6.897: Selected Topics in Cryptography
Lectures 7 and 8

Lecturer: Ran Canetti

Highlights of past lectures

- Presented a basic framework for analyzing the security of protocols for multi-party function evaluation.
- Presented the notion of modular composition.
- Stated and proved the non-concurrent. composition theorem
- Showed how to capture ZK and PoK within this framework.
- Used the composition theorem to analyze a basic ZK protocol.
- Mentioned some limitations of the notion…

Lectures 7 and 8

The UC Framework and composition theorem

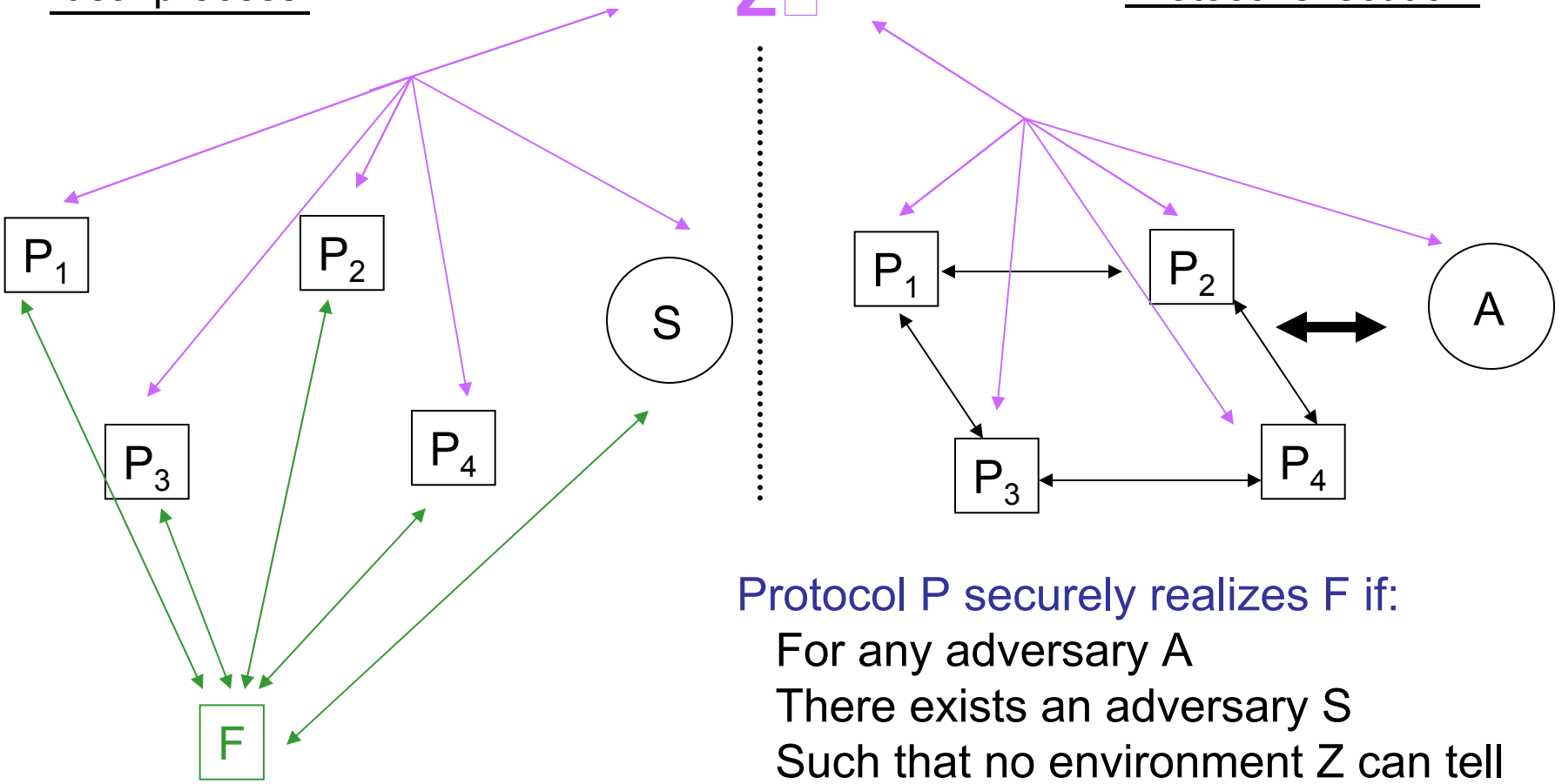
- Motivation for a new framework
- The UC framework:
  - The basic system model
  - The real-life model for protocol execution
  - Ideal functionalities and the ideal process
- Alternative formalizations
- Universal composition:
  - The hybrid model
  - The composition operation
  - The UC theorem
  - Interpretations
  - Proof
- Some ideal functionalities

Review of the definition:

Z

Ideal process:

Protocol execution:



Protocol P securely realizes F if:

For any adversary A

There exists an adversary S

Such that no environment Z can tell whether it interacts with:

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

Note:

There exist protocols for securely evaluating any multi-party function under this definition:

- Goldreich-Micali-Wigderson [GMW87,G98,G04], for any number of faults, computational.
- BenOr-Goldwasser-Wigderson [BGW88], for honest majority, algebraic, “information-theoretic”.
- Many other protocols…

Characteristics of the basic definition

- Simplistic system model: Fixed set of parties, fixed identities, fixed number of protocols.
- Fixed set of corrupted parties.
- Only function evaluation.
- Environment interacts with the computation only at the beginning and the end.
- Only non-concurrent composition.

Wish-list for a more general framework

- Deal with more “real-life” settings such as:
  - Asynchronous communication
  - Unreliable and unauthenticated communication
  - Variable (even unbounded) number of participants
  - Variable identities
- Deal with reactive tasks (e.g., encryption, signatures, commitments, secret-sharing…)
- Deal with adaptive break-ins to parties
- Deal with concurrent composition
- Allow proving security of “natural protocols”

➔ The UC framework is aimed at all of the above goals (except perhaps the last one… but not really)

Presenting the UC framework

- Generalize the underlying computational model ("systems of ITMs")
- Generalize the real-life model of protocol execution
- Generalize the notion of a "trusted party" and the ideal process
- Generalize the notion of protocol emulation

- ## Interactive Turing machines (ITMs):

  An ITM is a TM with some special tapes:

  - Incoming communication tape
  - Incoming subroutine output tape
  - Identity tape, contains ID=(SID,PID), where:
    - SID is the “session ID”
    - PID is the “party ID”
  - security parameter tape

  An activation of an ITM is a computation until a “waiting” state is reached.

- ## Polytime ITMs:

  An ITM M is polytime if at any time the overall number of steps taken is polynomial in the security parameter plus the overall input length (ie, # of bits on the *input tape*).

Systems of interacting ITMs (Variable number of ITMs):

A system of interacting ITMs is a pair (M_0,C) where is the initial ITM and C is a "control function" from sequences of requests to {0,1}. A run of the system is the following process:

- M_0 starts with some external input and value for the security parameter.
- In each activation an ITM may request to write to *at most one* tape of another ITM. A request includes:
  - Identity of the requesting ITM
  - Identity of the target ITM and tape, code for the target ITM.
  - Contents

  If the control function C allows the tuple (source id, target id, code, tape) then the instruction is carried out. *If no ITM with target id exists then a copy is invoked, with said code, identity, and sec. param.*
- The machine written to is the next to be activated. If none then M_0 is activated next.
- The output is the output of the initial ITM M_0.

➔ Notice: the identity of each ITM is globally unique.

- Convention:

  If an ITM M is invoked because M' wrote to its input tape then M is called a subroutine of M' and M' is the invoker of M.

- States of systems of ITMs
  - :A state of a system describes an instance in the run of the system, including local states of all ITMs.

- Multi-party protocols:
  - A multi-party protocol is a (single) ITM.
  - An instance of a protocol P with SID sid, in a state s of a system, is the set of all ITMs in s whose code is P and whose SID is sid.

(some technicalities are pushed under the rug…)

The "real-life model" for protocol execution

The real-life model for executing P with environment Z is the following system of interacting ITMs:

- Initial ITM:
  - Environment Z (the initial ITM, with fixed ID)
- Control function:
  - Z can activate a (single copy of) an ITM A (adversary), and multiple ITMs running P, *all having the same SID*, and write to their input tapes.
  - A can write to the incoming comm. tapes of all parties and to the subroutine output tape of Z.
  - All other ITMs can write to the incoming comm. tape of A, can invoke new ITMs, and can write to the subroutine output tape of their invoker and the input tapes of their subroutines.
  - Modeling corruptions: A can write a "corrupt" message on incoming comm. Tape of ITM M. Then:
    - M writes "Corrupted" on subr. output tape of Z
    - From now on, in each activation M sends its entire state to A
    - A assumes all write privileges of M.

- Notes:
  - Z interacts with A freely throughout the computation.
  - All communication between parties is done “via A”.
  - Asynchronous communication, no authenticity/reliability guarantee.
  - Z creates new parties adaptively, sets their identities.
  - Adaptive corruptions.

- Notation:
  - $EXEC_{P,A,Z}(k,z,r)$: output of Z after above interaction with P,A, on input z and randomness r for the parties with s.p. k. (r denotes randomness for all parties)
  - $EXEC_{P,A,Z}(k,z)$: The output distribution of Z after above interaction with P,A, on input z and s.p. k, and uniformly chosen randomness for the parties.
  - $EXEC_{P,A,Z}$:

    The ensemble of distributions $\{EXEC_{P,A,Z}(k,z)\}_{k \text{ in } N,\ z \text{ in } \{0,1\}^*}$

Towards the ideal process: Ideal Functionalities and dummy parties

An ideal functionality F is a PPT ITM with the following conventions:

- The PID of F is unused (set to 0)
- F takes inputs from multiple parties – but only from parties whose SID is identical to the local one.
- F can write outputs to all parties that write inputs to it, and invoke new dummy parties with the same SID.

A dummy party for F does:

- Copy all inputs to the input tape of the copy of F with the same SID as the local one.
- Copy all outputs from F to the subroutine output tape of its invoker.

The ideal process

The ideal process for evaluating functionality F with environment Z and adversary S is the following system of interacting ITMs:

- Initial ITM:
  - Environment Z (the initial ITM, with fixed ID)
- Control function:
  - Z can activate a (single copy of) an ITM A (adversary), and multiple copies of ITMs running P, and write to their input tapes. However, here A is "ideally replaced" with an ITM S, and the parties running P are "ideally replaced" by dummy parties for F.
  - F can write to the comm. tape of S and to the subroutine output tapes of all dummy parties.
  - S can write to the incoming comm. tape of F and to the subroutine output tape of Z.
  - Modeling corruptions: A can write a "corrupt M" message on incoming comm. tape of F. Then, F does as it wishes… (typically, F will:
    - Write "Corrupted" on subroutine output tape of M
    - Reveal some information to S
    - Let S influence the output that F provides to M)

- Notes:
  - Communication from Z to F and back is immediate (F has to explicitly "include S in the loop" if it so wishes).
  - F knows who is corrupted… furthermore, the allowed information leakage upon corruption has to be explicitly specified.

- Notation:
  - $\text{IDEAL}^{f}_{S,Z}(k,z,r)$: output of Z after above interaction with F,S, on input z and randomness r for the parties with s.p. k. (r denotes randomness for all parties, ie, $r= r_Z, r_S, r_f$.)
  - $\text{IDEAL}^{f}_{S,Z}(k,z)$: The output distribution of Z after above interaction with f,S, on input z, s.p. k, and uniform randomness for the parties.
  - $\text{IDEAL}^{f}_{S,Z}$:
    The ensemble $\{\text{IDEAL}^{f}_{S,Z}(k,z)\}_{(k \text{ in } N,\ z \text{ in } \{0,1\}^*)}$

Definition of security:

Protocol P emulates the ideal process for F if for any adversary A there exists an adversary S such that for all Z we have:

$$IDEAL^{F}_{S,Z} \sim EXEC_{P,A,Z}.$$

In this case we say that protocol P securely realizes F.

Note: There is no parameterization via the "set of corrupted parties". (In the current formulation there is no need… since F knows who is corrupted then the security properties of the protocol under corruptions can be explicitly expreseed in the code of F.)

Variants

- Passive (semi-honest) adversaries: The corrupted parties continue running the original protocol.
- Unconditional security: Allow Z, A to be computationally unbounded. (S should remain polynomial in Z,A,P, otherwise weird things happen…)
- Perfect security: Z's outputs in the two runs should be identically distributed.
- Other variants (e.g., secure channels, authenticated channels, synchronous communication) are captured as ideal functionalities within the existing framework, *without changing the framework itself.*

Equivalent formulations (same as for the basic definition):

- Z outputs an arbitrary string (rather than one bit) and Z's outputs of the two executions should be indistinguishable.
- Z, A are limited to be deterministic.
- Change order of quantifiers: S can depend on Z.

Another equivalent formulation: security w.r.t. a dummy adversary

- Consider the following adversary (the "dummy adversary") A_d:
  - When receiving (from Z) an input "deliver m to party id", A_d writes m on the communication tape of party id.
  - When receiving an incoming message m from party id, A_d writes "got m from party id" on the subroutine tape of Z.

  (This description captures also the case of party corruptions.)

- Say that protocol P realizes functionality F w.r.t. the dummy adversary if there exists an ideal-process adversary S such that for all Z we have $IDEAL^F_{S,Z} \sim EXEC_{P,Ad,Z}$.

- Claim: P realizes F w.r.t. the dummy adversary iff it realizes F.

Proof:

- If P realizes F then it also realizes F w.r.t the dummy adversary.
- Assume P realizes F then w.r.t. dummy adversaries. That is, there exists an ideal-process adversary S_d such that $IDEAL^F_{Sd,Z} \sim EXEC_{P,Ad,Z}$. Now, let A be an arbitrary adversary.

  Construct the following ideal process adversary S. S runs simulated copies of A, S_d. Next:
  - All inputs from Z are forwarded to A, and all outputs from A go to Z.
  - When A sends message m to party id, S gives input "send m to id" to S_d.
  - When S_d outputs "got m from id", write "m from id" to comm. tape of A.
  - All messages from F are forwarded to S_d, and all messages from S_d to F are forwarded.

Analysis of S_d:

Want to show that $IDEAL^F_{S,Z} \sim EXEC_{P,A,Z}$. This is done as follows:

- Construct the following environment Z_d:
  - Runs simulated copies of Z and A.
  - Forwards all inputs/outputs from Z to the parties and back
  - Forwards all inputs/outputs from Z to A and back
  - Whenever A delivers message m to party id, Z_d activates the actual adversary with input "send m to party id".
  - When receives an output "got m from id" from the adversary, Z_d writes ("m from id") on the communication tape of A.
  - Output whatever Z outputs.
- Can see:

$$IDEAL^F_{S,Z} = IDEAL^F_{Sd,Zd} \sim EXEC_{P,Ad,Zd} = EXEC_{P,A,Z}$$

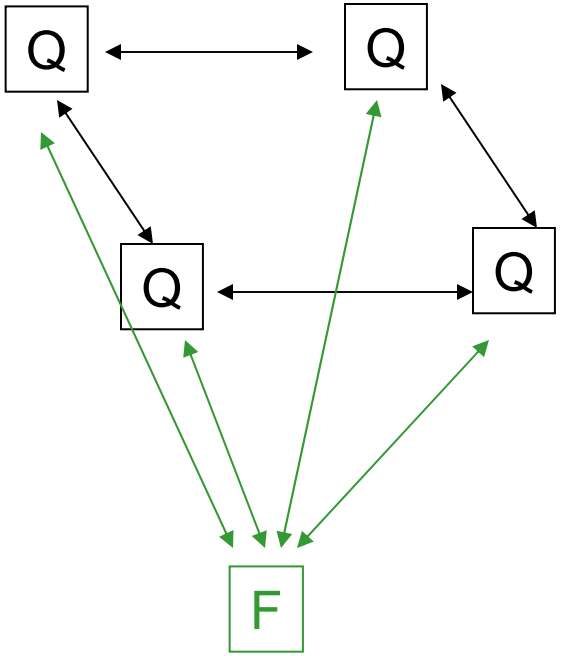
■

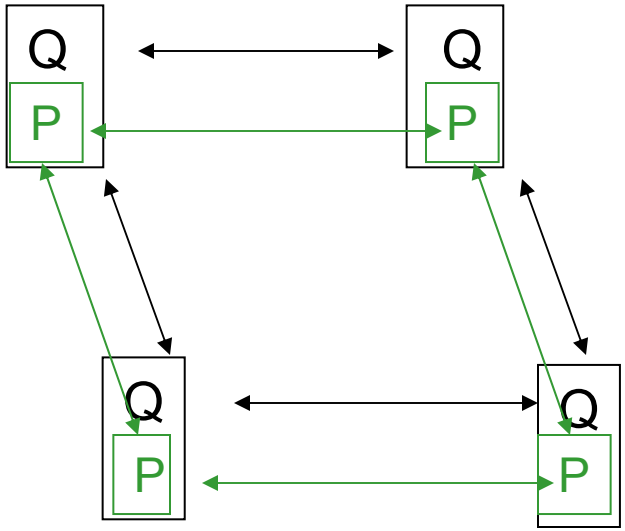
The UC theorem

Will proceed in the usual steps:

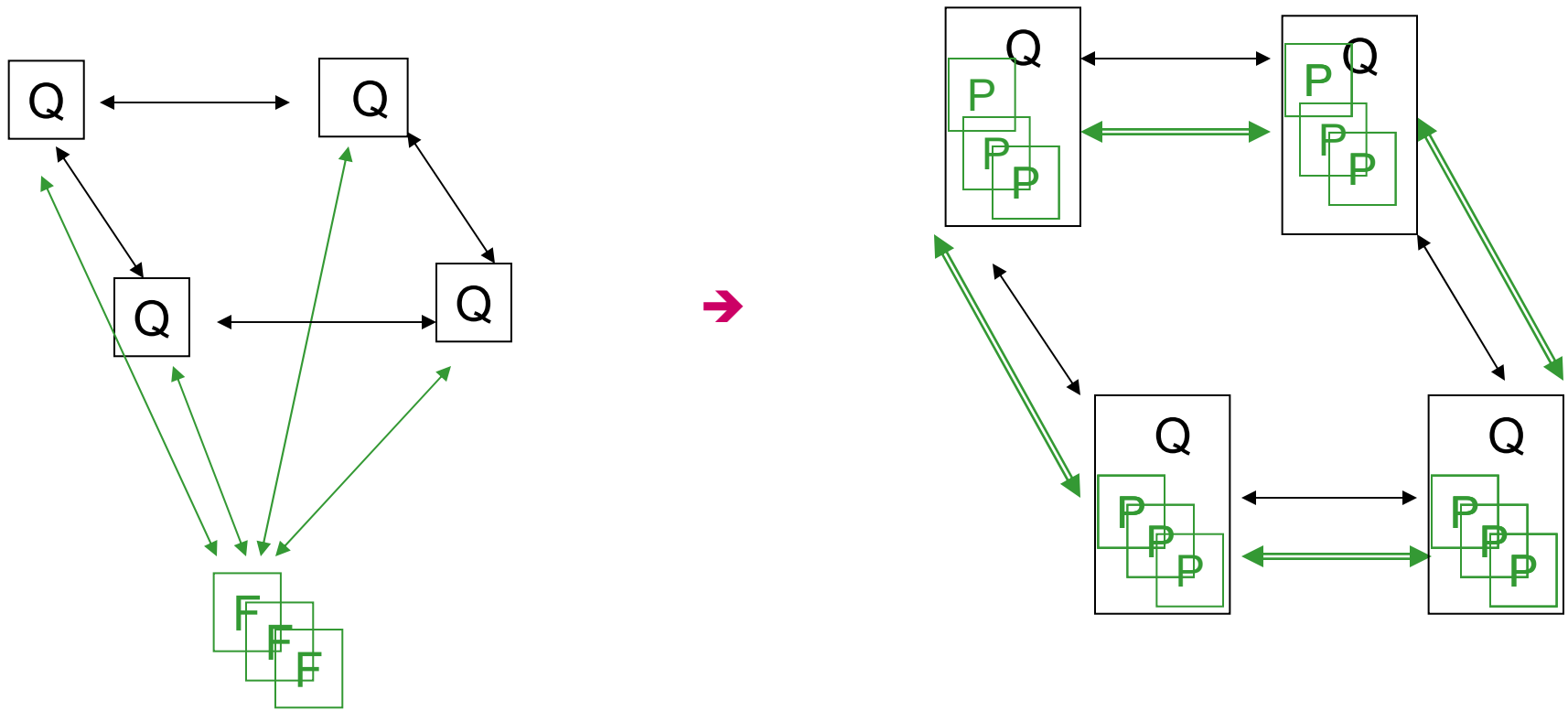
- Present the hybrid model
- Present the UC operation
- State the UC theorem
- Discuss some implications
- Prove the theorem

Modular composition:
The basic idea for a single copy of f





The basic idea for multiple calls to F:



The "hybrid model" for protocol execution

The hybrid model for executing P with ideal functionality F and environment Z is the following system of interacting ITMs:

- Initial ITM:
  - Environment Z (the initial ITM, with fixed ID)
- Permission list:
  - Z can activate a (single copy of) an ITM A (adversary), and multiple ITMs running P, *all having the same SID*, and write to their input tapes.
  - A can write to the incoming comm. tapes of all parties and to the subroutine output tape of Z.
  - All other parties can write to the incoming comm. tape of A, can invoke new parties, and can write to the subroutine output tape of their invoker and the input tapes of their subroutines.
  - In addition, parties can provide inputs to and get outputs from multiple copies of F, as follows:
    - To send input x to copy of F with SID sid, when playing the role of PID pid, a party sends the input x to the dummy party for F with ID=(sid,pid).
    - Outputs from the copy of F with ID=(sid,0) are received via dummy parties for F with SID=sid (and some PID).

The “hybrid model” for protocol execution (continued)

- Modeling corruptions: A can write a “corrupt” message on incoming comm. Tape of ITM M. Then:
  - M writes “Corrupted” on subr. output tape of Z
  - From now on, in each activation M sends its entire state to Z
  - A assumes all write privileges of M.
  - Corruption messages to copies of F are treated as in the ideal process (I.e., up to the discretion of F), with the exception that the “corrupted” outputs are written directly on the subroutine tape of Z.

The composition operation: universal composition

Start with:

- Protocol Q in the F-hybrid model
- Protocol P that securely realizes F.

Construct the composed protocol Q^P:

- Each input to a dummy party for F with ID=(sid,pid) is replaced with an input to an ITM running P with ID=(sid,pid).
- Each output of the ITM (sid,pid) is treated as usual (I.e., as an output coming from a dummy party for F with ID=(sid,pid).

Note:

- In Q^P there may be multiple copies of P running concurrently.
- If P is a protocol in the real-life model then so is Q^P. If P is a protocol in the F'-hybrid model for some functionality F', then so is Q^P.

The universal composition theorem:

Let Q be a protocol that works in the F-hybrid model, and let P be a protocol that securely realizes F. Then the protocol Q^P emulates protocol Q. That is, for any t-limited adversary A there is a t-limited adversary H such that for any Z we have

$$EXEC^F_{Q,H,Z} \sim EXEC_{Qp,A,Z} .$$

Corollary: If protocol Q securely realizes functionality F'' (in the F-hybrid model) then protocol Q^P securely realizes F'' as well.

Proof: Let A be an adversary that operates against Q^P. Then since emulates there is an adversary H such that

$$EXEC^F_{Q,H,Z} \sim EXEC_{Qp,A,Z} .$$

Since Q realizes F'' there exists an adversary S such that

$$IDEAL^F_{S,Z} \sim EXEC_{Q,H,Z} .$$

The corollary follows.

Implications of the UC theorem

1. Can design and analyze protocols in a modular way:
   - Partition a given task T to simpler sub-tasks $T_1 \ldots T_k$
   - Construct protocols for realizing $T_1 \ldots T_k$.
   - Construct a protocol for T assuming ideal access to $T_1 \ldots T_k$.
   - Use the composition theorem to obtain a protocol for T from scratch.

(Now can be done concurrently and in parallel.)

Implications of the UC theorem

2. Assume protocol P securely realizes ideal functionality F. Can deduce security of P in any multi-execution protocol environment:

   As far as the collection of external protocols are concerned, interacting with multiple instances of P is equivalent to interacting with multiple copies of F.

Proof outline:

(Will use the alternative formulation of the definition: security w.r.t. the dummy adversary).

From the fact that P realizes F, we know that there exists an ideal-process adversary S such that $\text{IDEAL}^{F}_{S,Z} \sim \text{EXEC}_{Q,Ad,Z}$.

Consider the protocol Q^{P} in the real-life model.

We will construct an adversary H that interacts with protocol Q in the F-hybrid model such that no Z can tell the difference between the interaction with H,Q,F and the interaction with Ad,Q^{P}.

Then we will show that H is valid: Given an environment Z that distinguishes between the two interactions with non-negligible probability, we construct an environment Z_p that distinguishes between an interaction with P and A_d, and an interaction with F and S. (Here S is the ideal-process adversary that is guaranteed by the security of P.)

Adversary H :

The goal is to mimic the behavior of the dummy adversary A_d. This is done as follows:

– Messages sent to and from the parties of Q (and their subroutines) are forwarded to the actual parties I the F-hybrid model. (Here H behaves *exactly* like A_d.)

– Messages sent to and from each instance of P are treated as follows:

- For each instance of P, H keeps a simulated copy of S.
- All messages from Z to parties of P are forwarded to the corresponding instance of S.
- Messages generated by each instance of S are forwarded to Z.
- Messages from each instance of S to its (only) copy of F are forwarded to the corresponding copy of F.
- Messages from each copy of F in the external interaction are forwarded to the corresponding copy of S.

Analysis of H :

Assume there is an environment Z that on input z distinguishes with some probability e between a run of H with Q in the F-hybrid model and a run of A_d with Q^P in the plain real-life model.

Construct an environment Z_P that distinguishes with a related probability between a single run of S in the ideal process for F, and a run of A_d with P (in contradiction to the security of P).

We use a hybrid argument:

– Consider m "hybrid systems": in the i-th system, the first i instances of F are replaced by copies of P. (Here m is a bound on the number of instances of P with this Z.) Then:

- The 0-th hybrid system is identical to a run of Q in the F-hybrid model.
- The m-th hybrid system is identical to a run of Q^P.

➔ There exists an i such that Z distinguishes with proability e/m between an interaction with the I-th system and an interaction with the I+1st system.

Z_P uses this fact similarly to the non-concurrent case. (Details on the board.)

■

Some ideal functionalities

“standard” ideal functionalities:

An ideal functionality is called “standard” if it consists of an “outer shell” and a “main program”, with the following properties:

Allowing S to delay receiving inputs and sending outputs:

- Whenever receiving an input from party (id), the outer shell notifies S that it received input from (id). When receiving “ok” from S, the shell forwards the input to the main program.
- Whenever the main program wishes to write an output to party (id), the shell tells S that it wants to give output to (id). When receiving “ok” from S, the shell forwards the output to (id).

Dealing with corruptions:

- When S asks to corrupt party (id), hand an output “corrupted” to (id), and hand S all the inputs and outputs received/sent to (id) so far. Also, from now on take all of (id)’s inputs from S, and send S all of (id)’s outputs.

➔ From now on, functionalities are standard unless said otherwise…

Example: The authenticated message transmission functionality, F_{auth}

1. Receive input (sid,pid_s,pid_r,m) from party (sid,pid_s). Then:
   1. Output (sid,pid_s,pid_r,m) to party (sid,pid_r)
   2. Send (sid,pid_s,pid_r,m) to S
   3. Halt.

Example:
The *secure* message transmission functionality, F_{smt}

1. Receive input (sid,pid_s,pid_r,m) from party (sid,pid_s). Then:
   1. Output (sid,pid_s,pid_r,m) to party (sid,pid_r)
   2. Send $(sid,pid_s,pid_r,|m|)$ to S
   3. Halt.

Example: The key-exchange functionality F_{KE} (I)

Wait to receive:

- $(sid, pid_a, \text{“exchange”}, pid_b)$ from party (sid, pid_a)
- $(sid, pid_b, \text{“exchange”}, pid_a)$ from party (sid, pid_b)

Then:

- Choose $a \leftarrow \{0,1\}^k$
- Output (sid, pid_a, pid_b, a) to the two parties, (sid, pid_a) and (sid, pid_a)
- Send (sid, pid_a, pid_b) to S
- Halt.

Example: The key-exchange functionality F_{KE} (II)

Wait to receive:

- $(sid, pid_a, \text{“exchange”}, pid_b)$ from party (sid, pid_a)
- $(sid, pid_b, \text{“exchange”}, pid_a)$ from party (sid, pid_b)

Then:

- If one of the parties is corrupted then obtain a value a from S. Else, choose $a \leftarrow \{0,1\}^k$
- Output (sid, pid_a, pid_b, a) to the two parties, (sid, pid_a) and (sid, pid_a)
- Send (sid, pid_a, pid_b) to S
- Halt.

Example:
The ZKPoK functionality F_{zk} (for relation R),

1. Receive (sid, pid_p,pid_v,x,w) from (sid,pid). Then:
   1. Output (sid, pid_p, pid_v, x, R(x,w)) to (sid,pid)
   2. Send (sid, pid_p, pid_v, x, R(x,w)) to S
   3. Halt

Example: The commitment functionality, F_{com}

1. Upon receiving (sid,pid_p,pid_v ,"commit",x) from (sid,pid_p), do:
   1. Record x
   2. Output (sid,pid_p,pid_v, "receipt") to (sid,pid_v)
   3. Send (sid,pid_p,pid_v, "receipt") to S
2. Upon receiving (sid,"open") from (sid,pid_p), do:
   1. Output (sid,x) to (sid,pid_v)
   2. Send (sid,x) to S
   3. Halt.

Example:
The Synchronous Communication Functionality, F_{synch}
(parameterized by a set T of PIDs of participants)

1. In the first activation set round number r ←0.
2. When receiving input "report" from party (sid,pid) where pid is in T, do:
   - Output r to the party, plus all the messages addressed to it that were not yet delivered.
   - Obtain from the party a list of messages to be delivered in the next round. Send this list to S.
3. Once all the parties in T have sent their messages for this round, increment r ← r+1 and return to Step 2.