Jointly Restraining Big Brother:

Using cryptography to reconcile privacy with data aggregation

Ran Canetti
IBM Research

Privacy-sensitive interactions

The basic problem: Parties want to perform some joint computation while preserving privacy of local data.

Examples:

- Elections
- Obtaining statistical data on private records, e.g.:
  - Medical records
  - Shopping patterns and preferences
  - Whereabouts and travel patterns of individuals
- Pooling information from different sources

A general approach for solution:

1. Formalize the required functionality in terms of a “centralized trusted service”.
2. Run a cryptographic protocol that realizes the “centralized trusted service” functionality.
   - Can use a generic construction (typically inefficient)
   - Can design more efficient protocols for a given trusted-service.

The “trusted service” solution

- Assume all parties have “ideally secure channels” to an incorruptible trusted party.
- The trusted party processes inputs coming from the parties and provides the desired outputs.

Note: Trusted party can be *reactive:* Can get inputs and generate outputs throughout the computation.

Example: Elections

Tasks of trusted party:

- Receives votes, verifies credentials
- Publicizes tallies, required statistics
- Revokes privacy of misbehaving individuals
- …

Example: Medical records

Tasks of trusted party:

- Obtains full records from individuals and doctors
- Provides full information on records with authorization by individual
- Provides statistical information on records (possibly limited/perturbed)
- Allows pooling some information with other depositories
- …

Challenges (I):

- Specification design (write the trusted party code): Exactly what is revealed and when?
  - What aggregates are “ok”, what perturbations
  - When to revoke identity, how much to revoke
  - How to resolve disputes
  - …

That’s the “non-cryptographic” part. Often hardest…

(But can assume a trusted party!)

Challenges (II):

- Efficiency of the cryptographic solution:
  - Communication patterns:

    Are third parties involved? Which parties need to be on-line?
  - Communication complexity: rounds, bandwidth, etc.
  - Computational complexity
- Security of the solution:
  - Based on what assumptions?
  - What security properties are guaranteed?

Stand-Alone Security

- Security is interpreted as “emulating the trusted service solution” [GMW87]: “Whatever damage that can be done to the protocol could have been done to the trusted party solution”.

However:

- The “classic” formalizations of this intuitive notion (e.g. [GL90,MR91,B91,C95,C00]) guarantee security only when a single protocol execution takes place at any time.
- In contrast, in today’s networks:
  - Multiple copies of a protocol may be running concurrently
  - A protocol is run concurrently with other protocols
  - Parties may be unaware of other executions, protocols, parties.

Stand-alone security does not suffice!

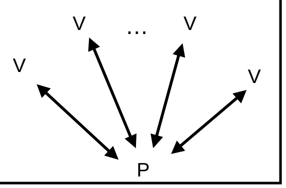
Example: Concurrent Zero-Knowledge

[F90,DNS98]

– Original notion of ZK [GMR85] does not guarantee security when the prover interacts with many verifiers concurrently.

– Best known solution: O(log n) rounds
  [RK99,PRS02]

– Lower bound of (log n) rounds
  (for black-box simulation)
  [CKPR01]



Example: Malleability of commitments

[DDN91]

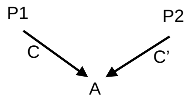
Stand-alone notions do not guarantee “independence” among committed values.

Example: Malleability of commitments

[DDN91]

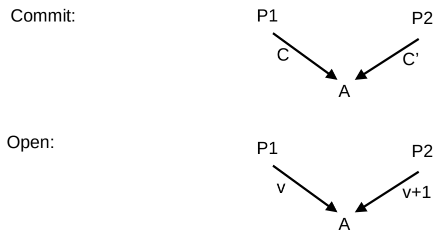
Stand-alone notions do not guarantee “independence” among committed values.

Commit:



Example: Malleability of commitments
[DDN91]

Stand-alone notions do not guarantee “independence” among committed values.



How to guarantee security in complex protocol environments?

Traditional approach: keep writing more sophisticated definitions, that capture more scenarios…

- Ever more complex
- No guarantee that “we got it all”.
- No general view

An alternative approach:

- Prove security of a protocol as stand-alone (single execution, no other parties).
- Use a general **secure composition theorem** to deduce security in arbitrary execution environments.

Universally Composable Security [C01]

Provides a framework where:

1. Can capture the security requirements of practically any cryptographic task.
2. Can prove a general, “universal composition” theorem that:
   - Guarantees security in arbitrary multi-protocol, multi-execution environments.
   - Enables modular design and analysis of protocols.

The composition operation

(Originates with [MR91])

Start with:

- Protocol ρ^F that uses ideal calls to a “trusted party” F
- Protocol π that “emulates” F

Construct the composed protocol ρ^{π}:

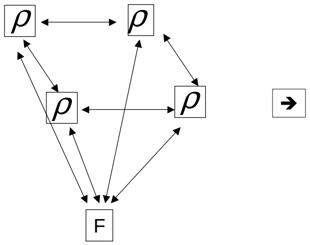
- Each call to F is replaced with an invocation of π.
- Each value returned from π is treated as coming from F.

Note: In ρ^F parties may call many copies of F.

➔ In ρ^{π} many copies of π run concurrently.

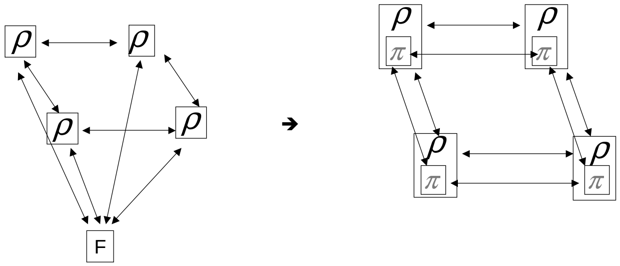
The composition operation

(single call to F)



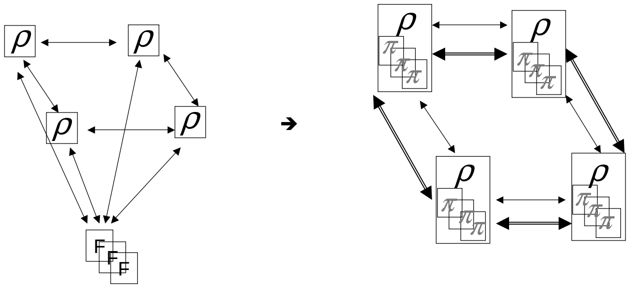
The composition operation

(single call to F)



The composition operation

(multiple calls to F)



The universal composition (UC) theorem:

Protocol ρ^{π} “emulates” protocol ρ^{F}.

(That is, for any adversary A there exists an adversary A` such that no Z can tell whether it is interacting with ( , A) or with ( $^{F}$,A`).)

Corollary: If ρ^{F} securely realizes functionality G then so does ρ^{π} .

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.

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

Implications of the UC theorem

2. Assume protocol π "emulates" a trusted service F. Can deduce security of π in any multi-execution environment:

As far as the "rest of the network" is concerned, interacting with (multiple copies of) π *is equivalent to interacting with (multiple copies of) F.*

Questions:

- do
- Are known protocols UC-secure?

  (Do these protocols "emulate" the trusted services associated with the corresponding tasks?)

- How to design UC-secure protocols?zcyk02]

Existence results: Honest majority

Thm: Can realize *any trusted service* in a UC way.

(e.g. use the protocols of [BGW88, RB89,CFGN96]).

Usages:

- All parties actively participate in computation
- Use a set of servers to realize the trusted service (secure as long as only a minority is corrupted).

What if there is no honest majority?
(e.g., two-party protocols)

- Known protocols (e.g., [Y86,GMW87]) do not work.
  (“black-box simulation with rewinding” cannot be used).
- Many interesting functionalities (commitment, ZK, coin tossing, etc.) cannot be realized in plain model.
- In the “common random string model” can do:
  - UC Commitment, UC Zero-Knowledge
    [CF01, DDOPS01,CLOS02, DN02, DG03]
  - Emulate any trusted service [CLOS02]

The [GMW87] paradigm:

1) Construct a protocol secure against *semi-honest* adversaries (who follow the protocol specification):
   - -Represent the “trusted party code” as a Boolean circuit (state represented as “feedback lines”)
   - -Each party shares its input among all others (using a simple sum scheme)
   - -The parties evaluate the circuit gate by gate. Each gate evaluation needs 1-out-of-4 oblivious transfer between any pair of parties.
   - -Output lines are revealed to the corresponding parties. Shares of “feedback lines” kept.

-Works even in the UC model.

The [GMW87] paradigm:

1) F
2) Construct a *compiler* that transforms protocols secure in the semi-honest model to protocols secure against malicious adversaries.

[GMW87] Protocol Compilation

- Aim: force the malicious parties to follow the protocol specification.
- How?
  - Parties **commit** to inputs
  - Parties **commit** to *uniform* random tapes (use secure coin-tossing to ensure uniformity)
  - Parties use **zero-knowledge** protocols to prove that every message sent is according to the protocol (and consistent with the committed input and random-tape).

Constructing a UC "[GMW87] compiler"

- Problem: In [GMW87], both commitment and ZK are not UC.
- First attempt: Replace commitment and ZK with UC counterparts.

Constructing a UC "[GMW87] compiler"

- Problem: In [GMW87], both commitment and ZK are not UC.
- First attempt: Replace commitment and ZK with UC counterparts.
  - Doesn't work… (cannot make ZK proofs on "ideal commitments")

The "Commit-and-Prove" primitive

- Define a single primitive where parties can:
  - Commit to values
  - Prove "in ZK" statements regarding the committed values
- Can realize "C&P" in the CRS model (using UC commitment and UC ZK).
- Given access to ideal "C&P", can do the [GMW87] compiler *without computational assumptions.*

To sum up:

- Can "emulate" any trusted service in a universally composable way, with any number of faults.
- Main problem: Solution is typically very inefficient (to the point of being unrealistic)…

Application to privacy

- Any privacy problem that has a “trusted service” solution is solvable in principle.
- Challenges:
  - Good specification of the “trusted privacy service.”
  - More realistic protocols.