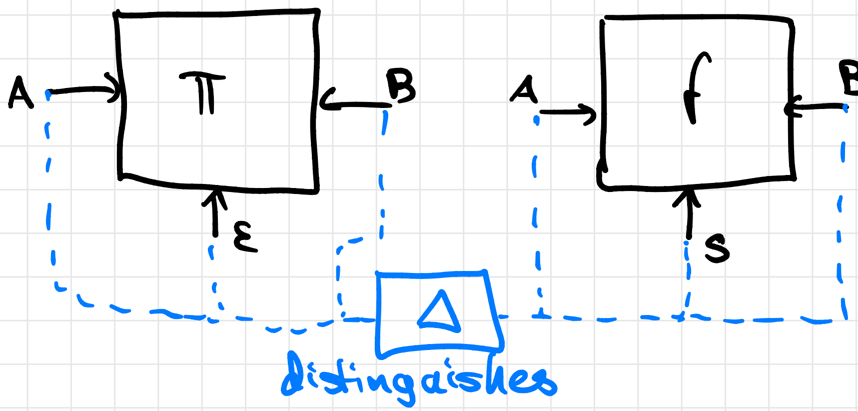
Last week we saw a model which achieves sequential composability, and forms a stepping stone along the way to UC. Broadly speaking, we have parties (e.g. $\{A, B\}$) who choose inputs according to some distribution and a function f specifying the output for each party and the adversary. A protocol π is a set of PPT interactive algorithms for each party to run.

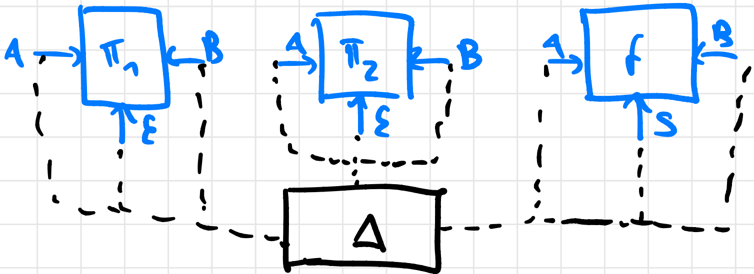
We said that π securely realises f if for all PPT adversaries $\mathcal{E}$, there exists a PPT simulator S such that A interacting with π produces outputs indistinguishable from those of S interacting with f.

Note that we can think of an environment as a prob. algorithm Δ that chooses the inputs and sees the outputs, and decides whether they were generated by $\mathcal{E}$ and π $(G(\mathcal{E},\pi))$ or S and f $(G(S,f))$. Security is then the statement that no environment can succeed with non-negligible probability.



The environment is meant to capture everything external to the current protocol that could be used to distinguish the two worlds. As we saw, concurrent protocol execution is not allowed: the environment only interacts before and after the protocol executes, otherwise no security is guaranteed.

UC extends this model by allowing the adversary and environment to interact at (more or less) any time.



In the case of concurrent executions, the distinguisher can tell the worlds apart by using outputs from π_1 as inputs to π_2.

We now build up the communication model we'll need to handle concurrent executions, and in fact almost-arbitrary interleavings of messages. Strap yourselves in...

Machines → Protocols → Execution → Emulation

We will consider protocols made up of several abstract machines ($\approx$ probabilistic interactive Turing machines). Each machine μ has the following properties:

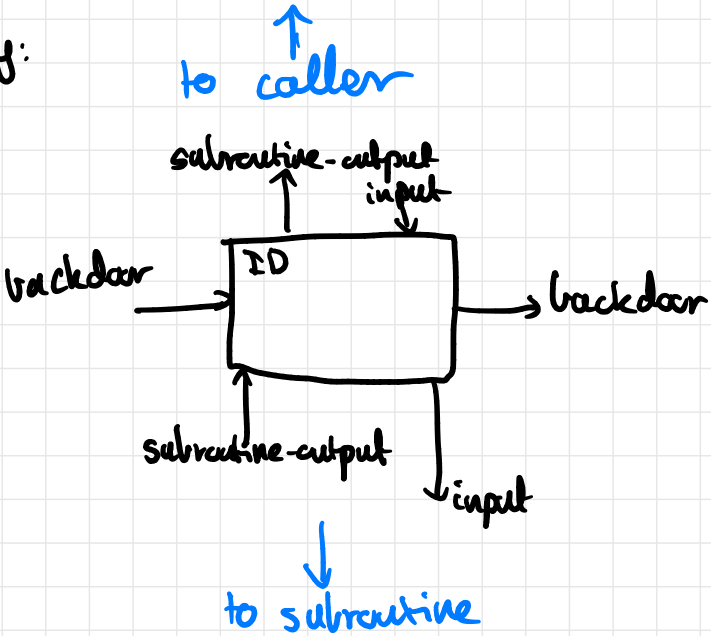
1) an identity: an element of $\mathbb{N}$ that is visible to μ and constant
2) incoming information is either input (from a "caller"), subroutine-output, or backdoor (communication w/ adversary)
3) a communication set: which identities μ can send information to and of which kind.

For example, a secure channel S from machine X to machine Y has communication set $C = \{(X, \text{input}), (Y, \text{subroutine-output}), (\mathcal{A}, \text{backdoor})\}$.

Defn 1

A machine is a triple $\mu = (ID, C, \tilde{\mu})$ where ID is the identity, C is a comm. set, and $\tilde{\mu}$ is the "program" (e.g. formal prob. ITM).

Pictorially:

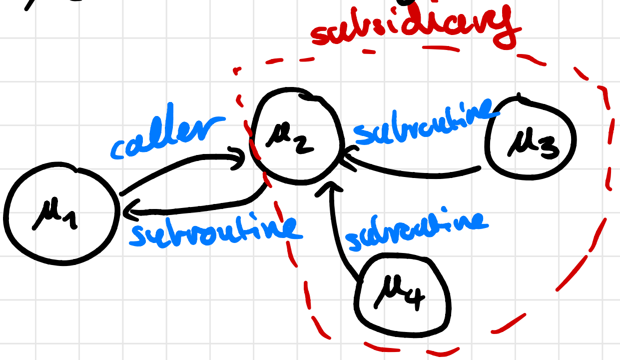


To define protocols we model relationships that machines can have.

Def^n 2

For a set of machines $\{\mu_1, \dots, \mu_n\}$ where $\mu_i = (ID_i, C_i, \tilde{\mu}_i)$, we define μ_i to be

- a caller of μ_j if $(\text{input}, ID_j) \in \mu_i$
- a subroutine of μ_j if $(\text{subroutine\_output}, ID_j) \in \mu_i$
- a subsidiary of μ_j if it is a subroutine of μ_j, or of μ_k where μ_k is a subsidiary of μ_j.



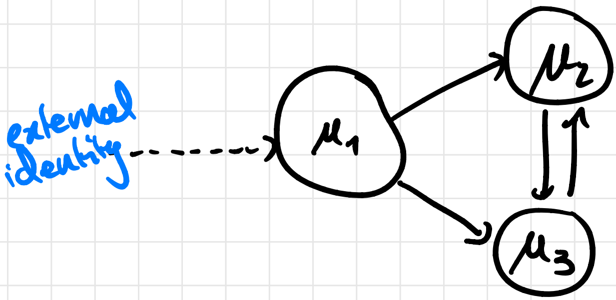
We need certain coherence conditions for a protocol to make sense.

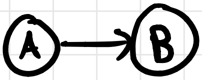
Def^n 3

A tuple of machines $\pi = (\mu_1, \dots, \mu_n)$ is a protocol if

- for each μ_i that is a caller of μ_j, μ_j is a subroutine of μ_i
- for each μ_i that is a subroutine of identity ID, if $ID_j = ID$ for some j, then μ_j is a caller of μ_i

If there is no j, we say μ_i is a main machine of π, and ID is an external identity of μ wrt. π. A machine that is not a main machine is an internal machine of π.





: A is caller of B
B is subroutine of A

Execution

Given a protocol π, we add two more machines:

- the environment $\mathcal{E}$ with identity 0
- the adversary $\mathcal{A}$ with identity 1

$\mathcal{E}$ is a caller of $\mathcal{A}$ and all main machines of π. $\mathcal{A}$ can give backdoor information to all machines, and we augment all machines in π to give backdoor info. to $\mathcal{A}$.

We think of both $\mathcal{E}$ and $\mathcal{A}$ as adversarial. $\mathcal{E}$ represents interaction through inputs and outputs, and $\mathcal{A}$ represents information leakage through other means.

We assume $\mathcal{E}$ has a binary output variable (e.g. special place on its tape).

Def^n 4

An **execution** of protocol π with env. $\mathcal{E}$ and adv. A on initial input z begins by running $\mathcal{E}$ on z. Machines take turns: once $\mu = (ID, c, a)$ executes a "transmit to ID'" instruction, μ is suspended. Then:

1) If $\mu = \mathcal{E}$, the message is written to μ' with the label "input" and some external identity. If $\mu = A$, no external identity is added. Then μ' runs.

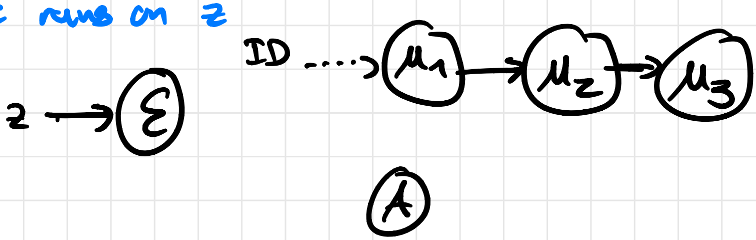
2) If $\mu \neq \mathcal{E}$ and ID' is an external identity, the message is written to $\mathcal{E}$ with identities ID and ID', and $\mathcal{E}$ resumes.

3) Otherwise, the message is written to μ' with the given label and identity ID, and μ runs.

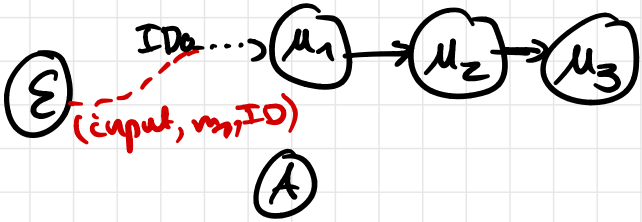
4) If any machine pauses without transmitting, $\mathcal{E}$ resumes.

Execution ends when $\mathcal{E}$ halts, with the given output. We write $EXEC_{\pi, A, \mathcal{E}}(z)$ for the random variable of the output over choices of machines run on input z.

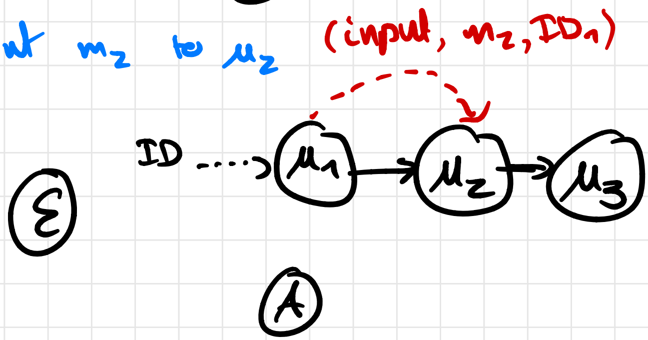
1. $\mathcal{E}$ runs on z



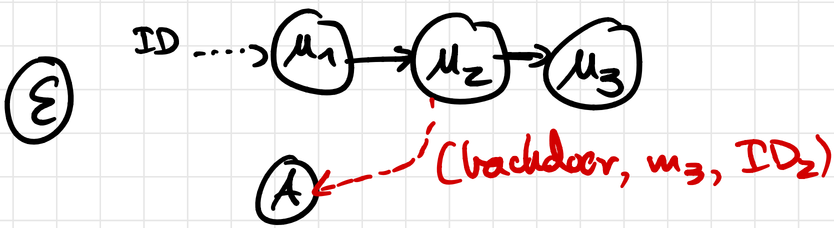
2. $\mathcal{E}$ transmits m_1 to μ_1



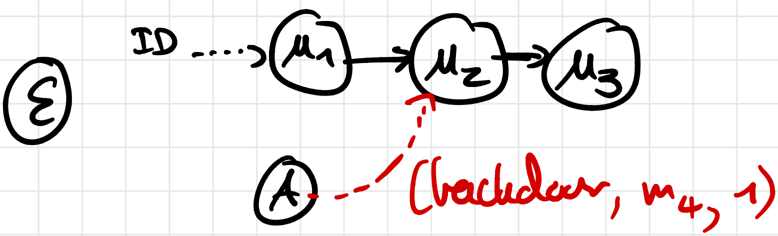
3. μ_1 gives input m_2 to μ_2



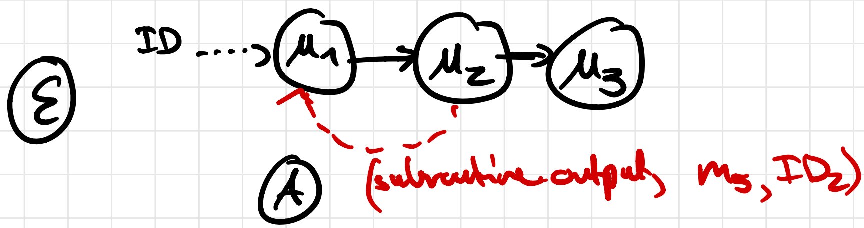
4. μ_2 transmits m_3 to $\mathcal{A}$



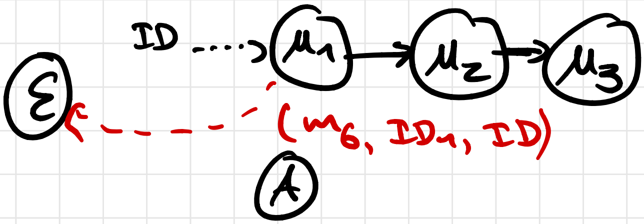
5. $\mathcal{A}$ transmits m_4 to μ_2



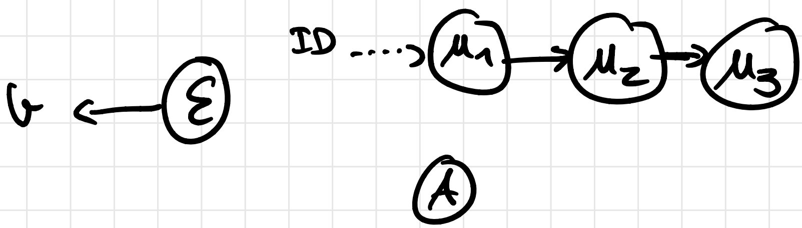
6. μ_2 gives output m_5 to μ_1



7. μ_1 gives output m_6 to $\mathcal{E}$



8. $\mathcal{E}$ halts and outputs v



Def^n 5 UC-emulation (restricted)

Protocol π UC-emulates protocol ϕ if for all polynomial-time (in the length of the input, $\lambda = |z|$) adversaries $\mathcal{A}$, there exists a polytime adversary S such that for all polytime environments $\mathcal{E}$:

$$\max_z \left\{ \left| \Pr[EXEC_{\pi,\mathcal{A},\mathcal{E}}(z)=1] - \Pr[EXEC_{\phi,S,\mathcal{E}}(z)=1] \right| \right\} = negl(\lambda)$$

Next time: the gory details...