} \otimes \mathcal{Q}) = \text{Inst}_{C'}(\mathcal{P}) \otimes \text{Inst}_{C}(\mathcal{Q})$$

Then, since $\mathcal{Q} =_{\mathcal{C}} \mathcal{Q}'$, we have:

$$\text{Inst}_{C'}(\mathcal{P}) \otimes \text{Inst}_{C}(\mathcal{Q}) = \text{Inst}_{C'}(\mathcal{P}) \otimes \text{Inst}_{C}(\mathcal{Q}') = \text{Inst}_{C'}(\mathcal{P} \otimes \mathcal{Q}')$$

concluding our proof.

2. The proof here is similar to part 1. For any $C' \in \mathcal{C}'$, there exists a compatible $C \in \mathcal{C}$, and then we get:

$$\text{Inst}_{C'}(\mathcal{P} \otimes \mathcal{Q}) = \text{Inst}_{C'}(\mathcal{P}) \otimes \text{Inst}_{C}(\mathcal{Q})$$

Since $\mathcal{Q} \overset{\epsilon}{\approx}_{\mathcal{C}} \mathcal{Q}'$, we have:

$$\text{Inst}_{C'}(\mathcal{P}) \otimes \text{Inst}_{C}(\mathcal{Q}) \overset{\epsilon}{\approx} \text{Inst}_{C'}(\mathcal{P}) \otimes \text{Inst}_{C}(\mathcal{Q}')$$

since $\otimes$ for systems respects this operation. We can then conclude with

$$\text{Inst}_{C'}(\mathcal{P}) \otimes \text{Inst}_{C}(\mathcal{Q}') = \text{Inst}_{C'}(\mathcal{P} \otimes \mathcal{Q}')$$

3. Once more, for any $C' \in \mathcal{C}'$, there exists a compatible $C \in \mathcal{C}$ giving us:

$$\text{Inst}_{C'}(\mathcal{P} \otimes \mathcal{Q}) = \text{Inst}_{C'}(\mathcal{P}) \otimes \text{Inst}_{C}(\mathcal{Q})$$

We then apply our assumption that $\mathcal{Q} \overset{\epsilon}{\rightsquigarrow}_{\mathcal{C}} \mathcal{Q}'$ to get:

$$\text{Inst}_{C'}(\mathcal{P}) \otimes \text{Inst}_{C}(\mathcal{Q}) \overset{\epsilon}{\approx} \text{Inst}_{C'}(\mathcal{P}) \otimes ((S \otimes 1(\ldots)) \circ \text{Inst}_{C}(\mathcal{Q}'))$$

Next, we apply interchange to get:

$$\begin{matrix} 1(\text{Out}(\text{Inst}_{C'}(\mathcal{P}))) \circ \text{Inst}_{C'}(\mathcal{P}) \\ \otimes \\ ((S \otimes 1(\ldots)) \circ \text{Inst}_{C}(\mathcal{Q}')) \end{matrix} = \begin{pmatrix} 1(\text{Out}(\text{Inst}_{C'}(\mathcal{P}))) \\ \otimes \\ S \\ \otimes \\ 1(\text{Out}(\text{Inst}_{C}(\mathcal{Q}))/\text{Out}(S)) \end{pmatrix} \circ \begin{pmatrix} \text{Inst}_{C'}(\mathcal{P}) \\ \otimes \\ \text{Inst}_{C}(\mathcal{Q}') \end{pmatrix}$$

Applying concurrent breakdown in reverse, we get that the right hand side is $\text{Inst}_{C'}(\mathcal{P} \otimes \mathcal{Q})$, and that the left hand side is the simulator showing that $\mathcal{P} \otimes \mathcal{Q} \overset{\epsilon}{\rightsquigarrow}_{\mathcal{C}'} \mathcal{P} \otimes \mathcal{Q}'$. The left hand side is a valid simulator because $\text{Out}(\text{Inst}_{C}(\mathcal{Q})) = \text{Out}(\text{Inst}_{C'}(\mathcal{Q}))$, and all of the honest parts of $\mathcal{P}$ are left untouched, since all of it is.

■

Theorem 4.13 (Horizontal Composition Theorem). For any protocols $\mathcal{P}, \mathcal{Q}$ with $\mathcal{P} \lhd \mathcal{Q}$ well defined and closed, and for any compatible corruption classes $\mathcal{C}, \mathcal{C}'$, we have:

1. $\mathcal{Q} =_{\mathcal{C}} \mathcal{Q}' \implies \mathcal{P} \lhd \mathcal{Q} =_{\mathcal{C}'} \mathcal{P} \lhd \mathcal{Q}'$
2. $\mathcal{Q} \overset{\epsilon}{\approx}_{\mathcal{C}} \mathcal{Q}' \implies \mathcal{P} \lhd \mathcal{Q} \overset{\epsilon}{\approx}_{\mathcal{C}'} \mathcal{P} \lhd \mathcal{Q}'$

Furthermore, if $\mathcal{C}'$ is *strictly* compatible with $\mathcal{C}$, we have:

3. $\mathcal{Q} \overset{\epsilon}{\rightsquigarrow}_{\mathcal{C}} \mathcal{Q}' \implies \mathcal{P} \lhd \mathcal{Q} \overset{\epsilon}{\rightsquigarrow}_{\mathcal{C}'} \mathcal{P} \lhd \mathcal{Q}'$

Proof: As one might expect, Theorem 4.7(horizontal breakdown) will be critical to proving each of these statements.

One crude summary of the theorem, in the case that the protocols are closed, is that given compatible corruption models C, C', there's a system Stuff such that

$$\text{Inst}_{C'}(\mathcal{P} \lhd \mathcal{Q}) = \text{Stuff} \circ \text{Inst}_{C}(\mathcal{Q})$$

This summary suffices to prove a couple statements already.

1. By assumption, for any $C' \in \mathcal{C}'$, there exists a compatible $C \in \mathcal{C}$. In this case, we have:

$$\text{Inst}_{C'}(\mathcal{P} \lhd \mathcal{Q}) = \text{Sutff} \circ \text{Inst}_{C}(\mathcal{Q})$$

If we then apply $\mathcal{Q} =_{\mathcal{C}} \mathcal{Q}'$, we get:

$$\text{Stuff} \circ \text{Inst}_{C}(\mathcal{Q}) = \text{Stuff} \circ \text{Inst}_{C}(\mathcal{Q}')$$

and then, applying breakdown in reverse, we end up with $\text{Inst}_{C'}(\mathcal{P} \lhd \mathcal{Q}')$, completing our proof.

2. We apply the same reasoning, with the difference that:

$$\text{Stuff} \circ \text{Inst}_C(\mathcal{Q}) \overset{\epsilon}{\approx} \text{Stuff} \circ \text{Inst}_C(\mathcal{Q}')$$

rather than being strictly equal.

3. At this point our crude summary of the breakdown theorem is not sufficient anymore. We start with the same reasoning. For any $C' \in \mathcal{C}'$, there exists a *strictly* compatible $C \in \mathcal{C}$, and we have:

$$\text{Inst}_{C'}(\mathcal{P} \lhd \mathcal{Q}) = \text{Stuff} \circ \text{Inst}_C(\mathcal{Q})$$

then, we apply our assumption that $\mathcal{Q} \overset{\epsilon}{\rightsquigarrow}_{\mathcal{C}} \mathcal{Q}'$, giving us:

$$\text{Stuff} \circ \text{Inst}_C(\mathcal{Q}) \overset{\epsilon}{\approx} \text{Stuff} \circ (S \otimes 1(\ldots)) \circ \text{Inst}_C(\mathcal{Q})$$

Our strategy will be to rearrange the right hand side to get

$$(S' \otimes 1(\ldots)) \circ \text{Inst}_{C'}(\mathcal{P} \lhd \mathcal{Q}')$$

We start by unrolling Stuff, using strict compatability, to get:

$$1(O) \circ \begin{pmatrix} \circledast_{i \in [\mathcal{P}.n]} \text{Routed}(\text{Corrupt}'_{C'}(\mathcal{P}.P_i)) \\ * \\ \mathcal{R}_{\mathcal{P}} \\ \otimes \\ 1(\text{Leakage}, L_{\mathcal{Q}'}) \end{pmatrix} \circ \begin{pmatrix} \mathcal{P}.F \\ \otimes \\ 1(\text{Out}(\mathcal{R}_q)) \\ \otimes \\ 1(\mathcal{Q}'.\text{Leakage}) \\ \otimes \\ \bigotimes_{i \in [\mathcal{Q}'.n]} 1_i \end{pmatrix} \circ \begin{pmatrix} S \\ \otimes \\ 1(O_{\bar{S}}) \end{pmatrix} \circ \text{Inst}_C(\mathcal{Q}')$$

with $O_{\bar{S}} := \text{Out}(\text{Inst}_C(\mathcal{Q}'))/\text{Out}(S)$, and with each $1_i := 1(\text{Out}(\text{Inst}_C(\mathcal{Q}'.P_i)))$.
we can apply interchange a few times to get:

$$1(O) \circ \begin{pmatrix} \begin{pmatrix} \circledast_{C'_i \neq \text{H}} \begin{pmatrix} \text{Routed}(\text{Corrupt}'_{C'}(\mathcal{P}.P_i)) \\ \otimes \\ 1(L_i) \\ \otimes \\ 1(\text{Leakage}) \end{pmatrix} \end{pmatrix} \circ \begin{pmatrix} S \\ \otimes \\ 1(O_S) \end{pmatrix} \circ \begin{pmatrix} \circledast_{C_i \neq \text{H}} \text{Routed}(\text{Corrupt}_C(\mathcal{Q}'.P_i)) \\ \otimes \\ 1(\text{Out}(\mathcal{P}.F), \text{Out}(\mathcal{Q}.F)) \end{pmatrix} \\ * \\ \circledast_{C'_i = \text{H}} \text{Routed}(\text{Corrupt}_{C'}((\mathcal{P} \lhd \mathcal{Q}').P_i)) \\ * \\ \mathcal{R}_{\mathcal{P}} \circ \mathcal{R}_{\mathcal{Q}'} \end{pmatrix} \circ \begin{pmatrix} \mathcal{P}.F \\ \otimes \\ \mathcal{Q}'.F \end{pmatrix}$$

with $O_S := O_{\bar{S}} \cup \text{Out}(\mathcal{P}.F)$ and L_i as per the horizontal breakdown theorem. The only functions that S masks are the leakage, the malicious corruption functions, and the logs from semi-honest corruption. Semi-honest corruption does

not use any outputs of S, instead relying on the $\mathcal{Q}'.P_i$, accessible via $1(O_S)$. In the case of malicious corruption, since $\text{Corrupt}'_{C'}(\mathcal{P}.P_i)$ omits the Call_{F_i} functions, the system also has no dependencies on the output of S. Since none of these corrupted players depend on S, we can slide it forward, using interchange, to get:

$$1(O)\circ\left(\begin{array}{c}\begin{pmatrix}S\\\otimes\\1(\ldots)\end{pmatrix}\circ\left(\begin{array}{c}\ast\\{\scriptstyle C'_i\neq \text{H}}\end{array}\begin{pmatrix}\text{Routed}(\text{Corrupt}'_{C'}(\mathcal{P}.P_i))\\\otimes\\1(L_i)\\\otimes\\1(\text{Leakage})\end{pmatrix}\right)\circ\left(\begin{array}{c}\ast\\{\scriptstyle C_i\neq \text{H}}\end{array}\begin{array}{c}\text{Routed}(\text{Corrupt}_C(\mathcal{Q}'.P_i))\\\otimes\\1(\text{Out}(\mathcal{P}.F),\text{Out}(\mathcal{Q}.F))\end{array}\right)\\\ast\\\begin{array}{c}\ast\\{\scriptstyle C'_i=\text{H}}\end{array}\text{Routed}(\text{Corrupt}_{C'}((\mathcal{P}\lhd\mathcal{Q}').P_i))\\\ast\\\mathcal{R}_{\mathcal{P}}\circ\mathcal{R}_{\mathcal{Q}'}\end{array}\right)\circ\begin{pmatrix}\mathcal{P}.F\\\otimes\\\mathcal{Q}'.F\end{pmatrix}$$

which becomes:

$$\begin{pmatrix}S\\\otimes\\1(\text{Out}(\text{Inst}_{C'}(\mathcal{P}\lhd\mathcal{Q}'))/\text{Out}(S))\end{pmatrix}\circ\text{Inst}_{C'}(\mathcal{P}\lhd\mathcal{Q}')$$

From this chain of equalities we conclude that $\mathcal{P}\lhd\mathcal{Q}'\overset{\epsilon}{\rightsquigarrow}\mathcal{P}\lhd\mathcal{Q}'$

■

4.3 Global Functionalities

Definition 4.18 (Relatively Closed Protocols). A protocol $\mathcal{P}$ is *closed relative to* a game G if:

- $\text{In}(\mathcal{P})=\emptyset$
- $\text{IdealIn}(\mathcal{P})\subseteq\text{Out}(G)$

□

Definition 4.19 (Relative Instantiation). Given a closed protocol $\mathcal{P}$ relative to G, we can define, for any corruption model C, the relative instantiation:

$$\text{Inst}_C^G(\mathcal{P}):=\begin{pmatrix}\text{Inst}_C(\mathcal{P})\\\otimes\\1(\text{Out}(G))\end{pmatrix}\circ G$$

We can also extend this to the case of simulated instantiation, defining, for any simulator S:

$$\text{SimInst}_{S,C}^G(\mathcal{P}):=\begin{pmatrix}\text{SimInst}_{S,C}(\mathcal{P})\\\otimes\\1(\text{Out}(G))\end{pmatrix}\circ G$$

□

Definition 4.20 (Relative Notions of Equality). Given closed protocols $\mathcal{P}, \mathcal{Q}$ relative to G, with the same shape, and a corruption class $\mathcal{C}$ for these protocols, we define:

- $\mathcal{P} =_{\mathcal{C}}^{G} \mathcal{Q} \iff \forall C \in \mathcal{C}.\ \text{Inst}_C^G(\mathcal{P}) = \text{Inst}_C^G(\mathcal{Q})$
- $\mathcal{P} \overset{\epsilon}{\approx}_{\mathcal{C}}^{G} \mathcal{Q} \iff \forall C \in \mathcal{C}.\ \text{Inst}_C^G(\mathcal{P}) \overset{\epsilon}{\approx} \text{Inst}_C^G(\mathcal{Q})$
- $\mathcal{P} \overset{\epsilon}{\rightsquigarrow}_{\mathcal{C}}^{G} \mathcal{Q} \iff \forall C \in \mathcal{C}.\ \exists S.\ \text{Inst}_C^G(\mathcal{P}) \overset{\epsilon}{\approx} \text{SimInst}_{S,C}^G(\mathcal{Q})$

□

Theorem 4.14 (Relative Equality Hierarchy). For any corruption class $\mathcal{C}$ and game G, we have:

1. $\mathcal{P} =_{\mathcal{C}}^{G} \mathcal{Q} \implies \mathcal{P} \overset{0}{\approx}_{\mathcal{C}}^{G} \mathcal{Q}$.
2. $\mathcal{P} \overset{\epsilon}{\approx}_{\mathcal{C}}^{G} \mathcal{Q} \implies \mathcal{P} \overset{\epsilon}{\rightsquigarrow}_{\mathcal{C}}^{G} \mathcal{Q}$.

Proof:

1. This follows from the fact that $A = B \implies A \overset{0}{\approx} B$ for any systems A, B.

2. In the proof of Theorem 4.8, we used the existence of a simulator S such that $\text{SimInst}_{S,C}(\mathcal{P}) = \text{Inst}_C(\mathcal{P})$. This simulator will also satisfy $\text{SimInst}_{S,C}^G(\mathcal{P}) = \text{Inst}_C^G(\mathcal{P})$, and can thus be used directly to prove this relation.

■

Theorem 4.15 (Transitivity of Relative Equality). For any protocols $\mathcal{L}$, $\mathcal{P}$, $\mathcal{Q}$ closed relative to a game G, and for any corruption class, we have:

1. $\mathcal{L} =_{\mathcal{C}}^{G} \mathcal{P}, \mathcal{P} =_{\mathcal{C}}^{G} \mathcal{Q} \implies \mathcal{L} =_{\mathcal{C}}^{G} \mathcal{Q}$,
2. $\mathcal{L} \overset{\epsilon_1}{\approx}_{\mathcal{C}}^{G} \mathcal{P}, \mathcal{P} \overset{\epsilon_2}{\approx}_{\mathcal{C}}^{G} \mathcal{Q} \implies \mathcal{L} \overset{\epsilon_1+\epsilon_2}{\approx}_{\mathcal{C}}^{G} \mathcal{Q}$,
3. $\mathcal{L} \overset{\epsilon_1}{\rightsquigarrow}_{\mathcal{C}}^{G} \mathcal{P}, \mathcal{P} \overset{\epsilon_2}{\rightsquigarrow}_{\mathcal{C}}^{G} \mathcal{Q} \implies \mathcal{L} \overset{\epsilon_1+\epsilon_2}{\rightsquigarrow}_{\mathcal{C}}^{G} \mathcal{Q}$.

Proof: Once again, the first two parts follow directly from Lemma 3.5, by considering the systems $\text{Inst}_C^G(\mathcal{L})$, $\text{Inst}_C^G(\mathcal{P})$, $\text{Inst}_C^G(\mathcal{Q})$ for any $C \in \mathcal{C}$.

For part 3, given any $C \in \mathcal{C}$, there exists S_1, S_2 such that:

- $\text{Inst}_C^G(\mathcal{L}) \overset{\epsilon_1}{\approx} \text{SimInst}_{S_1,C}^G(\mathcal{P})$,
- $\text{Inst}_C^G(\mathcal{P}) \overset{\epsilon_2}{\approx} \text{SimInst}_{S_2,C}^G(\mathcal{Q})$.

Next, observe that for any protocol $\mathcal{P}$, we can write:

$$\text{SimInst}_C^G = \begin{pmatrix} S \\ \otimes \\ 1(O) \end{pmatrix} \circ \text{Inst}_C^G(\mathcal{P})$$

where $O = \text{Out}(\text{Inst}_C(\mathcal{P}))/\text{Out}(S) \cup \text{Out}(G)$.

We then apply transitivity for systems and interchange get:

$$\text{Inst}_C^G(\mathcal{L}) \overset{\epsilon_1+\epsilon_2}{\approx} \begin{pmatrix} S_1 \circ S_2 \\ \otimes \\ 1(O) \end{pmatrix} \circ \text{Inst}_C^G(\mathcal{Q})$$

And the left side is simply $\text{SimInst}^G_{(S_1 \circ S_2),C}(\mathcal{Q})$, concluding our proof.

■

Theorem 4.16 (Global Malicious Completeness). Let $\mathcal{P}$ and $\mathcal{Q}$ closed protocols relative to G with the same shape. Given any class of corruptions $\mathcal{C}$, let $\mathcal{C}'$ be a related class, containing models in $\mathcal{C}$ with some malicious corruptions replaced with semi-honest corruptions. We then have:

1. $\mathcal{P} =_C^G \mathcal{Q} \implies \mathcal{P} =_{C'}^G \mathcal{Q}$,
2. $\mathcal{P} \overset{\epsilon}{\approx}_C^G \mathcal{Q} \implies \mathcal{P} \overset{\epsilon}{\approx}_{C'}^G \mathcal{Q}$,

Proof: We proceed similarly to Theorem 4.10 (malicious completeness). In that theorem, the key observation was that for any $C' \in \mathcal{C}'$ and the related $C \in \mathcal{C}$, it holds that:

$$\text{Inst}_{C'}(\mathcal{P}) = \text{SimInst}_{S_{\text{SH}},C}(\mathcal{P})$$

(this observation also doesn't depend on $\mathcal{P}$ being fully closed, allowing us to use it here).

Now, this clearly implies that:

$$\text{Inst}^G_{C'}(\mathcal{P}) = \text{SimInst}^G_{S_{\text{SH}},C}(\mathcal{P})$$

And then, using our observation from Theorem 4.15, we can write this as:

$$\text{Inst}^G_{C'}(\mathcal{P}) = \begin{pmatrix} S_{\text{SH}} \\ \otimes \\ 1(O) \end{pmatrix} \circ \text{Inst}^G_C(\mathcal{P})$$

where $O = \text{Out}(\text{Inst}_C(\mathcal{P}))/\text{Out}(S) \cup \text{Out}(G)$.

This immediately implies parts 1 and 2.

■

Theorem 4.17 (Global Vertical Composition Theorem). For any protocol $\mathcal{P}$ and game F, such that $\mathcal{P} \circ F$ is well defined and closed relative to G, and for any corruption class $\mathcal{C}$, we have:

1. $F = F' \implies \mathcal{P} \circ F =^{G}_{\mathcal{C}} \mathcal{P} \circ F'$
2. $F \overset{\epsilon}{\approx} F' \implies \mathcal{P} \circ F \overset{\epsilon}{\approx}^{G}_{\mathcal{C}} \mathcal{P} \circ F'$

Proof: The proof of Theorem 4.11 will be the basis for what we do here. Using it, we can write:

$$\text{Inst}^{G}_{C}(\mathcal{P} \circ F) = \begin{pmatrix} A \circ F \\ \otimes \\ 1(\text{Out}(G)) \end{pmatrix} \circ G$$

for some system A. At this point, the theorem immediately holds, since $\circ$ and $\otimes$ (for systems) respect both $=$ and $\approx$.

■

Theorem 4.18 (Global Concurrent Composition Theorem). Let $\mathcal{P}, \mathcal{Q}$ be closed protocols relative to G, with $\mathcal{P} \otimes \mathcal{Q}$ well defined. For any compatible corruption classes $\mathcal{C}, \mathcal{C}'$ it holds that:

1. $\mathcal{Q} =^{G}_{\mathcal{C}} \mathcal{Q}' \implies \mathcal{P} \otimes \mathcal{Q} =^{G}_{\mathcal{C}'} \mathcal{P} \otimes \mathcal{Q}'$
2. $\mathcal{Q} \overset{\epsilon}{\approx}^{G}_{\mathcal{C}} \mathcal{Q}' \implies \mathcal{P} \otimes \mathcal{Q} \overset{\epsilon}{\approx}^{G}_{\mathcal{C}'} \mathcal{P} \otimes \mathcal{Q}'$
3. $\mathcal{Q} \overset{\epsilon}{\rightsquigarrow}^{G}_{\mathcal{C}} \mathcal{Q}' \implies \mathcal{P} \otimes \mathcal{Q} \overset{\epsilon}{\rightsquigarrow}^{G}_{\mathcal{C}'} \mathcal{P} \otimes \mathcal{Q}'$

Proof: We start by using Theorem 4.6, giving us:

$$\text{Inst}^{G}_{C'}(\mathcal{P} \otimes \mathcal{Q}) = \begin{pmatrix} \text{Inst}_{C'}(\mathcal{P}) \\ \otimes \\ \text{Inst}_{C}(\mathcal{Q}) \\ \otimes \\ 1(\text{Out}(G)) \end{pmatrix} \circ G = \begin{pmatrix} \text{Inst}_{C'}(\mathcal{P}) \\ \otimes \\ 1(\text{Out}(\text{Inst}_{C}(\mathcal{Q}))) \\ \otimes \\ 1(\text{Out}(G)) \end{pmatrix} \circ \text{Inst}^{G}_{C}(\mathcal{Q})$$

We can then immediately derive parts 1 and 2.

For part 3, we apply the hypothesis to the last part of the above relation, to get:

$$\text{Inst}^{G}_{C'} \overset{\epsilon}{\approx} \begin{pmatrix} \text{Inst}_{C'}(\mathcal{P}) \\ \otimes \\ 1(\text{Out}(\text{Inst}_{C}(\mathcal{Q}))) \\ \otimes \\ 1(\text{Out}(G)) \end{pmatrix} \circ \text{SimInst}^{G}_{S,C}(\mathcal{Q})$$

Then, we unroll $\text{SimInst}^G_{S,C}(\mathcal{Q})$, to get:

$$\begin{pmatrix} \text{Inst}_{C'}(\mathcal{P}) \\ \otimes \\ 1(\text{Out}(\text{Inst}_C(\mathcal{Q}))) \\ \otimes \\ 1(\text{Out}(G)) \end{pmatrix} \circ \begin{pmatrix} \begin{pmatrix} S \\ \otimes \\ 1(\ldots) \end{pmatrix} \circ \text{Inst}_C(\mathcal{Q}) \\ \otimes \\ 1(\text{Out}(G)) \end{pmatrix} \circ G$$

Then, we apply interchange to get:

$$\begin{pmatrix} \begin{pmatrix} 1(\ldots) \\ \otimes \\ S \\ \otimes \\ 1(\ldots) \end{pmatrix} \circ \begin{pmatrix} \text{Inst}_{C'}(\mathcal{P}) \\ \otimes \\ \text{Inst}_C(\mathcal{Q}) \end{pmatrix} \\ \otimes \\ 1(\text{Out}(G)) \end{pmatrix} \circ G$$

But this is just $\text{SimInst}^G_{S',C'}(\mathcal{P} \otimes \mathcal{Q})$, for some simulator S', applying concurrent breakdown in reverse.

■

Theorem 4.19 (Global Horizontal Composition Theorem). For any protocols $\mathcal{P}, \mathcal{Q}$ closed relative to G, with $\mathcal{P} \lhd \mathcal{Q}$ well defined, and for any compatible corruption classes $\mathcal{C}, \mathcal{C}'$, we have:

1. $\mathcal{Q} =^G_{\mathcal{C}} \mathcal{Q}' \implies \mathcal{P} \lhd \mathcal{Q} =^G_{\mathcal{C}'} \mathcal{P} \lhd \mathcal{Q}'$
2. $\mathcal{Q} \overset{\epsilon}{\approx}^G_{\mathcal{C}} \mathcal{Q}' \implies \mathcal{P} \lhd \mathcal{Q} \overset{\epsilon}{\approx}^G_{\mathcal{C}'} \mathcal{P} \lhd \mathcal{Q}'$

Furthermore, if $\mathcal{C}'$ is *strictly* compatible with $\mathcal{C}$, we have:

3. $\mathcal{Q} \overset{\epsilon}{\rightsquigarrow}^G_{\mathcal{C}} \mathcal{Q}' \implies \mathcal{P} \lhd \mathcal{Q} \overset{\epsilon}{\rightsquigarrow}^G_{\mathcal{C}'} \mathcal{P} \lhd \mathcal{Q}'$

Proof: This proof is similar to that of Theorem 4.13. By compatability, for any $C' \in \mathcal{C}'$, we have a compatible $C \in \mathcal{C}$.

A crude summary of the horizontal breakdown theorem is that:

$$\text{Inst}_{C'}(\mathcal{P} \lhd \mathcal{Q}) = \text{Stuff} \circ \begin{pmatrix} \text{Inst}_C(\mathcal{Q}) \\ \otimes \\ 1(\text{In}(\mathcal{P}.F)) \end{pmatrix}$$

Using the fact that being closed relative to G means $\text{In}(\mathcal{P}.F) \subseteq \text{Out}(G)$, we get:

$$\text{Inst}^G_{C'}(\mathcal{P} \lhd \mathcal{Q}) = \begin{pmatrix} \text{Stuff} \\ \otimes \\ 1(\text{Out}(G)) \end{pmatrix} \circ \text{Inst}^G_C(\mathcal{Q})$$

Part 1 and 2 both follow immediately from this decomposition.

For part 3, we dig a bit deeper into the proof of Theorem 4.13. In that proof, it was actually shown that:

$$\text{Stuff} \circ \text{SimInst}_{S,C}(\mathcal{Q}') = \text{SimInst}_{S',C'}(\mathcal{P} \lhd \mathcal{Q}')$$

for some appropriate simulator S'.

We can start to apply this, first by using our hypothesis:

$$\text{Inst}^G_{C'}(\mathcal{P} \lhd \mathcal{Q}) = \begin{pmatrix} \text{Stuff} \\ \otimes \\ 1(\text{Out}(G)) \end{pmatrix} \circ \text{Inst}^G_C(\mathcal{Q}) \overset{\epsilon}{\approx} \begin{pmatrix} \text{Stuff} \\ \otimes \\ 1(\text{Out}(G)) \end{pmatrix} \circ \text{SimInst}^G_C(\mathcal{Q}')$$

Next, we unroll the right side, to get:

$$\begin{pmatrix} \text{Stuff} \\ \otimes \\ 1(\text{Out}(G)) \end{pmatrix} \circ \begin{pmatrix} \text{SimInst}_{S,C}(\mathcal{Q}') \\ \otimes \\ 1(\text{Out}(G)) \end{pmatrix} \circ G$$

Then, apply interchange, to get:

$$\begin{pmatrix} \text{Stuff} \circ \text{SimInst}_{S,C}(\mathcal{Q}') \\ \otimes \\ 1(\text{Out}(G)) \end{pmatrix} \circ G$$

And finally, apply the fact we dug up above, to get:

$$\begin{pmatrix} \text{SimInst}_{S',C'}(\mathcal{P} \lhd \mathcal{Q}) \\ \otimes \\ 1(\text{Out}(G)) \end{pmatrix} \circ G$$

which is none other than $\text{SimInst}^G_{S',C'}(\mathcal{P} \lhd \mathcal{Q})$.

■

4.4 Hopping Ideal Functionalities

Lemma 4.20 (Deidealization Lemma). Given a closed protocol $\mathcal{P}$ with an ideal functionality $F \otimes G$, there exists protocols $\mathcal{P}'$ and $\mathcal{G}$ such that:

$$\mathcal{P} \equiv \mathcal{P}' \lhd \mathcal{G}$$

and $\mathcal{P}'$ has ideal functionality F.

Proof: The players of $\mathcal{P}'$ are those of $\mathcal{P}$, except that each P_i's call to a function $g \in \text{Out}(G)$ is replaced with a renamed function g_i. $\mathcal{G}$ will have one player for

each player in $\mathcal{P}'$. Each player $\mathcal{G}.P_i$ exports a function g_i for each input g_i of $\mathcal{P}'.P_i$, which immediately calls $g \in \text{Out}(G)$, and returns the result. The leakage of $\mathcal{G}$ will simply be $\mathcal{P}.\text{Leakage} \cap \text{Out}(G)$. From this definition, it's clear that $\mathcal{P}$ is literally equal to $\mathcal{P}' \lhd \mathcal{G}$, as when the players in the latter are formed, the calls to the intermediate g_i disappear, with each player calling $g \in \text{Out}(G)$ directly

■

Lemma 4.21 (Embedding Lemma). Given a protocol $\mathcal{P}$ closed relative to a game G, there exists a protocol $\text{Embed}_G(\mathcal{P})$ such that for any corruption model C, we have:

$$\text{Inst}_C^G(\mathcal{P}) = \text{Inst}_C(\text{Embed}_G(\mathcal{P}))$$

Proof: This one is quite simple. $\text{Embed}_G(\mathcal{P})$ has the same players as $\mathcal{P}$, with the ideal functionality becoming:

$$\begin{pmatrix} \mathcal{P}.F \\ \otimes \\ 1(\text{Out}(G)) \end{pmatrix} \circ G$$

and the leakage being $\mathcal{P}.\text{Leakage} \cup \text{Out}(G)$. The two instantiations will then clearly be equal under any corruption model.

■

4.5 Some Syntactical Conventions

5 Examples

5.1 Constructing Private Channels

In this section, we consider the problem of constructing a *private* channel from a *public* channel. A public channel leaks all messages sent over it to an adversary, whereas a private channel leaks a minimal amount of information: in our case, essentially just the length of messages sent over the channel.

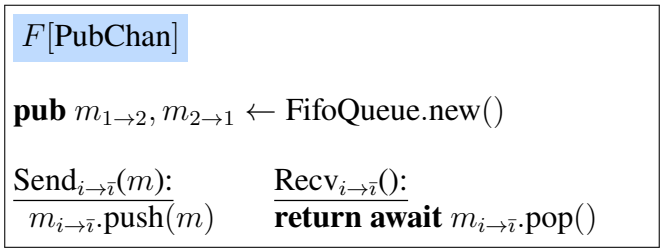
We'll be constructing a two-party private channel from a public channel using an encryption scheme, and will also show that this construction is secure, even if one of the two parties using the channel is corrupted.

Let's start with the ideal functionality representing a public channel, as Game 5.1.

A few clarifications on the notation in this game:

- For $i \in \{1, 2\}$, we let $\bar{i}$ denote either 2 or 1, respectively.
- There are two versions of Send_i and Recv_i, for $i \in \{1, 2\}$.

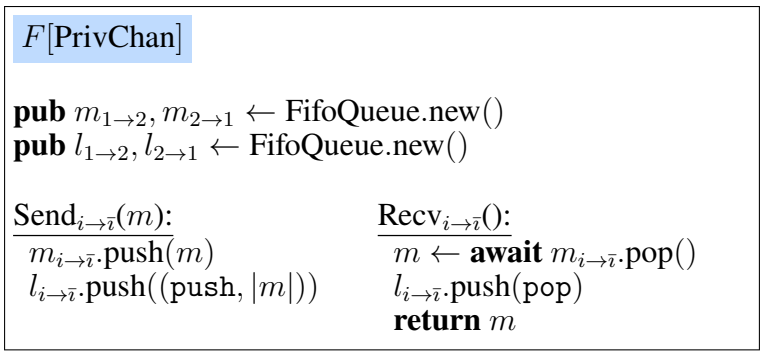
- The pop function on queues is asynchronous, meaning that we wait until the queue is not empty to remove the oldest element from it.
- The queues are public in an *immutable* fashion: they can be read but not modified outside the package.



Game 5.1: Public Channel Functionality

The idea behind this functionality is that each party can send messages to, or receive messages from the other party. However, at any point, the currently stored messages are readable by the adversary. Note that this assignment of which functions are usable by which entities is not defined by the functionality *itself*, but rather merely suggested by its syntax, and enforced only by how protocols will eventually use the functionality.

Next, we look at a functionality for *private* channels, captured by Game 5.2.



Game 5.2: Private Channel Functionality

The crucial difference is the nature of the leakage. Now, rather than being able to see the current state of either message queue, including the messages themselves, now the adversary can only see a historical log for each queue, describing only the *length* of the messages inserted into the queue. The reason for having a historical log, rather than just a snapshot of the lengths of the current messages, is to make the simulator's job easier in the eventual proof of security.

Now, we need to define the protocols. One protocol will use the private channel

to send messages, and the other will try and implement the same behavior, but using only the public channel, aided by an encryption scheme.

Let's start with the simpler private channel protocol, which we'll call, for lack of imagination, $\mathcal{Q}$, and defined via Protocol 5.3

$\mathcal{Q}$ is characterized by:

- Leakage $:= \{l_{1\to 2}, l_{2\to 1}\}$,
- $F :=$ PrivChan,
- And two players defined via the following system (for $i \in \{1, 2\}$):

P_i

$\underline{\text{Send}_i(m)\text{:}}$
$\text{Send}_{i\to\bar{i}}(m)$

$\underline{\text{Recv}_i()\text{:}}$
return await $\text{Recv}_{\bar{i}\to i}()$

Protocol 5.3: Private Channel Protocol

This protocol basically just provides each player access with their corresponding functions in the functionality, and leaks the parts of the functionality that the adversary should have access to, as expected.

Next, we need to define a protocol providing an encrypted channel. Thanks to a brilliant stroke of creativity, we'll call this one $\mathcal{P}$. The basic idea is that $\mathcal{P}$ will encrypt messages before sending them over the public channel. We'll be using public-key encryption, as defined in todo. For the sake of simplicity, we'll be relying on an additional global functionality, Keys, which will be used to setup each party's key pair, and allow each party to agree on the other's public key.

This functionality is defined in Game 5.4. The basic idea is that a key pair is generated for each party, and that party can see their secret key, along with the public key for the other party.

Keys

$(\text{sk}_1, \text{pk}_1) \xleftarrow{\$} \text{Gen}()$
$(\text{sk}_2, \text{pk}_2) \xleftarrow{\$} \text{Gen}()$

$\underline{\text{Keys}_i()\text{:}}$
return $(\text{sk}_i, \text{pk}_{\bar{i}})$

Game 5.4: Keys Functionality

With this in hand, we can define $\mathcal{P}$ itself, in Protocol 5.5.

$\mathcal{P}$ is characterized by:

- Leakage $:= \{m_{1\to 2}, m_{2\to 1}\}$,
- $F := 1(\text{Keys}) \otimes \text{PrivChan}$,
- And two players defined via the following system (for $i \in \{1, 2\}$):

P_i

$(\text{sk}_i, \text{pk}_{\bar{i}}) \leftarrow \text{Keys}_i()$

$\underline{\text{Send}_i(m)\text{:}}$	$\underline{\text{Recv}_i()\text{:}}$
$\text{Send}_{i\to\bar{i}}(\text{Enc}(\text{pk}_{\bar{i}}, m))$	$c \leftarrow$ **await** $\text{Recv}_{\bar{i}\to i}()$
	return $\text{Dec}(\text{sk}_i, c)$

Protocol 5.5: Encrypted Channel Protocol

Each player will encrypt their message for the other player before sending it, and then decrypt it using their secret key after receiving it. The Keys functionality isn't embedded into the protocol, but is instead considered to be a global functionality.

6 Differences with UC Security

7 Further Work

8 Conclusion

References

[Mei22] Lúcás Críostóir Meier. MPC for group reconstruction circuits. Cryptology ePrint Archive, Report 2022/821, 2022. `https://eprint.iacr.org/2022/821`.

A Additional Game Definitions

In this section, we include explicit definitions of several games we use throughout the rest of this work. While we expect these notions to be familiar, we think the precise details are worth spelling out here.

A.1 Encryption

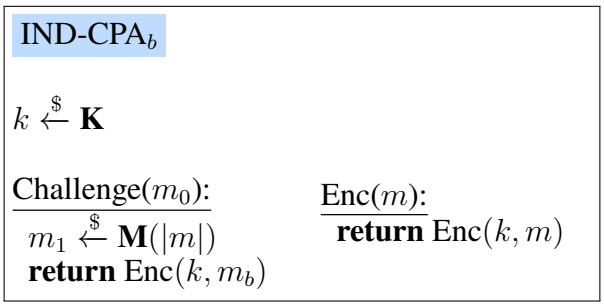
An encryption scheme consists of types $\mathbf{K}, \mathbf{M}, \mathbf{C}$, along with probabilistic functions $\text{Enc} : \mathbf{K} \times \mathbf{M} \xleftarrow{\$} \mathbf{C}$ and $\text{Dec} : \mathbf{K} \times \mathbf{C} \to \mathbf{M}$. By $\mathbf{M}(|m|)$ we denote the distribution of messages with the same length as m. We require that $\mathbf{K}$ and $\mathbf{M}(|m|)$ are efficiently sampleable.

The encryption scheme must satisfy a correctness property:

$$\forall k \in \mathbf{K},\ m \in \mathbf{M}.\ P[\text{Dec}(\text{Enc}(k, m)) = m] = 1$$

Encrypting and then decrypting a message should return that same message.

The security of an encryption scheme can be captured by the following game:



In essence, an adversary cannot distinguish between an encryption of a message of their choice and that of a random message, even if they can encrypt messages at will.