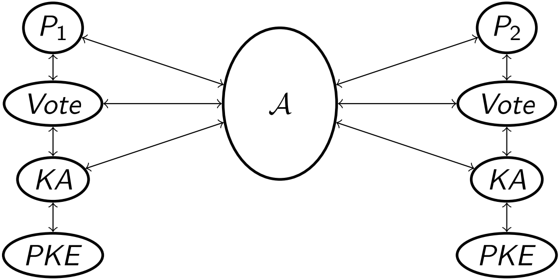
Introduction to Universally Composable Security

Olivier Pereira

Cosyproofs – April 2009

Protocol composition...



- Protocols usually do not execute alone
- Is security proven in a stand-alone setting preserved under composition?
- Are security definitions proposed in stand-alone setting useful under composition?

Secure Function Evaluation (SFE)

"Most general" protocol problem (due to [Yao82]):

- Parties $P_1, \ldots, P_n$, each having an input x_i
- Each P_i wants $y_i = f_i(x_1, \ldots, x_n)$
- The protocol gives y_i to P_i and nothing more

Examples (P_2 takes role of adversary when needed):

- Authentic communication: $(-, m, m) = f(m, -, -)$
- Secure communication: $(-, ||m||, m) = f(m, -, -)$
- Key agreement: $(k, -, k) = f(-, -, -)$
- Vote: $(\sum_{i=1}^{n} v_i, \ldots, \sum_{i=1}^{n} v_i) = f(v_1, \ldots, v_n)$
- ...

(Limitation: Does not directly capture functions keeping an internal state between multiple activations)

Protocol Specification

Most simple case (conceptually):

- Two parties evaluate a function
- Authentic communications
- One party can be malicious

Requirements:

- Correctness: P_i receives $f_i(x_1, x_2)$
- Privacy: no party learns about the other party input

Protocol Specification

Probably not that simple:

Consider $(x_1 \oplus x_2, x_1 \oplus x_2) = f(x_1, x_2)$

- No privacy if we want correctness
- Suppose P_1 sends x_1 to P_2.
  P_2 can then fix $x_1 \oplus x_2$ the way he wants!

Requirements:

- Correctness: each party receives $f_i(x_1, x_2)$
- Privacy: no party learns about the other party input
- Input independence: no party should be able to choose his input as a function of the other party input

Protocol Specification

Probably not that simple:

Consider $(r, r) = f(-, -)$, with random $r \in QR_n$

- No privacy needed: no input!
- Suppose P_1 selects random $x \in [1, n]$, and sends $r = x^2 \mod n$.
- Correctness ok, but P_1 knows a secret trapdoor information on r: its SQRT!

Requirements:

- Correctness: each party receives $f_i(x_1, x_2)$
- Privacy: no party learns about the other party input
- Input independence
- Output computation process should be controlled

Protocol Specification

Requirements:

- Correctness: each party receives $f_i(x_1, x_2)$
- Privacy: no party learns about the other party input
- Input independence
- Output computation process should be controlled
- ...

Two approaches:

1. problem specific
2. general framework

Problem Specific

Two approaches:

1. problem specific

Example: Authenticated key exchange [BR93, BR95, ...]

- Parties interact on a network controlled by $\mathcal{A}$
- $\mathcal{A}$ can decide to output a *test* query to a party
  - which has not been corrupted
  - s.t. no matching participant was corrupted
- coin b is flipped
  - if $b = 0$ then session key is sent to $\mathcal{A}$
  - if $b = 1$ then random key is sent to $\mathcal{A}$
- $\mathcal{A}$ has to guess b with non negligible probability

Problem Specific

Two approaches:

1. problem specific

Pros:

- easy to manipulate

Cons:

- Errors can be dangerous:
    - security says what $\mathcal{A}$ cannot do, not what the protocol should do

  $\Rightarrow$ risk to forget giving some power to $\mathcal{A}$
- Security and communication models interleave

General Framework

Two approaches:

2. general framework

Example: [Yao86, GMW87, . . . , Can01, PW01, . . .]

- describe the protocol task (e.g., function to evaluate)
- prove that a protocol realizes that task, in some fixed communication, corruption, . . . models

General Framework

Two approaches:

2. general framework

Pros:

- task definition and security separated
- unified framework for all protocol tasks
- typically on the safe side
    - forgetting things makes the protocol "too secure"

Cons:

- More complex to handle...
- Specifications can be too strong

Two-Party Tasks

Two-Party Tasks:

- Two parties evaluate a function
- Authentic communications
- One is malicious

Security Definition

What do we want?

- In an ideal world:
  - A trusted component $\mathcal{F}$ is available for evaluating f
  - Parties (P_i and $\mathcal{A}$) give it their inputs
  - $\mathcal{F}$ returns the result
- a protocol is secure if it *emulates* this behavior

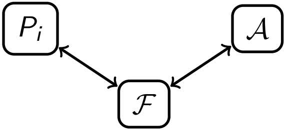
Motivation:

- seems a natural way to say what we want
- seems to capture everything we discussed:
  - correctness, privacy
  - no party can use the other party input
  - no way to learn more than the output

Ideal World

Ideal world behavior:

- Assume a trusted ITM $\mathcal{F}$ computing f
- Parties (P_i and $\mathcal{A}$) give it their inputs
- $\mathcal{F}$ returns the result



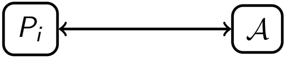
By definition:

- Every behavior of $\mathcal{A}$ in IW is harmless

Real World

Real world behavior:

- No trusted party
- P_i and $\mathcal{A}$ interact



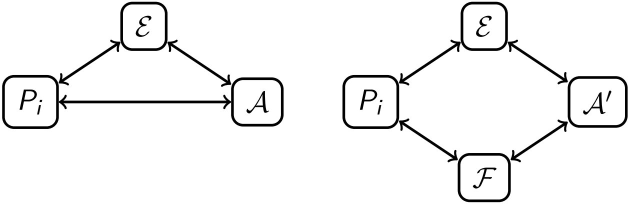
Security definition:

- Real-world protocol is secure if it *emulates* ideal behavior
- $\forall \mathcal{A}$ in RW, $\exists \mathcal{A}'$ in IW: behaviors of two systems cannot be distinguished

Security Definition

Security definition:

- $\forall \mathcal{A}$ in RW, $\exists \mathcal{A}'$ in IW: behaviors of two systems cannot be distinguished
- we need one guy to check this indistinguishability
- indistinguishability should hold $\forall$ inputs!
  (i.e., even adversarially chosen!)



- $\forall \mathcal{A}$ in RW, $\exists \mathcal{A}'$ in IW:
  no $\mathcal{E}$ can distinguish the 2 worlds

Security Definition

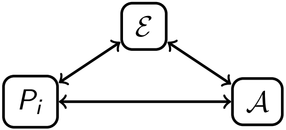
Security definition:

- $\forall \mathcal{A}$ in RW, $\exists \mathcal{A}'$ in IW: no $\mathcal{E}$ can distinguish RW/IW

How do we play this?

In Real World:

1. $\mathcal{E}$ sends whatever input he wants to P_i and $\mathcal{A}$
2. P_i and $\mathcal{A}$ play the protocol
3. P_i and $\mathcal{A}$ send their output to $\mathcal{E}$
4. $\mathcal{E}$ outputs a bit



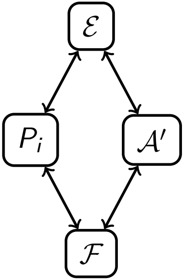
Security Definition

Security definition:

- $\forall \mathcal{A}$ in RW, $\exists \mathcal{A}'$ in IW: no $\mathcal{E}$ can distinguish RW/IW

In Ideal World (**attempt**):

1. $\mathcal{E}$ sends whatever input he wants to P_i and $\mathcal{A}'$
2. P_i forwards input to $\mathcal{F}$
3. $\mathcal{A}'$ sends something to $\mathcal{F}$
4. $\mathcal{F}$ sends output to P_i and $\mathcal{A}'$
5. $\mathcal{A}'$ and P_i send result to $\mathcal{E}$
6. $\mathcal{E}$ outputs a bit



Potentially too strong: in this IW, P_i always provide an output, while $\mathcal{A}$ is typically able to make the protocol fail

Security Definition

Security definition:

- $\forall \mathcal{A}$ in RW, $\exists \mathcal{A}'$ in IW: no $\mathcal{E}$ can distinguish RW/IW

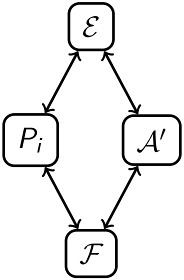
In Ideal World:

1. $\mathcal{E}$ sends whatever input he wants to P_i and $\mathcal{A}'$
2. P_i forwards input to $\mathcal{F}$
3. $\mathcal{A}'$ sends something to $\mathcal{F}$

$\Rightarrow$ 4. $\mathcal{F}$ sends his output to $\mathcal{A}'$

$\Rightarrow$ 5. When $\mathcal{A}'$ says *ok*, $\mathcal{F}$ sends his output to P_i

4. $\mathcal{A}'$ and P_i send result to $\mathcal{E}$
5. $\mathcal{E}$ outputs a bit



Security Definition

Security definition:

- $\forall \mathcal{A}$ in RW, $\exists \mathcal{A}'$ in IW: $\forall \mathcal{E}$:

$$\textsc{Exec}(P_i, \mathcal{A}, \mathcal{E}) \approx \textsc{Exec}(\mathcal{F}, \mathcal{A}', \mathcal{E})$$

Observations:

- $\mathcal{E}$ outputs a single bit
- $\mathcal{E}$ takes behavior of P_i into account
- $\mathcal{E}$ can decide to send P_i's input to $\mathcal{A}$
- Asymmetric definition: not every $\mathcal{A}'$ needs to be matched!
- $\mathcal{A}'$ controls if P_i receives its output: no fairness!
- Any notion of indistinguishability can be chosen...

Adversary vs. Simulator

Security definition:

- $\forall \mathcal{A}$ in RW, $\exists \mathcal{A}'$ in IW: $\forall \mathcal{E}$:

$$\textsc{Exec}(P_i, \mathcal{A}, \mathcal{E}) \approx \textsc{Exec}(\mathcal{F}, \mathcal{A}', \mathcal{E})$$

Observations:

- $\mathcal{A}$ does no harm proved by saying harmless $\mathcal{A}'$ can do the same thing
- $\mathcal{A}'$ simulates the real-world execution with $\mathcal{A}$
- $\mathcal{A}'$ usually called *Simulator* $\mathcal{S}$

General Tasks

Several limitations until now:

- Two party vs. multi-party (unbounded)
- Adversary is a party vs. Protocols played against network
- One-shot tasks vs. reactive tasks
- $\mathcal{A}$ ignores whatever might have happened in the rest of the world during protocol execution

General Tasks

We need a more general model:

- Security definitions look ok
- We need more general protocol tasks and interactions

General model:

- Multi-party: just allow more parties
- Protocols against network: all network communications go through $\mathcal{A}$
- Reactive tasks: $\mathcal{F}$ can be any process
  ($\mathcal{F}$ can leak information to $\mathcal{A}$, guarantee fairness, ...)
- Concurrent execution: $\mathcal{A}$ interacts freely with $\mathcal{E}$

Execution Model

Execution: Same process in real and ideal world

- $\mathcal{E}$ creates as many parties it wants, interacts freely with them
- Parties and $\mathcal{A}$ interact freely through network
- $\mathcal{A}$ interacts freely with $\mathcal{E}$ through I/O channel
- $\mathcal{E}$ outputs a bit

Observations:

- $\mathcal{A}$ very powerful: controls all communications between parties! Can be mitigated:
  - by adding appropriate functionalities, or
  - by adding some constraints on $\mathcal{A}$
- $\mathcal{F}$ and $\mathcal{A}$ can interact freely
  - can be used for authorized leakages
  - can be used for regulating timing

Example: key exchange

Key exchange $\mathcal{F}_{KE}$:

1. Upon input $(Initiate, I, R)$ from I,
   - record (I, R)
   - send public delayed output $Initiate, I$ to R
2. Upon input *Respond* from R,
   - send respond to $\mathcal{S}$
3. Upon input $(Corrupt, P)$ from $\mathcal{S}$,
   - Update $Corrupted := Corrupted \cup P$
4. Upon input $(Key, P, \tilde{k})$ from $\mathcal{S}$,
   - If no recorded key, generate random k
   - If $Corrupted \neq \emptyset$, send $\tilde{k}$, else k to P

Observations:

- More tricky! But forgetting things is "harmless" here...

Protocol Composition

How do protocols behave when composed with other protocols?

Different composition modes:

- Timing:
  - sequential, non-concurrent, parallel, concurrent
- Protocol
  - Self-composition, general composition
- Number of executions
  - Constant, polynomial, unbounded
- State relation
  - Separate states, joint states
- Inputs
  - Same inputs, fixed inputs, adaptively chosen inputs

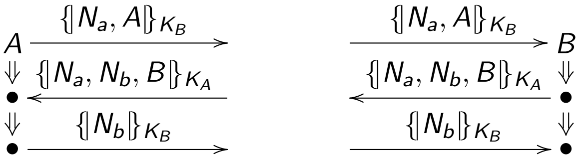
Does composition preserve local security?

- We want ideal-world behavior preserved under composition

Possible Problems

Key Exchange:

- Suppose KE produces a key used to encrypt m_0 or m_1
- One-time pad should be ok for encryption!
- Needham-Schroeder-Lowe public-key protocol



- N_a and N_b could be secret keys
- Suppose A sends $N_b \oplus m_i$ to B
- Attacker can make a guess on N_b, test using last message, and check for error signal from B

Universal Composition

Universal composition :

- Suppose ρ is a protocol that uses functionality ϕ
- Suppose π is a protocol with same interface as ϕ
- $\rho^{\pi/\phi}$ is the operation that replaces all instances of ϕ with instances of π

- Essentially: procedure call in programs
- Can be used to cover all composition cases
  (Just as adversarial control of network can be used for all variants)
- E.g., Sequential composition is ρ restricted to sequential calls

Universally Composable Security

Universal composition theorem:

- Suppose π emulates ϕ. Then $\rho^{\pi/\phi}$ emulates ρ.

Proof idea:

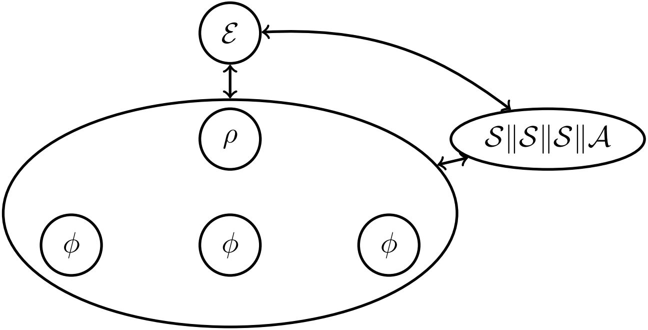
- Fix any $\mathcal{E}$ and $\mathcal{A}$ for ρ
- ρ can invoke at most $p(k)$ instances of π
- Suppose $\mathcal{S}$ is simulator for π with transparent forwarding adversary
- ...

Universally Composable Security

Universal composition theorem:

- Suppose π emulates ϕ. Then $\rho^{\pi/\phi}$ emulates ρ.

Sketch (8):



Conclusions. . .

Real world / Ideal world paradigm:

- comes with strong composition theorems
  useful for sophisticated protocols, abstraction, . . .
- provides a way to separate security from communication and computation modeling issues

Conclusions...

Communication and computation models remain a central challenge
([Can01, PW01, MMS03, BPW04, PS04, Can05, HUMQ05, CCK+06, Küs06, HUMQ08, . . .])

- Communication:
  - Can we stick to purely probabilistic protocol executions?
  - If one allows nondeterminism, what should the scheduler know?
- Computation:
  - Polynomial. . . on the life time? per activation?
  - Polynomial in what?
    security parameter? + inputs from $\mathcal{E}$? + . . . ?
  - Polynomial time. . . worst-case, average-case, expected?
  - Polynomial time. . . if $\mathcal{F}$ is perfectly secure, can we use super-polynomial simulators?

Conclusions...

Is universal composition what we really want?

- All instances of π have their own state
- Protocol instances often share state variables (long-term keys, ...)
- We need composition with *joint states*!

Further Readings...

- *Security and Composition of Cryptographic Protocols: A Tutorial*, by Ran Canetti
  http://eprint.iacr.org/2006/465
- *Multiparty Computation, an Introduction*, by R. Cramer, I. Damgård and J. Nielsen
  http://www.daimi.au.dk/~ivan/mpc.pdf
- *Universally composable security: a new paradigm for cryptographic protocols*, by Ran Canetti
  http://eprint.iacr.org/2000/067
- *Compositional Security for Task-PIOAs*, by R. Canetti, L. Cheung, D. Kaynar, N. Lynch, O. Pereira
  http://www.dice.ucl.ac.be/crypto/task-pioa/index.php