6.897: Advanced Topics in Cryptography

Lecturer: Ran Canetti

Focus for first half (until Spring Break): Foundations of cryptographic protocols

Goal: Provide some theoretical foundations of secure cryptographic protocols:

- General notions of security
- Security-preserving protocol composition
- Some basic constructions

Overall:

Definitional and foundational slant

(but also constructions, and even some efficient ones…)

Notes

- Throughout, will try to stress conceptual points and considerations, and will spend less time on technical details.
- Please interrupt me and ask lots of questions – both easy and hard!
- The plan is only a plan, and is malleable…

Lecture plan

Lecture 1 (2/5/4): Overview of the course. The definitional framework of "classic" multiparty function evaluation (along the lines of [C00]): Motivation for the ideal-model paradigm. The basic definition.

Lecture 2 (2/6/4): Variants of the basic definition. Non-concurrent composition.

Lecture 3 (2/12/4): Example: Casting Zero-Knowledge within the basic definitional framework. The Blum protocol for Graph Hamiltonicity. Composability of Zero-Knowledge.

Lecture 4 (2/13/4): The universally composable (UC) security framework: Motivation and the basic definition (based on [C01]).

Lectures 5,6 (2/19-20/4): No lecture (TCC)

Lecture 7 (2/26/4): Alternative formulations of UC security. The universal composition theorem. Survey of feasibility results in the UC framework. Problem Set 1.

Lecture 8 (2/27/4): UC commitments: Motivation. The ideal commitment functionality. Impossibility of realizations in the plain model. A protocol in the Common Reference String (CRS) model (based on [CF01]).

Lecture 9 (3/4/4): The multi-commitment functionality and realization. UC Zero Knowledge from UC commitments. Universal composition with joint state. Problem Set 1 due.

Lecture 10 (3/5/4): Secure realization of any multi-party functionality with any number of faults (based on [GMW87,G98,CLOS02]): The semi-honest case. (Static, adaptive, two-party, multi-party.)

Lecture 11 (3/11/4): Secure realization of any multi-party functionality with any number of faults: The Byzantine case. (Static, adaptive, two-party, multi-party.) The case of honest majority.

Lecture 12 (3/12/4): UC signatures. Equivalence with existential unforgeability against chosen message attacks (as in [GMR88]). Usage for certification and authentication.

Lecture 13 (3/18/4): UC key-exchange and secure channels. (Based on [CK02]).

Lecture 14 (3/19/4): UC encryption and equivalence with security against adaptive chosen ciphertext attacks (CCA). Replayable CCA encryption. (Based on [CKN03].) Problem Set 2.

Scribe for today?

What do we want from a definition of security for a given task?

- Should be mathematically rigorous
  (I.e., should be well-defined how a protocol is modeled and whether a given protocol is “in” or “out”).
- Should provide an abstraction (“a primitive”) that matches our intuition for the requirements of the task.
- Should capture “all realistic attacks” in the expected execution environment.
- Should guarantee security when the primitive is needed elsewhere.
- Should not be over-restrictive.
- Should be based on the functionality of the candidate protocol, not on its structure.
- Nice-to-haves:
  - Ability to define multiple tasks within a single framework.
  - Conceptual and technical simplicity.

What do we want from a definition of security for a given task?

- Should be mathematically rigorous
  (I.e., should be well-defined how a protocol is modeled and whether a given protocol is "in" or "out").
- Should provide an abstraction ("a primitive") that matches our intuition for the requirements of the task.
- Should capture "all realistic attacks" in the expected execution environment.
- Should guarantee security when the primitive is needed elsewhere.
- Should not be over-restrictive.
- Should be based on the functionality of the candidate protocol, not on its structure.
- Nice-to-haves:
  - Ability to define multiple tasks within a single framework.
  - Conceptual and technical simplicity.

What do we want from a definition of security for a given task?

- Should capture “all realistic attacks” in the expected execution environment. Issues include:
  - What are the network characteristics? (synchrony, reliability, etc.)
  - What are the capabilities of the attacker(s)? (controlling protocol participants? The communication links?In what ways?)
  - What are the possible inputs?
  - What other protocols are running in the same system?
- Should guarantee security when the primitive is needed elsewhere:
  - Take a protocol that assumes access to the “abstract primitive”, and let it work with a protocol that meets the definition. The overall behavior should remain unchanged.

➔ Some flavor of “secure composability” is needed already in the basic desiderata.

First candidate: The “classic” task of multiparty secure function evaluation

- We have:
  - n parties, $p_1 \ldots p_n$, n>1, where each p_i has an input value x_i in D. Some of the parties may be corrupted. (Let’s restrict ourselves to static corruptions, for now.)
  - A probabilistic function $f: D^n \times R \rightarrow D^n$.
  - An underlying communication network
- Want to design a “secure” protocol where each p_i has output $f(x_1 \ldots x_n, r)_i$ .That is, want:
  - Correctness: The honest parties get the correct function value of the parties’ inputs.
  - Secrecy: The corrupted parties learn nothing other than what is computable from their inputs and prescribed outputs.

Examples:

- $F(x_1,\ldots,x_n) = x_1+\ldots+x_n$
- $F(x_1,\ldots,x_n) = \max(x_1+\ldots+x_n)$
- $F(-,\ldots,-) = r \leftarrow_U D$
- $F((x_0,x_1),b) = (-,x_b)$ (b in $\{0,1\}$)
- $F_R((x,w),-) = (-,(x,R(x,w)))$ (R(x,w) is a binary relation)

…

- But, cannot capture “reactive” tasks (e.g., commitment, signatures, public-key encryption…)

How to formalize?

How to define correctness?

Question: Based on what input values for the corrupted parties should the function be computed?
(ie, recall: P_i should output $f(x_1 \ldots x_{n,}, r)_i$. But what should be the x's of the corrupted parties?)

- If we require that f is computed on input values fixed from above then we get an unrealizable definition.
- If we allow the corrupted parties to choose their inputs then we run into problems.

  Example:

  Function: $f(x_1,x_2)=(x_1+x_{2,}, x_1+x_2)$.
  Protocol: P_1 sends x_1 to P_2 . P_2 sends $x_1+x_{2,}$ back.
  The protocol is both "correct" and "secret". But it's not secure…

➔ Need an "input independence" property, which blends secrecy and correctness…

How to formalize?

How to define secrecy?

An attempt: “It should be possible to generate the view of the corrupted parties given only their inputs and outputs.”

Counter example:

Function: $F(-,-) = (r \leftarrow_U D, -)$

Protocol: P_1 chooses $r \leftarrow_U D$, and sends r to P_2 .

The protocol is clearly not secret (P_2 learns r). Yet, it is possible to generate P_2 ‘s view (it’s a random bit).

➔ Need to consider the outputs of the corrupted parties together with the outputs of the uncorrupted parties. That is, correctness and secrecy are again intertwined.

The general definitional approach

[Goldreich-Micali-Wigderson87]

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

- Several formalizations exist (e.g. [Goldwasser-Levin90, Micali-Rogaway91, Beaver91, Canetti93, Pfitzmann-Waidner94, Canetti00, Dodis-Micali00,…])
- I’ll describe the formalization of [Canetti00]

  (in a somewhat different presentation).

Presenting the definition:

- Describe the model for protocol execution (the “real life model”).
- Describe the ideal process for evaluating a function with a trusted party.
- Describe the notion of “emulating an ideal process”.

I’ll describe the definition for the case of:

- Synchronous networks
- Active (Byzantine) adversary
- Static (non-adaptive) adversary
- Computational security (both adversary and distinguisher are polytime)
- Authenticated (but not secret) communication

Other cases can be inferred…

Some preliminaries:

- Distribution ensembles:

  A distribution ensemble $D = \{D_{k,a}\}$ (k in N, a in $\{0,1\}^*$)

  is a sequence of distributions, one for each value of k,a .
  We will only consider binary ensembles,

  i.e. ensembles where each $D_{k,a}$ is over $\{0,1\}$.

- Relations between ensembles:
  - Equality: $D=D'$ if for all k,a, $D_{k,a} = D'_{k,a}$.
  - Statistical closeness: $D \sim D'$ if for all $c,d>0$ there is a k_0 such that for all $k> k_0$ and all a with $|a|< k^d$ we have

    $$\text{Prob}[x \leftarrow D_{k,a}, x=1] - \text{Prob}[x \leftarrow D'_{k,a}, x=1] < k^{-c} .$$

- Multiparty functions:

  An n-party function is a function $f: N \times R \times (\{0,1\}^*)^{n+1} \rightarrow (\{0,1\}^*)^{n+1}$

- Interactive Turing machines (ITMs):

  An ITM is a TM with some special tapes:
  - Incoming communication tape
  - Incoming subroutine output tape
  - Identity tape, 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.
- Systems of interacting ITMs (Fixed number of ITMs):
  - A system of interacting ITMs is a set of ITMs, one of them the initial one, plus a set of “writing permissions”.
  - A Run of a system $(M_0 \ldots M_m)$:
    - The initial ITM M_0 starts with some external input.
    - In each activation an ITM may write to tapes of other ITMs.
    - The ITMs whose tapes are written to enter a queue to be activated next .
    - The output is the output of the initial ITM M_0.
- Multiparty protocols:

  An n-party protocol is a sequence of n ITMs, $P=(P_1 \ldots P_n)$.

The “real-life model” for protocol execution

A system of interacting ITMs:

- Participants:
  - An n-party protocol $P=(P_1 \ldots P_n)$. (any $n>1$)
  - Adversary A, controlling a set B of “bad parties” in P. (ie, the bad parties run code provided by A)
  - Environment Z (the initial ITM)
- Computational process:
  - Z gets input z
  - Z gives A an input a and each good party P_i an input x_i
  - Until all parties of P halt do:
    - Good parties generate messages for current round.
    - A gets all messages and generates messages of bad parties.
    - A delivers the messages addressed to the good parties.
  - Before halting, A and all parties write their outputs on Z’s subroutine output tape.
  - Z generates an output bit b in {0,1}.

- 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, ie, r= r_Z ,r_A ,r_1 …r_n.)
  - $EXEC_{P,A,Z}$ (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 in N, z in {0,1}*)

The ideal process for evaluation of f:

Another system of interacting ITMs:

- Participants:
  - “Dummy parties” $P_1 \ldots P_n$.
  - Adversary S, controlling the “bad parties” P_i in B.
  - Environment Z
  - A “trusted party” F for evaluating f
- Computational process:
  - Z gets input z
  - Z gives S an input a and each good party P_i an input x_i
  - Good parties hand their inputs to F
  - Bad parties send o F whatever S says. In addition, S sends its own input.
  - F evaluates f on the given inputs (tossing coins if necessary) and hands each party and S its function value. Good parties set their outputs to this value.
  - S and all parties write their outputs on Z’s subroutine output tape.
  - Z generates a bit b in {0,1}.

- 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 in N, z in {0,1}*)

- Notation:
  - Let **B** be a collection of subsets of {1..n}. An adversary is **B**-limited if the set B of parties it corrupts is in **B**.

Definition of security:

Protocol P **B**-emulates the ideal process for f if for any **B**-limited adversary A there exists an adversary S such that for all Z we have:

$$\mathrm{IDEAL}^{f}_{S,Z} \sim \mathrm{EXEC}_{P,A,Z} \, .$$

In this case we say that protocol P **B**-securely realizes f.

In other words: "Z cannot tell with more than negligible probability whether it is interacting with A and parties running P, or with S and the ideal process for f."

Or: "whatever damage that A can do to the parties running the protocol can be done also in the ideal process."

This implies:

- Correctness: For all inputs the good parties output the “correct function value” based on the provided inputs
- Secrecy: Whatever A computes can be computed given only the prescribed outputs
- Input independence: The inputs of the bad parties are chosen independently of the inputs of the good parties.

Equivalent formulations:

- 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.

Variants

- Passive (semi-honest) adversaries: The corrupted parties continue running the original protocol.
- Secure channels, unauthenticated channels:
  Change the "real-life" model accordingly.
- 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.
- Adaptive security: Both A and S can corrupt parties as the computation proceeds. Z learns about corruptions.
  Some caveats:
  - What information is disclosed upon corruption?
  - For composability, A and Z can talk at each corruption.

On protocol composition

So far, we modeled “stand-alone security”:

- Only a single execution of a single protocol
- No other parties, no other network activity

What about security “in conjunction with other protocol executions”?

- Other executions of the same protocol?
- Other executions of arbitrary other protocols?
- “Intended” (coordinated) executions?
- “unintended” (uncoordinated) executions?

Examples

- Composition of instances of the same protocol:
  - With same inputs/different inputs
  - Same parties/different parties/different roles
  - Sequential, parallel, concurrent (either coordinated or uncoordinated).
- "Subroutine composition" (modular composition): protocol Q calls protocol P as subroutine.
  - Non-concurrent, Concurrent
- General composition: Running in the same system with arbitrary other protocols (arbitrary network activity), without coordination.

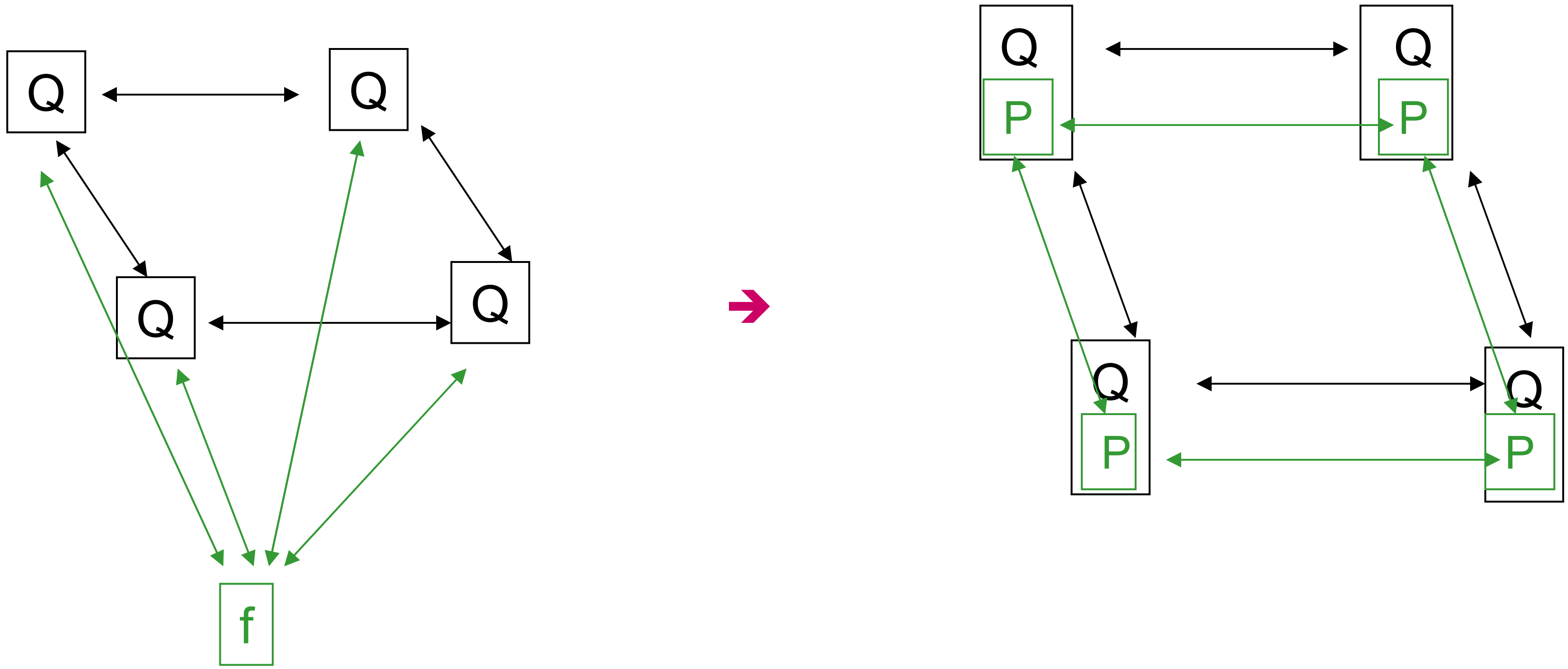
Is security maintained under these operations?

Examples

- Composition of instances of the same protocol:
  - With same inputs/different inputs
  - Same parties/different parties/different roles
  - Sequential, parallel, concurrent (either coordinated or uncoordinated).
- **“Subroutine composition” (modular composition): protocol Q calls protocol P as subroutine.**
  - **Non-concurrent**, Concurrent
- General composition: Running in the same system with arbitrary other protocols (arbitrary network activity), without coordination.

Is security maintained under these operations?

Modular composition: The basic idea



Towards the composition theorem

The hybrid model with ideal access to func. f (the f-hybrid model):

- Start with the real-life model of protocol execution.
- In addition, the parties have access to a trusted party F for f:
  - At pre-defined rounds, the protocol instructs all parties to sends values to F.
  - F evaluates f on the given inputs and hands outputs to parties
  - Once the outputs are obtained the parties proceed as usual.
- Notation: $\mathrm{EXEC}^{f}_{P,H,Z}$ is the ensemble describing the output of Z after interacting with protocol P and adversary H in the f-hybrid model.

Note:

- During the “ideal call rounds” no other computation takes place.
- Can generalize to a model where in each “ideal call round” a different function is being evaluated. But doesn’t really add power (can use a single universal functionality).

The composition operation: Modular composition

(Originates with [Micali-Rogaway91])

Start with:

- Protocol Q in the f-hybrid model
- Protocol P that securely realizes f

Construct the composed protocol Q^P:

- Each call to f is replaced with an invocation of P.
- The output of P is treated as the value of f.

Notes:

- In Q^P, there is at most one protocol active (ie, sending messages) at any point in time: When P is running, Q is suspended.
- It is important that in P all parties terminate the protocol at the same round. Otherwise the composition theorem does not work…
- 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 function f', then so is Q^P.

The non-concurrent modular composition theorem:

Protocol Q^P “emulates” protocol Q. That is:
For any **B**-limited adversary A there is a **B**-limited adversary H such that for any Z we have $\mathrm{EXEC}^f_{Q,H,Z} \sim \mathrm{EXEC}_{Qp,A,Z}$.

Corollary: If protocol Q t-securely realizes function f’’ (in the f-hybrid model) then protocol Q^P t-securely realizes f’’ (in the plain real-life model)$_{,}$.

Proof outline:

Let’s restrict ourselves to one subroutine call.

We have a **B**-limited adversary A that interacts with protocol Q^P in the real-life model.

We want to construct an adversary H that interacts with protocol Q in the f-hybrid model such that no Z can tell the difference between the two interactions.

We proceed In three steps:

1. Out of A, we construct an adversary A_P that interacts only with protocol P.
2. From the security of P, there is an adversary S_P in the ideal process for f such that $\mathrm{IDEAL}^f_{Sp,Z} \sim \mathrm{EXEC}_{P,A,Z}$.
3. Out of A and S we construct adversary H, and show that $\mathrm{EXEC}^f_{P,H,Z} \sim \mathrm{EXEC}_{Qp,A,Z}$.

Adversary A_P :

- Expect the input (coming from Z) to contain an internal state of A at the beginning of the round where protocol Q^P calls P. (If input is in the wrong format then halt.)
- Run A from this state, while interacting with parties running P.
- At the end of the run, output the current state of A.

From the security of P we have that there is an adversary S_P such that $\mathrm{IDEAL}^f_{Sp,Z} \sim \mathrm{EXEC}_{P,A,Z}$.

Note: Here it is important that the input of A_P is general and not only the inputs of the bad parties to the function.

Adversary H :

- Until the round where the parties in Q call f, run A. (Indeed, up to this point the two protocols are identical.)
- At the point where Q calls f, run S_P:
  - Play Z for S_P, and give it the current state of A as input.
  - When S_P generates f-inputs, forward these inputs to f.
  - Forward the outputs obtained from f to S_P .
- Once S_P generates its output, continue running A from the state that appears in the output of S_P.
- Halt when A halts, and output whatever A outputs.

Analysis of H :

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

Construct an environment Z_P that, on input z, distinguishes with the same probability between a run of S_P in the ideal process for f, and a run of A_P with P (in contradiction to the security of P).

Environment Z_P (on input z):

- Run Z on input z, and orchestrate for Z an interaction with parties running Q^P and with adversary A.
- At the round when P is called, start interacting with the external system:
  - Give to the external good parties the inputs that the simulated good parties would give to P.
  - Give the current state of A to the external adversary
- When the external outputs are generated, continue the simulated interaction between A and the parties running Q^p: the good parties use their outputs from the external system as the outputs of P, and A runs from the state in the output of the external adversary.
- When the internal outputs are generated, hand them to Z and outputs whatever Z outputs.

Analysis of Z_P :

Can verify:

- If the “external system” that Z_P interacts with is an ideal process for f with adversary S_P then the simulated Z sees exactly an interaction with H and Q in the f-hybrid model.
- If the “external system” that Z_P interacts with is an execution of P with adversary A_P then the simulated Z sees exactly an interaction with A and Q^P in the plain real-life model.

Thus, Z_P distinguishes with the same probability that Z distinguishes.

■

Implication of the theorem

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.

(Analogous to subroutine composition for correctness of programs, but with an added security guarantee.)