MPRI C.2.3 - Concurrency

Probabilistic models and applications
Lecture 5

Kostas Chatzikokolakis

Jan 24, 2014

Outline of the lectures

- Dec 13
- Dec 20
- Jan 10
- Jan 17
- Jan 24

Outline of the lectures

- The need for randomization
- Probabilistic automata
- Probabilistic bisimulation
- Probabilistic calculi
- Testing equivalence
- Introduction to probabilistic model checking and PRISM
- Metrics for probabilistic processes
- PRISM: a probabilistic model checker
- Verification of anonymity protocols: Dining Cryptographers, Crowds

Questions from the last lecture

Question 1:

If m is a pseudometric then $\hat{m}$ is also a pseudometric.

Questions from the last lecture

Question 2:

Let δ_s denote the Dirac distribution assigning pb 1 to s.
Then $\hat{m}(\delta_s, \delta_{s'}) = m(s, s')$ for all $s, s' \in S$.

The metric extension of bisimulation

- Define $F : \mathcal{M} \to \mathcal{M}$. $F(m)$ is smallest metric s.t.
  - $\forall s \xrightarrow{a} \mu \; \exists t \xrightarrow{a} \mu' : \hat{m}(\mu, \mu') \leq F(m)(s, t)$
  - $\forall t \xrightarrow{a} \mu \; \exists s \xrightarrow{a} \mu' : \hat{m}(\mu, \mu') \leq F(m)(s, t)$
- Alternatively

$$F(m)(s, t) = \max_a \hat{m}(s \xrightarrow{a}, t \xrightarrow{a})$$

where $s \xrightarrow{a} = \{\mu \mid s \xrightarrow{a} \mu\}$

- Take $m_\sim$ to be the maximum fixpoint of F

The metric extension of bisimulation

Lemma

R: equivalence relation on S, m: metric on S.

$$m(s,t) = 0 \Leftrightarrow sRt$$

implies

$$\hat{m}(\mu,\mu') = 0 \Leftrightarrow \mu R \mu'$$

Theorem

$m \sim t$ iff $m_{\sim}(s,t) = 0$

PRISM

- A probabilistic model checker
- Supports (among other things) MDPs and PCTL
- Given an MDP M and a PCTL formula φ it can automatically decide whether $M \models \varphi$
- It can also compute the minimum λ s.t. $s \models \mathcal{P}_{\leq\lambda}\psi$
  - Exercise: how?
- Open source: `www.prismmodelchecker.org`

PRISM language

- A language to specify MDPs
- A model contains variables with a given domain:

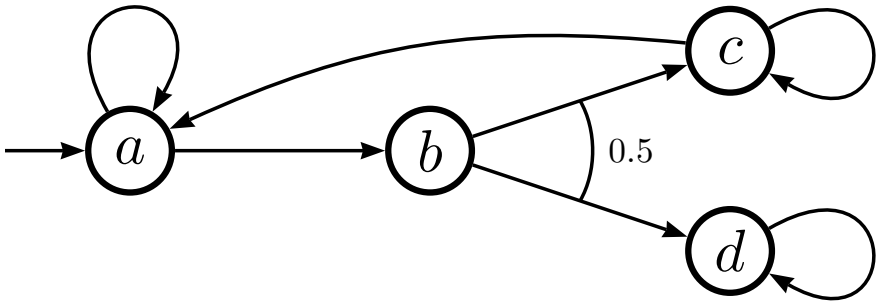
```
global var1:  [1..10] init 1;
global var2:  [1..5] init 2;
```

- A state is an assignment of values to variables,
  eg `var1 = 1, var2 = 2`
- In the above example S contains 50 states
- The initial state is given by the initial values of the variables

PRISM language

- Transitions are given by update rules
  ```
  [] guard -> pr_1:(update_1) + ...+ pr_n:(update_n);
  ```
- The guard determines on which states the transition is possible
- The update rules determine the target state, eg
  ```
  [] var1 = 1 & var2 < 5 ->
          0.5:(var2' = var2 + 1) + 0.5:(var1' = 2);
  ```
- This creates a transition from any state $(1, n)$ with $n < 5$ to $(1, n+1)$ and $(2, n)$, each with probability 0.5
- Multiple transitions can be defined for the same state (nondeterministic choice)

Example




```
global state:[1..4] init 1;

[] state = 1 -> 1:(state' = 1);
[] state = 1 -> 1:(state' = 2);

[] state = 2 -> 0.5:(state' = 3) + 0.5:(state' = 4);

[] state = 3 -> 1:(state' = 3;
[] state = 3 -> 1:(state' = 1);

[] state = 4 -> 1:(state' = 4);
```

PRISM properties

- An atomic proposition is any boolean expression on variables:

```
(var1 < 5) & (var2 < var1)
```

- We can give them labels

```
label "fail" = var2 > var1;
```

- Properties are given as PCTL formulas, eg

```
P < 0.01 [ F "fail" ]
```

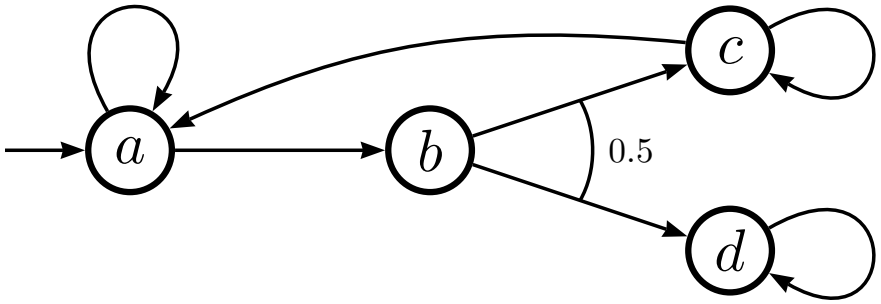
- Properties are evaluated on all states, but ...

```
"someState" => P < 0.01 [ F "fail" ]
```

- We can also compute the min/max prob of satisfying path formulas

```
Pmax = ?  [ F "fail" ]
Pmax = ?  [ F "fail" {"someState"} ]
```

Example




```
label "a" = state = 1;
label "b" = state = 2;

"b" => P <= 0.5 [ F "a" ]
Pmax=?[ F "a" {"b"} ]
```

The two envelopes puzzle

- I select two real numbers in some arbitrary way
- I put them in two envelopes, you select one of them (in any way you want)
- You see the number and you have 2 options: keep it, or exchange it with the other envelope
- Your goal is to select the bigger number
- Solution: chooze z using any distribution that selects any number with nonzero pb, keep the number if $\geq z$.

The two envelopes puzzle

```
global num1:[0..4] init 0;
global num2:[0..4] init 0;

global chosen:[0..4] init 0;
global not_chosen:[0..4] init 0;

global z:[0..4] init 0;

global finished:bool init false;
```

The two envelopes puzzle

```
[] num1 = 0 -> 1:(num1' = 1);
[] num1 = 0 -> 1:(num1' = 2);
[] num1 = 0 -> 1:(num1' = 3);
[] num1 = 0 -> 1:(num1' = 4);

[] num2 = 0 -> 1:(num2' = 1);
[] num2 = 0 -> 1:(num2' = 2);
[] num2 = 0 -> 1:(num2' = 3);
[] num2 = 0 -> 1:(num2' = 4);

[] num1 != 0 & num2 != 0 & chosen = 0 ->
0.5:(chosen' = num1) & (not_chosen' = num2) +
0.5:(chosen' = num2) & (not_chosen' = num1);
```

The two envelopes puzzle

```
[] chosen != 0 & z = 0 ->
0.02:(z' = 1) +
0.57:(z' = 2) +
0.16:(z' = 3) +
0.25:(z' = 4);

[] !finished & z != 0 & chosen <= z ->
1:(chosen' = not_chosen) & (not_chosen' = chosen) &
  (finished' = true);
[] !finished & z != 0 & chosen > z ->
1:(finished' = true);
```

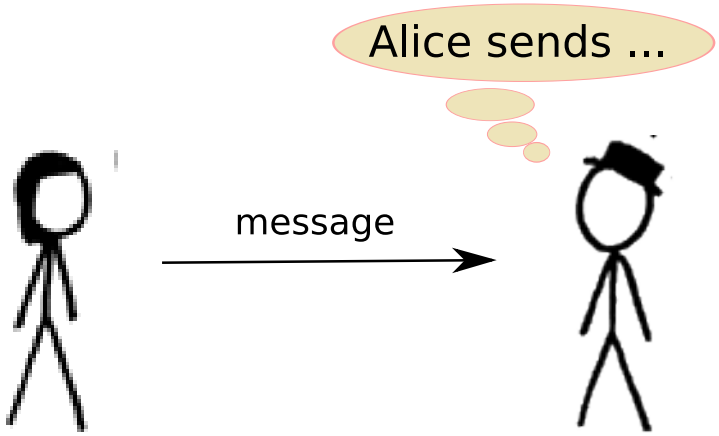
The two envelopes puzzle

```
label "won" = finished & chosen >= not_chosen;
label "start" = num1 = 0 & num2 = 0;

"start" => P>0.5 [ F "won" ];

Pmax=? [ F "won" { "start" } ];
Pmin=? [ F "won" { "start" } ];
```

The notion of anonymity



Alice does not want Bob to know that she is the sender.

Applicatons

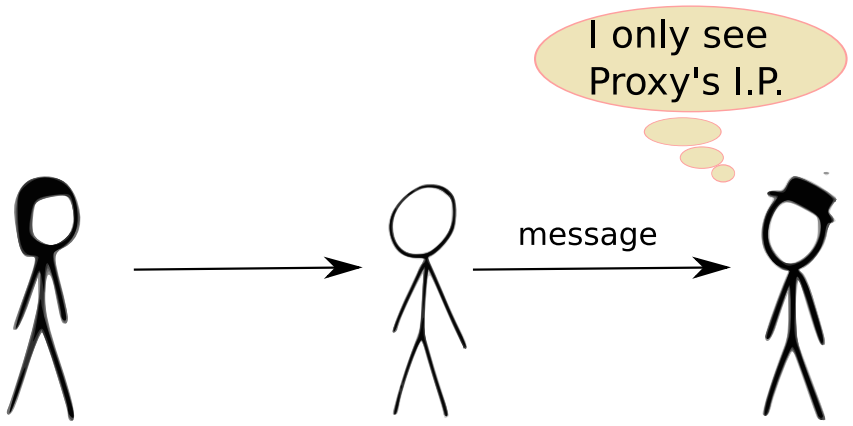
- General applications
  - Web browsing
  - Censorship resistance
  - SPAM avoidance
  - Profiling avoidance
- Specialized applications
  - Electronic voting
  - Incident reporting
  - Auctions / stock market
  - File sharing

Why anonymity is difficult?

- Visit `www.hostip.info`
- Sender's IP address included in all IP packets
- Already enough to trace someone to ISP/region level
- Can be traced down to individuals using ISP's logs (obtained with ISP's co-operation, subpoenas, . . .)
- Similarly for ethernet (MAC address) and other protocols
- Application identity leakage (eg cookies)

Methodologies for connection-level anonymity

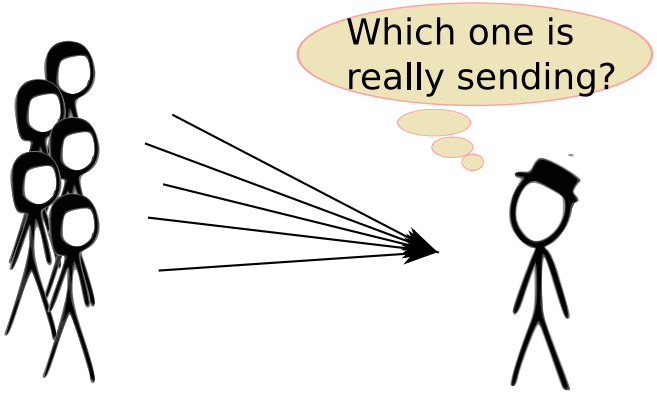
Simple solution: use an anonymous proxy



- eg. `www.anonymizer.com`
- Major disadvantage: we need to trust a single agent

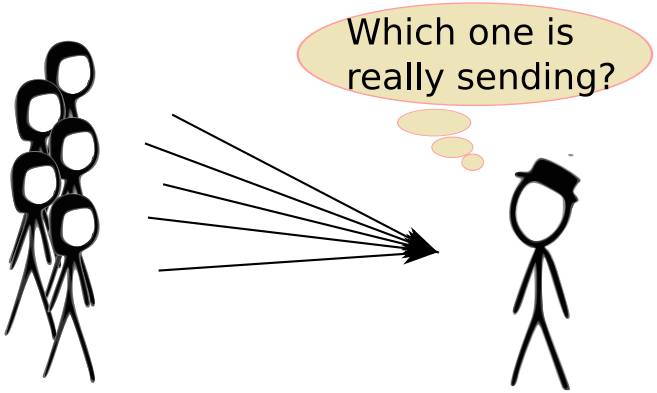
Approaches without a trusted party

Hide message in other traffic

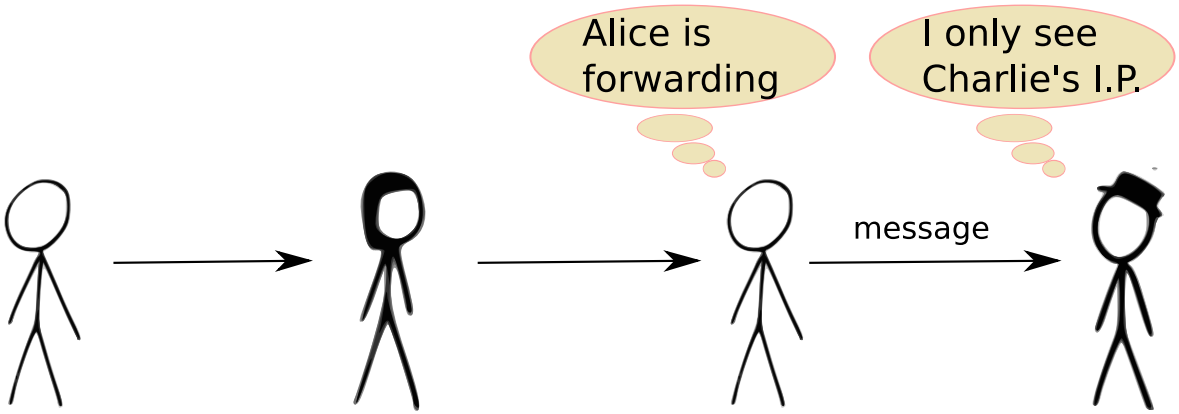


Approaches without a trusted party

Hide message in other traffic

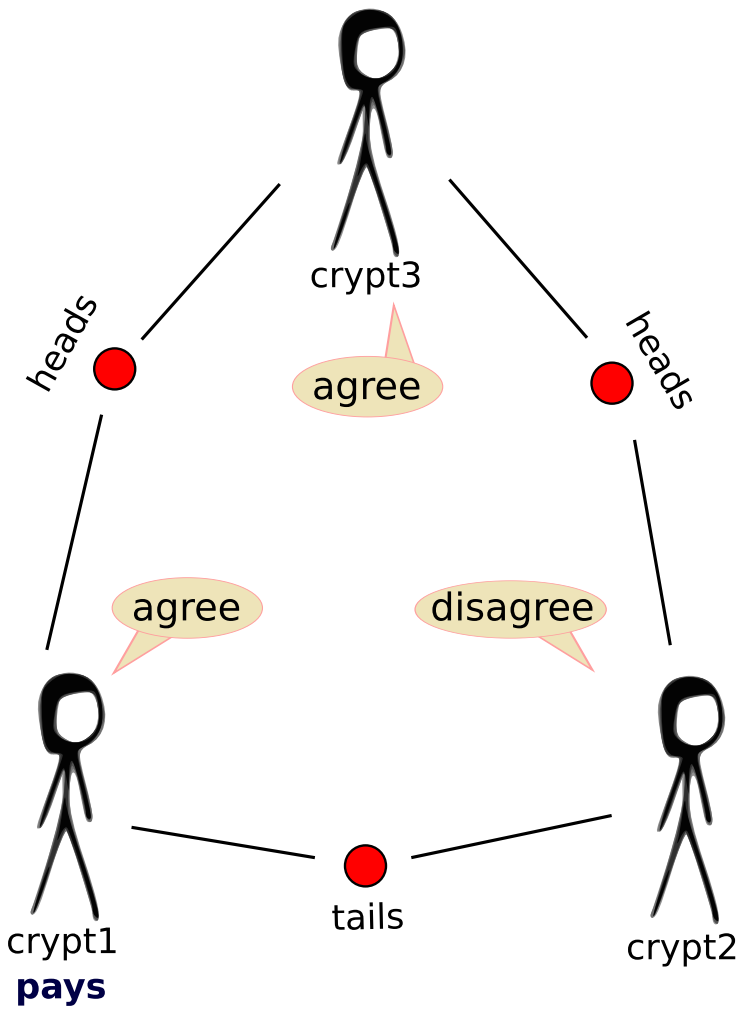


Forward message through other users



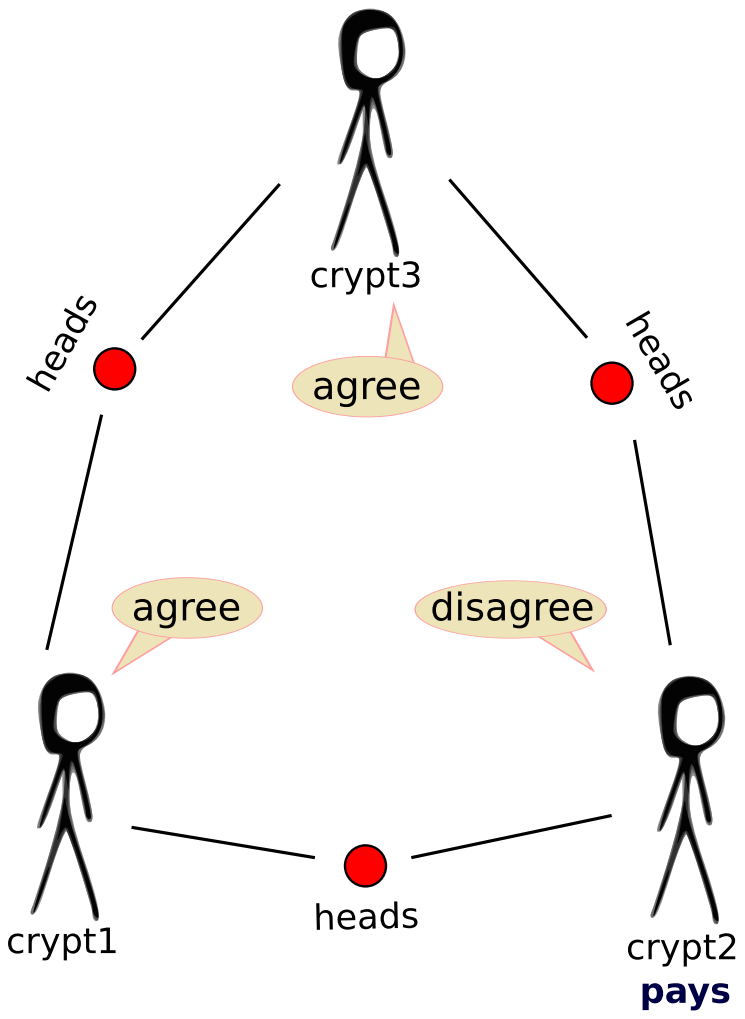
The dining cryptographers protocol

- Goal: find whether a cryptographer pays without revealing who
- Coins are fair and only visible to adjacent cryptographers
- Announce agree/disagree, the payer says the opposite
- A cryptographer is the payer $\Leftrightarrow$ the number of disagrees is odd



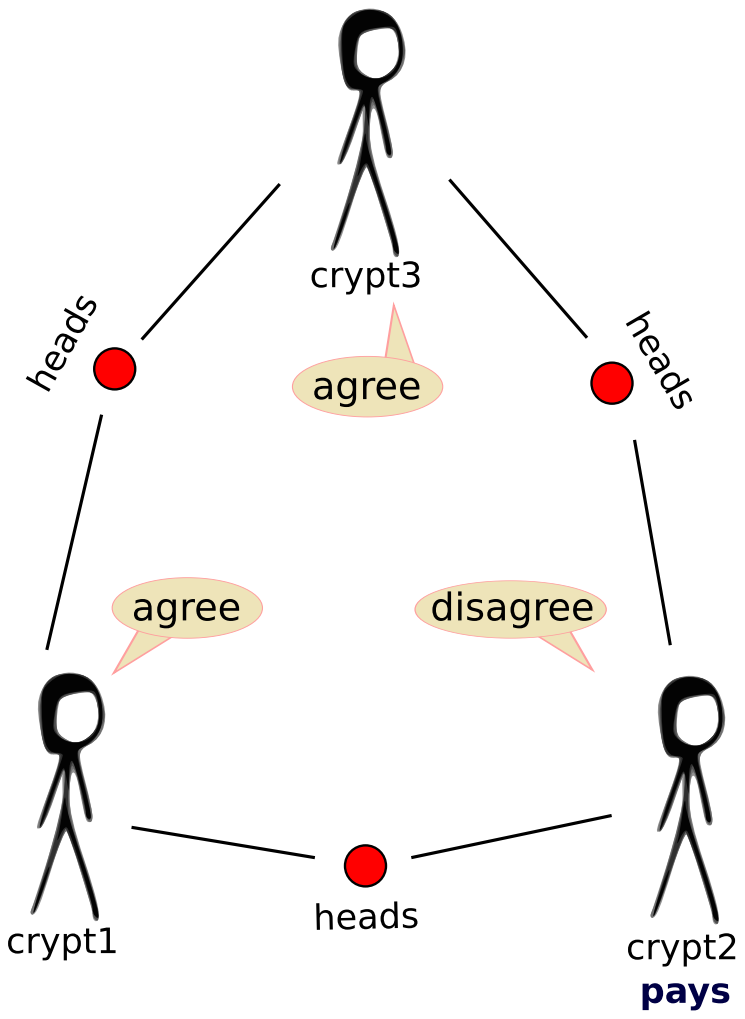
Anonymity of the DC protocol (intuition)

1. Attacker is an outside observer
   - Assume that crypt2 is the payer
   - This is impossible given the previous coins
   - BUT: there is a coin outcome that makes the same announcement valid!
   - The attacker cannot distinguish the 2 cases
   - The two coin outcomes have the same probability



Anonymity of the DC protocol (intuition)

2. Attacker is cryptographer 3
   - Now the attacker knows the 2 coins
   - But he is still confused
   - The coin that makes the announcement valid under crypt2 is not visible to crypt3



Anonymity properties

- Many notions of anonymity, depending on our needs
- Two orthogonal questions:
  - What we want to hide?
  - To what degree we want to hide it?

Anonymity properties

What we want to hide?

- Sender anonymity: the attacker should not know that a user sent a message
- Receiver anonymity: the attacker should not know that a user received a message
- Unlinkability: the attacker might know that some users sent messages and some users received messages, but cannot link the sender and the receiver

Anonymity properties

To what degree we want to hide it?

- Beyond suspicion (strong anonymity)

  From the attacker's point of view, the sender appears no more likely to be the originator of the message than any other potential sender in the system.

- Probable innocence

  From the attacker's point of view, the sender appears no more likely to be the originator of the message than to not be the originator.

- Possible innocence

  From the attacker's point of view, there is a nontrivial probability that the real sender is someone else.

- Something in between?

A general probabilistic framework

Common features of anonymity protocols

- There is information that we want to hide
  - DC: Who pays
- There is information that is made public
  - DC: the agree/disagree announcements
- Randomization is used to hide the link between the two
  - DC: Coin tossing

A general probabilistic framework

- We define a set $\mathcal{A}$ of anonymous events
- Assumption: on each execution exactly one $a \in \mathcal{A}$ happens
- The attacker tries to find out which one
- $\mathcal{A}$ models what we want to hide
  - Sender anonymity: $\mathcal{A} = \{sender_1, sender_2, \ldots\}$
  - Receiver anonymity: $\mathcal{A} = \{reicever_1, receiver_2, \ldots\}$
  - Unlinkability:
    $\mathcal{A} = \big\{\{(s_1, r_1), (s_2, r_2)\}, \{(s_1, r_2), (s_2, r_1)\}, \ldots\big\}$

A general probabilistic framework

- We define a set $\mathcal{O}$ of observable events
- On each execution exactly one $o \in \mathcal{O}$ happens
- This models what the attacker can observe
- For the dining cryptographers:

$$\mathcal{O} = \{aad, ada, daa, ddd\}$$

A general probabilistic framework

- For each $a \in \mathcal{A}$, $o \in \mathcal{O}$ we define the probabilities

$$p(o|a) = \text{probability of observing } o \text{ when } a \text{ happens}$$

$$p(a) = \text{probability that } a \text{ happens}$$

- $p(o|a)$ models the behaviour of the protocol
- $p(a)$ models the behaviour of the users
- With these we can define the probability of any event involving $\mathcal{A}, \mathcal{O}$, eg $p(o), p(a \wedge o), p(a|o)$

Strong anonymity

- Informal definition

  From the attacker's point of view, the sender appears no more likely to be the originator of the message than any other potential sender in the system.

- First idea: $\forall a, a', o : p(a|o) = p(a'|o)$
- This is too strong: consider a case where $p(a) = 99\%, p(a') = 1\%$
- We want a definition independent from $p(a)$

Strong anonymity

- A different approach

  The attacker should get no information from the protocol about the originator of the message

- Formally, a protocol satisfies strong anonymity iff

$$\forall a \in \mathcal{A}, o \in \mathcal{O} : p(a|o) = p(a)$$

- This does not forbid $p(a) = 99\%$

- An alternative definition

$$\forall a, a' \in \mathcal{A}, o \in \mathcal{O} : p(o|a) = p(o|a')$$

- The two definitions are equivalent (exercise: proof)

- The last one is independent from $p(a)$

Strong anonymity: Dining Cryptographers

Attacker is an outside observer

- $\mathcal{A} = \{crypt_1, crypt_2, crypt_3\}$
- $\mathcal{O} = \{aad, ada, daa, ddd\}$
- We can show that

$$p(aad|crypt_1) = p(aad|crypt_2) = p(aad|crypt_3) = 1/4$$

  and similarly for the other observables
- These values can be computed using PRISM (exercise)
- Strong anonymity holds (second definition)

Strong anonymity: Dining Cryptographers

Attacker is cryptographer 3

- $\mathcal{A} = \{crypt_1, crypt_2\}$
- $\mathcal{O} = \{(aad, h, t), (aad, t, h), (ada, h, h), (ada, t, t), \ldots\}$
- In this case:

$$p((aad, h, t)|crypt_1) = p((aad, h, t)|crypt_2) = 1/8$$

  and similarly for the other observables

- Strong anonymity still holds

The Crowds protocol

- DC is not practical for a large number of users
- In practice we might want to trade anonymity for efficiency
- Crowds offers a weaker notion of anonymity called probable innocence
- Designed for anonymous web surfind

The Crowds protocol

The initiator:

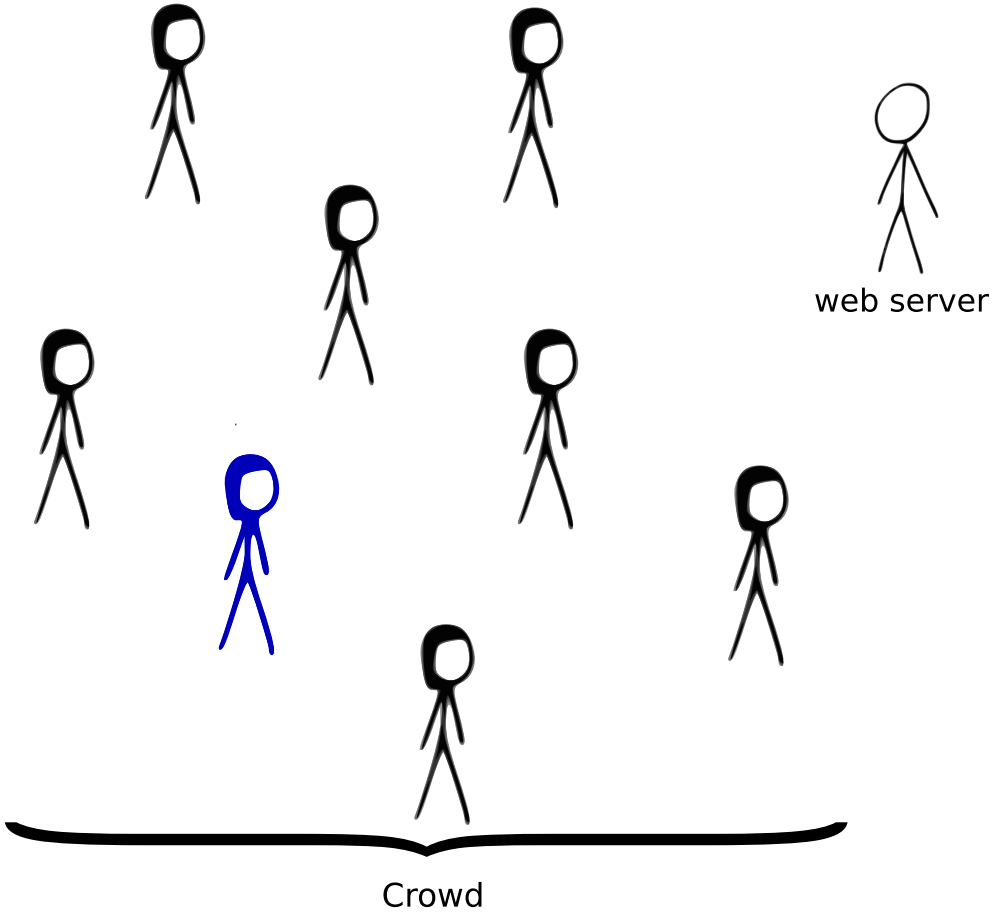
- Forwards the message

A forwarder:

- With pb p_f forwards
- With pb $1 - p_f$ delivers

- The path is used in the opposite direction for the reply

- The same path is used in future requests



The Crowds protocol

The initiator:

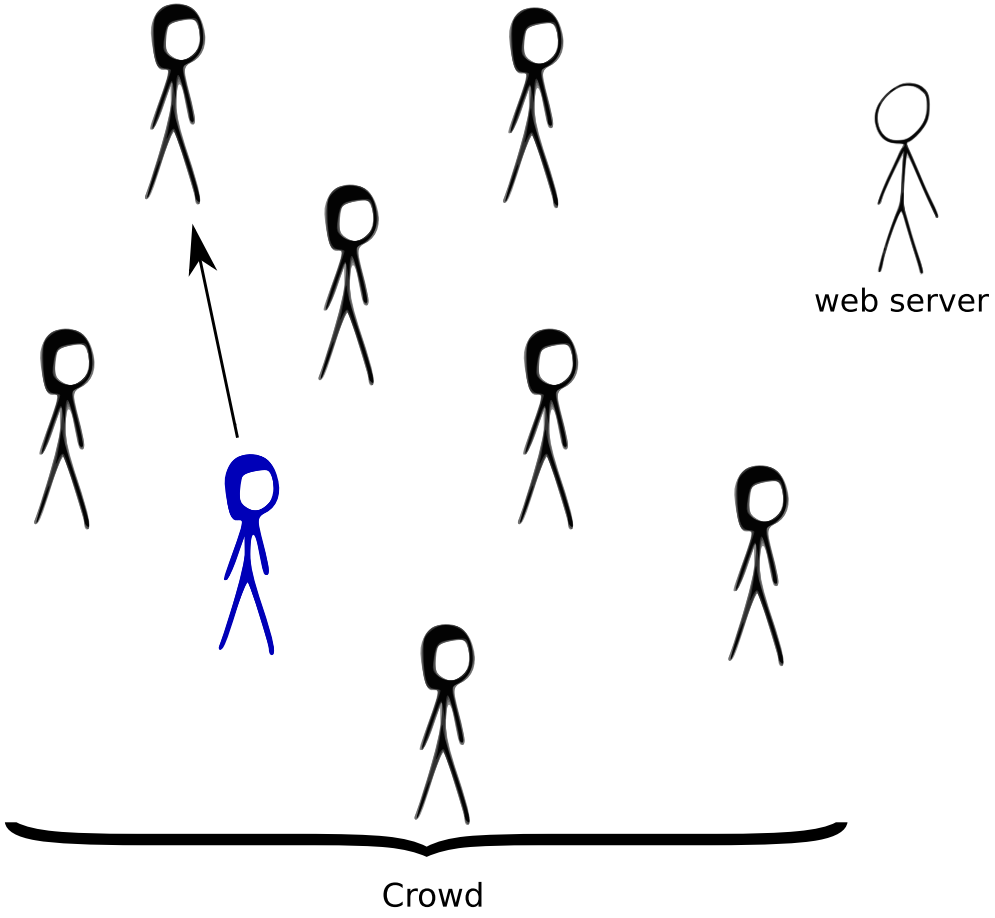
- Forwards the message

A forwarder:

- With pb p_f forwards
- With pb $1 - p_f$ delivers

- The path is used in the opposite direction for the reply

- The same path is used in future requests



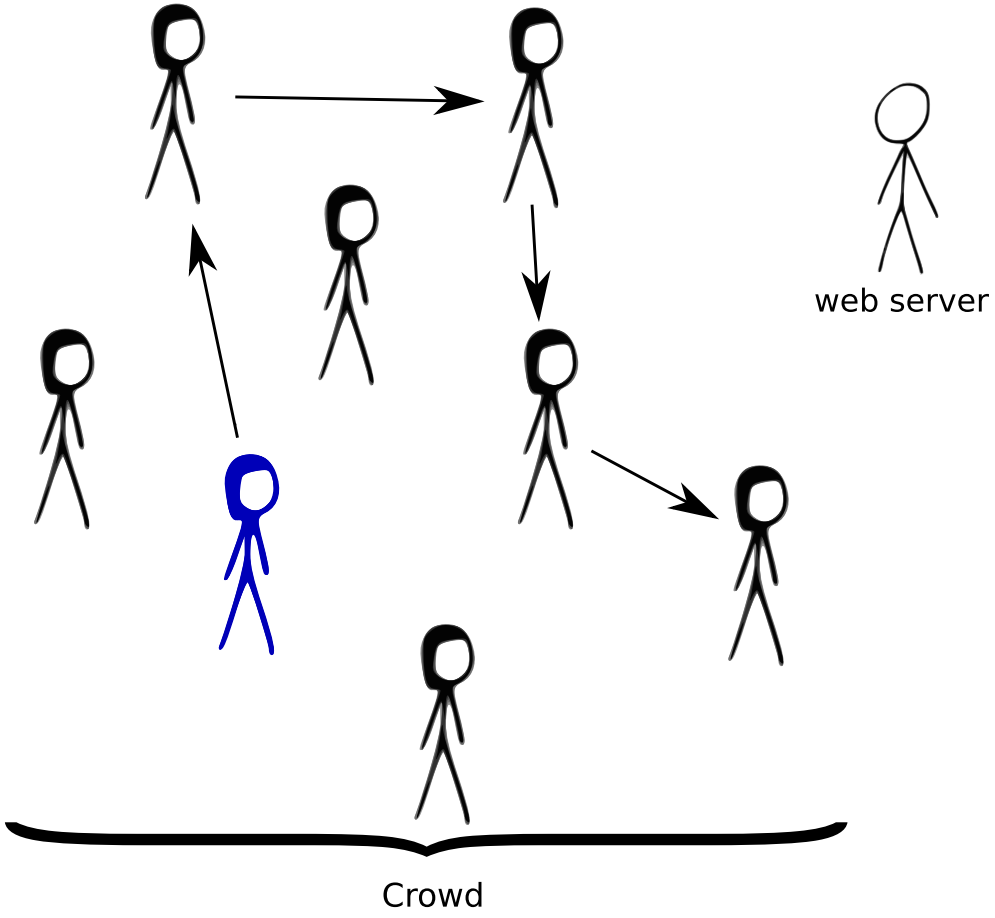
The Crowds protocol

The initiator:

- Forwards the message

A forwarder:

- With pb p_f forwards
- With pb $1 - p_f$ delivers
- The path is used in the opposite direction for the reply
- The same path is used in future requests



The Crowds protocol

The initiator:

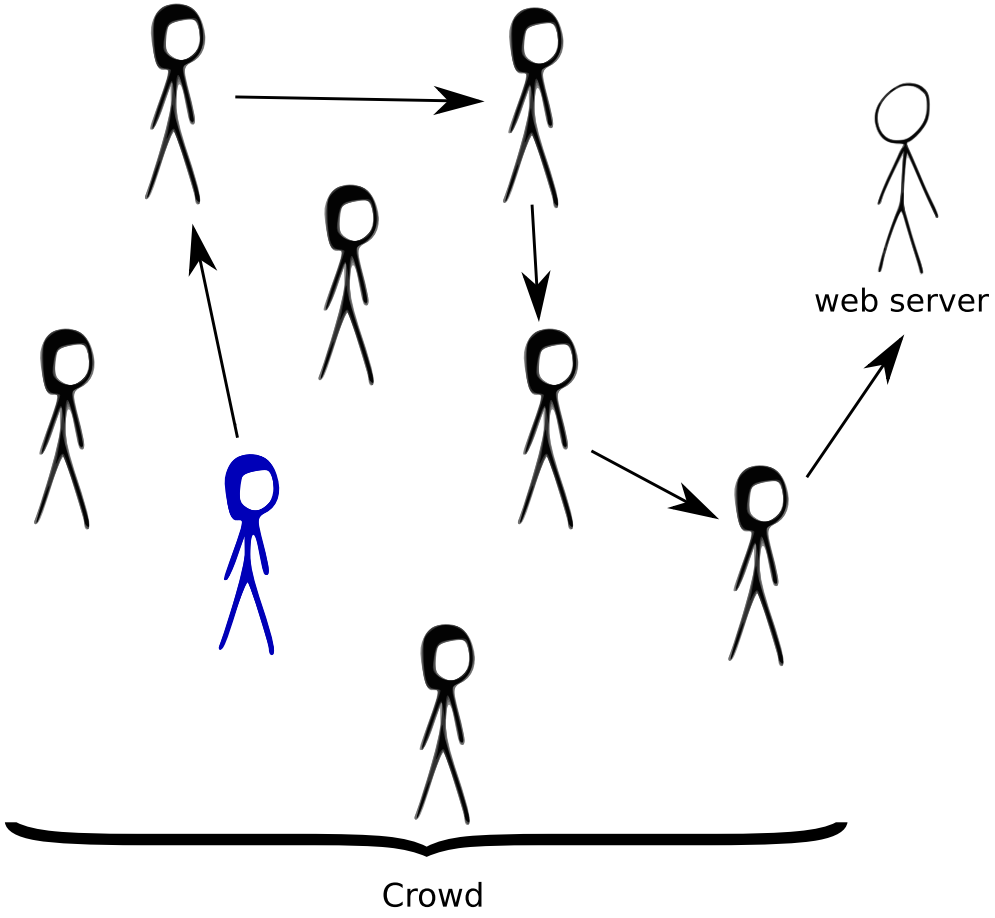
- Forwards the message

A forwarder:

- With pb p_f forwards
- With pb $1 - p_f$ delivers

- The path is used in the opposite direction for the reply

- The same path is used in future requests



The Crowds protocol

The initiator:

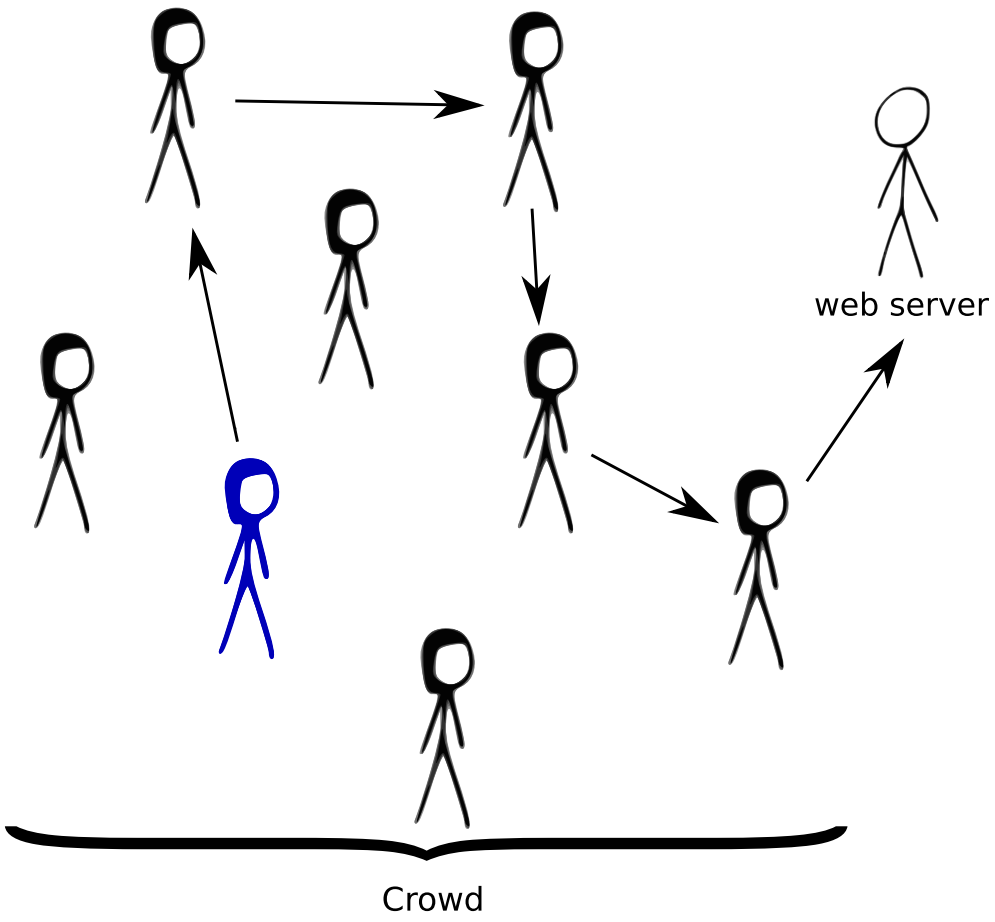
- Forwards the message

A forwarder:

- With pb p_f forwards
- With pb $1 - p_f$ delivers

- The path is used in the opposite direction for the reply

- The same path is used in future requests



The Crowds protocol

The initiator:

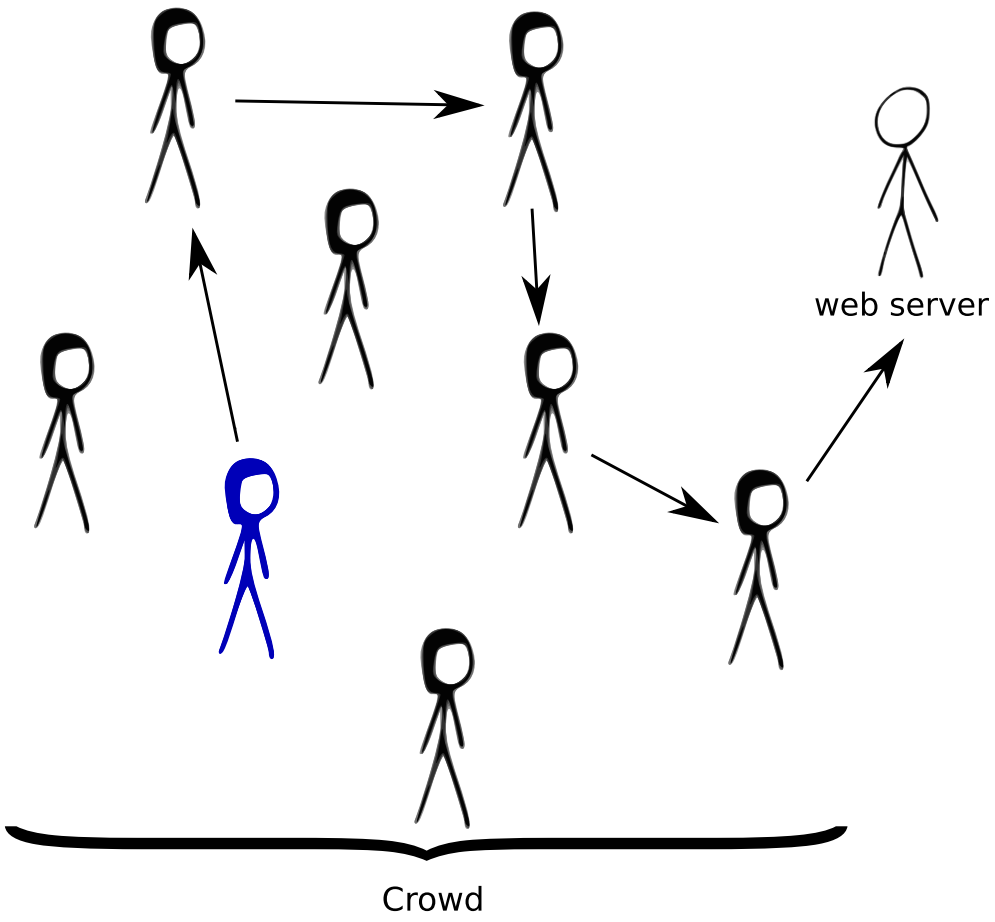
- Forwards the message

A forwarder:

- With pb p_f forwards
- With pb $1 - p_f$ delivers

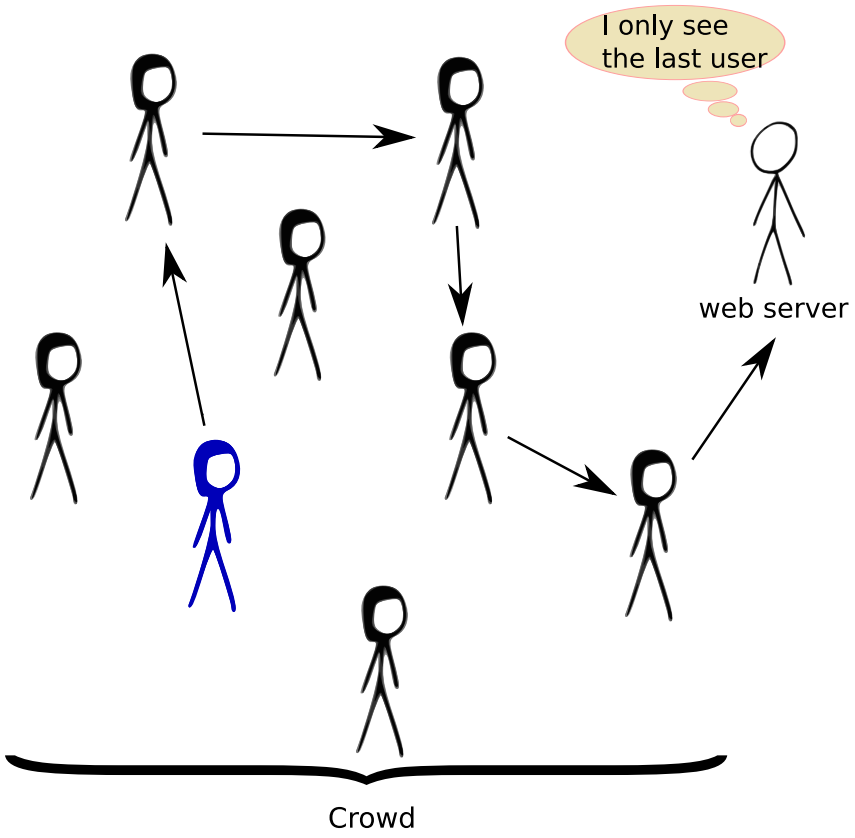
- The path is used in the opposite direction for the reply

- The same path is used in future requests



The Crowds protocol: anonymity

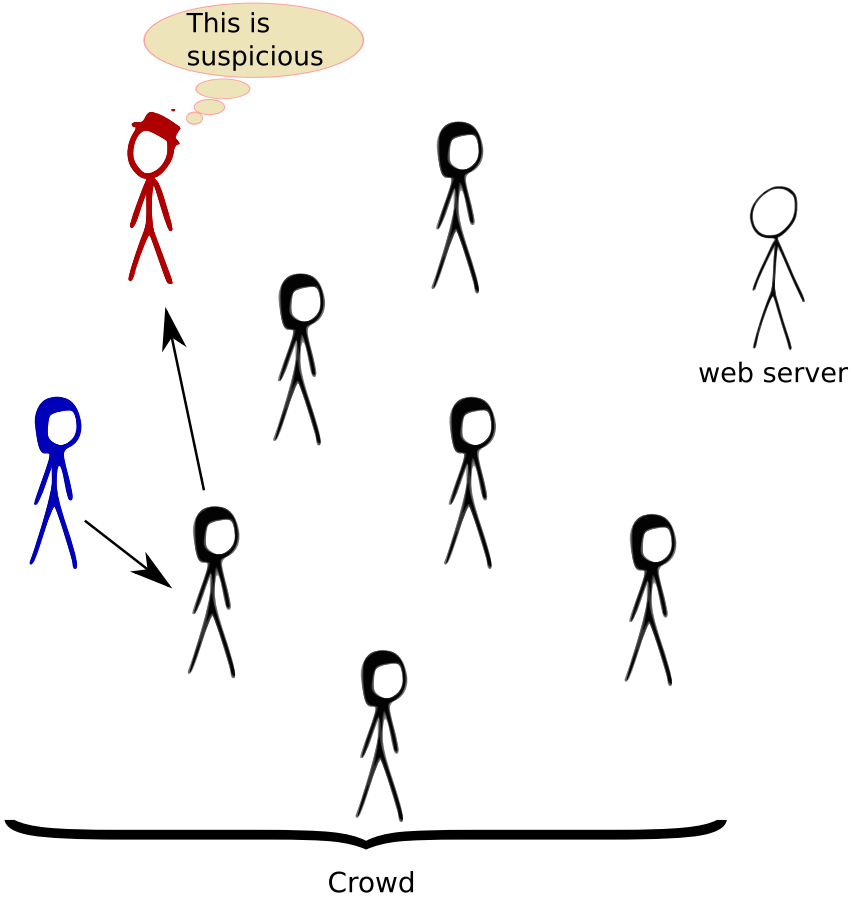
- We consider sender anonymity
- Attacker model
  - Cannot see the whole network
  - Only messages sent to him
- The server:
  - only sees the last user
  - Strong anonymity is satisfied



The Crowds protocol: anonymity

Corrupted users:

- They can see forwarding requests and "detect" a user i
- User i can still claim that he was forwading the message for user j
- Is strong anonymity satisfied?
- Compare the probability to detect i:
  - when i is the payer
  - when j is the payer
- They are different: strong anonymity is violated



The Crowds protocol: anonymity

The attacker is the server:

- m users
- $\mathcal{A} = \{user_1, \ldots, user_m\}$
- $\mathcal{O} = \{last_1, \ldots, last_m\}$
  where $last_i$ means "user i was the last in the path"
- Strong anonymity: $p(last_j | user_i) = 1/m, \forall i, j$

The Crowds protocol: anonymity

There are c corrupted users and $n = m - c$ honest:

- $\mathcal{A} = \{user_1, \ldots, user_n\}$
- $\mathcal{O} = \{det_1, \ldots, det_n\}$
  where det_i means "user i was detected" (forwarded the message to a corrupted user)
- The protocol is highly symmetric

$$p(det_i|user_i) = p(det_j|user_j) \qquad \forall i, j$$
$$p(det_i|user_j) = p(det_k|user_l) \qquad \forall i \neq j, k \neq l$$

Model checking Crowds

- We want to analyze an instance of the protocol (for a fixed number of total and corrupted users)
- Problem: infinite executions
- But we can still model it using a finite number of states
- Intuition: we do not need to keep the complete path of the message, the protocol's behaviour depends only on the last user of the path
- We also consider that a corrupted user does not forward messages. Thus the number of observations is finite

Model checking Crowds

```
const double pf;

const int honestNo = 2;
const int userNo = 3;

global initiator: [-1..honestNo-1] init -1;

global curUser: [-1..userNo-1] init -1;    // has the messa
global prevUser: [-1..honestNo-1] init -1; // sent last

global delivered: bool init false;
global detected: [-1..honestNo-1] init -1;
```

Model checking Crowds

```
// select an initiator
[] initiator = -1 & curUser = -1 -> (initiator' = 0);
[] initiator = -1 & curUser = -1 -> (initiator' = 1);

// forward the message from the initiator to a user
[] initiator != -1 & curUser = -1 ->
   1/userNo: (prevUser' = initiator) & (curUser' = 0) +
   1/userNo: (prevUser' = initiator) & (curUser' = 1) +
   1/userNo: (prevUser' = initiator) & (curUser' = 2);
```

Model checking Crowds

```
// Honest user
[] curUser >= 0 & curUser < honestNo & !delivered ->
   1-pf: (delivered' = true) +
   pf/userNo: (prevUser' = curUser) & (curUser' = 0) +
   pf/userNo: (prevUser' = curUser) & (curUser' = 1) +
   pf/userNo: (prevUser' = curUser) & (curUser' = 2);

// Corrupted user
[] curUser >= honestNo & !delivered ->
   (delivered' = true) &
   (detected' = prevUser);
```

Model checking Crowds, Properties

We can use PRISM to compute the conditional prob. $p(o|a)$

```
label "observ0" = delivered & detected = 0;
label "observ1" = delivered & detected = 1;

label "user0" = initiator = 0 & curUser = -1;
label "user1" = initiator = 1 & curUser = -1;

Pmax=? [ F "observ0" { "user0" } ]
Pmax=? [ F "observ1" { "user0" } ]
```

Strong anonymity is satisfied only when $c = 0$

Probable innocence

- Informal definition

  From the attacker's point of view, the sender appears no more likely to be the originator of the message than to not be the originator.

- Definition given for Crowds (Reiter & Rubin): probable innocence is satisfied iff

$$\forall i : p(det_i | user_i) \leq 1/2$$

- This implies

$$\sum_{j \neq i} p(det_j | user_i) \geq 1/2$$

- It is more probable to detect someone else, than the real sender

Probable innocence

- Can be generalized as:

$$\forall o, a, a' : (n-1)p(o|a) \geq p(o|a')$$

- Intuition: compare one user to all the others together
- Strong anonymity $\Rightarrow$ probable innocence

$$p(o|a) = p(o|a') \quad \Rightarrow \quad (n-1)p(o|a) \geq p(o|a')$$

Model checking Crowds, Probable Innocence

We want to compare the probability of detecting the initiator to the probability of detecting some other user

```
label "detectInit" = delivered & detected = initiator;
label "detectOther" = delivered &
                      detected != -1 & detected != initiato

Pmax=? [ F "detectOther" ]
Pmax=? [ F "detectInit" ]
```

Model checking Crowds, Probable Innocence

Crowds on a complete graph satisfies probable innocence iff

$$m \geq (1 + \frac{1/2}{p_f - 1/2})(c + 1)$$

But on an arbitrary graph, finding an analytic expression is challenging, while model checking is equally easy.