Security and Composition of Cryptographic Protocols: A tutorial

Ran Canetti

Tel Aviv University

Cryptographic protocol problems

Two or more parties want to perform some joint computation, while guaranteeing "security" against "adversarial behavior".

Some security concerns

- Correctness of local outputs:
  - As a function of all inputs
  - Distributional and bias guarantees
  - Unpredictability
- Secrecy of local data and inputs
- Privacy
- Fairness
- Accountability
- Availability

Cryptographic protocol problems:

Secure Communication

Two (or more) parties want to communicate “securely”:

- Authentication: Recipient will accept only messages that were sent by the specified sender.
- Secrecy: Contents of messages should remain unknown to third parties.

Related tasks:

- Key-exchange: The parties agree on a random key that remains secret to eavesdroppers.
- Encryption (shared key, public key)
- Digital signatures (shared key, public key)

Cryptographic protocol problems:

Two-party tasks

- Coin tossing [Blum 82]

  Generate a common uniformly distributed bit (or string).

  The output should be “unbiased” and “unpredictable”.

- Zero-Knowledge [Goldwasser-Micali-Rackoff 85]

  P proves to V that $x \in L$ “without revealing extra info”.

- Commitment [Blum 82]

  Two stage process:

  - C gives x to V “in an envelope”

    (i.e., x is fixed but remains unknown to V).

  - C enables V to “open the envelope” (i.e., retrieve x).

(Here the parties typically dont trust each other.)

Cryptographic protocol problems:

Multiparty tasks and applications

- Electronic voting:

  Correctness, acountability, privacy, coersion-freeness...
- "E-commerce": *Fairness, accountability*
  - On-line Auctions, trading and financial markets, shopping
- On-line gambling: *Unpredictability, accountability...*
- Computations on databases: *Privacy*
  - Private information retrieval
  - Database pooling
- Secure distributed storage: *Availability, integrity, secrecy*
- Cloud computing: *integrity, secrecy, correctness*

But, what does “security” really mean?

Rigorously capturing the intuitive security concerns is a tricky business…

Main stumbling points:

- Unexpected inter-dependence between security requirements
- Unexpected bad interactions between different protocol instances in a system

→ Security is very sensitive to the execution environment.

Developing notions of security

- Initially, notions of security were problem-specific (e.g., Encryption, Coin-tossing, Zero-Knowledge, Signatures...)
- Subsequently, general frameworks for caprturing security of tasks were developed, e.g. [Goldwasser-Levin90,Micali-Rogaway91,Beaver91,C95,C00, Dodis-Micali00, Pfitzmann-Waidner 00&01, C01, Mitchell-Scedrov-Ramanathan-Teague01, Mateus-Mitchell-Scedrov03, Kuesters06]
  - All of these frameworks follow (in one way or another) a single paradigm: "The trusted party paradigm".

In favor of a general notion of security

- Provides better understanding of security concerns, various primitives, and the relations among them.
- Provides a basis for making claims about behavior of protocols in unknown environments:
  - "A protocol that realizes a task can be used in conjunction with any protocol that uses the task"
  - "Protocols that realize this task continue to realize it in any execution environment"

Today:

Part 1: Basic security

- Motivate and present the “Trusted party paradigm”.
- Review a basic formalization of the approach (based on [C, JoC 00]). See examples. Discuss feasibility.

Part 2: Security and composition

- Discuss secure protocol composition:
  - Show what can go wrong
  - Discuss settings and requirements
- Demonstrate the limited compositional properties of the basic definition

Part 3: Universally Composable security

- Present the notion (based on [C, FOCS'01])
- Demonstrate secure composability properties
- Discuss feasibility, possible relaxations.
- Explore connections to “formal analysis” of protocols.

Defining security: First attempts

Let's start with a simple setting:

- Two parties
- Function evaluation:
  - There is a known function f
  - Party P_i (i=1,2) has input x_i
  - Both parties wish to “securely” obtain $y=f(x_1,x_2)$.
- The only potentially adversarial entities are the parties themselves.

What are the security requirements?

What are the security requirements?

- Correctness: The honest party (parties) output a correct function value of the inputs of the parties.

- Secrecy: Each party learns only the function value. ("The view of each party can be generated given only its input and output.")

What are the security requirements?

- Correctness: The honest party (parties) output a correct function value of the inputs of the parties.

- Secrecy: Each party learns only the function value. ("The view of each party can be generated given only its input and output.")

Are we done?

But...

The “function value” depends on the inputs provided by the parties. How to define these inputs?

- Given from above?

  Unrealizable...

- Chosen during the run of the protocol? Dangerous...

Example

Function: $f(x_1,x_2) = x_1$ xor x_2

Protocol:

- P_1 sends x_1 to P_2.
- P_2 sends x_2 to P_1.
- Both parties output x_1 xor x_2.

Is the protocol secure?

- P_2 can decide the function value as a function of x_1.
- But: the protocol satisfies:
  - Correctness: The parties output the correct function of the inputs
  - Secrecy: trivial, since x_1 is computable from x_2 and x_1 xor x_2.

Conclusions:

- The definition should also limit the way the inputs to the computation are chosen.
- Secrecy and correctness are “weaved together”:
  - Correctness requires some flavor of secrecy
  - Secrecy depdends on the correctness

Conclusions:

- The definition should also limit the way the inputs to the computation are chosen.
- Secrecy and correctness are “weaved together”:
  - Correctness requires some flavor of secrecy
  - Secrecy depdends on the correctness

What about the case where parties are guaranteed to follow the protocol? Here the inputs are well defined and correctness seems straighforward. Is the definition adequate there?

Another example

Function: $f(x_1,x_2) = r \in_R \{0,1\}^k$

Protocol:

- Let f be a one-way permutation on $\{0,1\}^k$.
- P_1 chooses $s \in_R \{0,1\}^k$, and sends $r=f(s)$ to P_2.
- Both parties output r.

Is the protocol secure?

- P_1 knows a “trapdoor information” on r.
- P_2 cannot feasibly compute this information.

(Why is this bad?)

- But: the protocol satisfies:
  - Correctness: The output is distributed uniformly in $\{0,1\}^k$.
  - Secrecy: Trivial, there are no inputs.

Conclusion:

The definition should also specify the process by which the outputs are chosen, not just the distribution.

The general definitional approach

[Goldreich-Micali-Wigderson87]

'A protocol is secure for some task if it "emulates" an "ideal process" where the parties hand their inputs to a "trusted party", who locally computes the desired outputs and hands them back to the parties.'

But, how to formalize?

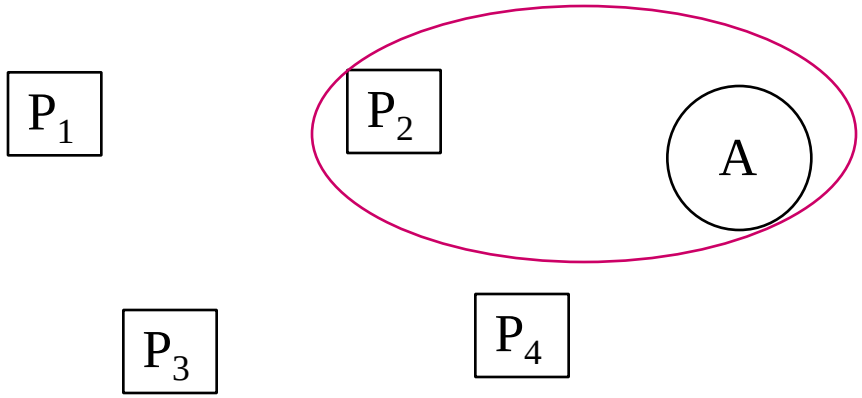
A basic formalization

- Formalize the process of protocol execution in presence of an adversary
- Formalize the “ideal process” for realizing the functionality
- Formalize the notion of “a protocol emulates the ideal process for a functionality.”

The model for protocol execution:

Participants:

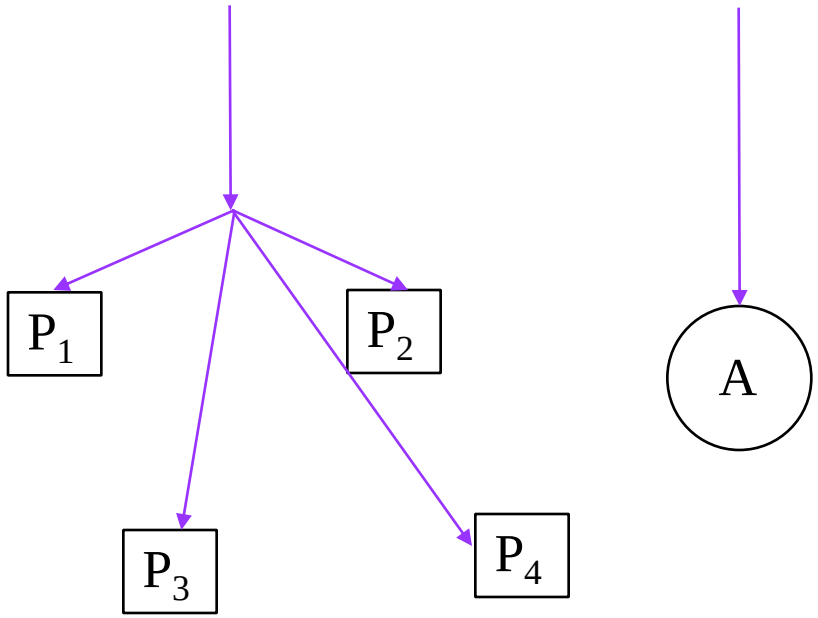
- Parties $P_1 \ldots P_n$
- Adversary A, controlling the corrupted parties.



The model for protocol execution:

Participants:

- Parties $P_1 \ldots P_n$
- Adversary A, controlling the corrupted parties.

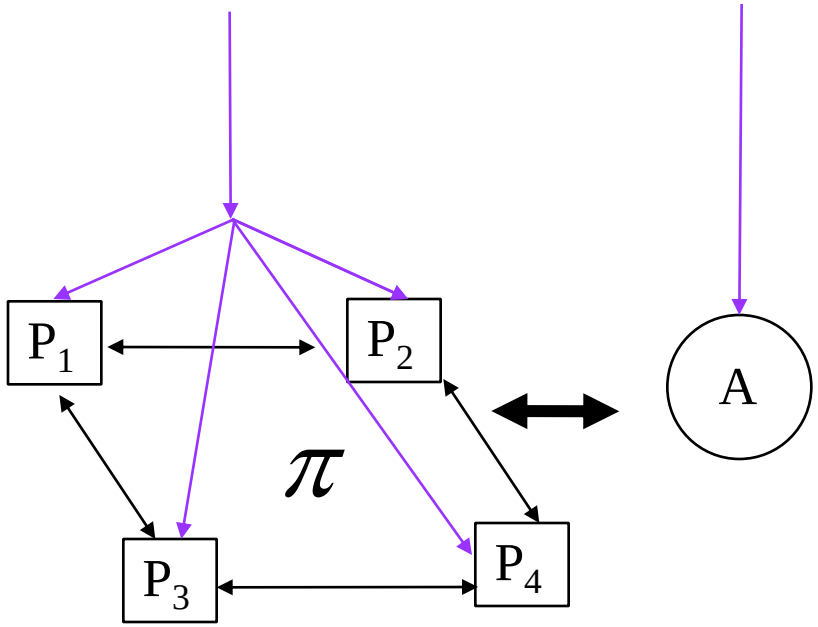


- The parties and the adversary get external input.

The model for protocol execution:

Participants:

- Parties $P_1 \ldots P_n$
- Adversary A, controlling the corrupted parties.



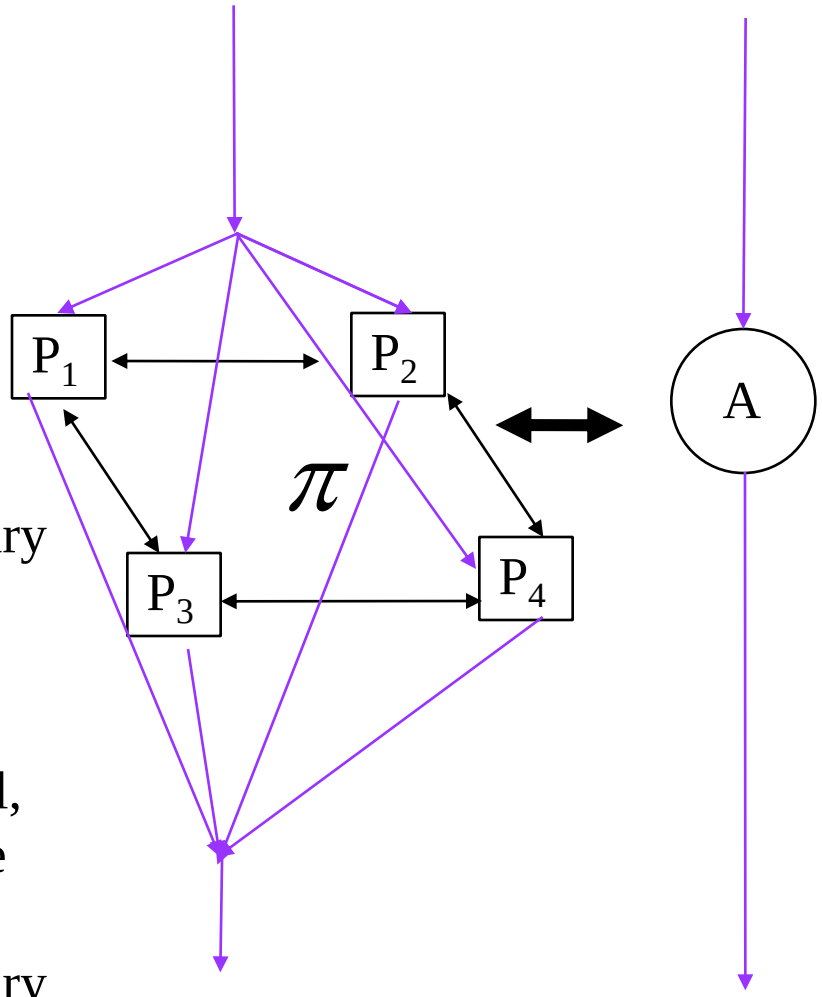
- The parties and the adversary get external input.

- The parties and adversary interact (parties running the protocol, A interferes according to the model.)

The model for protocol execution:

Participants:

- Parties $P_1 \ldots P_n$
- Adversary A, controlling the corrupted parties.

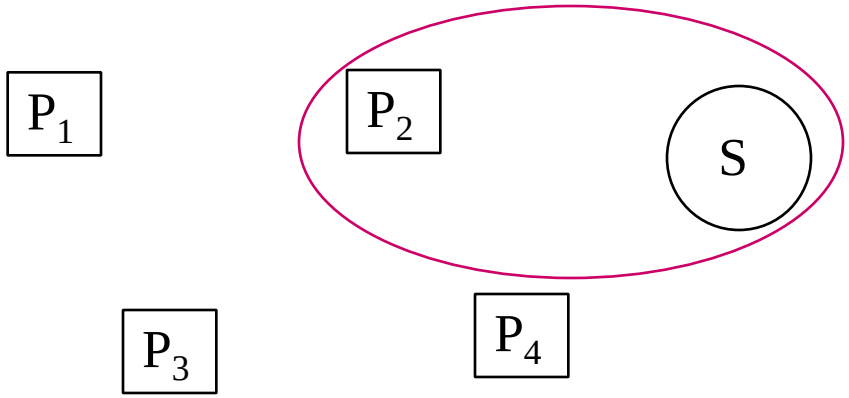


- The parties and the adversary get external input.
- The parties and adversary interact (parties running the protocol, A interferes according to the model.)
- The parites and the adversary generate their local outputs.

The ideal process (for evaluating function F):

Participants:

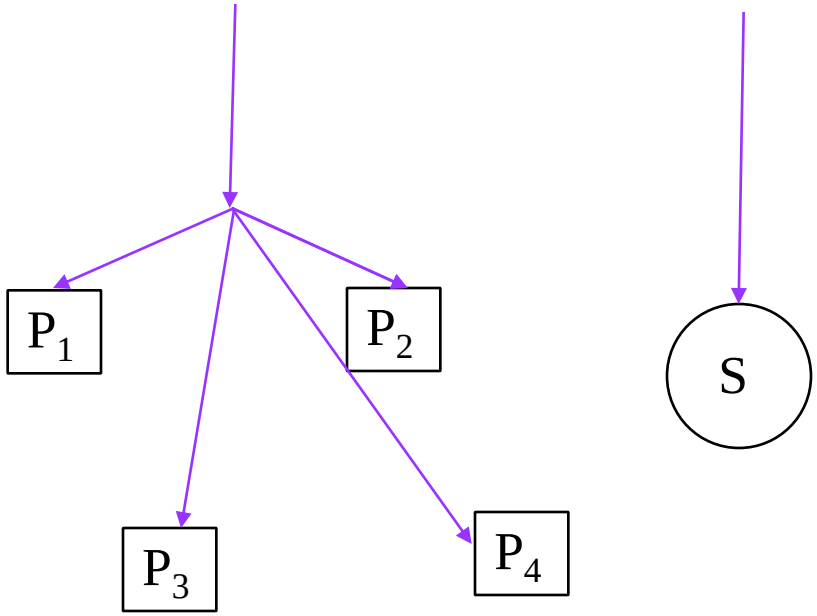
- Parties $P_1 \ldots P_n$
- Adversary S, controlling the corrupted parties.



The ideal process (for evaluating function F):

Participants:

- Parties $P_1 \ldots P_n$
- Adversary S, controlling the corrupted parties.

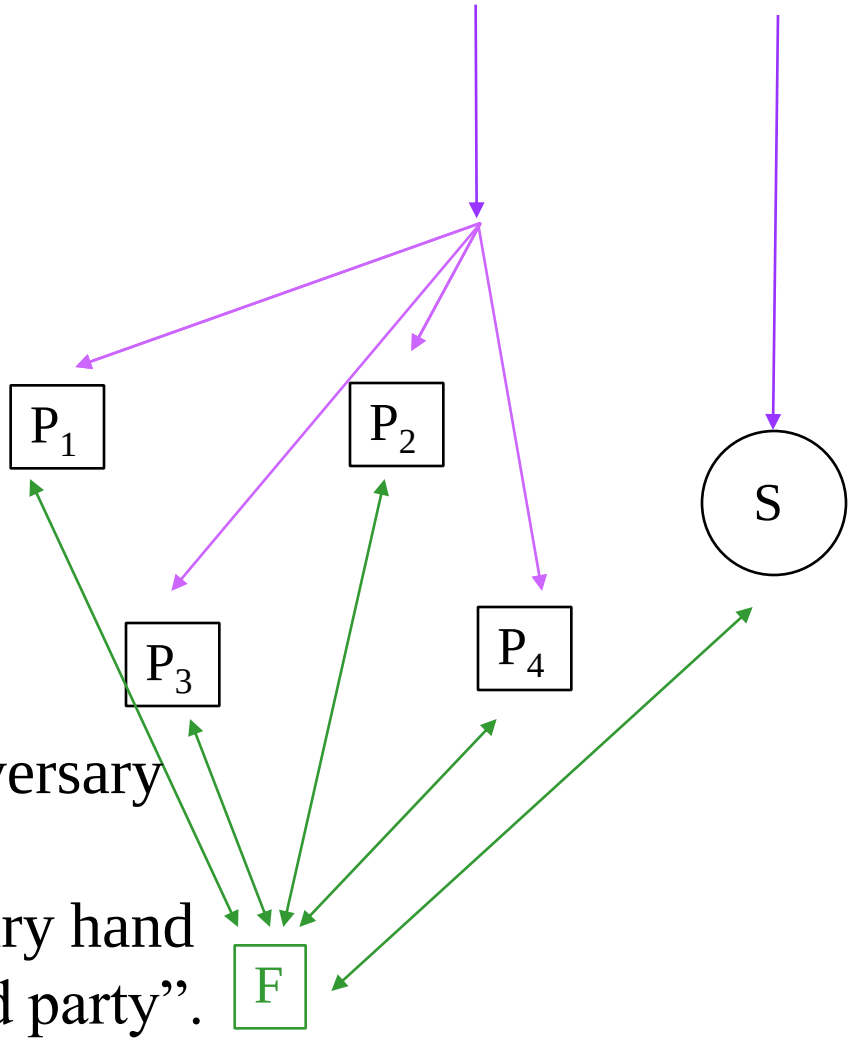


- The parties and the adversary get external input.

The ideal process (for evaluating function F):

Participants:

- Parties $P_1 \ldots P_n$
- Adversary S, controlling the corrupted parties.

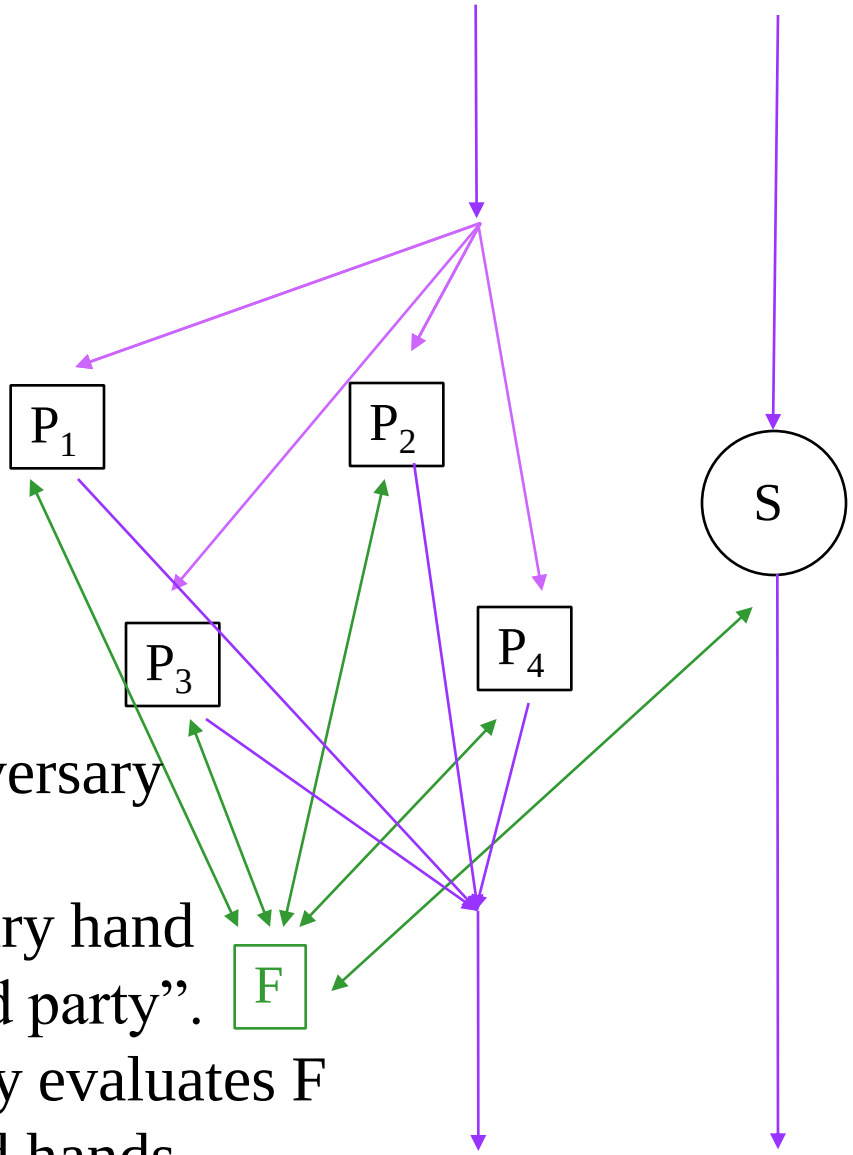


- The parties and the adversary get external input.
- The parties and adversary hand Their inputs to a "trusted party".
The trusted party locally evaluates F on the parties' inputs and hands each party Its prescribed output.

The ideal process (for evaluating function F):

Participants:

- Parties $P_1 \ldots P_n$
- Adversary S, controlling the corrupted parties.



- The parties and the adversary get external input.
- The parties and adversary hand Their inputs to a "trusted party".
- The trusted party locally evaluates F on the parties' inputs and hands each party Its prescribed output.

Definition of security

A protocol π *emulates* the ideal process for evaluating F if for any (PPT) adversary A there exists a (PPT) adversary S such that for any set of external inputs:

- The outputs of the uncorr. parties running π with A ~ The outputs of the uncorr. parties in the ideal process with S
- The output of A ~ The output of S

In this case, we say that π *securely realizes* functionality F.

Implications of the definition

Correctness: In the ideal process the parties get the "correct" outputs, based on the inputs of all parties. Consequently, the same must happen in the protocol execution (or else the first condition will be violated).

Secrecy: In the ideal process the adversary learns nothing other than the outputs of bad parties. Consequently, the same must happen in the protocol execution (or else the second condition will be violated).

"Input independence:" The bad parties cannot choose their inputs based on the inputs of the honest parties (since they cannot do so in the ideal process).

…

Example:
The millionaires functionality

1. Receive (x_1) from party P_1
2. Receive (x_2) from party P_2
3. Set b = ($x_1 > x_2$). output b to both parties.

Note:

- Both parties are assured that they receive the correct bit
- Neither party learns any information other than the bit b.

Example:
The xor function

1. Receive (x_1) from party P_1
2. Receive (x_2) from party P_2
3. Set b = (x_1 xor x_2). output b to both parties.

Example:
The xor function

1. Receive (x_1) from party P_1
2. Receive (x_2) from party P_2
3. Set b = (x_1 xor x_2). output b to both parties.

What about the above "bad protocol"?

Reminder: The bad protocol

- P_1 sends x_1 to P_2.
- P_2 sends x_2 to P_1.
- Both parties output x_1 xor x_2.

Example: The xor function

1. Receive (x_1) from party P_1
2. Receive (x_2) from party P_2
3. Set b = (x_1 xor x_2). output b to both parties.

The above "bad protocol" is no longer secure. Assume x_1 is random:

• In the protocol execution A (controlling P_2) can always force the output of P_1 to be 0.

• In the ideal process, P's output is always random (since P's input is independent from x_1).

Example:
The coin tossing functionality

1. Receive “start” from P_1
2. Receive “start” from P_2
3. Choose $r \in_R \{0,1\}^k$, output r to the parties.

Example:
The coin tossing functionality

1. Receive “start” from P_1
2. Receive “start” from P_2
3. Choose $r \in_R \{0,1\}^k$, output r to the parties.

What about the above “bad protocol” ?

Reminder: The bad protocol

- Let f be a one-way permutation on $\{0,1\}^k$.
- P_1 chooses $s \in_R \{0,1\}^k$, and sends $r=f(s)$ to P_2.
- Both parties output r.

Example: The coin tossing functionality

1. Receive “start” from P_1
2. Receive “start” from P_2
3. Choose $r \in_R \{0,1\}^k$, output s to the parties.

The bad protocol still satisfies the definition:

- The uncorrupted party (P_2) outputs a random value in both cases
- In the ideal process, S gets r from F. But it can ignore r, and instead choose a random s and output (s,f(s)). This would have the right distribution...

So, what's wrong?

Reminder: Definition of security

A protocol π *emulates* the ideal process for evaluating F if for any (PPT) adversary A there exists a (PPT) adversary S such that for any set of external inputs:

- The outputs of the uncorr. Parties running π with A ~ The outputs of the uncorr. parties in the ideal process with S
- The output of A ~ The output of S

Reminder: Definition of security

A protocol π *emulates* the ideal process for evaluating F if for any (PPT) adversary A there exists a (PPT) adversary S such that for any set of external inputs:

- The outputs of the uncorr. Parties running π with A ~ The outputs of the uncorr. parties in the ideal process with S
- The output of A ~ The output of S

Weakness: Need to "tie together" the outputs of the adversary and of the uncorrupted parties

Corrected definition of security

A protocol π *emulates* the ideal process for evaluating F if for any (PPT) adversary A there exists a (PPT) adversary S such that for any set of external inputs:

[The inputs and outputs of all parties and A in π] ~

[The inputs and outputs of all parties and S in ideal process for F]

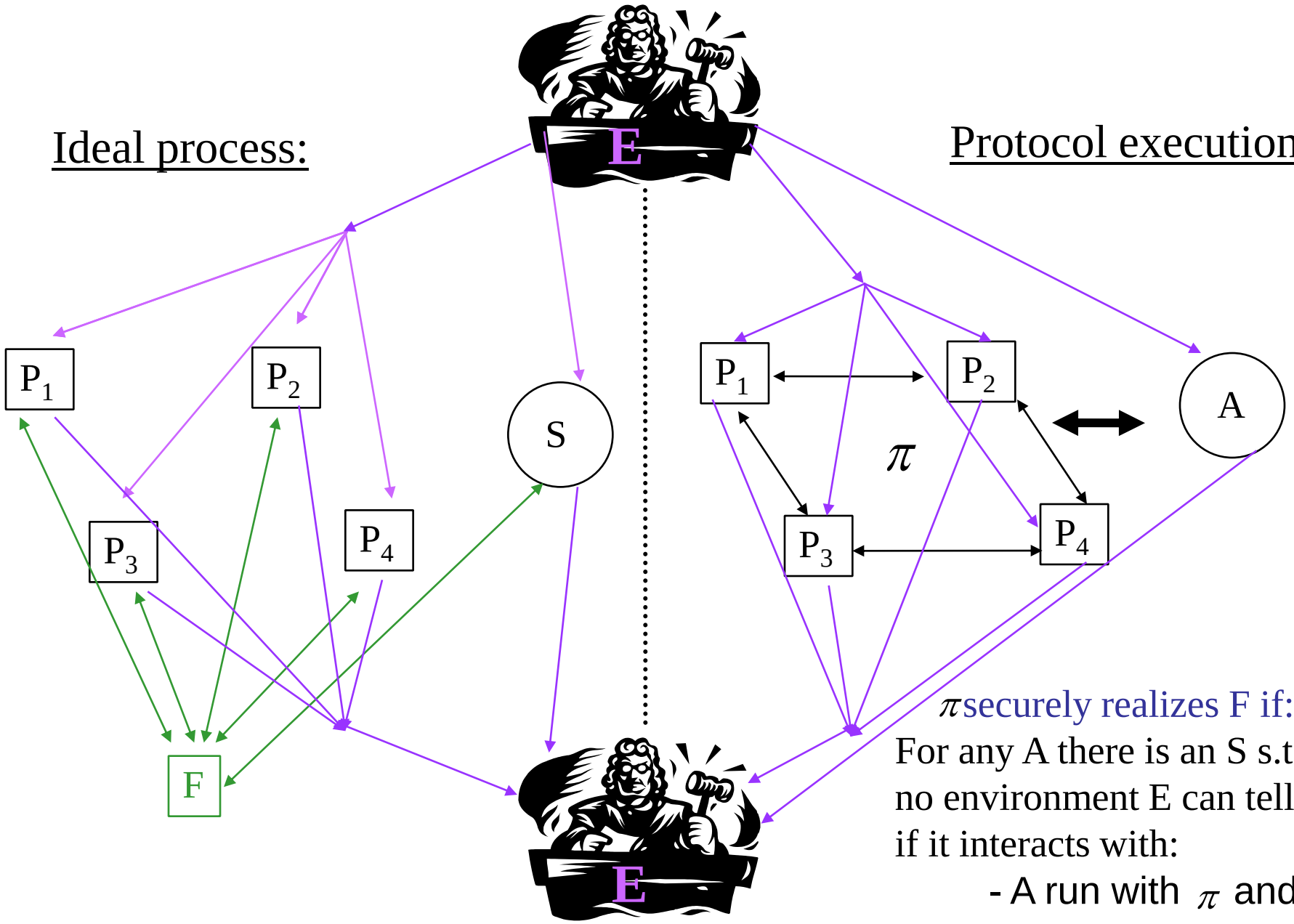
Coin tossing once more

1. Receive “start” from P_1
2. Receive “start” from P_2
3. Choose $r \in_R \{0,1\}^k$, output r to the parties.

The bad protocol no longer satisfies the definition:

- The “global output” of the protocol execution is $(r, f^{-1}(r))$
- In the ideal process, S gets r from F; to satisfy the definition, S now has to output $f^{-1}(r)$...

An alternative formulation:



π securely realizes F if:
For any A there is an S s.t
no environment E can tell
if it interacts with:

- A run with π and A
- A run with F and S

Another example: The ZK functionality, F_{ZK} (for relation R)

1. Receive (x,w) from party P ("the prover")
2. Receive (x) from party V ("the verifier")
3. Output R(x,w) to V.

Note:

- V is assured that it accepts only if R(x,w)=1 (soundness)
- P is assured that V learns nothing but R(x,w) (Zero-Knowledge)

Comparison with the traditional formulation

- Traditionally ZK is defined using three separate requirements (Completeness, Soundness, Zero-Knowledge)
- Here there is an apparent "proof of knowledge" requirement

Still the two formalizations are "essentially equivalent":

Theorem: A protocol securely realizes F_{ZK} for relation R if and only if it is a computationally sound ZK proof of knowledge for R (with non-black-box extractors).

(Assume that there is an input # s.t. R(x,#)=0 for all x, else augment R accordingly.)

Additional “definitional details”:

- What is the model of communication:
  - Asynchronous? Synchronous?
  - Secret? Authenticated? Unauthenticated?
  - Point-to-point? broadcast?
- How do parties address each other?
- Can we model “open systems” where parties join during the computation?
- Can we model reactive tasks?
- How to model PPT computation in a meaningful way?

Need a better model of computation...

The basic model: Highlights
(based on [C, iacr eprint 2000/067, Dec 05])

- The basic computing unit: PPT Interactive TM
  - Externally writable tapes: Input, incoming comm, subroutine output
  - Identity tape (identity "in software")
  - Polytime in input length, minus length of inputs written to others
- ITMs can:
  - invoke other ITMs (specifying code and identity of invokee)
  - write to tapes of other ITMs (one write per activation).

  (subject to restrictions specified by a "control function")
- Order of activations: An initial ITM is specified. ITM whose tape is last written to is activated next.

The protocol execution model: Highlights

- Environment: Invokes the adversary and as many parties as wants, gives inputs and identities, reads outputs.
- Parties: Run their code, send messages *only to adversary.*
- Adversary: Delivers arbitrary messages to parties.

Notes:

- Captures asynchronous, unauthenticated comm.

  (To capture other comm. models, add structure)
- Party corruptions modeled as special messages from adv.
- Env gives one input to and reads one output from adv

The ideal process: Highlights

- Modeled as a special protocol within the general model:
  - All parties copy their inputs to a special ITM ("Ideal functionality") F,
  - Copy the outputs from F to environment
- Can capture reactive tasks in a natural way (interactive F)
- F can interact directly with adversary, allowing for finer-grain specification of security properties:
  - Messages from Adv capture "allowed influence"
  - Messages to Adv captures "allowed info leakage"

Pro: very expressive. Con: very expressive.

Example: The commitment functionality

1. Upon receiving (“commit”,C,V,x) from party C, record x, and send (C, “receipt”) to V.
2. Upon receiving (“open”) from C, send (C,x) to V and halt.

Note:

- V is assured that the value it received in step 2 was fixed in step 1.
- P is assured that V learns nothing about x before it is opened.

Example: The commitment functionality

1. Upon receiving (“commit”,C,V,x) from party C, record x, and send (C, “receipt”) to V.
2. Upon receiving (“open”) from C, send (C,x) to V and halt.

But, need to allow the adverary to delay outputs of V, otherwise the requirement is unrealistically strong...

Example: The commitment functionality

1. Upon receiving (“commit”,C,V,x) from party C, record x, and send (C, “receipt”) to adv. When adv says “OK”, send (C, “receipt”) to V.
2. Upon receiving (“open”) from C, send (C,x) to V and halt.

Example: The commitment functionality

1. Upon receiving (“commit”,C,V,x) from party C, record x, and send (C, “receipt”) to adv. When adv says “OK”, send (C, “receipt”) to V.
2. Upon receiving (“open”) from C, send (C,x) to V and halt.

But, what if the commiter gets corrupted?

Example: The commitment functionality

1. Upon receiving (“commit”,C,V,x) from party C, record x, and send (C, “receipt”) to adv. When adv says “OK”, send (C, “receipt”) to V.
2. Upon receiving (“open”) from C, send (C,x) to V and halt.
3. Upon receiving a “corrupt C” from Adv, hand x to Adv.

Is that enough?

Example: The commitment functionality

1. Upon receiving (“commit”,C,V,x) from party C, record x, and send (C, “receipt”) to V. When adv says “OK”, send (C, “receipt”) to V.
2. Upon receiving (“open”) from C, send (C,x) to V and halt.
3. Upon receiving a “corrupt C” from Adv, hand x to Adv.

Is that enough?
What if the committer gets corrupted immediately after it was invoked, and the adversary changes the committed bit?

Example: The commitment functionality

1. Upon receiving (“commit”,C,V,x) from party C, record x, and send (C, “receipt”) to V. When adv says “OK”, send (C, “receipt”) to V.
2. Upon receiving (“open”) from C, send (C,x) to V and halt.
3. Upon receiving a “corrupt C” from Adv, hand x to Adv. If V didnt yet get “receipt” then allow Adv to change x.

Is this enough?

Example:
The commitment functionality

1. Upon receiving (“commit”,C,V,x) from party C, record x, and send (C, “receipt”) to V. When adv says “OK”, send (C, “receipt”) to V.
2. Upon receiving (“open”) from C, send (C,x) to V and halt.
3. Upon receiving a “corrupt C” from Adv, hand x to Adv. If V didnt yet get “receipt” then allow Adv to change x.

Is this enough?
Seems so...

General feasibility results

(with respect to this definition)

- [Yao86]: Can realize any two-party functionality for honest-but-curious parties.
- [GMW87]: Given authenticated communication, can realize any ideal functionality:
  - with any number of faults (without guarantee termination)
  - With up to n/2 faults (with guaranteed termination)
- [Ben-Or-Goldwasser-Wigderson88, Chaum-Crepeau-Damgard88]: Assuming secure channels, can do:
  - Unconditional security with n/3 faults
  - Adaptive security
- Many improvements and extensions [RB89,CFGN95,GRR97,…]

The basic technique

- Represent the code of the trusted party as a circuit.
- Evaluate the circuit gate by gate in a secure way against honest-but-curious faults.
- “Compile” the protocol to obtain resilience to malicious faults:
  - [GMW87]: Use general ZK proofs
  - [BGW88,CCD88]: Use special-purpose algebraic proofs

Part II:
Security under protocol composition

Is security preserved under protocol composition?

- So far, we considered only a single protocol execution, in isolation.
- Does security in this setting imply security in a multi-execution setting?

Is security preserved under protocol composition?

- So far, we considered only a single protocol execution, in isolation.
- Does security in this setting imply security in a multi-execution setting?

Why should we care?

- We can always directly analyze the entire system...

Two benefits of security-preserving composition of protocols

- Modular design and analysis of protocols:
  - Break down a complex system into simpler chunks.
  - Analyze protocols for each of the chunks (as stand-alone).
  - Deduce the security of the original, composite system.

- Security in unknown environments:
  - Guarantee security when the protocol is running alongside potentially unknown protocols (that may even be designed later, depending on the analyzed protocol).

Security under composition: 1st Example

Parallel composition of ZK protocols [Goldreich-Krawczyk88]:

- Assume the following gadget. The verifier V can generate "puzzles" such that:
  - The prover P can solve puzzles
  - V cannot distinguish a solution from a random element (even for puzzles that it generated).

Can be constructed either assuming P is unbounded, or under computational assumptions [Feige89].

Parallel composition of ZK protocols

Take a ZK protocol and add the following:

- P sends a puzzle p to V
- If V gives a solution s to p then reveal the secret witness.
- Else, if V returns a puzzle p' then send a solution s' for p'.

Parallel composition of ZK protocols

Take a ZK protocol and add the following:

- P sends a puzzle p to V
- If V gives a solution s to p then reveal the secret witness.
- Else, if V returns a puzzle p' then send a solution s' for p'.

- If the original protocol is ZK, then so is the modified protocol:
  - V will never solve the challenge puzzle p
  - The solution s' can be simulated by a random string

Parallel composition of ZK protocols

Assume V interacts with P in two sessions concurrently. Then there is an attack:

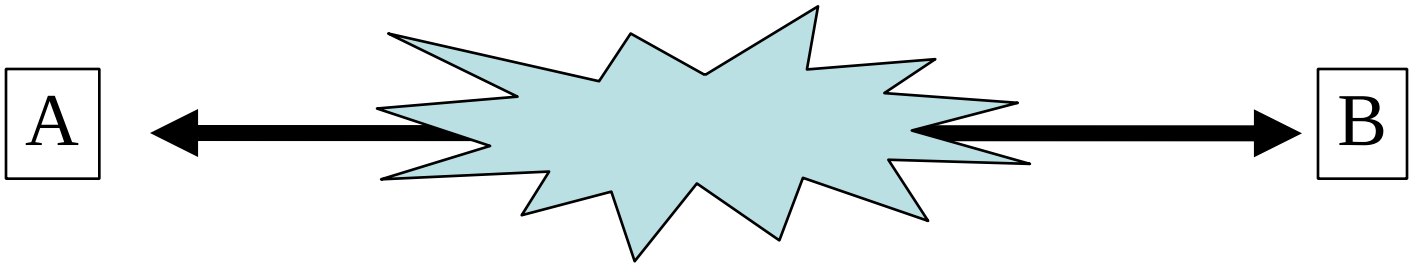
- V obtains p1 from P in session 1
- V gives p1 as its “p'” in session 2, gets a solution s1
- V gives s1 to P in session 1, and obtains the witness...

Conclusion: Running even *two* instances of the *same* protocol in parallel is not secure...

2nd example: Key exchange and secure channels

Authenticated Key Exchange

The goal: Two parties want to generate a common, random and secret key over an untrusted network.



- The main use is to set up a secure communication session: Each message is encrypted and authenticated using the generated key.

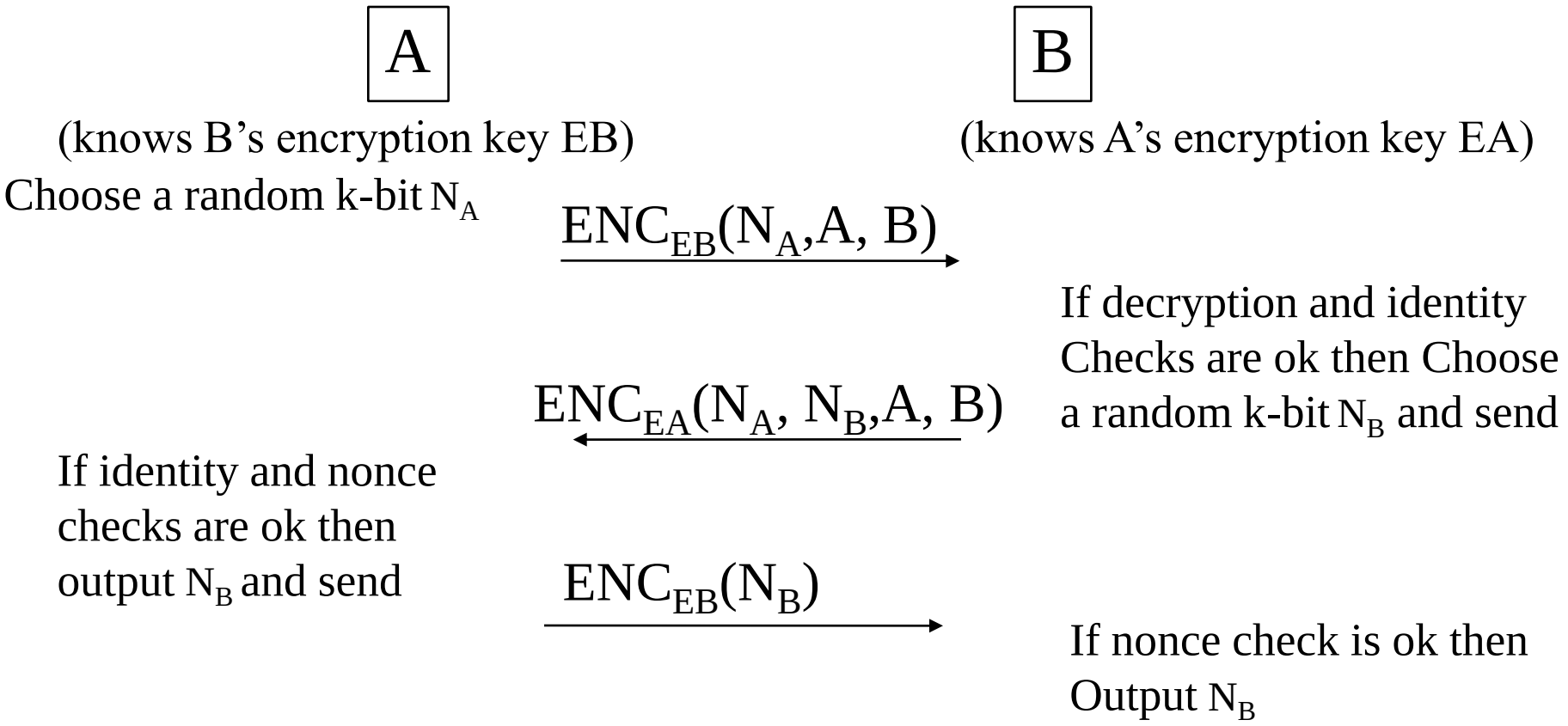
The basic security requirements

- Key agreement: If two honest parties locally generate keys associated with each other then the keys are identical.

- Key secrecy: The key must be unknown to an adversary.

Encryption-based protocol

[based on Needham-Schroeder-Lowe,78+95]



The protocol satisfies the requirements:

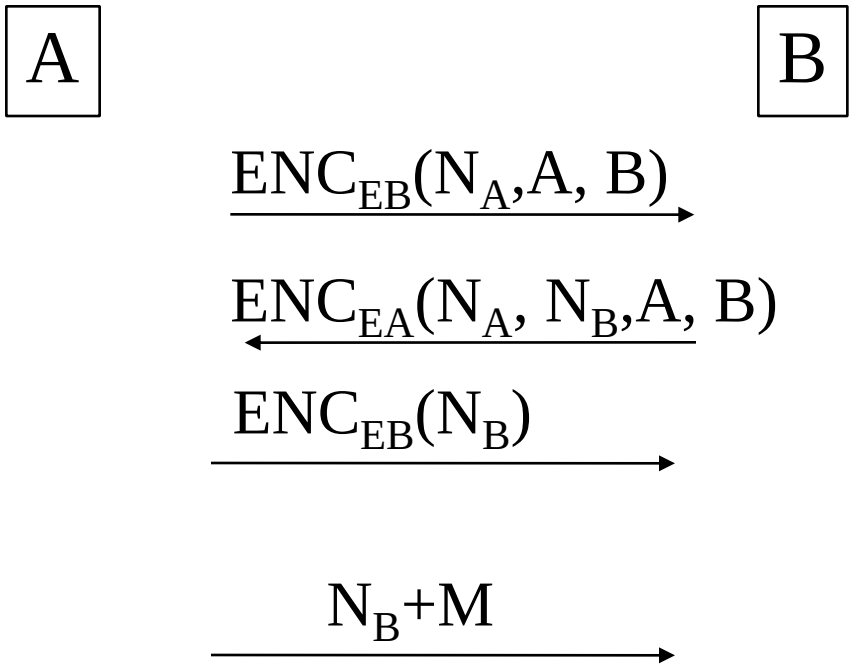
- Key agreement: If A, B locally output a key with each other, then this key must be N_B. (Follows from the "untamperability" of the encryption.)

- 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 securely realizes the KE functionality under the basic definition)

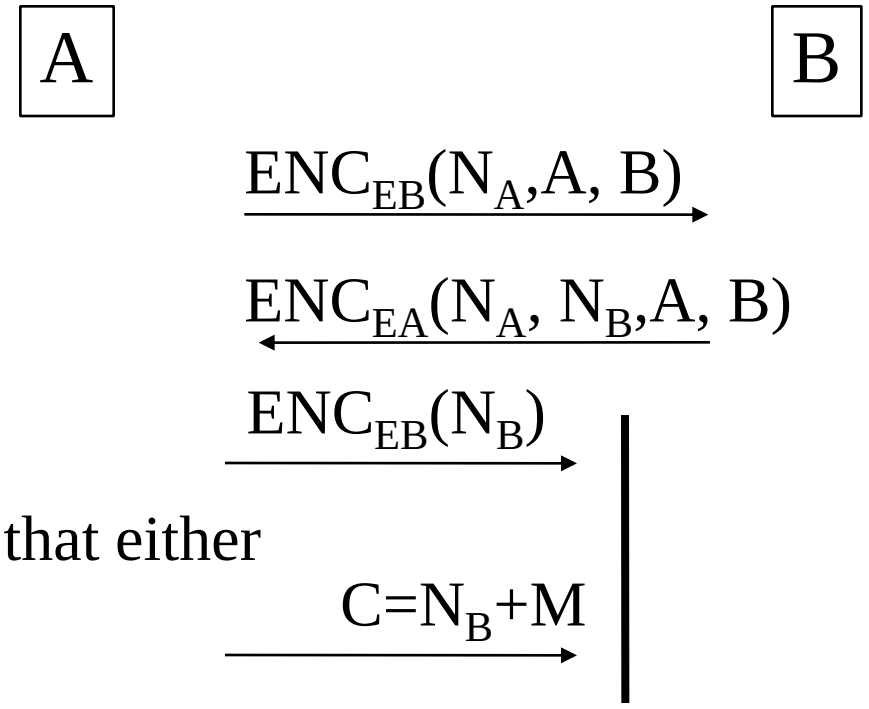
Attack against the protocol:

Assume that A uses the generated key to encrypt a buy/sell message M, using one-time-pad:



Attack against the protocol:

Assume that A uses the generated key to encrypt a buy/sell message M, using one-time-pad:



Attacker knows that either
$C=N_B+$ "sell", or
$C=N_B+$ "buy".
This can be checked:

If B accepts the exchange
then 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 posible candidates for the key...)

Example III:
Malleability of commitment

[Dolev-Dwork-Naor91]

A naive auction protocol using commitments:

Phase 1:
Each bidder publishes
a commitment to its bid.

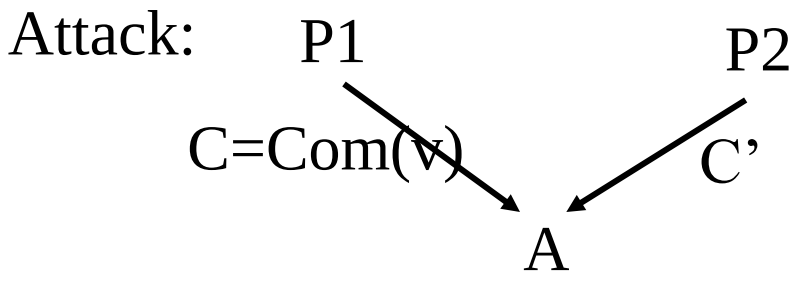
Phase 2:
Bidders open
their commitments.

Example III:
Malleability of commitments

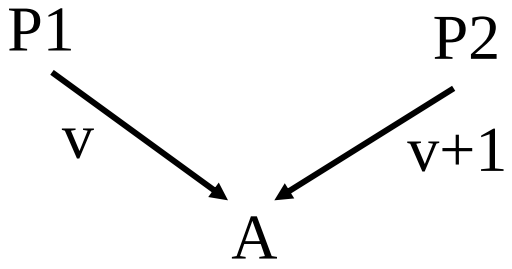
[Dolev-Dwork-Naor91]

A naive auction protocol using commitments:

Phase 1:
Each bidder publishes
a commitment to its bid.



Phase 2:
Bidders open
their commitments.



The problem: The stand-alone definition does not guarantee that the committed values in different instances are independent from each other.

This is a whole new security concern, that does not exist in the stand-alone model...

Non-malleable commitments [DDN91]

Guarantee “input independence” for commitments in the case where *two* instances of the *same* commitment protocol run concurrently.

Non-malleable commitments [DDN91]

Guarantee “input independence” for commitments in the case where *two* instances of the *same* commitment protocol run concurrently.

What about multiple instances? Different protocols?

Seems hopeless:

- Given a commitment protocol C, define the protocol C':
  - To commit to x, run C on x-1.
- Now, all the attacker has to do is to claim it uses C' and copy the commitment and decommitment messages...

Ways to compose protocols:

Ways to compose protocols: Salient parameters

- Timing coordination:
  - Sequential, Non-concurrent, Parallel, Concurrent
- Input coordination:
  - Same input, Fixed inputs, Adaptively chosen inputs
- Protocol coordination:
  - Self composition, General composition
- State coordination:
  - Indepdendent states, Shared state
- Number of instances:
  - Fixed, Bounded, Unbounded

Universal composition

(Idea originates in [Micali-Rogaway91])

The idea: Generalize "subroutine substitution" of sequential algorithms to distributed protocols.

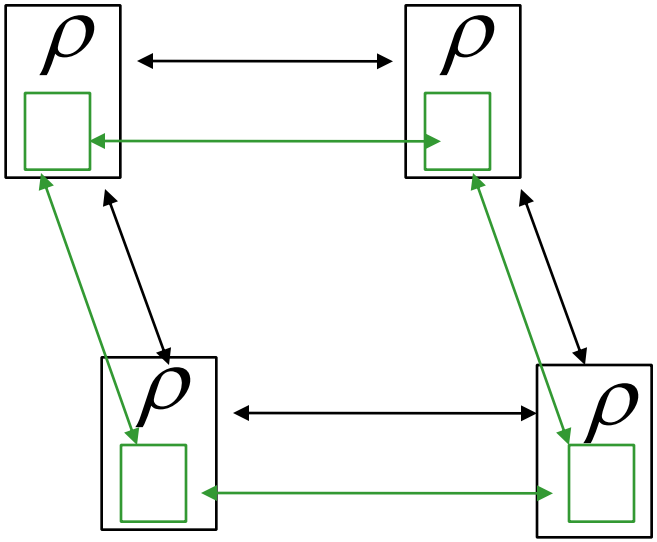
Start with:

- Protocol ρ that uses calls to ϕ
- Protocol π

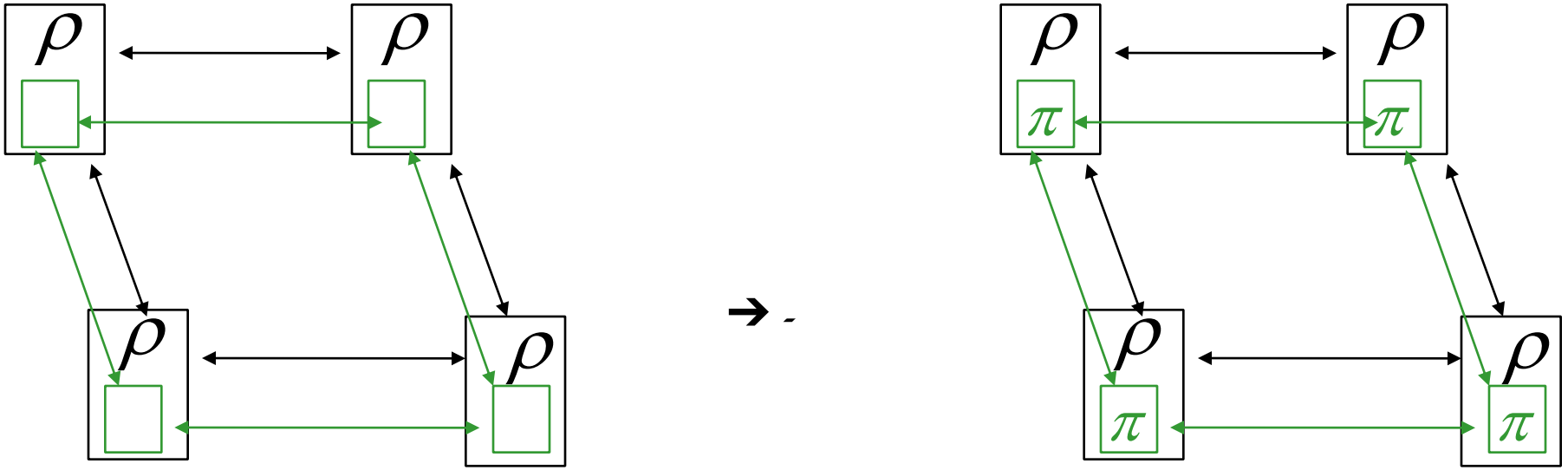
Construct the composed protocol $\rho^{\pi/\phi}$:

- Each call to ϕ is replaced with a call to π.
- Each value returned from π is treated as coming from ϕ.

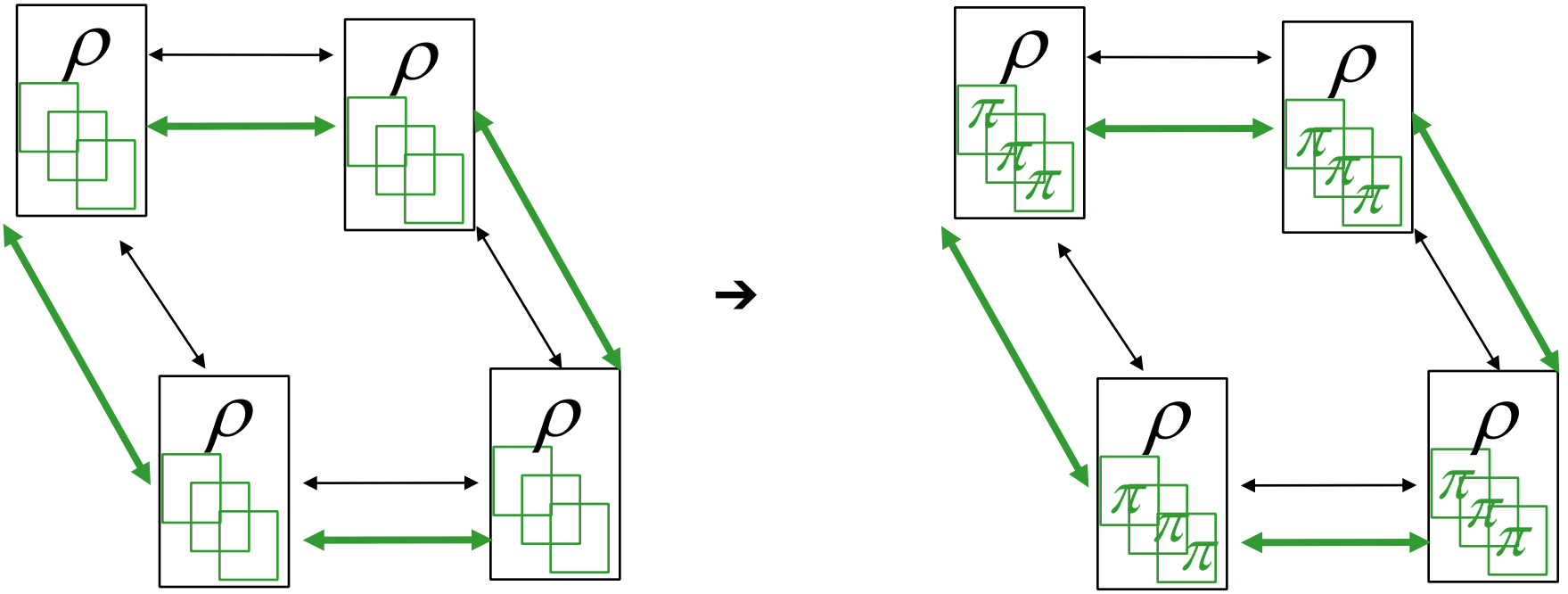
Universal Composition
(single subroutine call)



Universal Composition
(single subroutine call)



Universal Composition
(many subroutine calls)



Why study universal composition?

- Can represent any of the composition scenarios discussed in the literature, by using the appropriate "calling protocol,"ρ .

→ Enough to consider a single composition operation...

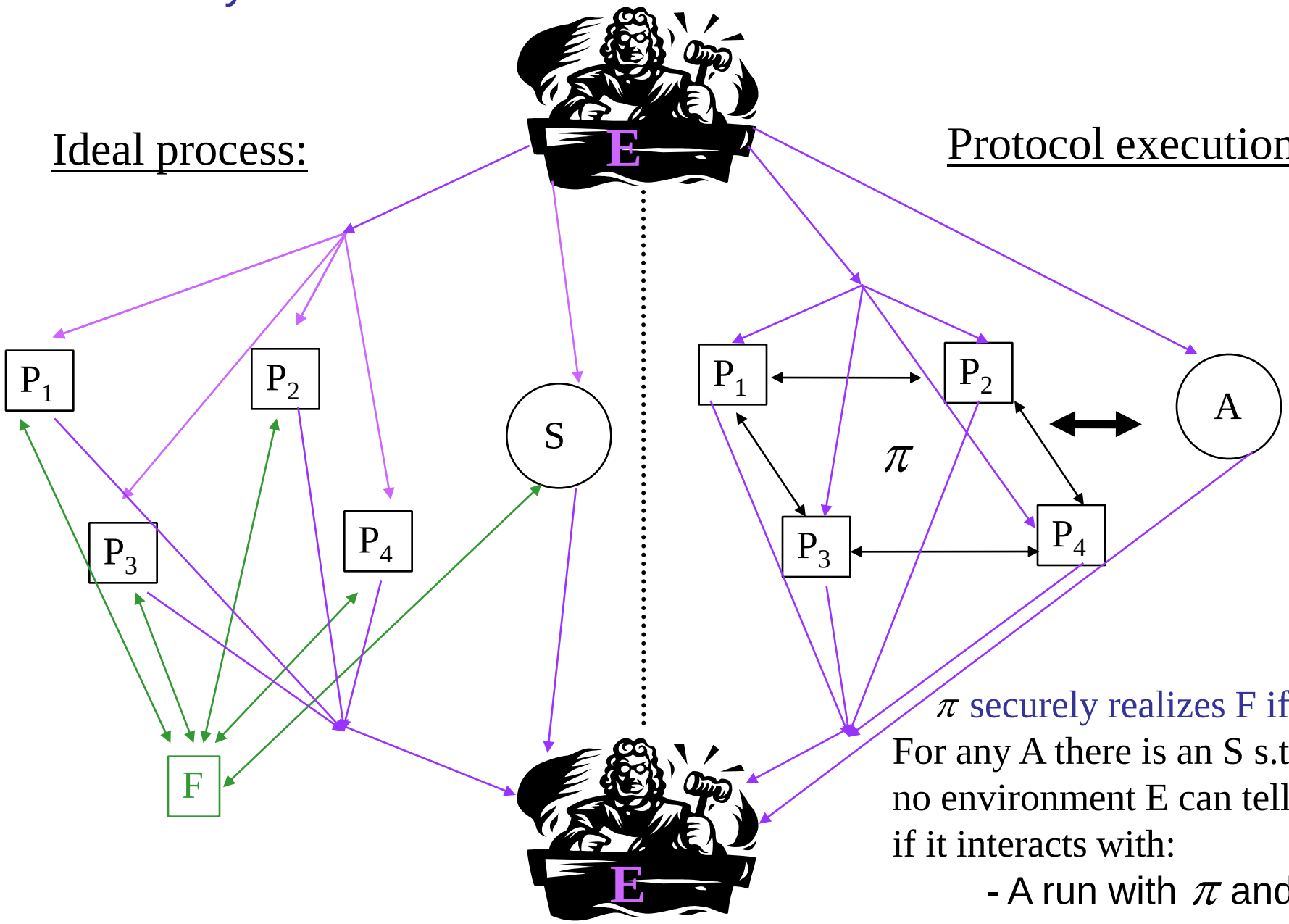
What should be the security requirements under protocol composition?

What should be the security requirements under protocol composition?

- Can have a list of properties to be preserved:
  - Correctness
  - Secrecy
  - Input independence
  - ...

→ But, again, how do we know we got it all...

Basic security:



π securely realizes F if:
For any A there is an S s.t
no environment E can tell
if it interacts with:

- A run with π and A
- A run with F and S

A proposed security requirement:

- Recall the notion of protocol emulation:

  π emulates ϕ if for any adversary A there exists an adversary S such that no environment E can tell if it runs with π and A or with ϕ and S.

Definition: π emulates ϕ with ρ-composable security if protocol ρ^{π} emulates protocol ρ^{ϕ}.

Goal: Find a definition that provides ρ-composable security for as many protocols ρ as possible.

What are the composability properties of basic security?

- Intuitively, should be composable: It is explicitly required that no environment can tell the difference between running π and running ...

- As long as subroutine calls are non-concurrent, the intuition holds:

The non-concurrent composition theorem:

[C. 00]

If π emulates $\mathcal{F}$ and in ρ no two protocol copies are running concurrently, then protocol $\rho^{\pi/\mathcal{F}}$ emulates protocol ρ.

Corollary: If ρ securely realizes functionality G then so does $\rho^{\pi/\mathcal{F}}$.

Proof idea:

- Given adv A against ρ^{π} , construct an adv A_{π} against a single instance of π .
- Obtain a simulator S_{π}.
- Construct an adversary that interacts with ρ^{f}, given A and S_{π}.
- Show validity by contradition...

What about concurrent subroutine calls?

- We already saw counter examples:
  - Zero-Knowledge
  - Key Exchange
  - Commitment

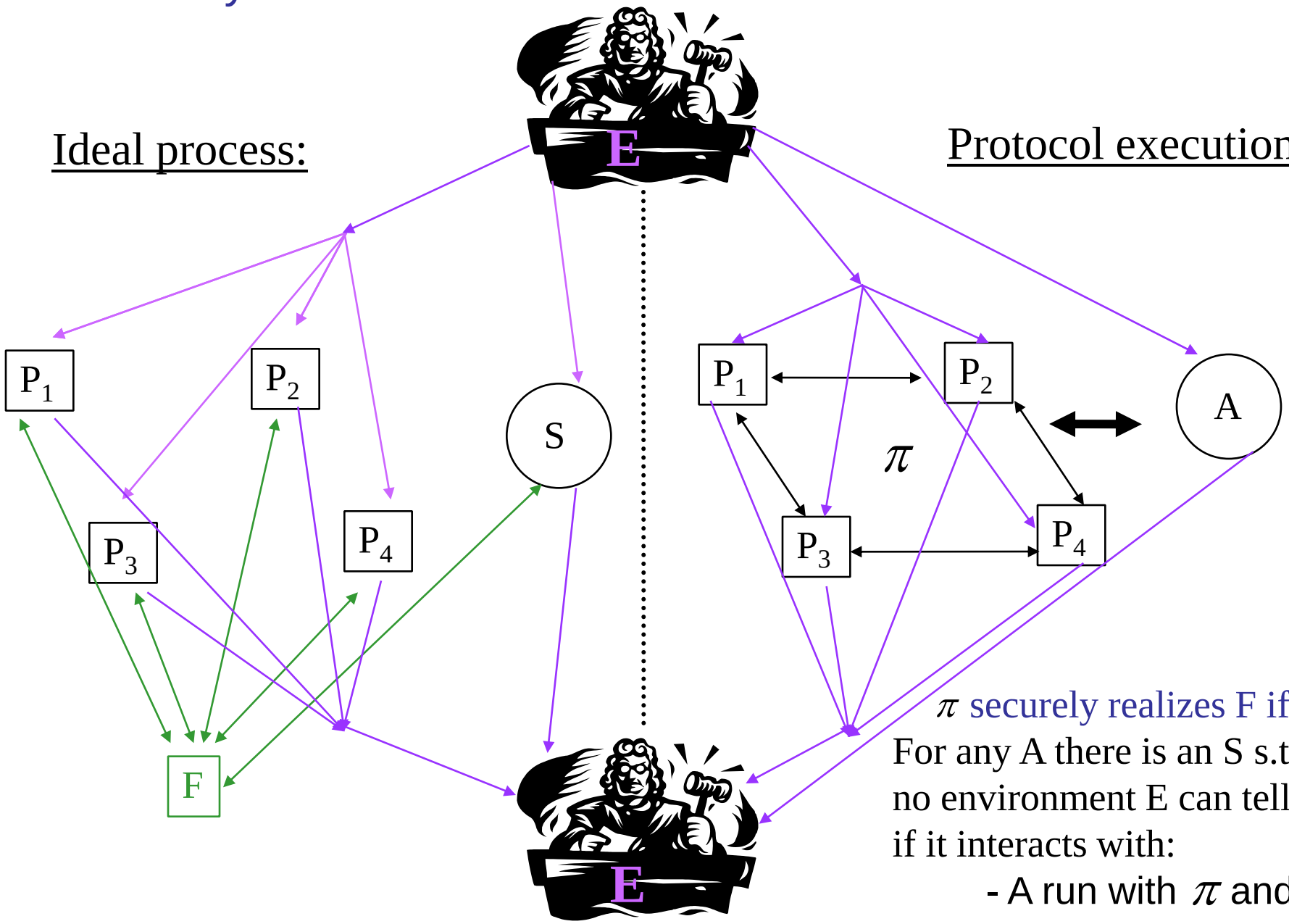
- Other examples exist, even with information-theoretic security, and even with a single subroutine call... [Lindell-Lysyanskaya-Rabin02, Kushilevitz-Lindell-Rabin06]

Part III:
Universally Composable Security

Why isn't basic security preserved under concurrent composition?

- Recall the definition...

Basic security:



π securely realizes F if:
For any A there is an S s.t
no environment E can tell
if it interacts with:

- A run with π and A
- A run with F and S

Why isn't basic security preserved under concurent composition?

- The “information flow” between the extenal environment and the adversary is limited:
  - Initial input
  - Final output
- Instead, in concurrent executions there is often “circular adversarieal information flow” among executions:

  execution 1 -> execution 2 -> execution 1

  these are not captured...

Universally Composable Security [C. 98,01]

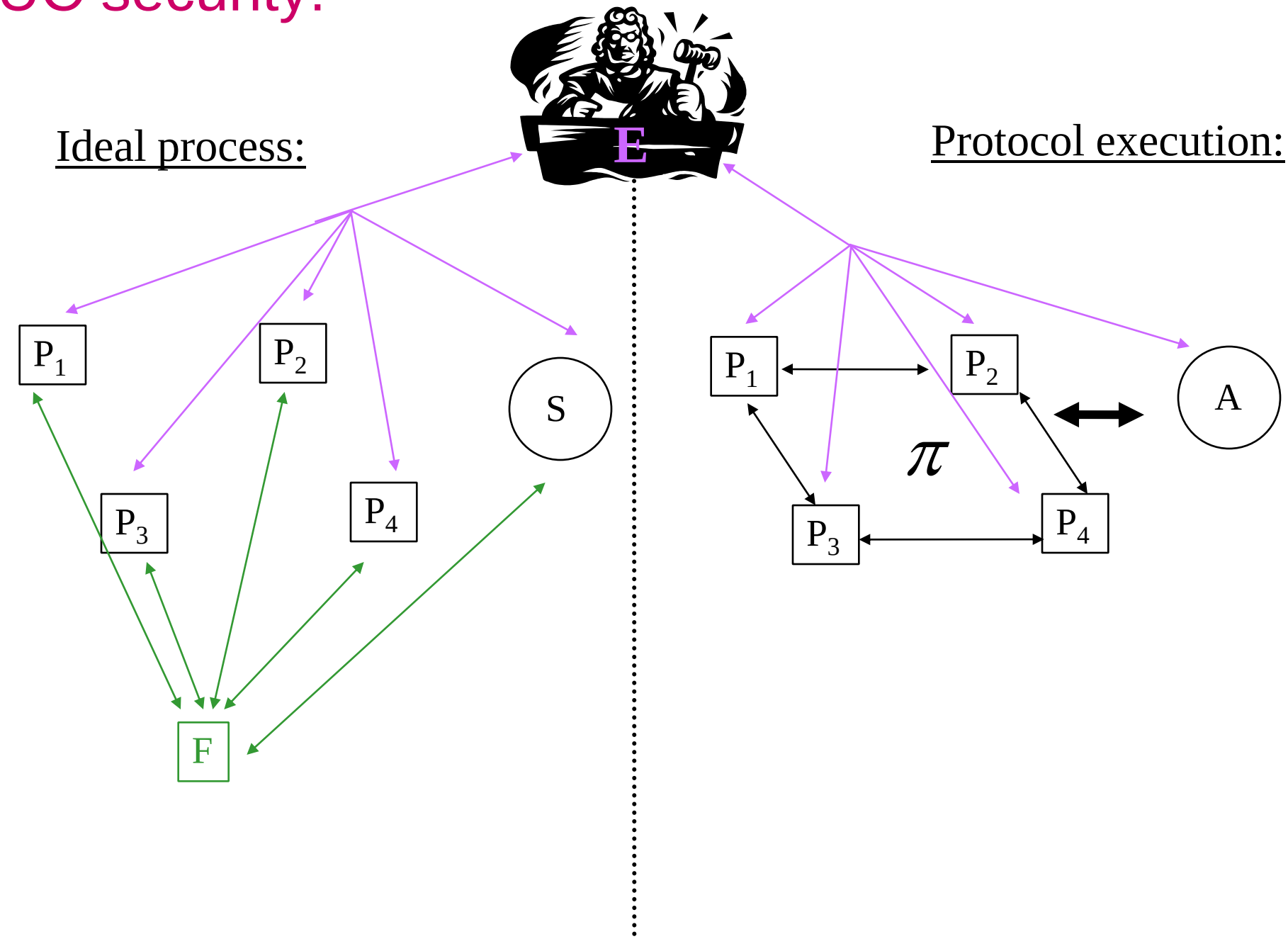
The main difference from the basic notion:

The environment interacts with the adversary in an arbitrary way throughout the protocol execution.

Also, add structure to the model to facilitate distinguishing among protocol instances in a system. (Add “a session identifier (SID)” as part of each ID.)

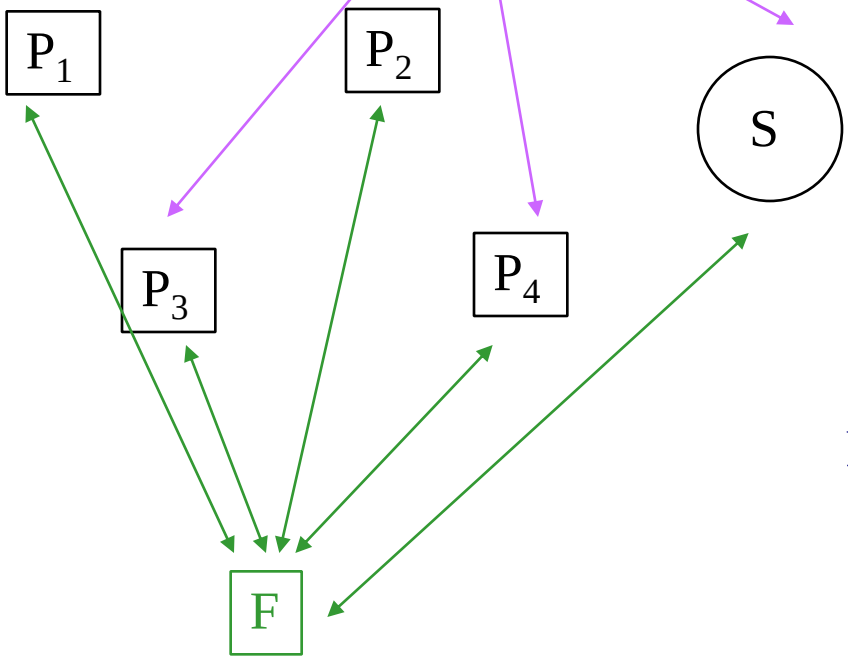
Similar ideas appear in [Pfitzmann-Waidner00,01]

UC security:

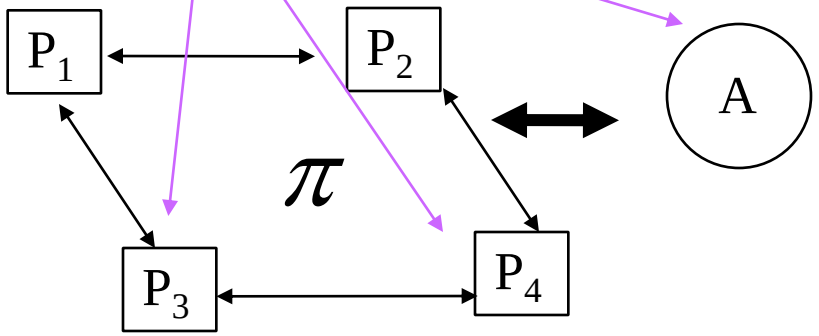

Ideal process:
E
Protocol execution:
P1
P2
S
P3
P4
F
P1
P2
A
π
P3
P4

UC security:

Ideal process:



Protocol execution:



Protocol π UC-realizes F if:

For any adversary A
There exists an adversary S
Such that no environment E can tell whether it interacts with:

- A run of π with A
- An ideal run with F and S

The universal composition theorem: [C. 01]

If π UC-emulates ϕ then protocol $\rho^{\pi/\phi}$ UC-emulates protocol ρ.

Corollary: If ρ UC-realizes functionality G then so does $\rho^{\pi/\phi}$.

Proof idea:

- Same outline as the non-concurrent case:
  - Given adv A against $\rho^{\pi/\phi}$, construct an adv A_π against a single instance of π .
  - Obtain a simulator S_π.
  - Construct an adversary that interacts with ρ, given A and *multiple instances* of S_π.
  - Show validity by contradition...

Implications of the UC theorem

- Modular protocol design and analysis
- Enabling sound formal and symbolic analysis
- Representing communication models as "helper" ideal functionalities

Modular protocol design

- So far, ideal functionalities were used as ways to specify security requirements.

- However, I.F.s can also capture security assumptions on underlying primitives (subroutines).
- Validity follows from UC theorem
- Simplifies design and analysis (subroutines are treated as “black boxes”)

Example: The authenticated message transmission functionality, F_auth

1. Upon receiving (A,B,sid,m) from A, send (A,B,sid,m) to the adversary.
2. When receiving "ok" from the adversary, output (A,B,sid,m) to B and halt.

Can be used as:

- A specification for message authentication protocols
- A way to formalize ideally authenticated communication

Design protocols that use (multiple instances of) F_auth as Subroutines.

(We call such protocols *F_auth-hybrid protocols*.)

Example: The *secure* message transmission functionality, F_smt

1. Upon receiving (A,B,sid,m) from A, send (A,B,sid,|m|) to the adversary.
2. When receiving “ok” from the adversary, output (A,B,sid,m) to B.

(need to add stuff for adaptive corruptions)

Can be used as:

- A specification for secure communication protocols
- A way to formalize ideally secret communication