Expressing Security Properties in CSP

- Security properties: the goals that a protocol is meant to satisfy, relatively to specific kinds and levels of threat – the intruders and their capabilities

- We will consider the following security properties:
  - **Secrecy**
    - messages, keys, etc. have not become known
  - **Authentication**
    - Guarantees about the parties involved in the protocol
  - **Non-repudiation**
    - Evidence of the involvement of the other party
  - **Anonymity**
    - Protecting the identity of agents wrt particular events

Anonymity

- We will model events as consisting of two components: the event itself, x, and the identity of the agent performing the event, a

$$a.x$$

- AnUsers: the users who want to remain secret
- Given x, define $A = \{ a.x \mid a \in \text{AnUsers} \}$

- **Definition:** A protocol described as a CSP system P provides anonymity if an arbitrary permutation of the events in A, applied to all the traces of P, does not alter the set of all possible traces of P

Anonymity

- **Traces of a process:** the sequences of visible actions in all possible runs

- Example: a -> b -> Stop ||| c -> d -> Stop

Traces: a.b.c.d   a.c.b.d   c.a.b.d   a.c.d.b   c.a.d.b   c.d.a.b

- Example: a -> b -> c -> Stop $||_{\{b\}}$ d -> b -> e -> Stop

Traces: a.d.b.c.d   d.a.b.c.d   a.d.b.d.c   d.a.b.d.c

Anonymity

- Let AnUsers = $\{p_1, p_2\}$

- Let A = $\{p_1.m, p_2.m\}$

- Example 1 $p_1.m \text{ -> } p_2.m \text{ -> Stop}$
- Example 2 $p_1.m \text{ -> Stop } ||| \; p_2.m \text{ -> Stop}$
- Example 3 $p_1.m \text{ -> Stop } + \; p_2.m \text{ -> Stop}$

- Question: for each system, say whether or not it provides anonymity wrt A

Anonymity

- **A more involved example:**

P = p_1.m -> a -> Stop [] p_2.m -> a -> Stop

$||_{\{p_1.m\ ,\ p_2.m\ \}}$

p_1.m -> b -> Stop [] p_2.m -> c -> Stop

Question: Does P provides anonymity wrt

A = {p_1.m, p_2.m}

Anonymity

- Answer: No

  P has traces $(p_1.m).b.a$, $(p_2.m).c.a$, ...

  but not $(p_2.m).b.a$, $(p_1.m).c.a$, ...

  The permutation $\{ p_1 \text{->} p_2 , p_2 \text{->} p_1 \}$ changes the traces.

- However, if we assume that the observer has no visibility of the actions **b** and **c**, then the system does provide anonymity wrt $A = \{p_1.m, p_2.m\}$

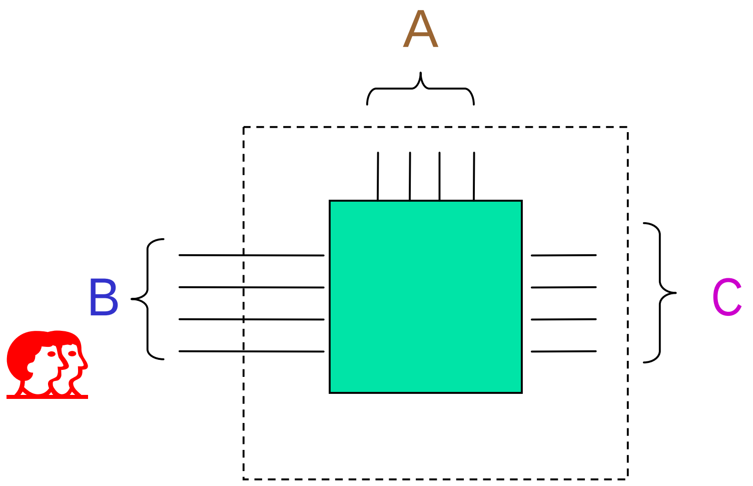
- One elegant way to formalize the concept of visibility in CSP is to use the the hiding operator:

  $P\backslash\{b, c\}$ provides anonymity wrt A

- Note: the above example shows that hiding A would not be enough

Anonymity

- In general, given P, consider the sets:
  - A = { a.x | a ε AnUsers } : the actions that we want to know only partially (we want to know x but not a)
  - B : the actions that we want to observe
  - C = Actions – (B U A) : The actions we want to hide

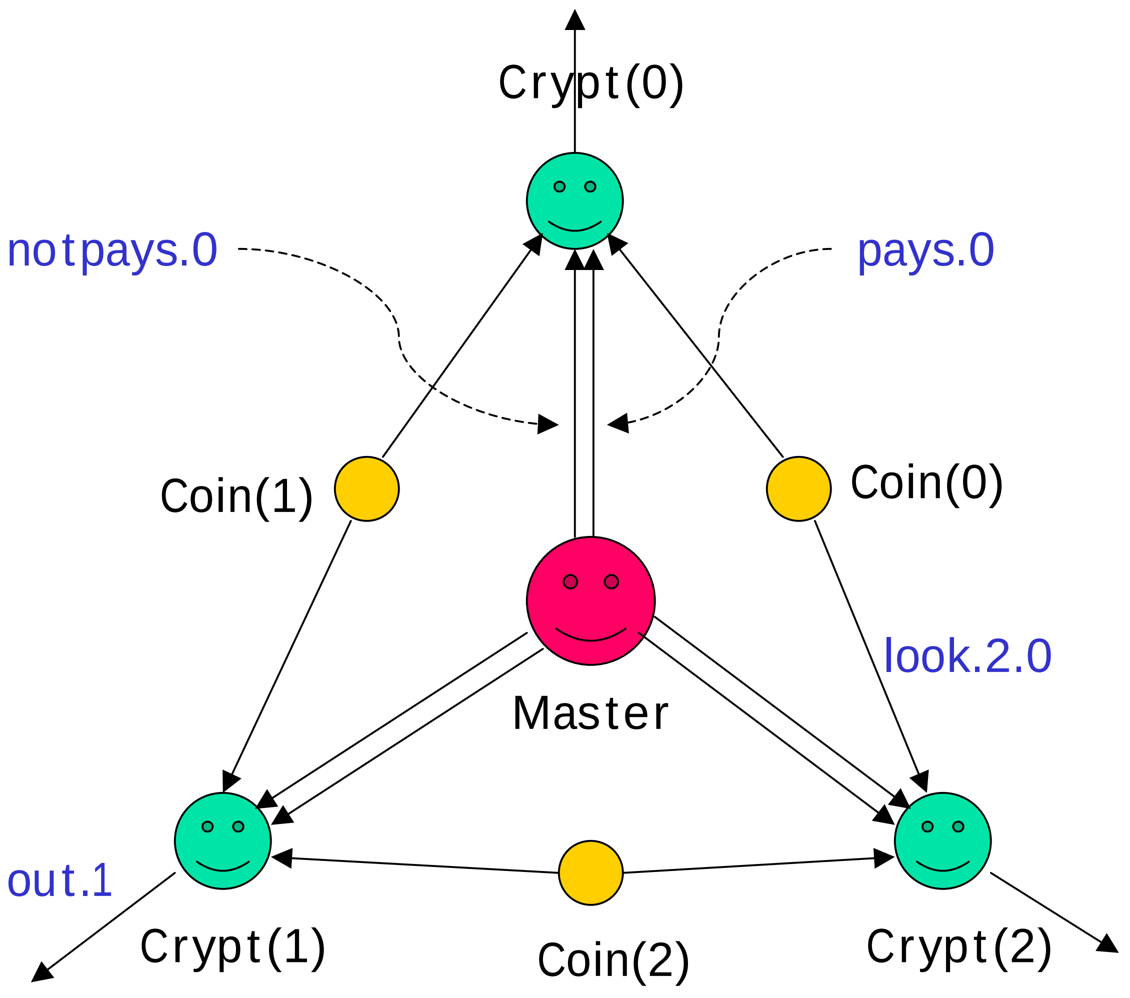


The system to consider for the Anonymity analysis: P\C

Method: for any perm ρ : A -> A
Check that ρ (P\C) $=_T$ P\C

$=_T$: trace equivalence

Example: The dining cryptographers



The dining cryptographers

- Three cryptographers share a meal
- The meal is paid either by the organization (master) or by one of them. The decision on who pays is taken by the master
- Each of them is informed by the master whether or not he is paying
- **GOAL**: The cryptographers would like to know whether the organization is paying or not, but without knowing the identity of the cryptographer who is paying (if any).

The dining cryptographers

- **Solution:** Each cryptographer tosses a coin. Each coin is in between two cryptographers.
- The result of each coin-tossing is visible to the adjacent cryptographers, and only to them.
- Each cryptographer examines the two adjacent coins
  - If he is not paying, he announces "agree" if the results are the same, and "disagree" otherwise.
  - If he is paying, he says the opposite

- **Claim:** if the number of "disagree" is even, then the master is paying. Otherwise, one of them is paying. In the latter case, the non paying cryptographers will not be able to deduce whom exactly is paying

The dining cryptographers

- Specification in CSP: Master and Coins

Master =

Σ_n pays.n -> notpays.(n+1) -> notpays (n+2) -> Stop

+ notpays.0 -> notpays.1 -> notpays.2 -> Stop

Coin(n) = Heads(n) + Tails(n)

Heads(n) = look.n.n.hd ->Stop ||| look.(n-1).n.hd ->Coin(n)

Tails(n) = look.n.n.tl -> Stop ||| look.(n-1).n.tl ->Coin(n)

- **Note:** the arithmetic operations are modulo 3

The dining cryptographers

- Specification in CSP: Cryptographers

```
Crypt(n) = notpays(n) -> Check(n)
              [] pays(n) -> Check’(n)

Check(n) =  look.n.n?x -> look.n.(n+1)?y ->
                          if (x=y) then out.n.agree -> Stop
                                   else out.n.disagree -> Stop

Check’(n) =  look.n.n?x -> look.n.(n+1)?y ->
                          if (x=y) then out.n.disagree -> Stop
                                   else out.n.agree -> Stop
```

The dining cryptographers

- Specification in CSP: The whole system

Crypts = Crypt(0) ||| Crypt(1) ||| Crypt(2)

Coins = Coin(0) ||| Coin(1) ||| Coin(2)

$\text{Meal} = \text{Master} \ ||_{\{\text{pays, notpays}\}} \ (\text{Coins} \ ||_{\{\text{look}\}} \ \text{Crypts})$

The dining cryptographers

- **The anonymity property**
  - A = { pays.0, pays.1, pays.2 }
  - B = { out }
  - C = Actions – (B U A) = {look,notpays}
- Theorem: For every permutation ρ: A -> A, we have

$$\rho(\mathrm{Meal}\backslash C) =_T \mathrm{Meal}\backslash C$$

  - $=_T$ here represents trace equivalence.
  - This theorem means that an external observer cannot infer which cryptographer has paid.
- This theorem can be proved by using the authomatic tool FDR.
- Of course, it can also be proved "by hand". Exercise

The dining cryptographers

- One can argue that previous result is not strong enough: a cryptographer has more information than an external observer. Let us then do the analysis for a cryptographer, say Crypt(0)

  A = { pays.1, pays.2 }
  B = { pays.0, notpays.0, look.0, out }
  C = Actions – (B U A)

- Theorem: For every permutation ρ: A -> A, we have

$$\rho(\mathrm{Meal} \backslash C) =_T \mathrm{Meal} \backslash C$$

- This means that if Crypt(1) or Crypt(2) pay, then Crypt(0) can't infer which of them has paid. The same can be shown for the other two. So Meal\C provides the desired anonymity property.

The dining cryptographers

- Example of a case in which the anonymity property does not hold.
- Assume that Crypt(0) can access the result of the third coin, namely has visibility of the result of the action look.2.2

A = { pays.1, pays.2 }

B = { pays.0, notpays.0, look.0, out } U { look.2.2 }

C = Actions – (B U A)

- We have that for some permutation ρ: A -> A,

$$\rho(Meal\backslash C) \;\; =/=_T \;\; Meal\backslash C$$

pays.2 notpays.0 look.00.heads look.0.1.heads look.2.2.heads out.2.disagree YES

pays.1 notpays.0 look.00.heads look.0.1.heads look.2.2.heads out.2.disagree NO