CS 591: Formal Methods in Security and Privacy

RHL and probabilistic computations

Marco Gaboardi
gaboardi@bu.edu

Alley Stoughton
stough@bu.edu

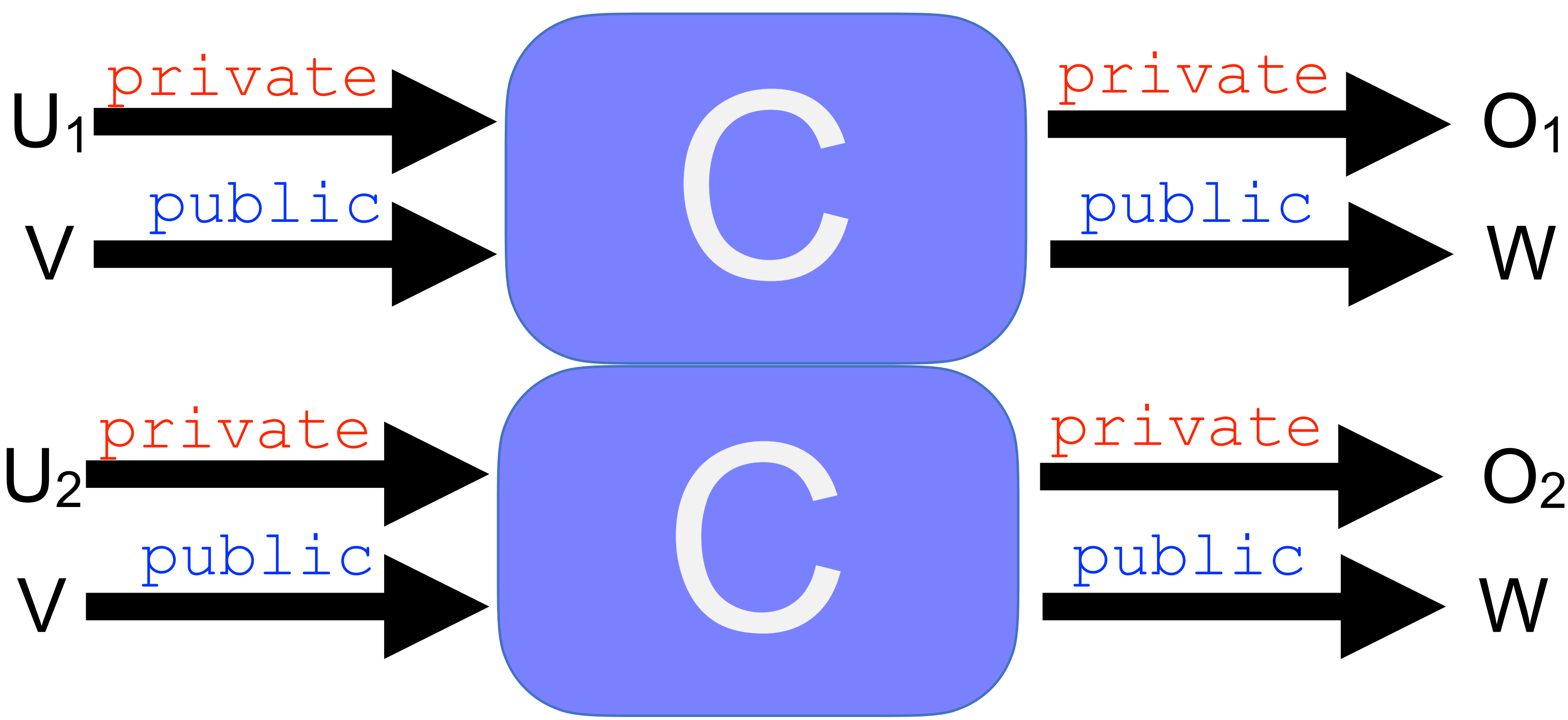
From the previous classes

NonInterference

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

Program$_1$ ~ Program$_2$

Postcondition

Precondition
(a logical formula)

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

Program   Program   Postcondition
(a logical formula)

Some Rules of Relational Hoare Logic

$$\frac{}{\vdash \mathtt{skip} \sim \mathtt{skip} : P \Rightarrow P}$$

$$\frac{}{\vdash \mathtt{abort} \sim \mathtt{abort} : \mathtt{true} \Rightarrow \mathtt{false}}$$

$$\frac{}{\vdash x_1 := e_1 \sim x_2 := e_2 : P[e_1\langle 1\rangle / x_1\langle 1\rangle, e_2\langle 2\rangle / x_2\langle 2\rangle] \Rightarrow P}$$

$$\frac{\vdash c_1 \sim c_2 : P \Rightarrow R \qquad \vdash c_1' \sim c_2' : R \Rightarrow S}{\vdash c_1 ; c_1' \sim c_2 ; c_2' : P \Rightarrow S}$$

$$\frac{P \Rightarrow S \qquad \vdash c_1 \sim c_2 : S \Rightarrow R \qquad R \Rightarrow Q}{\vdash c_1 \sim c_2 : P \Rightarrow Q}$$

Some Rules of Relational Hoare Logic

$$\frac{\vdash c_1 \sim c_2 : e_1\langle 1\rangle \wedge P \Rightarrow Q \qquad P \Rightarrow e_1\langle 1\rangle = e_2\langle 2\rangle \qquad \vdash c_1' \sim c_2' : \neg e_1\langle 1\rangle \wedge P \Rightarrow Q}{\vdash \begin{array}{c}\texttt{if } e_1 \texttt{ then } c_1 \texttt{ else } c_1' \\ \sim \\ \texttt{if } e_2 \texttt{ then } c_2 \texttt{ else } c_2'\end{array} : P \Rightarrow Q}$$

$$\frac{\vdash c_1 \sim c_2 : e_1\langle 1\rangle \wedge P \Rightarrow P \qquad P \Rightarrow e_1\langle 1\rangle = e2\langle 2\rangle}{\vdash \begin{array}{c}\texttt{while } e_1 \texttt{ do } c_1 \\ \sim \\ \texttt{while } e_2 \texttt{ do } c_2\end{array} : P \Rightarrow P \wedge \neg e_1\langle 1\rangle}$$

One-sided Rules

$$\frac{\vdash c_1 \sim c_2 : e\langle 1\rangle \wedge P \Rightarrow Q \qquad \vdash c_1' \sim c_2 : \neg e\langle 1\rangle \wedge P \Rightarrow Q}{\vdash \begin{array}{c}\texttt{if } e \texttt{ then } c_1 \texttt{ else } c_1' \\ \sim \\ c_2\end{array} : P \Rightarrow Q}$$

$$\frac{\vdash c_1 \sim c_2 : e\langle 2\rangle \wedge P \Rightarrow Q \qquad \vdash c_1 \sim c_2' : \neg e\langle 2\rangle \wedge P \Rightarrow Q}{\vdash \begin{array}{c} c_1 \\ \sim \\ \texttt{if } e \texttt{ then } c_2 \texttt{ else } c_2'\end{array} : P \Rightarrow Q}$$

How can we prove this?

```
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
```

Today: more on RHL and probabilistic computations

What do we do if our two programs have different forms? There are three pairs of *one-sided* rules.

Assignment — left

$$\frac{}{\vdash x := e \sim \mathtt{skip} : P[e\langle 1\rangle / x\langle 1\rangle] \Rightarrow P}$$

Assignment — right

$$\frac{}{\vdash \texttt{skip} \sim \texttt{x:=e} : \texttt{P[e<2>/x<2>]} \Rightarrow \texttt{P}}$$

Also pair of one-sided rules for while — we'll ignore for now

Rules of Relational Hoare Logic
Program Equivalence Rule

$$\dfrac{\models P : c_1' \equiv c_1 \qquad \models P : c_2' \equiv c_2 \qquad \vdash c_1' \sim c_2' : P \Rightarrow Q}{\vdash c_1 \sim c_2 : P \Rightarrow Q}$$

$\models P : c_1 \equiv c_2$ means $\{c_1\}_m = \{c_2\}_m$ for all m such that $P(m)$

Rules of Relational Hoare Logic
Program Equivalences

$\models$P : skip;c ≡ c

$\models$P : c;skip ≡ c

$\models$P:(c1;c2);c3 ≡ c1;(c2;c3)

...

Rules of Relational Hoare Logic

Combining Composition and Equivalence

We can combine the Composition and Program Equivalence Rules to split commands where we like:

$$\dfrac{\vdash c_1;c_2 \sim c_1' : P \Rightarrow R \qquad \vdash c_3 \sim c_2';c_3' : R \Rightarrow Q}{\vdash c_1;c_2;c_3 \sim c_1';c_2';c_3' : P \Rightarrow Q}$$

Rules of Relational Hoare Logic

Combining Composition and Equivalence

$$\frac{\dfrac{\vdash c_1 \sim \texttt{skip} : P \Rightarrow R \qquad \vdash c_2 \sim c_1' : R \Rightarrow Q}{\vdash c_1; c_2 \sim \texttt{skip}; c_1' : P \Rightarrow Q}}{\vdash c_1; c_2 \sim c_1' : P \Rightarrow Q}$$

Rules of Relational Hoare Logic

Combining Composition and Equivalence

$$\vdash c_1 \sim c_1' : P \Rightarrow R \qquad \vdash c_2 \sim \mathtt{skip} : R \Rightarrow Q$$

$$\frac{\qquad}{\vdash c_1;c_2 \sim c_1';\mathtt{skip} : P \Rightarrow Q}$$

$$\frac{\qquad}{\vdash c_1;c_2 \sim c_1' : P \Rightarrow Q}$$

Soundness

If we can derive $\vdash c_1 \sim c_2 : P \Rightarrow Q$ through the rules of the logic, then the quadruple $c_1 \sim c_2 : P \Rightarrow Q$ is valid.

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

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 $Q(m_1', m_2')$.

How do we check this?

Relative Completeness

If a quadruple $c_1 \sim c_2 : P \Rightarrow Q$ 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 $\vdash c_1 \sim c_2 : P \Rightarrow Q$ through the rules of the logic.

Soundness and completeness with respect to Hoare Logic

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

iff

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

Soundness and completeness with respect to Hoare Logic

$$\vdash_{\mathtt{RHL}} \mathtt{c}_1 \sim \mathtt{c}_2 : \mathtt{P} \Rightarrow \mathtt{Q}$$

iff

$$\vdash_{\mathtt{HL}} \mathtt{c}_1 ; \mathtt{c}_2 : \mathtt{P} \Rightarrow \mathtt{Q}$$

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

Possible projects

In Easycrypt

- Look at how to guarantee trace-based noninterference.
- Look at how to guarantee side-channel free noninterference.
- Look at the relations between self-composition and relational logic.

Not related to Easycrypt

- Look at type systems for non-interference.
- Look at other methods for relational reasoning
- Look at declassification

Probabilistic Language

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

Probabilistic Expressions

We extend the language with expression describing probability distributions.

$$d ::= f(e_1, \ldots, e_n, d_1, \ldots, d_k)$$

Where `f` is a distribution declaration

Some expression examples

`uniform(`$\{0,1\}^n$`)`   `gaussian(k,σ)`   `laplace(k,b)`

Semantics of Probabilistic Expressions

We would like to define it on the structure:

$$\{f(e_1,\dots,e_n,d_1,\dots,d_k)\}_m = \{f\}(\{e_1\}_m,\dots,\{e_n\}_m,\{d_1\}_m,\dots,\{d_k\}_m)$$

but is the result just a value?

Probabilistic Subdistributions

A discrete subdistribution over a set A is a function

$\mu : A \to [0, 1]$

such that the mass of μ,

$$|\mu| = \sum_{a \in A} \mu(a)$$

verifies $|\mu| \leq 1$.

The support of a discrete subdistribution μ,

$\mathrm{supp}(\mu) = \{a \in A \mid \mu(a) > 0\}$

is necessarily countable, i.e. finite or countably infinite.

We will denote the set of sub-distributions over A by D(A), and say that μ is of type D(A) denoted μ:D(A) if $\mu \in D(A)$.

Probabilistic Subdistributions

We call a subdistribution with mass exactly 1, a distribution.

We define the probability of an event $E \subseteq A$ with respect to the subdistribution $\mu : D(A)$ as

$$\mathbb{P}_{\mu}[E] = \sum_{a \in E} \mu(a)$$

Probabilistic Subdistributions

Let's consider μ∈D(A), and E⊆A, we have the following properties

$$\mathbb{P}_\mu[\varnothing] = 0$$

$$\mathbb{P}_\mu[A] \leq 1$$

$$0 \leq \mathbb{P}_\mu[E] \leq 1$$

E⊆F⊆A implies $\mathbb{P}_\mu[E] \leq \mathbb{P}_\mu[F]$

E⊆A and F⊆A implies $\mathbb{P}_\mu[E \cup F] \leq \mathbb{P}_\mu[E] + \mathbb{P}_\mu[F] - \mathbb{P}_\mu[E \cap F]$

We will denote by **O** the subdistribution μ defined as constant 0.

Operations over Probabilistic Subdistributions

Let's consider an arbitrary a∈A, we will often use the distribution unit(a) defined as:

$$\mathbb{P}_{\text{unit}(a)}[\{b\}] = \begin{cases} \text{1 if a=b} \\ \text{0 otherwise} \end{cases}$$

We can think about unit as a function of type unit:A → D(A)

Operations over Probabilistic Subdistributions

Let's consider a distribution $\mu \in D(A)$, and a function $M : A \rightarrow D(B)$ then we can define their composition by means of an expression let a =μ in M a defined as:

$$\mathbb{P}_{\text{let a =}\mu\text{ in M a}}[E] = \sum_{a \in \text{supp}(\mu)} \mathbb{P}_{\mu}[\{a\}] \cdot \mathbb{P}_{(Ma)}[E]$$

Semantics of Probabilistic Expressions - revisited

We would like to define it on the structure:

$$\{f(e_1,\ldots,e_n,d_1,\ldots,d_k)\}_m = \{f\}(\{e_1\}_m,\ldots,\{e_n\}_m,\{d_1\}_m,\ldots,\{d_k\}_m)$$

With input a memory m and output a subdistribution $\mu \in D(A)$ over the corresponding type A. E.g.

$$\{uniform(\{0,1\}^n)\}_m \in D(\{0,1\}^n)$$

$$\{gaussian(k,\sigma)\}_m \in D(Real)$$

Semantics of PWhile Commands

What is the meaning of the following command?

```
k:=$ uniform({0,1}^n); z:= x mod k;
```

Semantics of PWhile Commands

What is the meaning of the following command?

```
k:=$ uniform({0,1}^n); z:= x mod k;
```

We can give the semantics as a function between command, memories and subdistributions over memories.

$$\texttt{Cmd} * \texttt{Mem} \longrightarrow \texttt{D(Mem)}$$

We will denote this relation as:

$$\{c\}_m = \mu$$

Semantics of Commands

This is defined on the structure of commands:

Semantics of Commands

This is defined on the structure of commands:

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

Semantics of Commands

This is defined on the structure of commands:

$\{\texttt{abort}\}_{\texttt{m}} = \mathbf{0}$

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

Semantics of Commands

This is defined on the structure of commands:

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

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

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

Semantics of Commands

This is defined on the structure of commands:

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

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

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

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

Semantics of Commands

This is defined on the structure of commands:

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

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

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

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

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

Semantics of Commands

This is defined on the structure of commands:

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

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

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

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

$$\{\texttt{if e then c}_t \texttt{ else c}_f\}_m = \{\texttt{c}_t\}_m \text{ If } \{\texttt{e}\}_m\texttt{=true}$$

$$\{\texttt{if e then c}_t \texttt{ else c}_f\}_m = \{\texttt{c}_f\}_m \text{ If } \{\texttt{e}\}_m\texttt{=false}$$

Semantics of Commands

This is defined on the structure of commands:

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

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

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

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

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

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

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

Semantics of While

What about while

How did we handle the deterministic case?

Semantics of While

What about while

$$\{\texttt{while e do c}\}_{\mathtt{m}} = \texttt{???}$$

How did we handle the deterministic case?

Semantics of While

We defined it as

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

Where

$\mu_n =$

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

Semantics of While

We defined it as

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

Where

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

Is this well defined?

Semantics of Commands

This is defined on the structure of commands:

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

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

Semantics of Commands

This is defined on the structure of commands:

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

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

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

Semantics of Commands

This is defined on the structure of commands:

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

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

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

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

Semantics of Commands

This is defined on the structure of commands:

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

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

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

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

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

Semantics of Commands

This is defined on the structure of commands:

$\{\text{abort}\}_m = \mathbf{O}$

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

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

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

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

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

Semantics of Commands

This is defined on the structure of commands:

$\{abort\}_m = 0$

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

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

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

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

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

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

Semantics of Commands

This is defined on the structure of commands:

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

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

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

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

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

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

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

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

Semantics of Commands

This is defined on the structure of commands:

$\{abort\}_m = 0$

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

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

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

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

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

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

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

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

Revisiting the 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.

Revisiting the 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.

How do we formalize this?

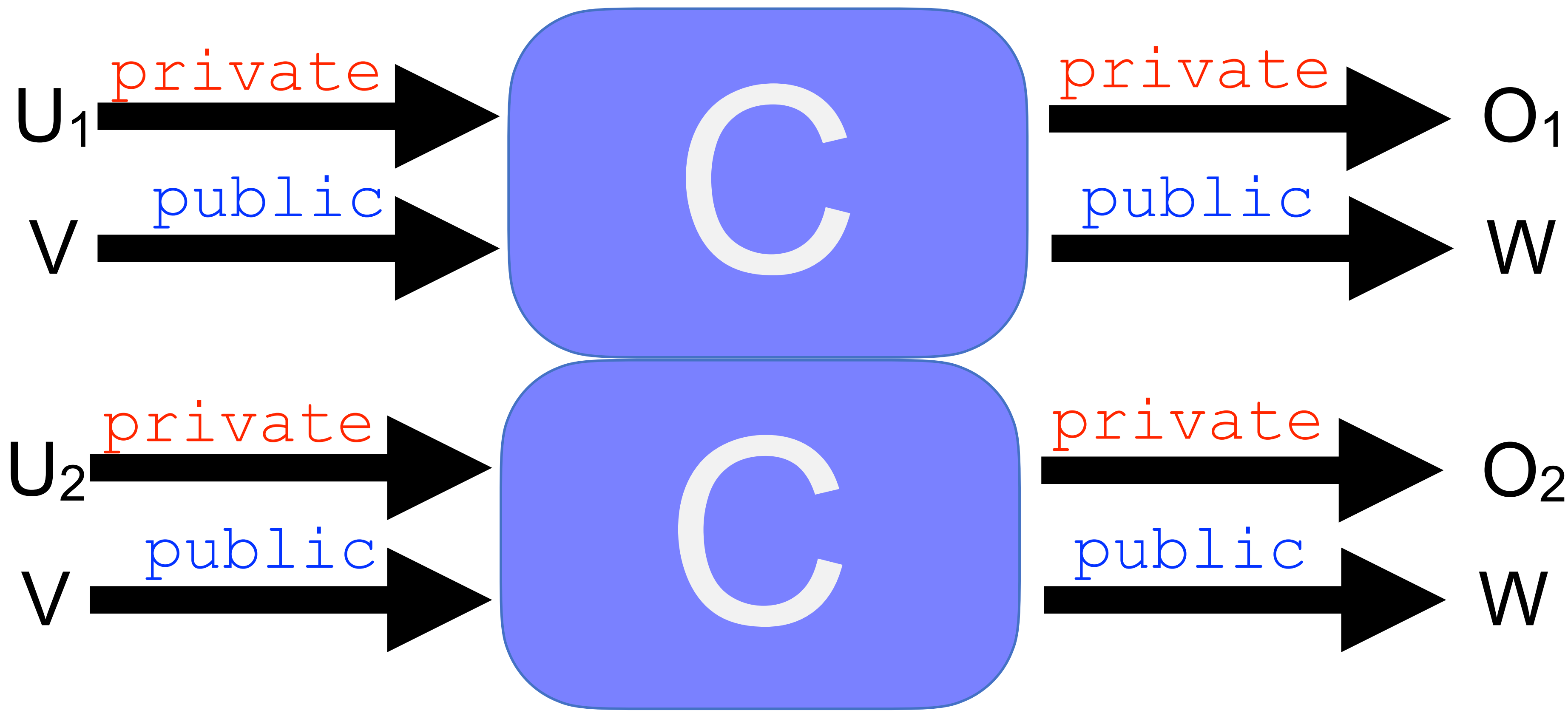
Probabilistic Noninterference

A program `prog` is probabilistically noninterferent if and only if, whenever we run it on two low equivalent memories m_1 and m_2 we have that the probabilistic distributions we get as outputs are the same on public outputs.

Noninterference as a Relational Property

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

$\{c\}_{m1} = \mu_1$ and $\{c\}_{m2} = \mu_2$ implies $\mu_1 \sim_{low} \mu_2$



Revisiting the example

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

Revisiting the example

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

How can we prove that this is noninterferent?