Universally Composable Security: A Tutorial

Ran Canetti

BU, March 18-19 2016

Intro

- Goal of the event:
  - Explain the rationale and workings of the UC framework to non-cryptographers
  - Alterior motive: Extend composable analysis beyond crypto
- People's backgrounds
- Plan for the event
- Website for products?
- Practicalities: Food, facilities

Lecture plan

Session 1: Background

The UC framework – general idea

Session 2: Details of the framework

Session 3: Capturing attacks and concerns: examples

Session 4: Introduction of projects

Session 5: Work in groups

What do we want from security analysis?

- Should faithfully represent realistic attacks
- Should specify the security *concerns* and *properties* in a *meaningful* and *precise* way
- Should capture “all realistic attacks” in the expected execution environment.
- Should not be over-restrictive.
- Guarantees should remain meaningful in many (any?) environment
- Should be technically manageable

➔ Should be modular:

  – Simplify the analytic process
  – Provide more meaningful security

Advocating a general model for security analysis

Pro:

- Provides better understanding of security
- Better expressibility, analysis is more meaningful
- Enables modularity and composability
- Overall simplification of the analytical work

Con:

- Model can be complex
- Hard to “get it right”

Frameworks for modeling distributed systems

- CSP [Hoare]
- pi-calculus [Milner] spi-calculus [Abadi-Gordon]
- I/O automata [Lynch]
- …

Pros: Much analytical work, verified, support modularity, some automated analysis

Cons:
- Not easy to model computational concerns
- Modeling a bit restrictive (scheduling, addressing)

Traditional cryptographic modeling

- Semantic security
- Zero Knowledge
- Commitment
- Secure function evaluation
- …

Pro:

- Captures cryptographic security (against computationally bounded attacks)
- relatively simple

Con: Not modular, security guarantees not always meaningful in a larger context.

Want:

- The best of both…

- Be able to play on the tradeoff:

Simple/Abstract ←→ Concrete/complex

Step 1: model computer systems and attacks

Model should:

- Allow capturing:
  - Realistic systems (processors, cores, ram, disks, networks, processes, os, applications, … delays, time… randomness…)
  - Realistic attacks: network, exploits, side channels, Human
  - Information seen by different components
  - efficiency, resource bounds
- Allow different levels of abstraction/detail
- Be simple, natural, intuitive…

➔ Very tricky… the root of many deficiencies

Step 2: Capture security properties

For instance:

- Trace properties ("correctness"):   "In each execution, if event C happens then event E happens"
- Probabilistic statements
- Secrecy /privacy
- Liveness
- Timing of events
- Costs and quantitative tradeoffs
- Combinations of the above

  Eg "attack can either learn or modify, but not both"

  "attack can learn/modify, but only after a certain event"

  "success of attack is proportional to the amount of resources expended

Step 3: Prove that a system satisfies a given set of properties

Questions

- By hand? Automated? How tractable?
- Based on what assumptions?
  - Model assumptions
  - Computational hardness assumptions
- Proof re-use:
  - Modularity?
  - Robustness?

The UC approach: Specification via an Ideal-Service

The idea:

- The security of a system is reflected only in its effects on the rest of the external environment.
- Therefore to capture the desired security of system P:
  - Write an “ideal system” F that captures the desired effect
  - System P is “secure for F” if it *“looks the same”* as F to *any external environment*.

Note: F need not be efficient or even realistically implemented.
All we care about is its responses to the environment.

Specification via an Ideal-Service

Pro:

- Expressive: Can naturally express any combination of properties
- Amenable to modular analysis

Con:

- Detailed, sometimes a bit roundabout

Specification via an Ideal-Service: Zoom in

- First attempt:

  P realizes F if for all environment E, E||P ~ E||F

(reminiscent of “observational equivalenve” [Milner])

➔ Correspondence is too tight.. So too restricted…
(eg, ~ is an equivalence relation)

How to relax?

Specification via an Ideal-Service: Adding simulation

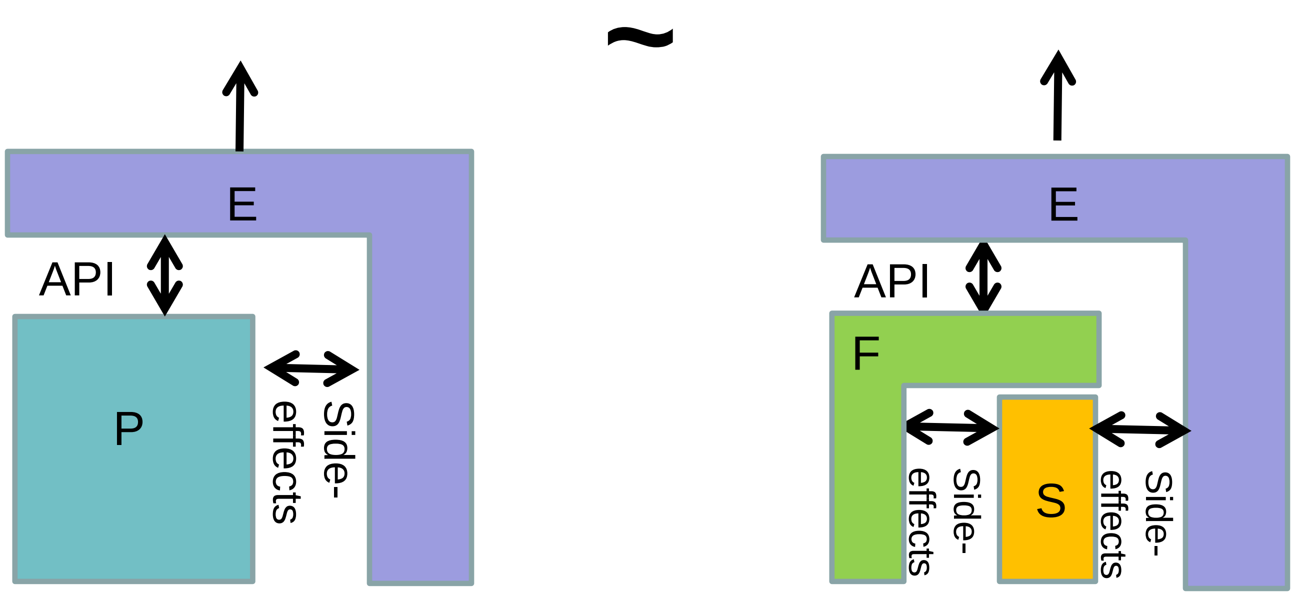
Idea:

- Split the interaction of the system P with the external world:
  - "Application Interface": the inputs from the users of P and the outputs to the users of P . (This is the "functionality" of P).
  - All the rest: consumed resources , communication, internal leakage of information, etc.
- Allow "fudging" E's view of the non-API interaction:

Def: P realizes F if there exists S such that for all E,

$$E||S||F \sim E||P$$

Recap: Simulation-based security specification



System P realizes specification F if there exists S such that for all E, $\mathrm{Exec}_{E,P} \sim \mathrm{Exec}_{E,S,F}$

($\mathrm{Exec}_{E,\ldots}$ returns the output of E from the execution …)

Specification via an Ideal-Service: Adding simulation

Def: P realizes F if there exists S such that for all E,

$$\mathrm{Exec}_{E,P} \sim \mathrm{Exec}_{E,S,F}$$

Rationale for adding S:

- “Any manipulation that P can do to E, could have done also by F (by adding S to E). Furthermore this can be done without modifying the API of F.”

Or:

- “Any manipulation that E can do to P, could have done also to F (by using S). Furthermore this can be done without modifying the API of F.”

➔ Definition is no longer symmetric

Compare with cryptographic-style simulation: Semantic Security of Encryption

Semantic security of encryption:

Want to capture "Enc(m) gives no knowledge on m"

- Game-based: An encryption algorithm Enc is sem. Secure if no (feasible) A wins w.p. >1/2+negl in game:

  $A \rightarrow m_1, m_2$

  $A \leftarrow Enc(m_b)$ $b \leftarrow \{1,2\}$

  $A \rightarrow b'$, wins if $b'=b$

- Simulation-based:

  For any A there is a simulator S such that for all m,

  $Enc(m) \sim S(|m|)$

Thm: Enc is Sim-Sem-Sec iff it is Game-Sem-Sec.

Cryptographic-style simulation: Zero-Knowledge & WI

[P,V] is an interactive protocol where P,V have joint input x, P has secret input w, (and V wants to learn whether R(x,w) for some relation R)

Want to capture "Interaction with P does not give V any knowledge on w"

- [P,V] is zero knowledge if for all V* there is a simulator S such that for all V*,x,w, [P(x,w),V*(x)] ~ S(x)

- [P,V] is "witness indistinguishable" (WI) if for all V*, x,w1,w2

  [P(x,w1),V*(x)] ~ [P(x,w2),V(x)]

Thm: ZK $\Rightarrow$ WI, but not vice versa!

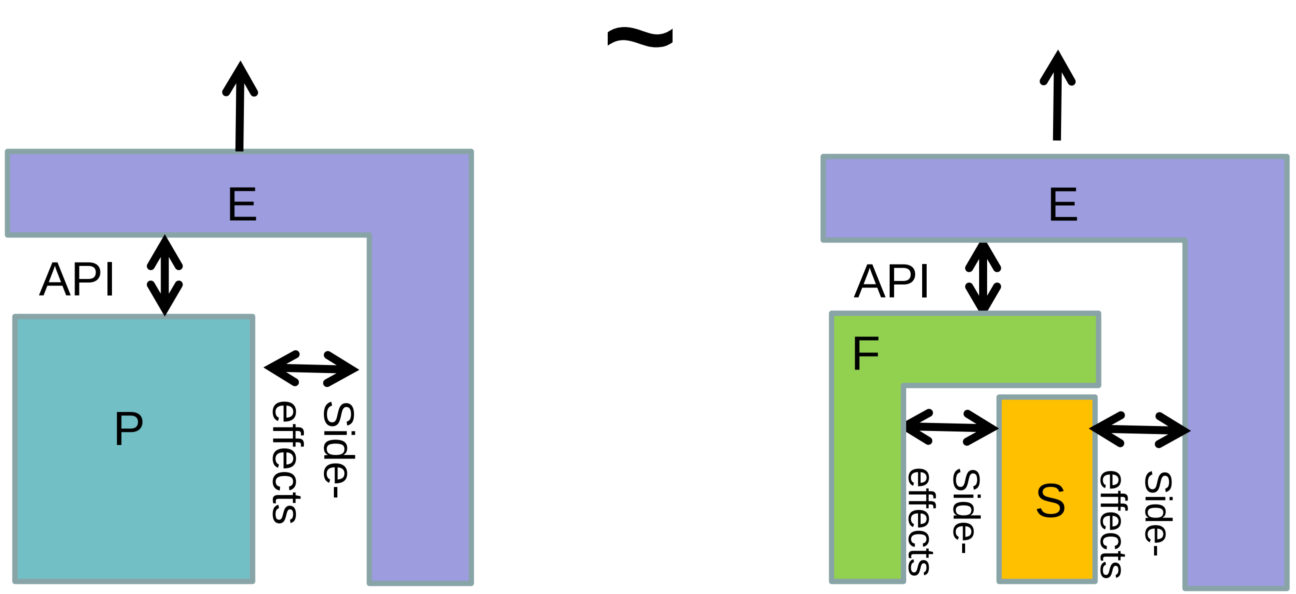
Differences from “traditional” cryptographic simulation

- Captures both secrecy and “correctness” guarantees
- Focus on the effect on the environment, rather than on protocol
- Require a single simulator (as opposed to a simulator per adversary)

Partial credits to this definitional style:

[Goldreich Micali Wigderson87,Goldwasser Levin 90,Micali Rogaway 91, Beaver 91, Canetti 92-95-00-01,Pfitzmann-Waidner 93-98-00…]

Recap: Simulation-based security specification



System P realizes specification F if there exists S such that for all E, $\text{Exec}_{E,P} \sim \text{Exec}_{E,S,F}$

Example:
Authenticated message transmission

F_{auth}:

- On input (Send,m,"B") from "A", output (Sent,m,"A") to "B".

➔ F has no side-effects, S needs to generate side-effects on its own, without knowing anything…

➔ Need to relax:

Example:
Authenticated message transmission

F_{auth}:

- On input (Send,m,"B") from "A", leak (A,B,m) to S
- When S returns "ok", output (Sent,m,"A") to "R".

→ S learns A,B,m, and *can delay delivery*

(Analysis of MAC-based protocol on board)

Example:
Secure message transmission

F_{smt}:

- On input (Send,m,"B") from "A", leak (A,B,|m|) to S
- When S returns "ok", output (Sent,m,"A") to "B".

➔S learns A,B,|m|, and *can delay delivery*

(Analysis of Enc-based protocol on board)

How to model leaky/imperfect encryption?

Example:
Zero Knowledge proofs

$F_{zk(R)}$:

- On input (Prove,x,w,B) from A, leak (A,B,x,R(x,w)) to S
- When S returns “ok”, output (Verified,A,x,R(x,w)) to B.

➔ B learns whether R(x,w)

➔ S,B learn only R(x,w), w remains secret.

Example:
Key Exchange

F_{ke}:

- On input (KE,B) from A, choose a key k and leak (A,B) to S.
- On input (KE,B) from A, leak (A,B) to S.
- When S returns (ok,P) for P={A,B}, output (A,B,k) to P.

➔ A,B obtain a fresh joint key

➔ S learns that A,B share a key.

Example:
File System with Integrity

F_{fsi}:

- On input (Init,fname,UID) record (fname,UID)
- On input (W,fname,update-contents,UID'):

  If UID'=UID then update the fname with update-contents,

  else return an error code.
- On input (read,fname,UID) leak fname to S. When S says ok, return the contents of fname to UID.

→ Write-control Integrity is guaranteed, no confidentiality guarantees.

Composition of protocols and systems

What happens to our security guarantees when the analyzed system runs alongside others?

- How do the systems interact?
  - Intentionally?
  - Adversarially?
- Do the systems have joint inputs? State? Modules?
- Do they run in parallel? concurrently?
- Does one system use the other?
- Are the systems coordinated? Same system?

What Can Go wrong?

- Protocols reuse state (eg, keying material)
- Security guarantees break due to bad interaction (ZK…)
- Security guarantees become inadequate (NM-Com)
- Security APIs don’t hold up
- …

Example: The Needham-Schroeder key exchange protocol

A	B
(knows B's public encryption key EB)	(knows A's public encryption key EA)

Choose a random k-bit N_A

$ENC_{EB}(N_A, A, B)$ $\longrightarrow$

If decryption and identity Checks are ok then Choose a random k-bit N_B and send

$ENC_{EA}(N_A, N_B, A, B)$ $\longleftarrow$

If identity and nonce checks are ok then output N_B and send

$ENC_{EB}(N_B)$ $\longrightarrow$

If nonce check is ok then Output N_B

The protocol satisfies the requirements:

- Key agreement: If A, B locally output a key with each other, then this key must be N_B. (Follows from the “untamperability” of the encryption.)

Key secrecy: The adversary only sees encryptions of the key, thus the key remains secret. (Follows from the secrecy of the encryption.)

Indeed, the protocol complies with early notions of security (e.g. [Dolev-Yao83, Bellare-Rogaway93, Datta-Derek-Mitchell-Warinschi06]).

Using the key for encrypting messages

Assume that the protocol is “composed” with an encryption protocol that uses the generated key to encrypt messages. Furthermore:
-The encryption protocol is one-time-pad
-The message is either “buy” or “sell”:

A B

$$A \xrightarrow{ENC_{EB}(N_A, A, B)} B$$

$$A \xleftarrow{ENC_{EA}(N_A, N_B, A, B)} B$$

$$A \xrightarrow{ENC_{EB}(N_B)} B$$

$$A \xrightarrow{N_B + M} B$$

An attack against the composed protocol:

A E B

$ENC_{EB}(N_A, A, B)$ →

← $ENC_{EA}(N_A, N_B, A, B)$

$ENC_{EB}(N_B)$ →

$C=N_B+M$ →

E can check whether
$C=N_B+$ “sell”, or $C=N_B+$ “buy”:
Let $C'=C+$”sell”.

$ENC_{EB}(C')$ →

Note: If M= ”sell” then $C'=(N_B+$”sell”$)+$”sell”$=N_B$. Else $C' \mathrel{!}= N_B$.
Thus, B accepts the exchange if and only if M= “sell”.

The problem: The adversary uses B as an “oracle” for whether it has the right key.

But the weakness comes to play only in conjunction with another protocol (which gives the adversary two possible candidates for the key...)

Consequently, need to explicitly incorporate the encryption protocol in the analysis of the key exchange protocol...

Want: A way to argue about the propagation of security in such situations

The methodology: Security preserving composition.

Will see:

- A (single) composition operation on systems
- Can express most other composition methods
- Preserves security

The composition operation: Universal Composition

Ingredients:

- Protocol (system) π that realizes ideal service φ
- Protocol (System) ρ that makes API calls to φ

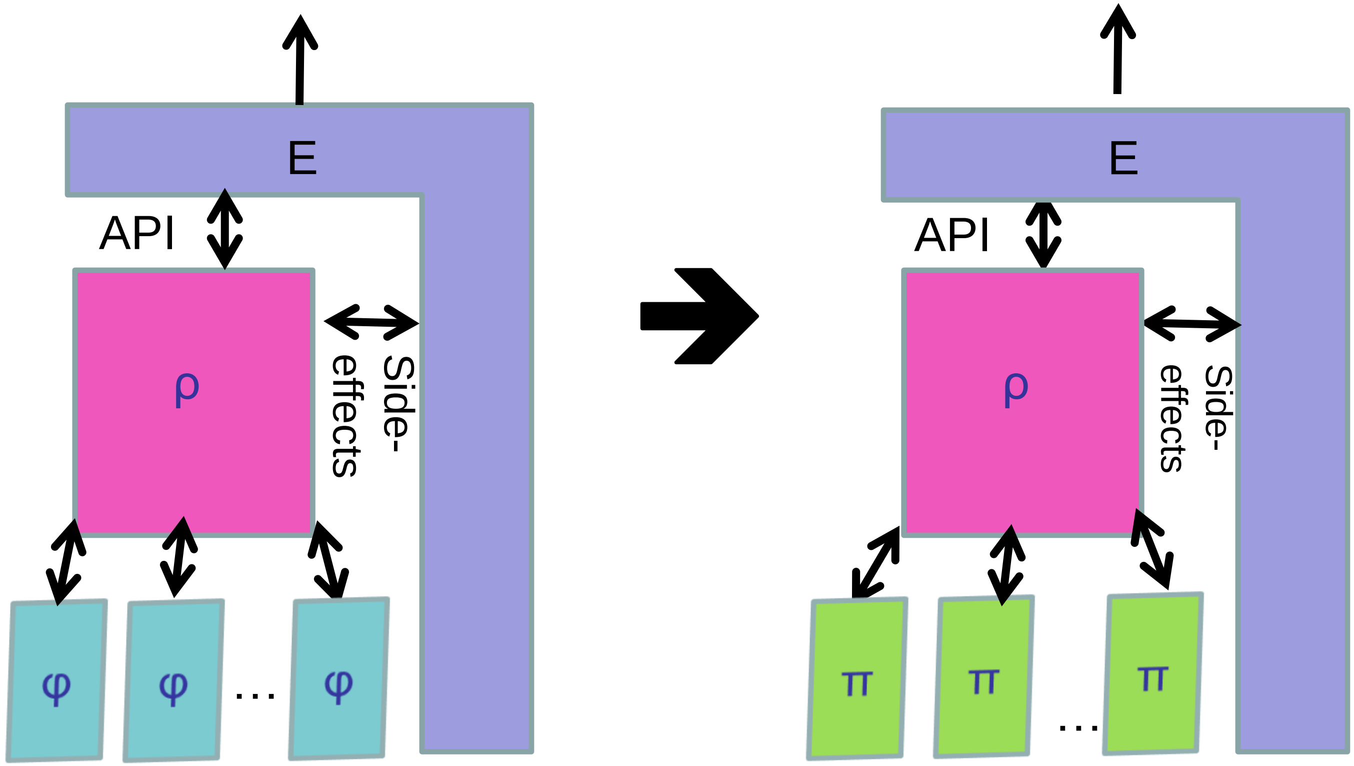
Result: A protocol $\rho^{\varphi \rightarrow \pi}$ Where:

- the calls to φ are replaced by calls to π
- Values returned from π are treated as coming from φ

Note:

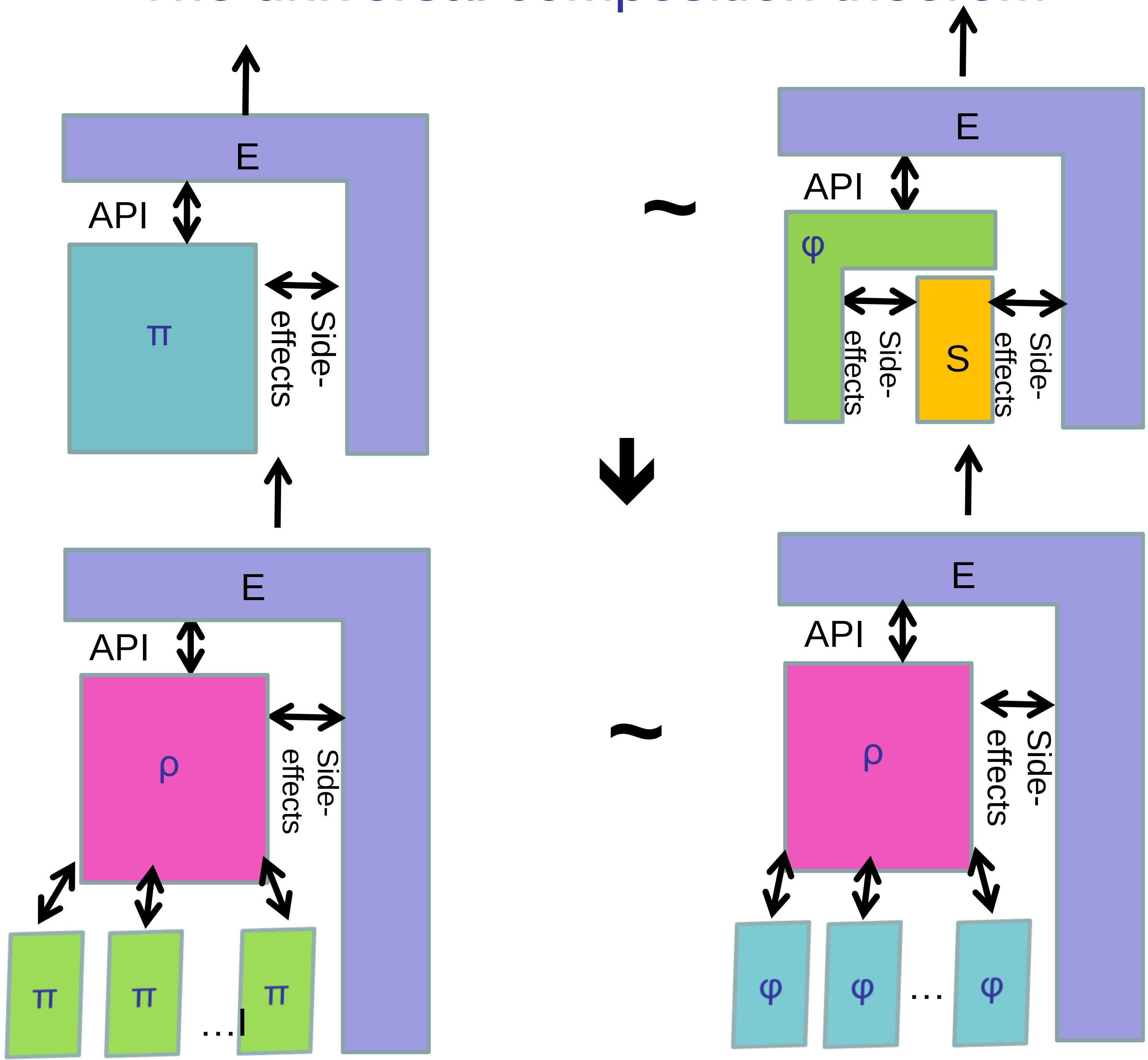
- *Just like subroutine substitution in sequential algorithms, except that each protocol/system may have many participants. (Still, calls are made locally by each participant.)*
- *There may be multiple instances of* φ and π.
- φ and π have similar API but very different “non-API” behavior (number of parties components, communication etc)

The universal composition operation



Note: each protocol can consist on many smaller components and parties.

The universal composition theorem



The composition theorem:

- If protocol π realizes ideal service φ and protocol ρ realizes Ideal Service γ, then protocol $\rho^{\varphi \to \pi}$ realizes γ.

More generally, protocol $\rho^{\varphi \to \pi}$ is “just as secure” as protocol ρ.

Corollaries:

- Allows for modular security analysis
- Allows arguing about security in arbitrary environments
- Gives concurrency “for free”…

Session 2

The actual framework

- The system model: Computing elements, scheduling, addressing, time bounds
- Model of protocol execution
- Protocol emulation, ideal services
- The composition operation and theorem

The basic computing unit: An interactive machine (IM)

- An abstract computing device
- Can model a node (cpu+RAM), a cluster of nodes, a process, an enclave,…

- Formally, an IM is a TM* with:
  - some special tapes (ports):
    - Identity tape (with code + id string, id=(pid,sid))
    - Input tape
    - Incoming communication tape
    - Incoming subroutine output tape
    - Outgoing message tape
  - An “external write” instruction (tbd)

* Can also think of an IM as a program in some higher language, e.g. Python or Java, with the appropriate data structures.

A system of IMs

- A system is a pair (I,C) where
  - I is an IM
  - C is a control function $C:\{0,1\}^* \rightarrow \{allow, disallow\}$
- An instance of an IM M is a pair $\mu=(M,id)$ where id is the contents of the identity tape
- A configuration of μ is the entire contents of tape and control
- An execution is a sequence of configurations:
  - In each config a single machien is active, initially I
  - Initial config is the initial config of I
  - The active machine runs till it performs external write. Then, the activation is suspended, the message on outgoing tape is delivered and the recipient machine is activated.
  - The execution ends when I halts. The output is output of I.

Message delivery and order of activations

The information on the outgoing message tape consists of:

- μ - ID of sending machine
- μ' =(M',id') - ID of target machine
- Tape name (input, incoming message, subroutine output)
- r ε {0,1} "reveal" bit
- m – message

Effect:

– If C(currrent execution prefix)=disallow then message is not delivered and I is activated.

– Else:

- If no MI with identity id' exists in the execution prefix then one is created and initialized with code M'. (unless M'=@, in which case I gets activated)
- If M'=@ or M' is the code of the MI with id' then the message is written to the appropriate tape of that MI
- Else (Mi with id' exists but with code different than M') then μ transitions to an error state and I is activated next.

Notes

- Number of MI’s is unbounded
- Allows dynamic code generation
- ID of each MI is unique in the system
- Need to know ID of an MI in order to send to it
- Mis know their IDs
- Scheduling is sequential and unfair…

More definitions

- *Extended systems:*

  C(exec prefix) = new ID and code for source and target

- μ is a *subroutine* of μ’ if μ wrote to the subroutine output tape for μ’ or μ’ wrote to the input tape of μ.

- A *protocol* is an interactive machine.

- An *instance* of protocol P in an execution prefix is a set of MI’s with program P and the same SID

Polynomial time

- A machine M is polynomial time if its runtime is bounded by a polynomial in N, where

  N = # bits written on M’s input tap –

  # bits that M wrote on input tapes of other machines.

- Can see: If all machines in a system of ITMS are polynomial (and are all bounded by the same polynomial) then the system halts within polynomial time.

- Parameterized systems: All machines get inputs of at least some polynomial length.

The model of protocol execution

On board

- The idea: Keep it simple. Run a single instance of the protocol, with an environment and an adversary.

- Participants: Environment E, adversary A, parties of protocol π.

  – E starts, invokes A, gives inputs to parties of a single instance of π, obtains outputs from parties of π.
  – Parties of π generate subroutines, outputs, and send messages to each other *via the adversary*.
  – Adversary obtains messages from parties, and either delivers some (arbitrarily modified) incoming messages to parties, or generates output for E.

(Rules are enforced by the control function.)

The model of protocol execution

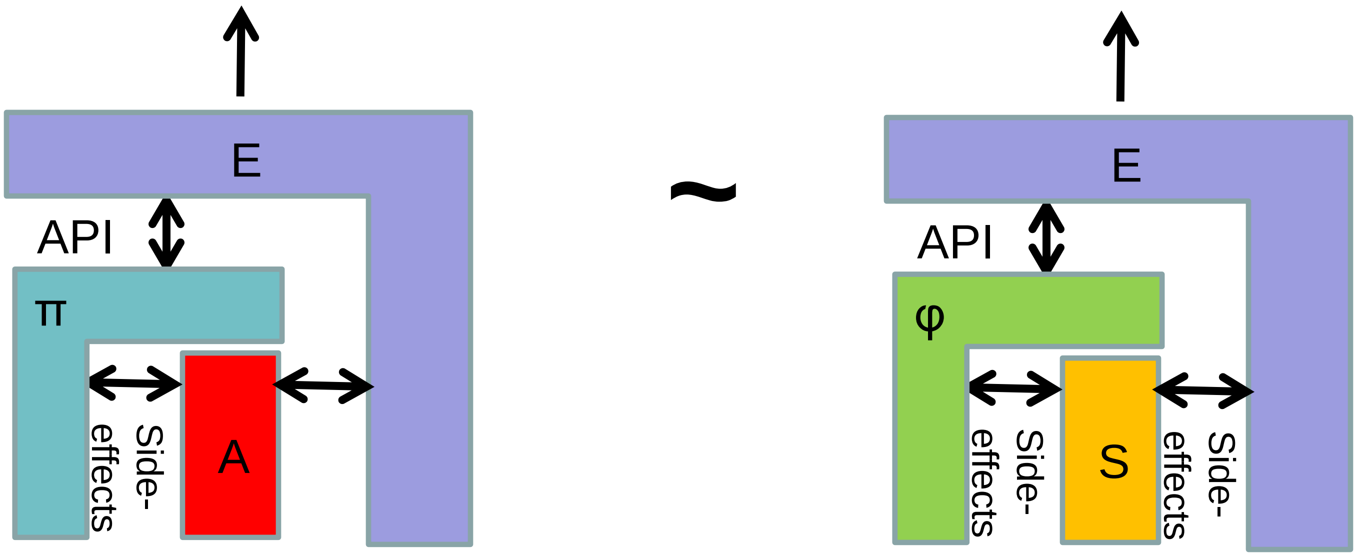
Note: The model is very rudimentary:

- Parties communicate only via the adversary
- No “party corruption” operations

➔Done for sake of simplicity and generality. Will add later.

(Note: Control F erases ID of E is from inputs to parties, and erases code of parties from outputs to E)

Protocol emulation



Protocol π *UC-emulates* protocol φ if for all A there exists S such that for all E*, $\mathrm{Exec}_{E,A,\pi} \sim \mathrm{Exec}_{E,S,\varphi}$.

($\mathrm{Exec}_{E,\ldots}$ returns the output of E from the execution …)

* Quantify only over "balanced" environments

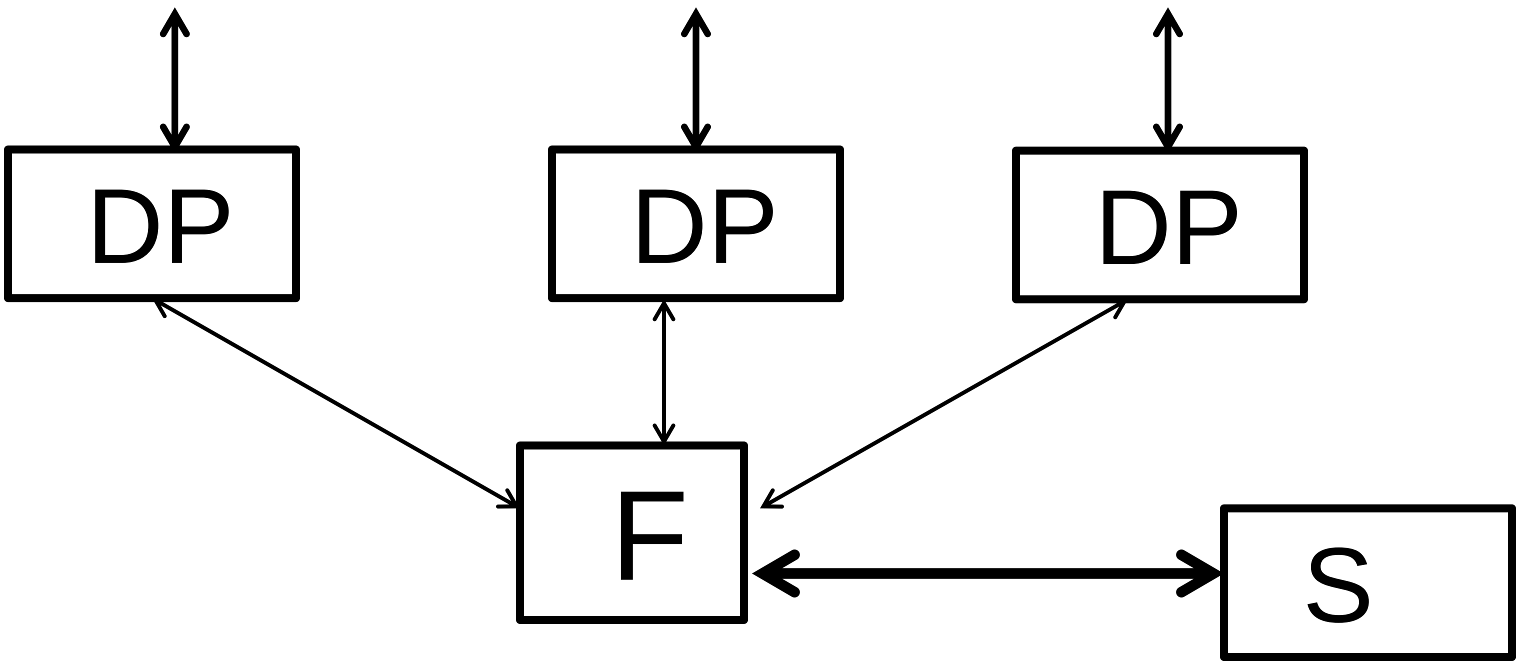
Notes

- Security with “dummy adversary”: Above def is equivalent to one where A is only a channel for E.

- Emulation is transitive: If protocol A UC-emulates protocol B and B UC-emulates protocol C then A UC-emulates C.

- Quantitative formulations: Measure the complexity overhead of S vs A, and the probability of distinguishing.

$IDEAL_F$: The ideal protocol for ideal service F



Dummy parties: Only transfer message from E to parties and back
Communication between A and F is critical!

P UC-realizes F if P UC-emulated $IDEAL_F$

Universal composition

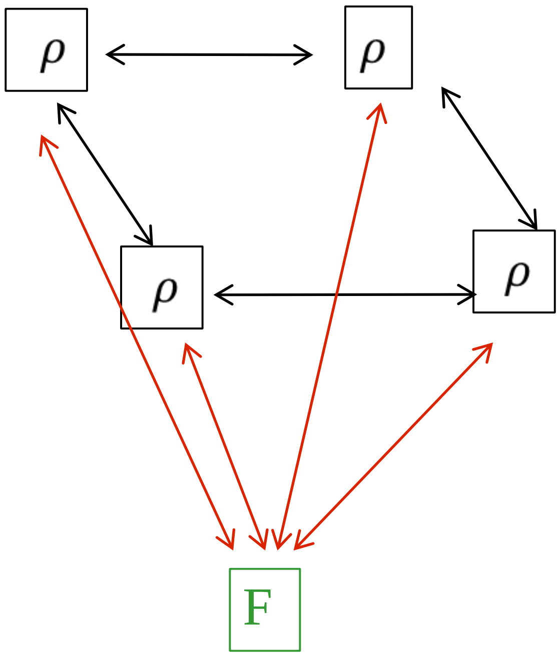
Ingredients:

- Protocol π that emulates protocol φ
- Protocol ρ that makes subroutine calls to φ

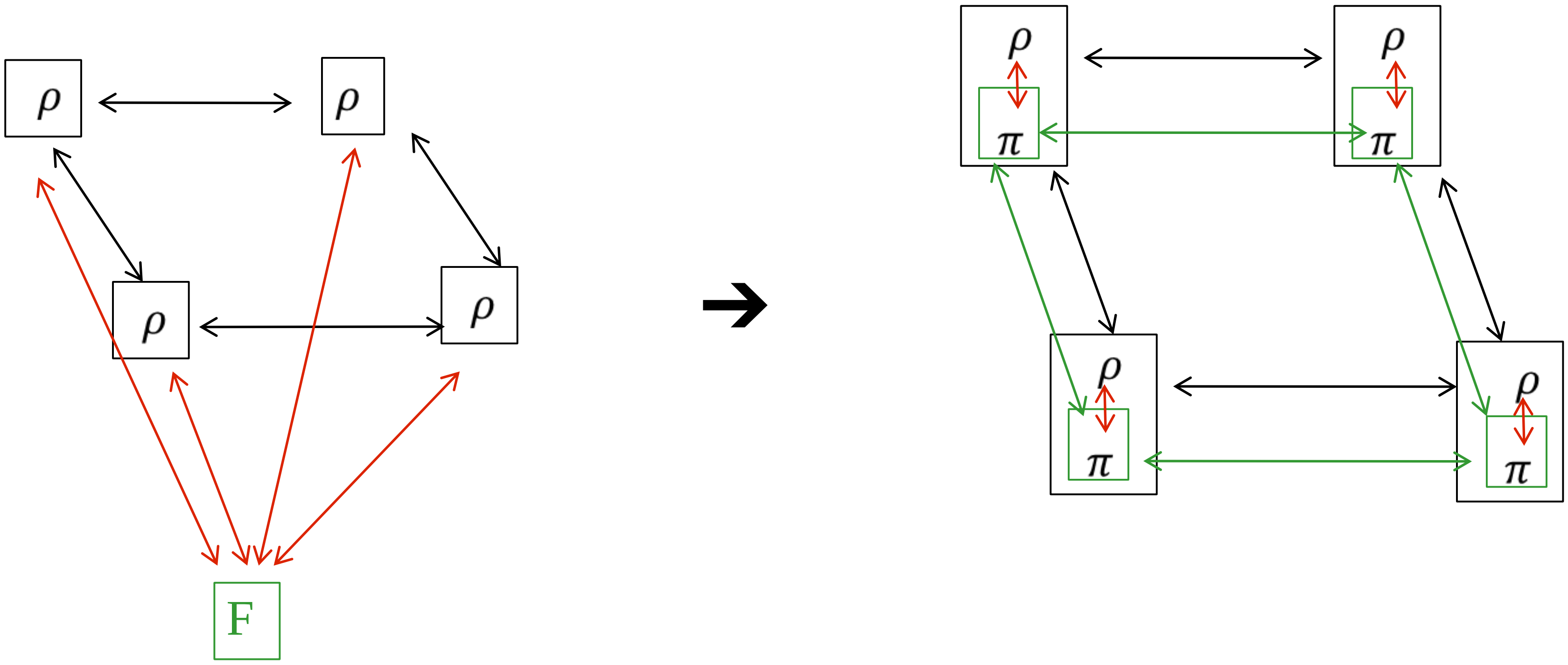
Result: A protocol $\rho^{\varphi \rightarrow \pi}$ Where:

- The calls to each MI of φ are replaced by calls to a MI of π with same id
- Values returned from each MI of π are treated as coming from a MI of φ with same id

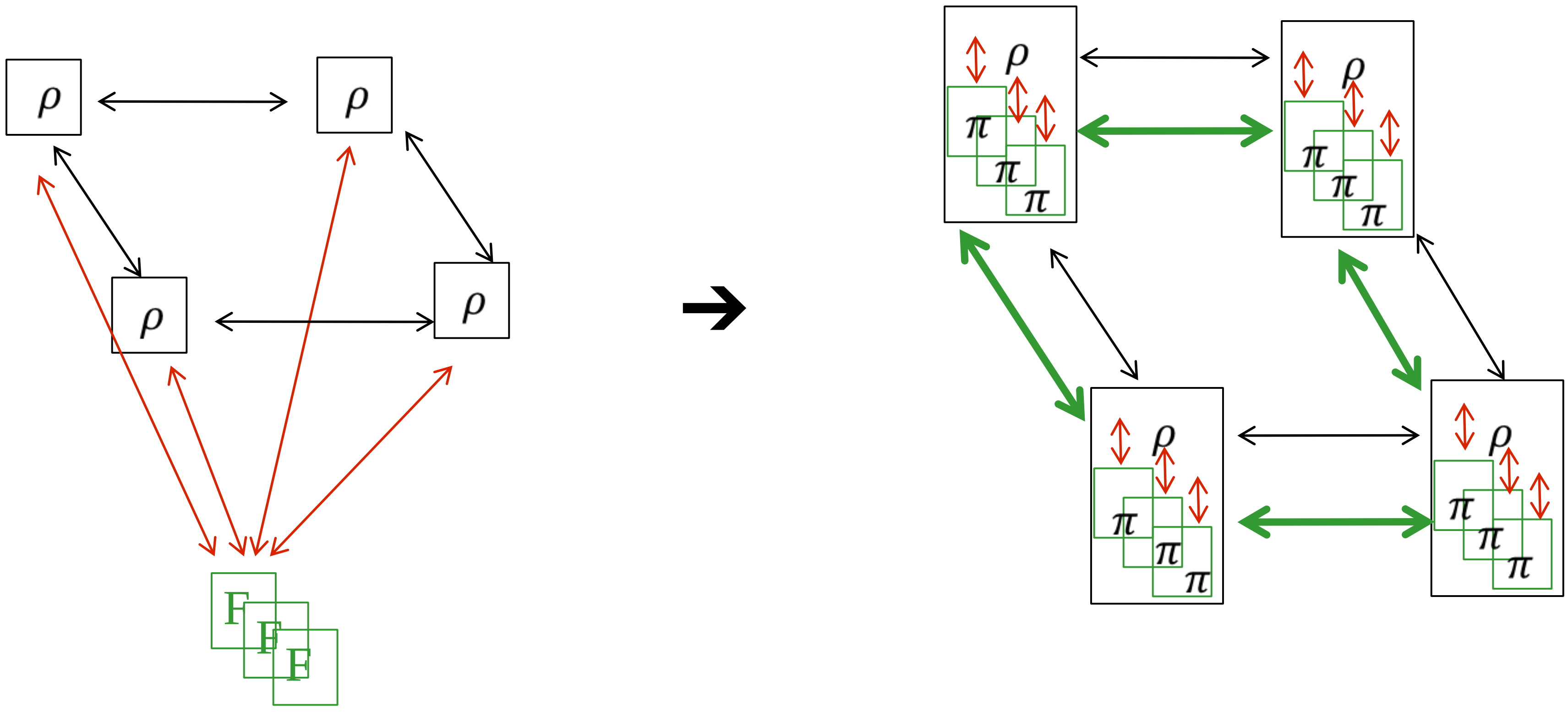
The composition operation (single call to F)



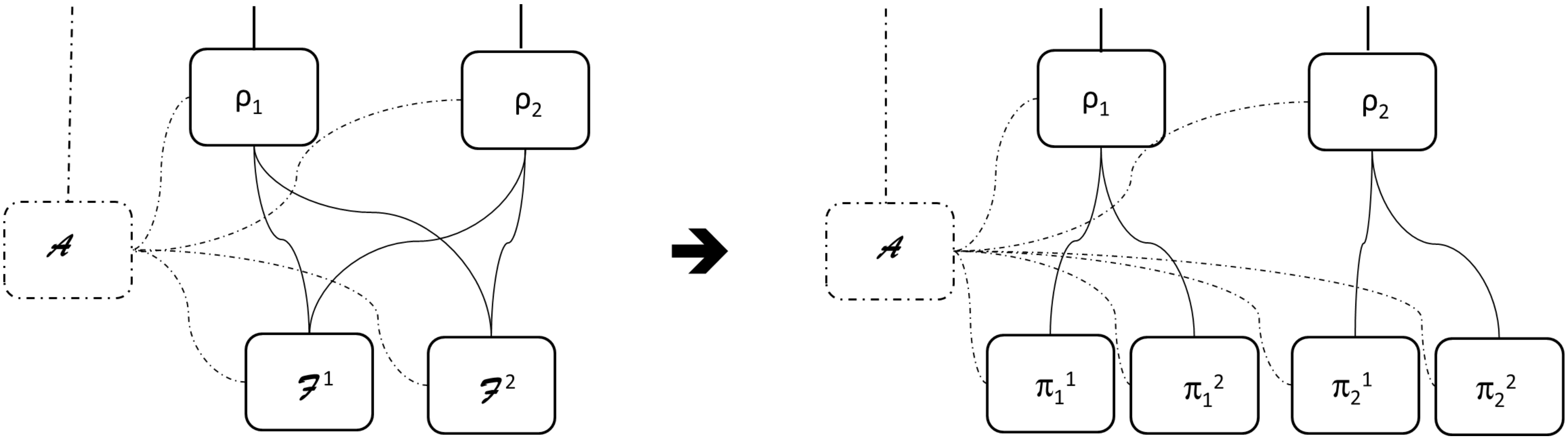
The composition operation (single call to F)



The composition operation (multiple calls to F)



The composition operation (multiple calls to F)



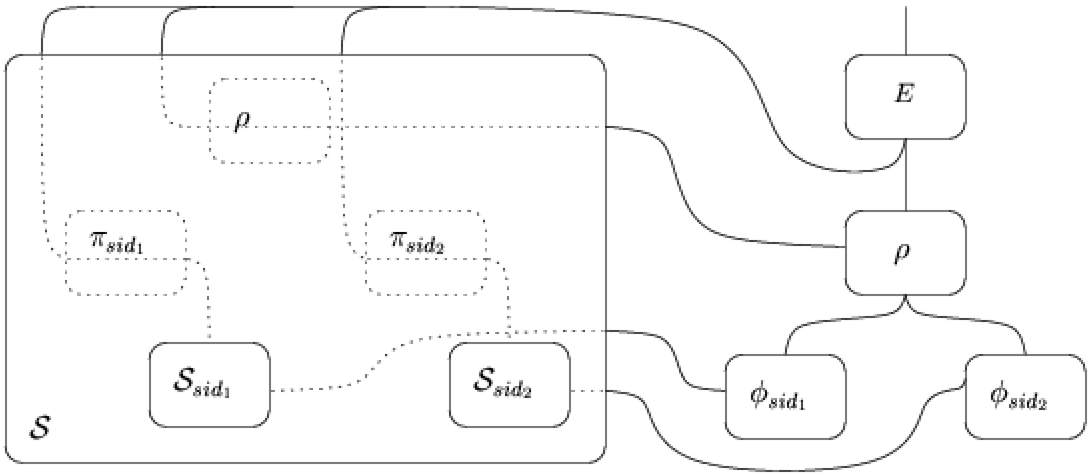
The composition theorem:

If protocol π UC-emulates protocol φ,

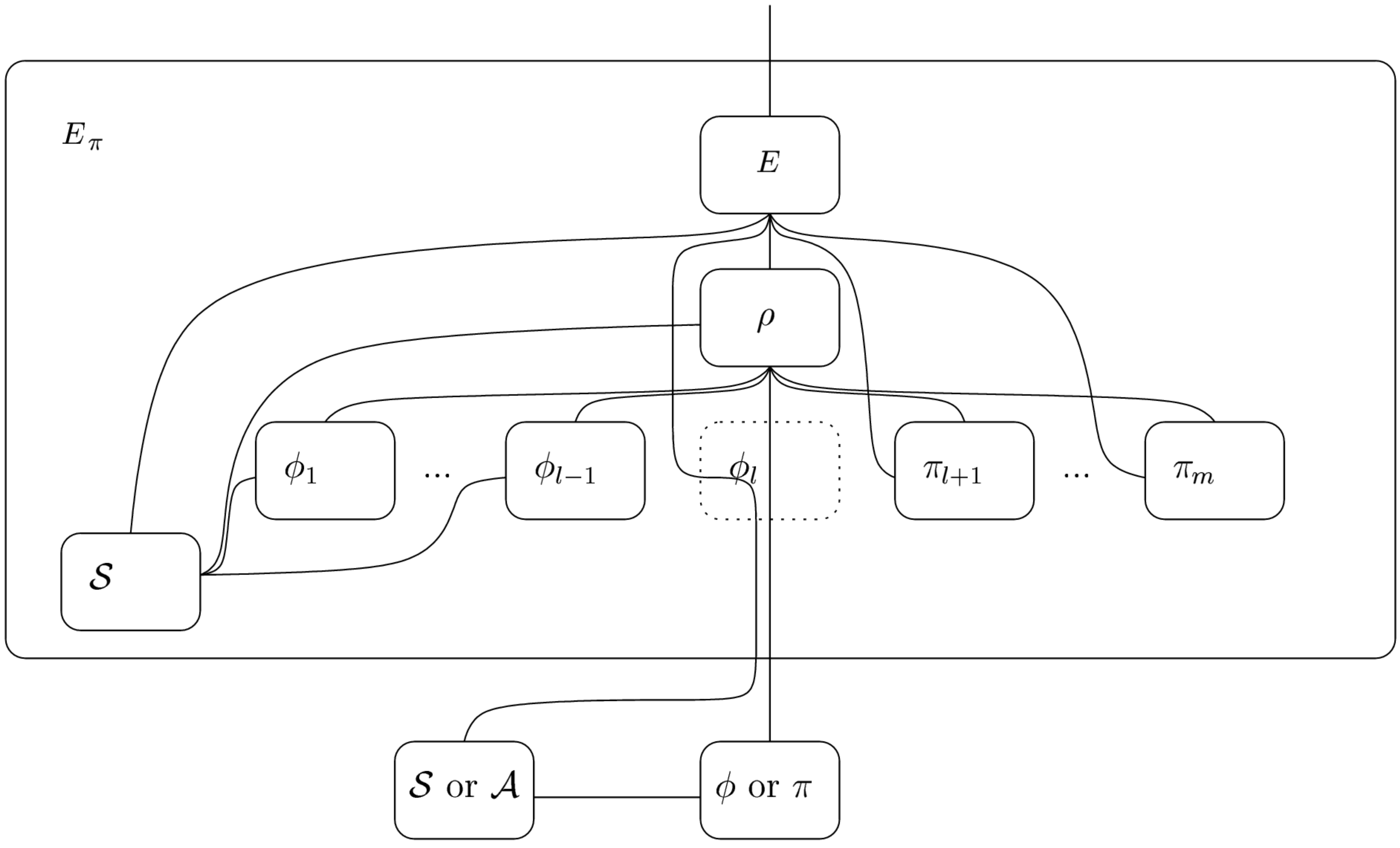
then protocol $\rho^{\varphi \rightarrow \pi}$ UC-emulates protocol ρ.

Proof outline: the combined simulator

On board



The distinguishing Environment



Modeling corruptions

- The shell construct: Allows encoding modeling instructions in a protocol
- Notification to Environment
- Types of corruption:
  - Byzantine
  - Honest-but-curious
  - Fail-stop
  - Transient
  - Leakage
  - Erasures
- Modeling aggregate information

synchronous communication

- On board

Key exchange & secure channels

- Auth + Enc
- KE
- Sig
- Cert
- PKI

- How to multiplex a single PKI over many sessions?

➔ Use JUC

De-composing “entangled systems”

Q: How can we de-compose systems to “independent components” even when the components have joint modules?

A: Can be done when the joint modules “behave like multiple independent instances” of a simpler module.

Example: Key exchange authenticated using PKI

Multi-session extensions

$\hat{\rho}$ is the *multi-session extension* of ρ if it "behaves like multiple independent sessions of ρ". That is:

- $\hat{\rho}$ runs multiple independent sessions of ρ, each with its own sub-session-id (ssid).
- Upon receiving message m=(s,m'), $\hat{\rho}$ activates session s of ρ with input m'.
- When session s of ρ wishes to send message m', $\hat{\rho}$ sends (s,m') to specified recipient.

Universal composition with Joint State (JUC)

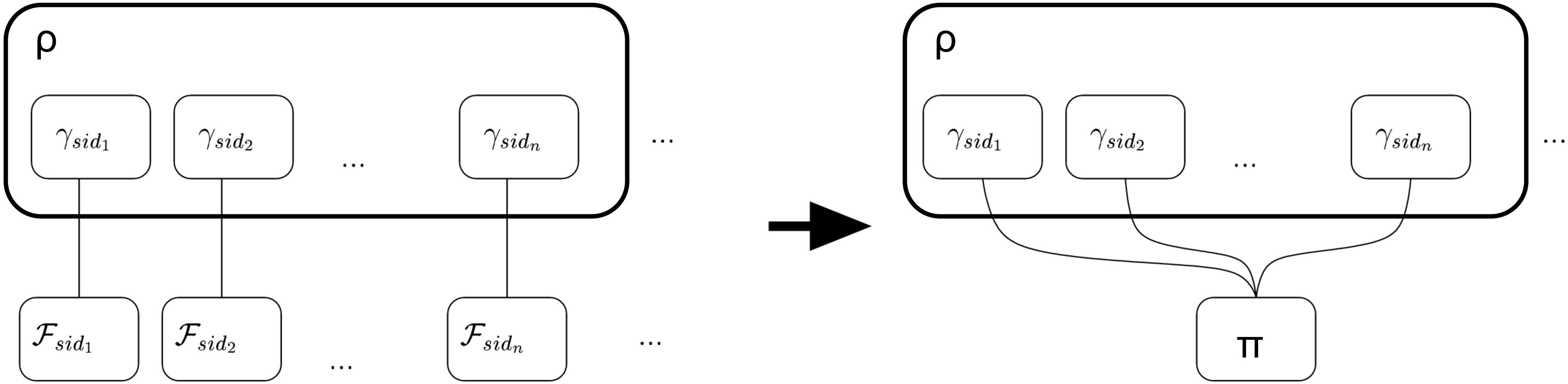
Ingredients:

- Protocol π that emulates protocol $\hat{\varphi}$
- Protocol ρ that makes subroutine calls to φ

Result: A protocol $\rho^{\hat{\varphi} \rightarrow \pi}$ Where:

- The calls to (the single instance of) $\hat{\varphi}$ are replaced by calls to (multiple instances of) π with:
  - same pid
  - ssid turns into sid
- Values returned from each MI of π are treated as coming from the MI of $\hat{\varphi}$ with same pid, sid turns to ssid.

Universal composition with Joint State



JUC Theorem [C-Rabin03]:

If protocol π UC-emulates protocol $\hat{\varphi}$,
then protocol $\rho^{\hat{\varphi} \rightarrow \pi}$ UC-emulates protocol ρ.

Authenticated filesystem

- On board