CS 591: Formal Methods in Security and Privacy

More on Differential Privacy

Marco Gaboardi
gaboardi@bu.edu

Alley Stoughton
stough@bu.edu

Course evaluation

The course evaluation is now available:

https://bu.campuslabs.com/courseeval/

Please fill it.

Recording

This is a reminder that we will record the class and we will post the link on Piazza.

This is also a reminder to myself to start recording!

From the previous classes

(ε,δ)-Differential Privacy

Definition

Given $\varepsilon, \delta \geq 0$, a probabilistic query $Q: X^n \to R$ is (ε,δ)-differentially private iff for all adjacent database D, D and for every $S \subseteq R$:

$$\Pr[Q(D) \in S] \leq \exp(\varepsilon)\Pr[Q(D') \in S] + \delta$$

(ε,δ)-indistinguishability

We can define a ε-skewed version of statistical distance. We call this notion ε-distance.

$$\Delta_{\varepsilon}(\mu 1,\mu 2)=\sup_{E\subseteq A}\max(\mu_1(E)-e^{\varepsilon}\mu_2(E),\ \mu_2(E)-e^{\varepsilon}\mu_1(E),0)$$

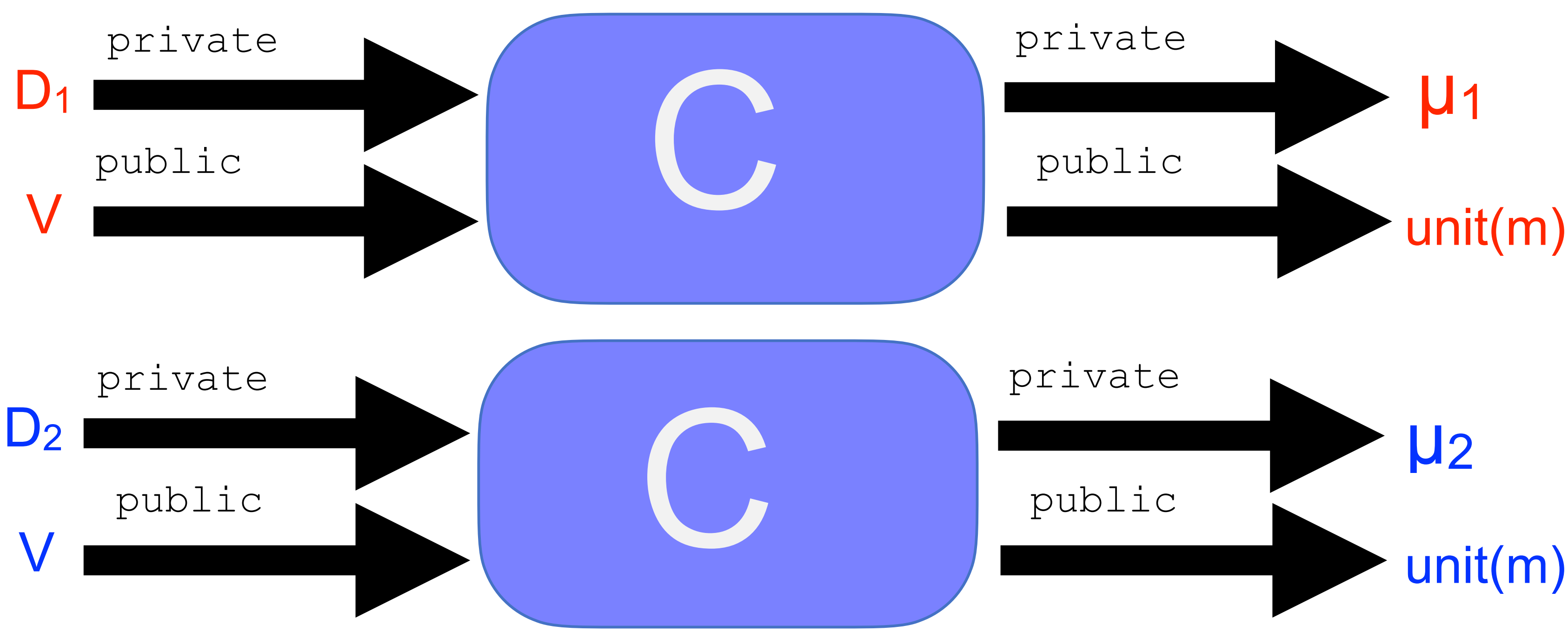
We say that two distributions $\mu_1, \mu_2 \in D(A)$, are at (ε,δ)-indistinguishable if:

$$\Delta_{\varepsilon}(\mu 1,\mu 2) \leq \delta$$

Differential Privacy as a Relational Property

c is differentially private if and only if for every $m_1 \sim m_2$ (extending the notion of adjacency to memories):

$\{c\}_{m1}=\mu_1$ and $\{c\}_{m2}=\mu_2$ implies $\Delta_\varepsilon(\mu_1,\mu_2) \leq \delta$



apRHL

Indistinguishability parameter

Precondition (a logical formula)

$$\vdash_{\epsilon,\delta} c_1 \sim c_2 : P \Rightarrow Q$$

Probabilistic Program

Probabilistic Program

Postcondition (a logical formula)

Validity of apRHL judgments

We say that the quadruple $\vdash_{\varepsilon,\delta} c_1 \sim c_2 : P \Rightarrow Q$ is valid if and only if for every pair of memories m_1, m_2 such that $P(m_1, m_2)$ we have:

$\{c_1\}_{m1} = \mu_1$ and $\{c_2\}_{m2} = \mu_2$ implies $Q_{\varepsilon,\delta}{}^*(\mu_1, \mu_2)$.

R-ε-δ-Coupling

Given two distributions $\mu_1 \in D(A)$, and $\mu_2 \in D(B)$, we have an R-ε-δ-coupling between them, for $R \subseteq A \mathrm{x} B$ and $0 \leq \varepsilon$ and $0 \leq \delta \leq 1$, if there are two joint distributions $\mu_L, \mu_R \in D(A\mathrm{x}B)$ such that:

1) $\pi_1(\mu_L) = \mu_1$ and $\pi_2(\mu_R) = \mu_2$,
2) the support of μ_L and μ_R is contained in R. That is, if $\mu_L(a,b)>0$, then $(a,b) \in R$, and if $\mu_R(a,b)>0$, then $(a,b) \in R$.
3) $\Delta_\varepsilon(\mu_L, \mu_R) \leq \delta$

Today: More on apRHL

Releasing privately the mean of Some Data

```
Mean(d : private data) : public real
   i:=0;
   s:=0;
   while (i<size(d))
      s:=s + d[i];
      i:=i+1;
   z:=$ lap eps s;
   z:= (s/i)+z;
   return z
```

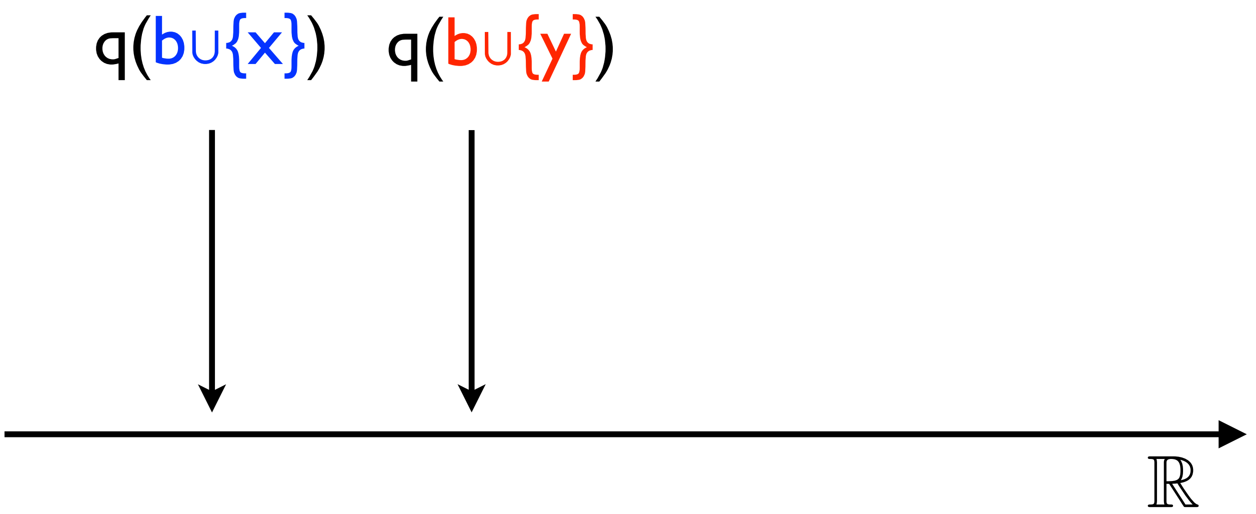
I am using the easycrypt notation here where `lap eps a` corresponds to adding to the value `a` noise from the Laplace distribution with `b=1/eps` and mean `mu=0`.

apRHL
Laplace

$$\frac{}{\vdash_{\varepsilon,0} \texttt{x}_1 \texttt{:=\$ Lap(}\varepsilon\texttt{,y}_1\texttt{)} \sim \texttt{x}_2 \texttt{:=\$ Lap(}\varepsilon\texttt{,y}_2\texttt{)} \texttt{ :|y}_1\texttt{-y}_2\texttt{|}\leq\texttt{1 ==> =}}$$

Global Sensitivity

$$GS_q = \max\{\,|q(D) - q(D')| \text{ s.t. } D \sim D'\}$$



Laplace in EasyCrypt

```
module M = {
  var x: int
  proc f (): int = {
    var r = 0;
      r <$ lap eps x;
    return r;
  }
}.

lemma lem1 : aequiv [ [eps & 0%r]
  M.f ~ M.f
 :  (`|M.x{1} - M.x{2}|<= 1)
==> res{2} = res{1} ].
proof.
  proc.
  seq 1 1 :(`|M.x{1} - M.x{2}|<= 1 /\ r{1}=r{2} /\ r{1}=0).
  toequiv.
  auto.
(*
   to prove this we can use the lap tactic which takes two
   parameters  k1 and k2 and generate two subgoals
   1) k2 * local_eps <= global eps
   2) |k1 + (M.x{1} - M.x{2})| <= k2
*)
  lap (0) 1.
qed.
```

apRHL
Generalized Laplace

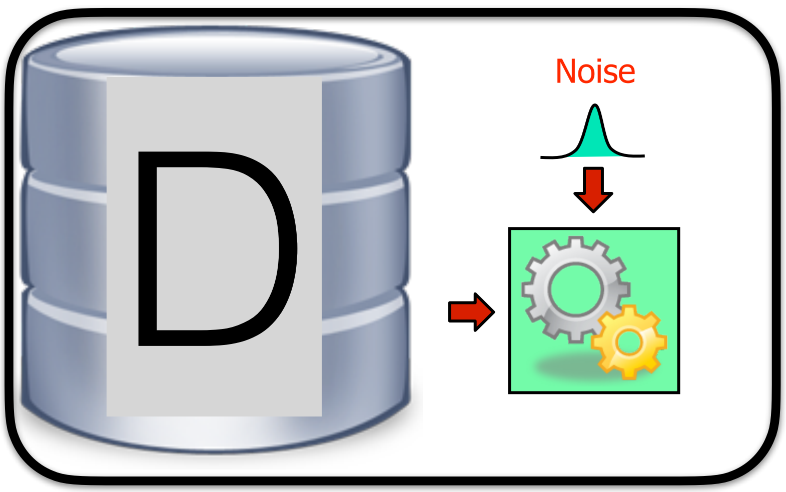
$$\vdash_{k2*\varepsilon,0} \begin{array}{l} x_1 := \$ \ \mathrm{Lap}(\varepsilon, e_1) \\ \sim \\ x_2 := \$ \ \mathrm{Lap}(\varepsilon, e_2) \\ : \ |k_1 + e_1\langle 1\rangle - y_2\langle 2\rangle| \le k_2 \\ \qquad \Longrightarrow x1\langle 1\rangle + k_1 = x\langle 2\rangle \end{array}$$

Releasing partial sums

```
DummySum(d : {0,1} list) : real list
   i:= 0;
   s:= 0;
   r:= [];
   while (i<size d)
       s:= s + d[i];
       z:=$ lap eps s;
       r:= r ++ [z];
       i:= i+1;
   return r
```

I am using the easycrypt notation here where `Lap(eps,a)` corresponds to adding to the value `a` noise from the Laplace distribution with `b=1/eps` and mean `mu=0`.

Composition



M_1 is (ε_1,δ_1)-DP

M_2 is (ε_2,δ_2)-DP

…

M_n is (ε_k,δ_k)-DP

The overall process is $(\varepsilon_1+\varepsilon_2+\ldots+\varepsilon_k,\delta_1+\delta_2+\ldots+\delta_k)$-DP

apRHL

Composition

$$\frac{\vdash_{\varepsilon 1,\delta 1} c_1 \sim c_2 : P \Rightarrow R \quad \vdash_{\varepsilon 2,\delta 2} c_1' \sim c_2' : R \Rightarrow S}{\vdash_{\varepsilon 1+\varepsilon 2,\delta 1+\delta 2} c_1; c_1' \sim c_2; c_2' : P \Rightarrow S}$$

This corresponds to EC command:

seq 1 1 : (postcondition R) <[eps$_1$ & del$_1$]>.

Releasing partial sums

```
DummySum(d : {0,1} list) : real list
   i:= 0;
   s:= 0;
   r:= [];
   while (i<size d)
       s:= s + d[i];
       z:=$ lap eps s;
       r:= r ++ [z];
       i:= i+1;
   return r
```

apRHL
awhile

```
P/\ e<1>≤0 => ¬b1<1>

⊢εk,δk c1~c2:P/\b1<1>/\b2<2>/\k=e<1>
      /\e<1>≤n
       ==> P /\ b1<1>=b2<2> /\k < e<1>
────────────────────────────────────────
⊢∑εk,∑δk while b1 do c1~while b2 do c2
         :P/\ b1<1>=b2<2>/\ e<1> ≤ n
          ==> P /\ ¬b1<1>/\ ¬b2<2>
```

Parallel Composition

Let M_1:DB $\rightarrow$R be a (ε_1,δ_1)-differentially private program and M_2:DB $\rightarrow$R be a (ε_2,δ_2)-differentially private program. Suppose that we partition D in a data-independent way into two datasets D_1 and D_2. Then, the composition $M_{1,2}$:DB$\rightarrow$R defined as

$$MP_{1,2}(D)=(M_1(D_1),M_2(D_2))$$

is $(\max(\varepsilon_1,\varepsilon_2),\max(\delta_1,\delta_2))$-differentially private.

Parallel Composition In EC

```
lemma lem2par : aequiv [ [eps & del]
  M1.f ~ M1.f
    :  (`|M1.d{1}.`1 - M1.d{2}.`1|<= 1 /\ M1.d{1}.`2 = M1.d{2}.`2)
==> res{2} = res{1}].
proof.
  proc.
  seq 3 3 :( `|z{1} - z{2}|<= 1/\ y{1} =y{2}).
  wp. toequiv. skip. trivial.
  seq 1 1 : (r{1}=r{2} /\ y{1} = y{2}) <[ eps & del ]>.
  lap (0) 1 => //.
  lap (0) 0.
(*
  we could have just avoid using laplace at all in the algorithm,
  but the point of this example is to show how the same algorithm
  can be analyzed in different ways. We will also see that this idea
  is behind the idea of parallel composition
*)
qed.
```

Releasing partial sums

```
DummySum(d : {0,1} list) : real list
    i:=0;
    s:=0;
    r:=[];
    while (i<size d)
        z:=$ lap eps d[i];
        s:= s + z;
        r:= r ++ [s];
        i:= i+1;
    return r
```

Parallel Composition In EC

```
lemma noisy_sum n j : 0<= j < n => aequiv [ [ eps & 0%r]
  M3.noisy_sum ~ M3.noisy_sum
    :  adjacent_e ls{1} ls{2} j /\ n=size ls{1}
      ==> res{2} = res{1} ].
    proof.
    move => H. proc.
    seq 3 3: (adjacent_e ls{1} ls{2} j /\ ={i, s, output} /\ i{1} = 0
              /\ s{1} = 0  /\ n=size ls{1}).
      toequiv; auto.
    (* notice the budget function we are using for epsilon *)
    awhile  [ (fun x => if j=n-x -1 then eps else 0%r ) & (fun _ => 0%r) ] n [n -i-1]
    (adjacent_e ls{1} ls{2} j /\ ={i, output} /\ 0 <= i{1} <= size ls{1} /\ n=size ls{1} /\
      (i{1}=j =>
    `|(nth 0 ls{1} i{1})  - (nth 0 ls{2} i{2}) | <= 1 /\
     eq_in_range ls{1} ls{2} (i{1}+1) (size ls{1} - 1)) /\
        0<=  size ls{1}); first 4 try (auto; progress;smt(ge0_eps)).
       search pred1. search iota_. rewrite /bigi.
       rewrite -big_mkcond. apply bigi_eps => //. rewrite sumr_const intmulr; smt().
      move => k.
        have H1: (j = n - k - 1)\/ (j <> n - k - 1). smt().
    elim H1 => [?|?].
        wp. progress.
        lap 0 1. smt().
       progress. smt().
       progress. smt().
       smt(). smt().
       smt(adjacent_sub_abs_bound).
       smt(adjacent_ne_sub_eq_after).
       smt(size_eq_adjacent).
       wp.
       lap 0 0. smt().
       progress.
       smt(size_eq_adjacent).
       auto; progress. smt().
       smt().
smt(adjacent_sub_abs_bound).
smt(adjacent_ne_sub_eq_after).
smt(size_eq_adjacent).
smt(size_eq_adjacent).
qed.
```

Pointwise Differential Privacy

Definition

Given $\varepsilon,\delta \geq 0$, a probabilistic query $Q: X^n \rightarrow R$ is (ε,δ)-differentially private iff

for all adjacent database D, D and for every $S \subseteq R$:

$$\Pr[Q(D) \in S] \leq \exp(\varepsilon)\Pr[Q(D') \in S] + \delta$$

Pointwise Definition

Given $\varepsilon,\delta \geq 0$, a probabilistic query $Q: X^n \rightarrow R$ is (ε,δ)-differentially private iff

for all adjacent database D, D and for every $r \in R$, we have δ_r such that:

$$\Pr[Q(D) \in S] \leq \exp(\varepsilon)\Pr[Q(D') \in S] + \delta_r$$

and $\sum \delta_r \leq \delta$

Above Threshold

```
aboveT (db :int list,n:int,t:int):int = {
    s<-0;
    i <- 0;
    r <- -1;
    nT <$ lap (eps/4%r) t;
    while (i < n) {
      s <$ lap (eps/2%r) (evalQ i db);
      if (nT < s /\ r = -1){
        r <- i;
      }
      i <- i + 1;
    }
    return r;
  }
```

Pointwise DP in Aprhl

$$\frac{\begin{array}{l}\texttt{forall } r \in R \\ \vdash_{\varepsilon,\delta r} c_1 \sim c_2 : P \texttt{ ==> } x\langle 1\rangle = r \texttt{ => } x\langle 2\rangle = r \\ \qquad\qquad \sum \delta r \leq \delta\end{array}}{\vdash_{\varepsilon,\delta}\ c_1 \sim c_2 : P \texttt{ ==> } x\langle 1\rangle = x\langle 2\rangle}$$

The corresponding tactic in EC is

`pweq(r,r)`

Review of the class

Formal Semantics

We need to assign a formal meaning to the different components:

formal semantics of specification conditions

Precondition

Program

Postcondition

formal semantics of programs

formal semantics of specification conditions

We also need to describe the rules which combine program and specifications.

Programming Language

```
c::= abort
   | skip
   | x:=e
   | c;c
   | if e then c else c
   | while e do c
```

$x, y, z, \ldots$ program variables

$e_1, e_2, \ldots$ expressions

$c_1, c_2, \ldots$ commands

Summary of the Semantics of Commands

$\{\texttt{abort}\}_m = \perp$

$\{\texttt{skip}\}_m = m$

$\{\texttt{x:=e}\}_m = m[x \leftarrow \{e\}_m]$

$\{\texttt{c;c'}\}_m = \{c'\}_{m'}$ If $\{c\}_m = m'$

$\{\texttt{c;c'}\}_m = \perp$ If $\{c\}_m = \perp$

$\{\texttt{if e then } c_t \texttt{ else } c_f\}_m = \{c_t\}_m$ If $\{e\}_m = \texttt{true}$

$\{\texttt{if e then } c_t \texttt{ else } c_f\}_m = \{c_f\}_m$ If $\{e\}_m = \texttt{false}$

$\{\texttt{while e do c}\}_m = \sup_{n \in Nat} \{\texttt{while}_n \texttt{ e do c}\}_m$

Hoare triple

Precondition
Program
Postcondition

Precondition
(a logical formula)

$$c : P \Rightarrow Q$$

Program

Postcondition
(a logical formula)

Validity of Hoare triple

We say that the triple $c : P \Rightarrow Q$ is valid

if and only if

for every memory m such that $P(m)$

and memory m' such that $\{c\}_m = m'$

we have $Q(m')$.

Is this condition easy to check?

An example

$\vdash$

```
x:=3;
y:=1;
while x > 1 do
  y := y+1;
  x := x-1;
```

$: \{true\} \Rightarrow \{y = 3\}$

Soundness

If we can derive $\vdash \mathtt{c} : \mathtt{P} \Rightarrow \mathtt{Q}$ through the rules of the logic, then the triple $\mathtt{c} : \mathtt{P} \Rightarrow \mathtt{Q}$ is valid.

Relative Completeness

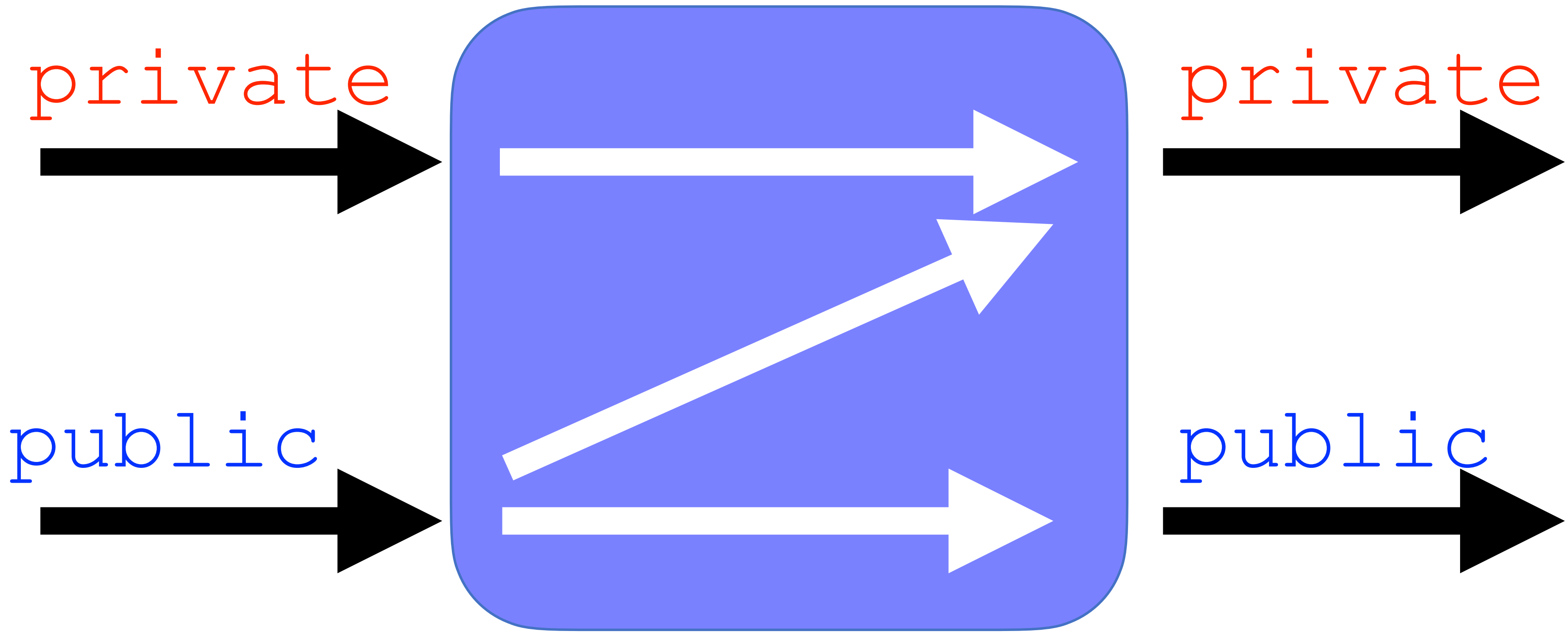
$$\frac{P \Rightarrow S \qquad \vdash c : S \Rightarrow R \qquad R \Rightarrow Q}{\vdash c : P \Rightarrow Q}$$

If a triple $c : P \Rightarrow Q$ is valid, and we have an oracle to derive all the true statements of the form $P \Rightarrow S$ and of the form $R \Rightarrow Q$, then we can derive $\vdash c : P \Rightarrow Q$ through the rules of the logic.

Noninterference

In symbols

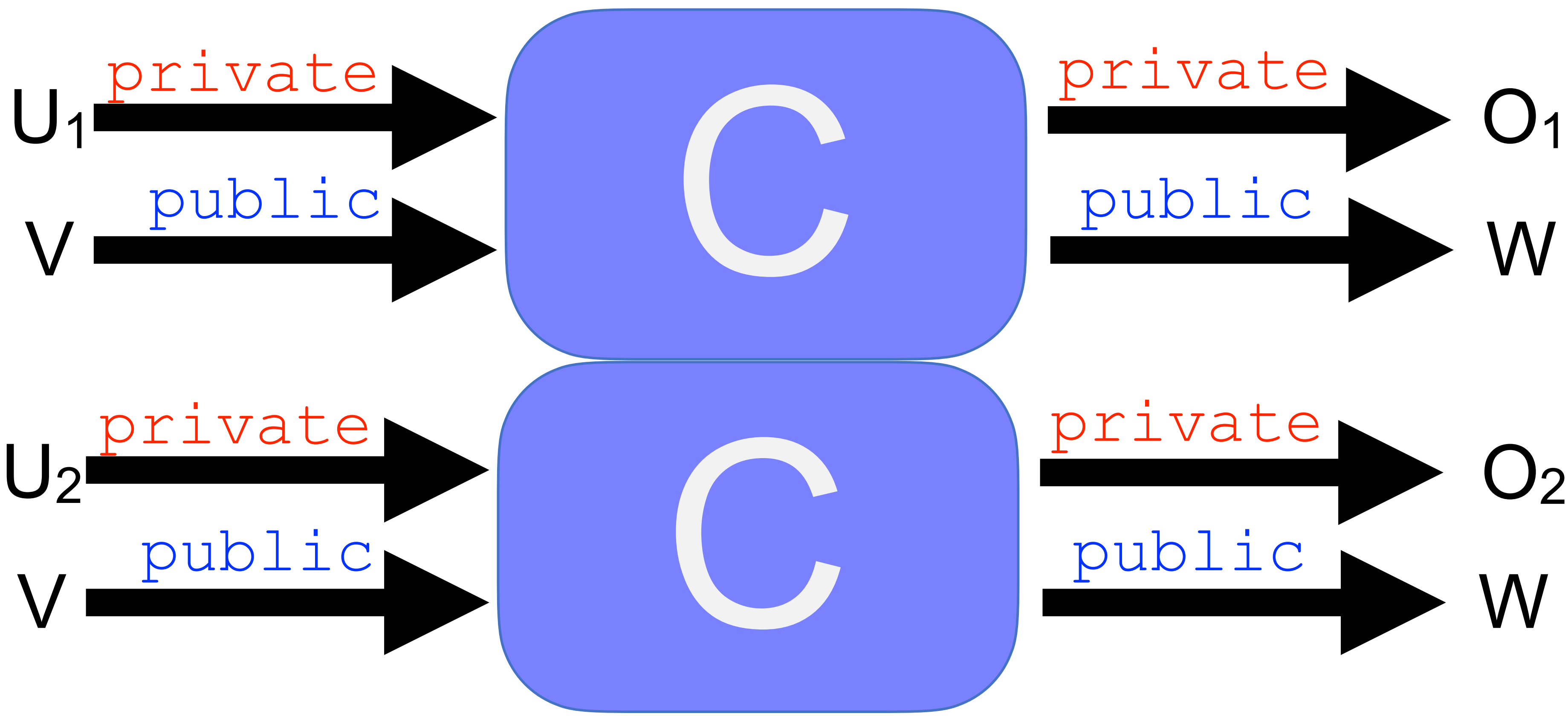
$m_1 \sim_{low} m_2$ and $\{c\}_{m1} = m_1'$ and $m_2'\{c\}_{m2} = m_2'$ implies $m_1' \sim_{low} m_2'$



Relational Property

In symbols, c is noninterferent if and only if for every $m_1 \sim_{low} m_2$:

1) $\{c\}_{m1}=\bot$ iff $\{c\}_{m2}=\bot$
2) $\{c\}_{m1}=m_1'$ and $\{c\}_{m2}=m_2'$ implies $m_1' \sim_{low} m_2'$



Relational Hoare Logic - RHL

Precondition
(a logical formula)

Precondition
$\texttt{Program}_1 \sim \texttt{Program}_2$
Postcondition

$$c_1 \sim c_2 : P \Rightarrow Q$$

Program

Program

Postcondition
(a logical formula)

Validity of Hoare quadruple

We say that the quadruple $c_1 \sim c_2 : P \Rightarrow Q$ is valid if and only if for every pair of memories m_1, m_2 such that $P(m_1, m_2)$ we have:

1) $\{c_1\}_{m1} = \perp$ iff $\{c_2\}_{m2} = \perp$
2) $\{c_1\}_{m1} = m_1'$ and $\{c_2\}_{m2} = m_2'$ implies $Q(m_1', m_2')$.

Is this easy to check?

An example

```
s1:public
s2:private
r:private
i:public

proc Compare (s1:list[n] bool,s2:list[n] bool)
i:=0;
r:=0;
while i<n do
 if not(s1[i]=s2[i]) then
     r:=1
 i:=i+1

: n>0 /\ =low ⇒ =low
```

Soundness and completeness with respect to Hoare Logic

$$\vdash_{\mathtt{RHL}} \mathtt{c_1} \sim \mathtt{c_2} : \mathtt{P} \Rightarrow \mathtt{Q}$$

iff

$$\vdash_{\mathtt{HL}} \mathtt{c_1;c_2} : \mathtt{P} \Rightarrow \mathtt{Q}$$

Under the assumption that we can partition the memory adequately, and that we have termination.

An example

```
OneTimePad(m : private msg) : public msg
   key :=$ Uniform({0,1}^n);
   cipher := msg xor key;
   return cipher
```

Learning a ciphertext does not change any a priori knowledge about the likelihood of messages.

Probabilistic While (PWhile)

```
c::= abort
   | skip
   | x:= e
   | x:=$ d
   | c;c
   | if e then c else c
   | while e do c
```

$d_1, d_2, \ldots$ probabilistic expressions

Semantics of Commands

This is defined on the structure of commands:

$\{\texttt{abort}\}_m = \mathbf{0}$

$\{\texttt{skip}\}_m = \texttt{unit}(m)$

$\{\texttt{x:=e}\}_m = \texttt{unit}(m[x \leftarrow \{e\}_m])$

$\{\texttt{x:=\$ d}\}_m = \texttt{let } a=\{d\}_m \texttt{ in unit}(m[x \leftarrow a])$

$\{\texttt{c;c}'\}_m = \texttt{let } m'=\{c\}_m \texttt{ in } \{c'\}_{m'}$

$\{\texttt{if e then } c_t \texttt{ else } c_f\}_m = \{c_t\}_m$ If $\{e\}_m = \texttt{true}$

$\{\texttt{if e then } c_t \texttt{ else } c_f\}_m = \{c_f\}_m$ If $\{e\}_m = \texttt{false}$

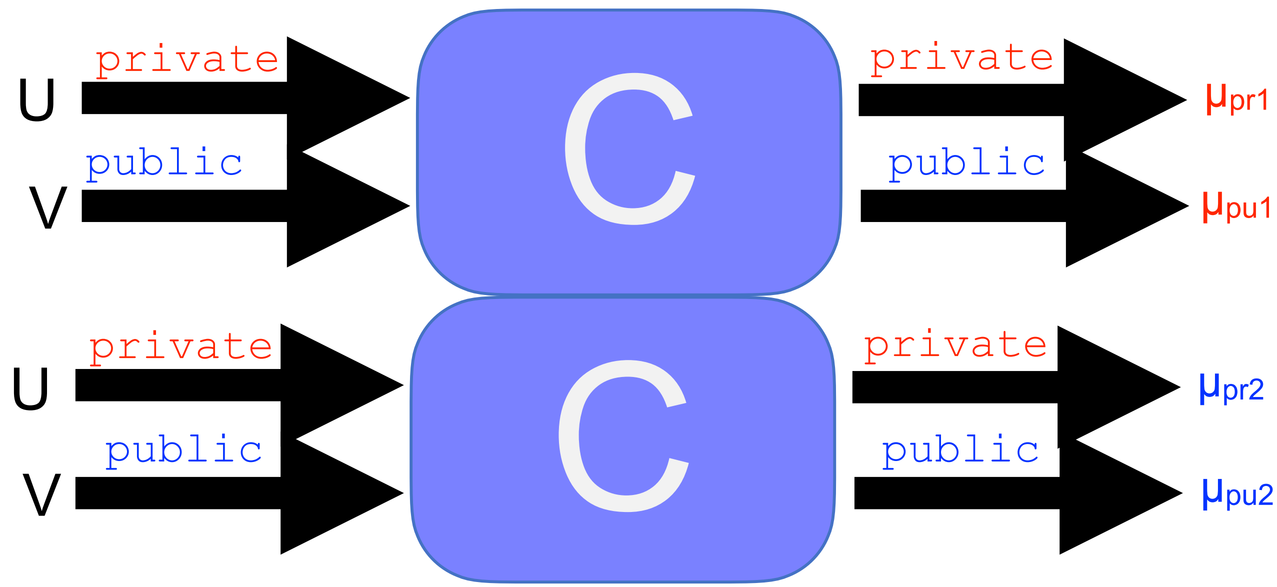
$\{\texttt{while e do c}\}_m = \sup_{n \in Nat} \mu_n$

$\mu_n =$

$\texttt{let } m'=\{(\texttt{while}^n \texttt{ e do c})\}_m \texttt{ in } \{\texttt{if e then abort}\}_{m'}$

Probabilistic Noninterference as a Relational Property

c is probabilistically noninterferent if and only if for every $m_1 \sim_{low} m_2$:
$\{c\}_{m1}=\mu_1$ and $\{c\}_{m2}=\mu_2$ implies $\mu_1 \sim_{low} \mu_2$



Probabilistic Relational Hoare Quadruples

Precondition

(a logical formula)

Precondition

Program₁ ~ Program₂

Postcondition

$$c_1 \sim c_2 : P \Rightarrow Q$$

Probabilistic Program

Probabilistic Program

Postcondition

(a logical formula)

Validity of Probabilistic Hoare quadruple

We say that the quadruple $c_1 \sim c_2 : P \Rightarrow Q$ is valid if and only if for every pair of memories m_1, m_2 such that $P(m_1, m_2)$ we have: $\{c_1\}_{m1} = \mu_1$ and $\{c_2\}_{m2} = \mu_2$ implies $Q^*(\mu_1, \mu_2)$.

Relational lifting of a predicate

We say that two subdistributions $\mu_1 \subseteq D(A)$ and $\mu_2 \subseteq D(B)$ are in the relational lifting of the relation `R`$\subseteq$AxB, denoted μ_1 `R*` μ_2 if and only if there exist a subdistribution $\mu \subseteq D(AxB)$ such that:

1) if μ`(a,b)>0, then (a,b)`$\in$`Q`.
2) $\pi_1(\mu) = \mu_1$ and $\pi_2(\mu) = \mu_2$

R-Coupling

Given two distributions $\mu_1 \in D(A)$, and $\mu_2 \in D(B)$, an R-coupling between them, for $R \subseteq AxB$, is a joint distribution $\mu \in D(AxB)$ such that:

1) the marginal distributions of μ are μ_1 and μ_2, respectively,
2) the support of μ is contained in R. That is, if μ`(a,b)>0, then (a,b)`$\in$R.

A sufficient condition for R-Coupling

Given two distributions $\mu_1 \in D(A)$, and $\mu_2 \in D(B)$, and a relation $R \subseteq A \times B$, if there is a mapping $h: A \to B$ such that:

1) h is a bijective map between elements in $supp(\mu_1)$ and $supp(\mu_2)$,
2) for every $a \in supp(\mu_1)$, $(a, h(a)) \in R$
3) $\Pr_{x \sim \mu 1}[x = a] = \Pr_{x \sim \mu 2}[x = h(a)]$

Then, there is an R-coupling between μ_1 and μ_2.
We write $h \lhd (\mu_1, \mu_2)$ in this case.

Probabilistic Relational Hoare Logic
Random Assignment

$$\frac{\begin{array}{l} h \lhd (\{d_1\},\{d_2\}) \\ P = \forall v, v \in supp(\{d_1\}) \\ \Rightarrow\ Q[v/x_1\langle 1\rangle, h(v)/x_2\langle 2\rangle] \end{array}}{\vdash x_1 :=\$\ d_1 \sim x_2 :=\$\ d_2 : P \Rightarrow Q}$$

Consequences of Coupling

Given the following pRHL judgment

$$\vdash c_1 \sim c_2 : \text{True} \Rightarrow Q$$

We have that:

if $Q \Rightarrow (R\langle 1 \rangle \iff S\langle 2 \rangle)$, then $\Pr[c_1 : R] = \Pr[c_2 : S]$

if $Q \Rightarrow (R\langle 1 \rangle \Rightarrow S\langle 2 \rangle)$, then $\Pr[c_1 : R] \leq \Pr[c_2 : S]$

A more realistic example

```
StreamCipher(m : private msg[n]) : public msg[n]
  pkey :=$ PRG(Uniform({0,1}^k));
  cipher := msg xor pkey;
  return cipher
```

Properties of PRG

We would like the PRG to increase the number of random bits but also to guarantee the result to be (almost) random.

We can express this as:

$\boxed{\text{PRG: } \{0,1\}^k \rightarrow \{0,1\}^n \text{ for } n>k}$

$\boxed{\Delta(\text{PRG}(\text{Uniform}(\{0,1\}^k),\text{Uniform}(\{0,1\}^n) \leq 2^{-n}}$

In fact this is a too strong requirement - usually we require that every polynomial time adversary cannot distinguish the two distributions in statistical distance

Approximate Probabilistic Relational Hoare Logic

Indistinguishability parameter

Precondition (a logical formula)

$$\vdash_{\delta} c_1 \sim c_2 : P \Rightarrow Q$$

Probabilistic Program

Probabilistic Program

Postcondition (a logical formula)

$\mathtt{R}$–δ–Coupling

Given two distributions $\mu_1 \in D(A)$, and $\mu_2 \in D(B)$, we have an $\mathtt{R}$-δ-coupling between them, for $\mathtt{R} \subseteq A\mathrm{x}B$ and $0 \leq \delta \leq 1$, if there are two joint distributions $\mu_{L}, \mu_R \in D(A\mathrm{x}B)$ such that:

1) $\pi_1(\mu_L)=\mu_1$ and $\pi_2(\mu_R)=\mu_2$,
2) the support of μ_L and μ_R is contained in $\mathtt{R}$. That is, if $\mu_L\mathtt{(a,b)>0}$, $\mathtt{then}$ $\mathtt{(a,b)} \in \mathtt{R}$, $\mathtt{and}$ if $\mu_R\mathtt{(a,b)>0}$, $\mathtt{then}$ $\mathtt{(a,b)} \in \mathtt{R}$.
3) $\Delta(\mu_L, \mu_R) \leq \delta$

Example of R-δ-Coupling

μ_1

00	0.2
01	0.25
10	0.25
11	0.3

$$R(a,b) = \{a \leq b\}$$

μ_2

00	0
01	0.40
10	0
11	0.6

μ_L	00	01	10	11
00		0.20		
01		0.25		
10				0.25
11				0.30

μ_R	00	01	10	11
00		0.20		
01		0.20		
10				0.3
11				0.3

$$\Delta(\mu_L, \mu_R) = 0.05$$

Approximate relational lifting of a predicate

We say that two subdistributions $\mu_1 \subseteq D(A)$ and $\mu_2 \subseteq D(B)$ are in the relational δ–lifting of the relation $R \subseteq A \times B$, denoted $\mu_1 \; R_\delta^* \; \mu_2$ if and only if there exist an R-coupling between them.

Validity of approximate Probabilistic Hoare judgments

We say that the quadruple $\vdash_\delta c_1{\sim}c_2 : P \Rightarrow Q$ is valid if and only if for every pair of memories m_1, m_2 such that $P(m_1, m_2)$ we have: $\{c_1\}_{m1} = \mu_1$ and $\{c_2\}_{m2} = \mu_2$ implies $Q_\delta{}^*(\mu_1, \mu_2)$.

Releasing the mean of Some Data

```
Mean(d : private data) : public real
   i:=0;
   s:=0;
   while (i<size(d))
       s:=s + d[i]
       i:=i+1;
   return (s/i)
```

(ε,δ)-Differential Privacy

Definition

Given $\varepsilon,\delta \geq 0$, a probabilistic query $Q: X^n \rightarrow R$ is (ε,δ)-differentially private iff

for all adjacent database b_1, b_2 and for every $S \subseteq R$:

$$\Pr[Q(b_1) \in S] \leq \exp(\varepsilon)\Pr[Q(b_2) \in S] + \delta$$

(ε,δ)-indistinguishability

We can define a ε-skewed version of statistical distance. We call this notion ε-distance.

$$\Delta_\varepsilon(\mu 1,\mu 2)=\sup_{E\subseteq A} \max(\mu_1(E)-e^{\varepsilon}\mu_2(E),\ \mu_2(E)-e^{\varepsilon}\mu_1(E),0)$$

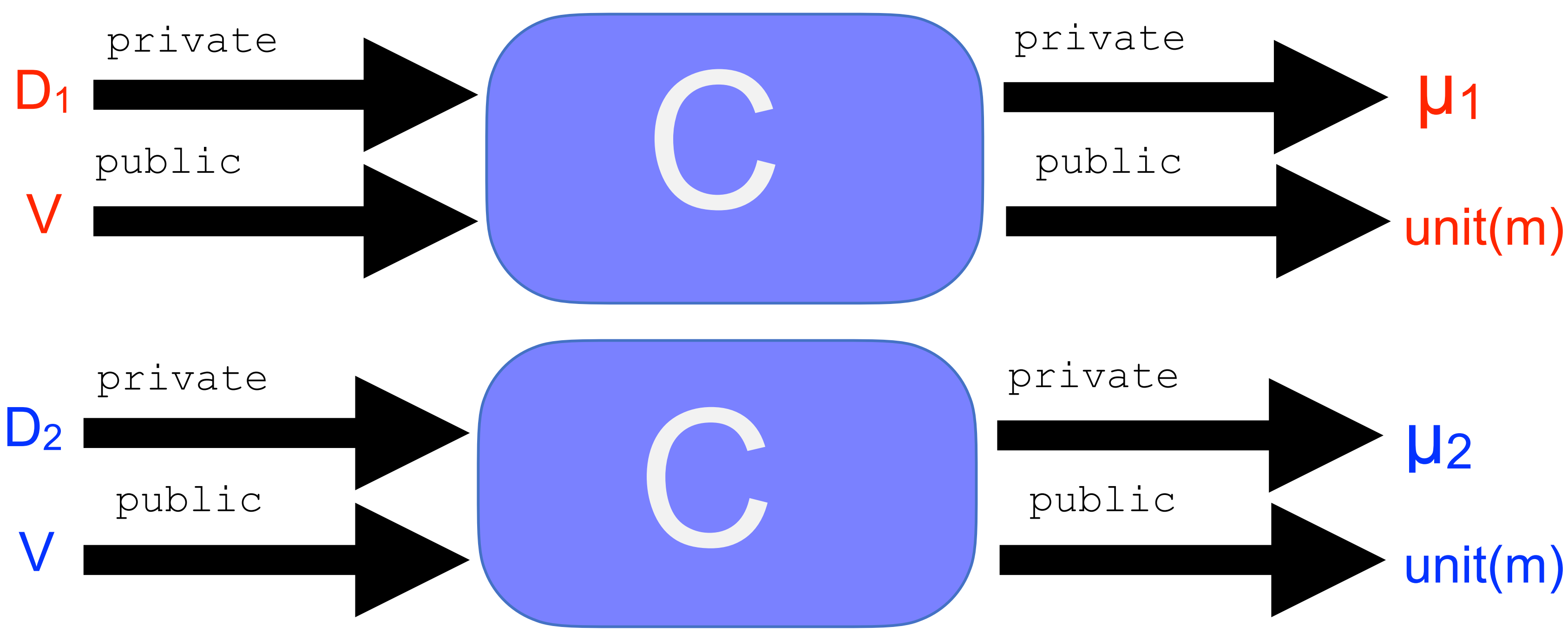
We say that two distributions $\mu_1, \mu_2 \in D(A)$, are at (ε,δ)-indistinguishable if:

$$\Delta_\varepsilon(\mu 1,\mu 2) \leq \delta$$

Differential Privacy as a Relational Property

c is differentially private if and only if for every $m_1 \sim m_2$ (extending the notion of adjacency to memories):

$\{c\}_{m1}=\mu_1$ and $\{c\}_{m2}=\mu_2$ implies $\Delta_\varepsilon(\mu_1,\mu_2) \leq \delta$



apRHL

Indistinguishability parameter

Precondition (a logical formula)

$$\vdash_{\epsilon,\delta} c_1 \sim c_2 : P \Rightarrow Q$$

Probabilistic Program

Probabilistic Program

Postcondition (a logical formula)

Validity of apRHL judgments

We say that the quadruple $\vdash_{\varepsilon,\delta} c_1 \sim c_2 : P \Rightarrow Q$ is valid if and only if for every pair of memories m_1, m_2 such that $P(m_1, m_2)$ we have:

$\{c_1\}_{m1} = \mu_1$ and $\{c_2\}_{m2} = \mu_2$ implies $Q_{\varepsilon,\delta}{}^*(\mu_1, \mu_2)$.

R–ε–δ–Coupling

Given two distributions $\mu_1 \in D(A)$, and $\mu_2 \in D(B)$, we have an R-ε-δ-coupling between them, for $R \subseteq A \times B$ and $0 \le \varepsilon$ and $0 \le \delta \le 1$, if there are two joint distributions $\mu_L, \mu_R \in D(A \times B)$ such that:

1) $\pi_1(\mu_L) = \mu_1$ and $\pi_2(\mu_R) = \mu_2$,
2) the support of μ_L and μ_R is contained in R. That is, if $\mu_L(a,b) > 0$, then $(a,b) \in R$, and if $\mu_R(a,b) > 0$, then $(a,b) \in R$.
3) $\Delta_\varepsilon(\mu_L, \mu_R) \le \delta$

Symmetric Encryption Schemes

- Our treatment of symmetric encryption schemes is parameterized by three types:

```
type key.   (* encryption keys, key_len bits *)
type text.   (* plaintexts, text_len bits *)
type cipher.   (* ciphertexts - scheme specific *)
```

- An encryption scheme is a *stateless* implementation of this module interface:

```
module type ENC = {
  proc key_gen() : key   (* key generation *)
  proc enc(k : key, x : text) : cipher   (* encryption *)
  proc dec(k : key, c : cipher) : text   (* decryption *)
}.
```

IND-CPA Game

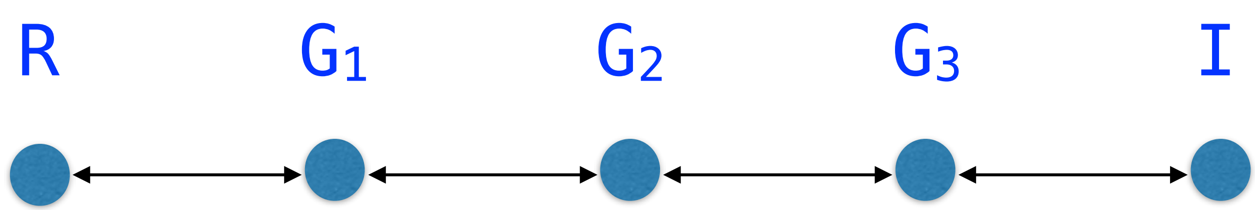
- The IND-CPA Game is parameterized by an encryption scheme and an encryption adversary:

```
module INDCPA (Enc : ENC, Adv : ADV) = {
  module EO = EncO(Enc)          (* make EO from Enc *)
  module A = Adv(EO)             (* connect Adv to EO *)
  proc main() : bool = {
    var b, b' : bool; var x1, x2 : text; var c : cipher;
    EO.init();                   (* initialize EO *)
    (x1, x2) <@ A.choose();      (* let A choose x1/x2 *)
    b <$ {0,1};                  (* choose boolean b *)
    c <@ EO.genc(b ? x1 : x2);   (* encrypt x1 or x2 *)
    b' <@ A.guess(c);            (* let A guess b from c *)
    return b = b';               (* see if A won *)
  }
}.
```

Sequence of Games Approach

- Our proof of IND-CPA security uses the *sequence of games approach*, which is used to connect a "real" game `R` with an *"ideal"* game `I` via a sequence of intermediate games.
- Each of these games is parameterized by the adversary, and each game has a `main` procedure returning a boolean.
- We want to establish an upper bound for

```
`| Pr[R.main() @ &m : res] - Pr[I.main() : res] |
```



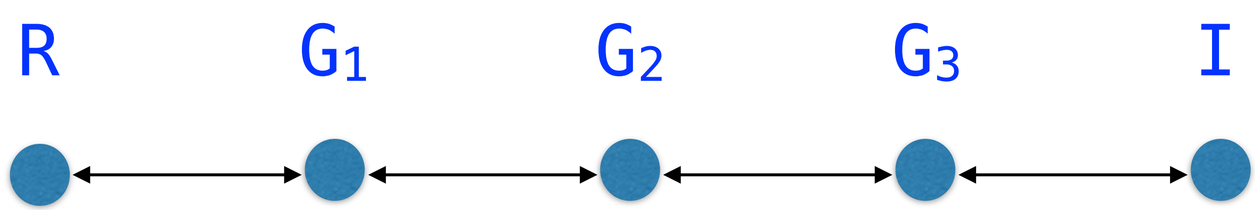
Sequence of Games Approach

- Suppose we can prove

```
`| Pr[R.main() @ &m : res] – Pr[G1.main() : res] | <= b1
`| Pr[G1.main() @ &m : res] – Pr[G2.main() : res] | <= b2
`| Pr[G2.main() @ &m : res] – Pr[G3.main() : res] | <= b3
`| Pr[G3.main() @ &m : res] – Pr[I.main() : res] | <= b4
```

for some b_1, b_2, b_3 and b_4. Then we can conclude

```
`| Pr[R.main() @ &m : res] – Pr[I.main() @ &m : res] | <=
   b1 + b2 + b3 + b4
```



Step 1: Replacing PRF with TRF

- In our first step, we switch to using a true random function instead of a pseudorandom function in our encryption scheme.
    - We have an exact model of how the TRF works.
- When doing this, we inline the encryption scheme into a new kind of encryption oracle, `EO_RF`, which is parameterized by a random function.
- We also instrument `EO_RF` to detect two kinds of "clashes" (repetitions) in the generation of the inputs to the random function.
        - This is in preparation for Steps 2 and 3.

Step 2: Oblivious Update in `genc`

- In Step 2, we make use of up to bad reasoning, to transition to a game in which the encryption oracle, **`EO_O`**, uses a true random function and "obliviously" ("O" for "oblivious") updates the true random function's map — i.e., overwrites what may already be stored in the map.

Step 3: Independent Choice in `genc`

- In Step 3, we again make use of up to bad reasoning, this time transitioning to a game in which the encryption oracle, `EO_I`, chooses the text value to be exclusive or-ed with the plaintext in a way that is “independent” (“I” for “independent”) from the true random function’s map, i.e., without updating that map.
- We no longer need to detect “pre” clashes (clashes in `genc` with a `u` chosen in a call to `enc_pre`).

Step 4: One-time Pad Argument

- In Step 4, we can switch to an encryption oracle `EO_N` in which the right side of the ciphertext produced by `EO_N.genc` makes no ("N" for "no") reference to the plaintext.
- We no longer need any instrumentation for detecting clashes.

Step 5: Proving G4's Probability

- When proving

```
local lemma G4_prob &m :
  Pr[G4.main() @ &m : res] = 1%r / 2%r.
```

we can reorder

```
b <$ {0,1};
c <@ EO_N.genc(text0);
b' <@ A.guess(c);
return b = b';
```

to

```
c <@ EO_N.genc(text0);
b' <@ A.guess(c);
b <$ {0,1};
return b = b';
```

- We use that Adv's procedures are lossless.

IND-CPA Security Result

```
lemma INDCPA (Adv <: ADV{Enc0, PRF, TRF, Adv2RFA}) &m :
  (forall (EO <: EO{Adv}),
   islossless EO.enc_pre => islossless Adv(EO).choose) =>
  (forall (EO <: EO{Adv}),
   islossless EO.enc_post => islossless Adv(EO).guess) =>
  `|Pr[INDCPA(Enc, Adv).main() @ &m : res] -
    1%r / 2%r| <=
  `|Pr[GRF(PRF, Adv2RFA(Adv)).main() @ &m : res] -
    Pr[GRF(TRF, Adv2RFA(Adv)).main() @ &m : res]| +
  (limit_pre%r + limit_post%r) / (2 ^ text_len)%r.
```

- Q: If we remove the restriction on `Adv` (`{Enc0, PRF, TRF, Adv2RFA}`), what would happen?
- A: Various tactic applications would fail; e.g., calls to the `Adv`'s procedures, as they could invalidate assumptions.

Any Question?