CS 591: Formal Methods in Security and Privacy

Probabilistic computations

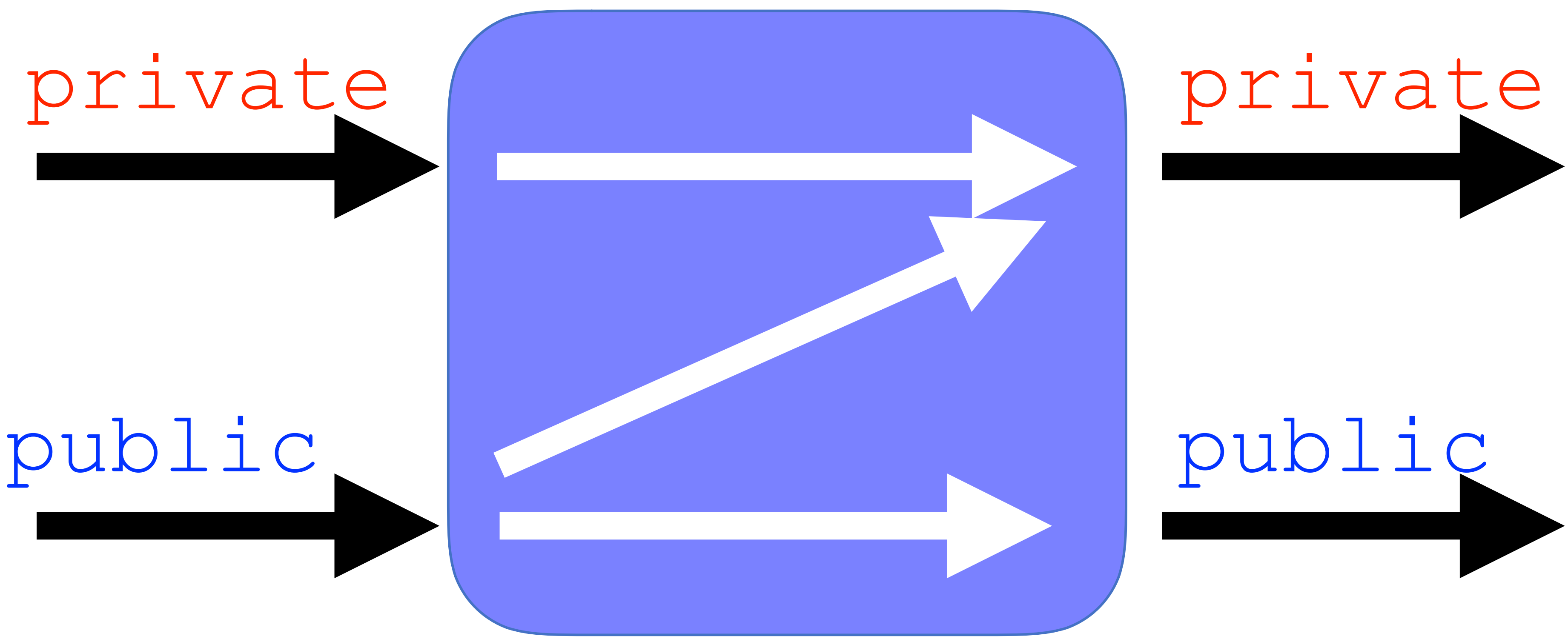
Marco Gaboardi
gaboardi@bu.edu

Alley Stoughton
stough@bu.edu

From the previous classes

Information Flow Control

We want to guarantee that confidential inputs do not flow to nonconfidential outputs.

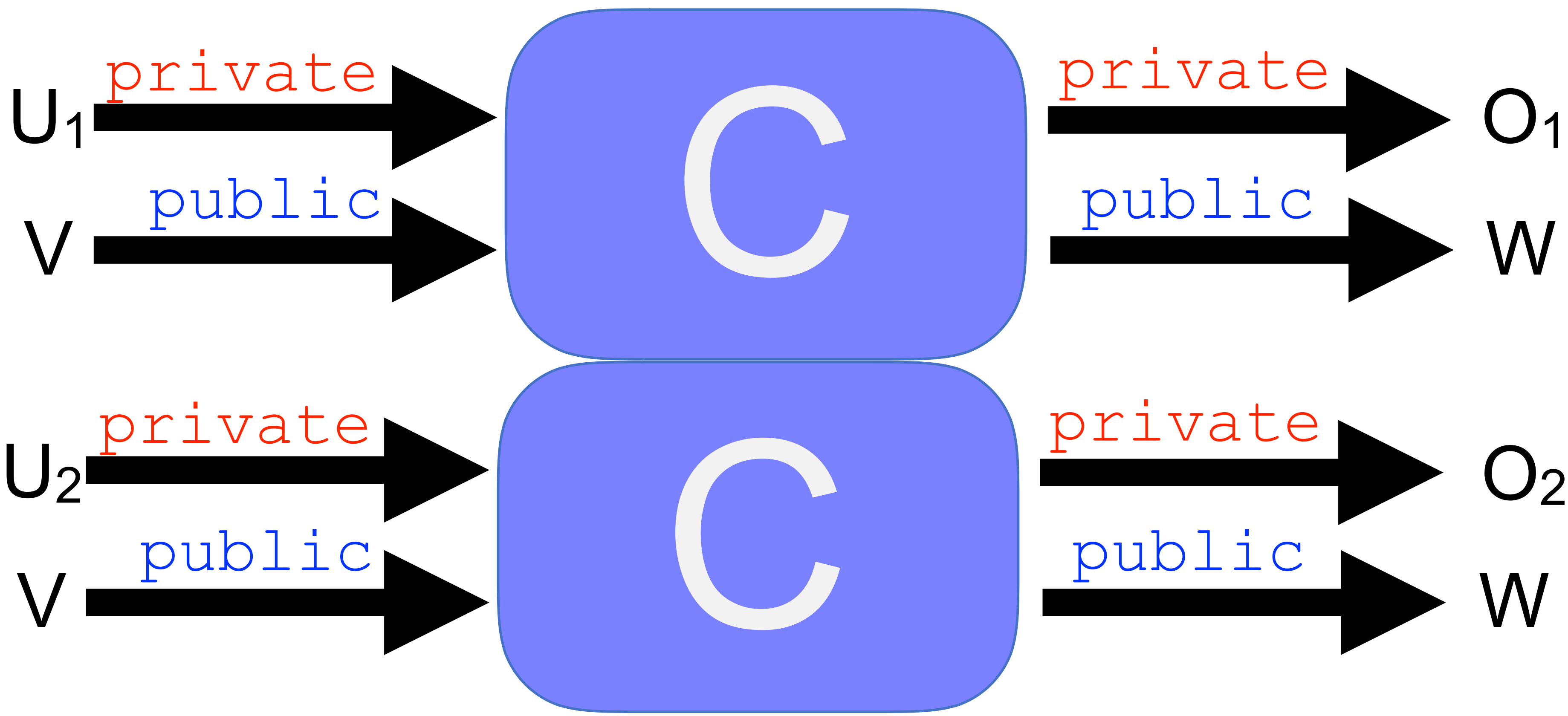


Noninterference as a Relational Property

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

1) $\{c\}_{m1} = \perp$ iff $\{c\}_{m2} = \perp$

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



Relational Hoare Quadruples

Precondition
(a logical formula)

`Precondition`
`Program`$_1$ `~` `Program`$_2$
`Postcondition`

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

Program

Program

Postcondition
(a logical formula)

Soundness

If we can derive $\vdash \texttt{c}_1 \sim \texttt{c}_2 : P \Rightarrow Q$ through the rules of the logic, then the quadruple $\texttt{c}_1 \sim \texttt{c}_2 : P \Rightarrow Q$ is valid.

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

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

iff

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

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.

Today:
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 Subdistributions

A discrete subdistribution over a set A is a function
$\mu : A \rightarrow [0, 1]$
such that the mass of μ,

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

verifies $|\mu| \leq 1$.

The support of a discrete subdistribution μ,
$supp(\mu) = \{a \in A \mid \mu(a) > 0\}$
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 $a \in A$, we will often use the distribution unit(a) defined as:

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

We can think about unit as a function of type unit:$A \rightarrow D(A)$

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.

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

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

Today:
Probabilistic Language

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:

$$\{\texttt{c}\}_{\texttt{m}} = \mu$$

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:

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

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

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

Semantics of Commands

This is defined on the structure of commands:

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

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

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

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

Semantics of Commands

This is defined on the structure of commands:

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

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

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

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

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

Semantics of Commands

This is defined on the structure of commands:

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

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

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

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

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

This is defined on the structure of commands:

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

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

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

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

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

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

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

How did we handle the deterministic case?

Semantics of While

We defined it as

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

Where

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

Semantics of While

We defined it as

$\{\texttt{while e do c}\}_{m} = \texttt{sup}_{n \in Nat}\ \mu_n$

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:

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

$\mu_n =$

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

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:

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

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

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

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

$\mu_n =$

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

Semantics of Commands

This is defined on the structure of commands:

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

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

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

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

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

$\mu_n =$

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

Semantics of Commands

This is defined on the structure of commands:

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

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

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

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

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

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

$\mu_n =$

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

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:

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

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

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

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

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

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

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

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

$\mu_n =$

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

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?

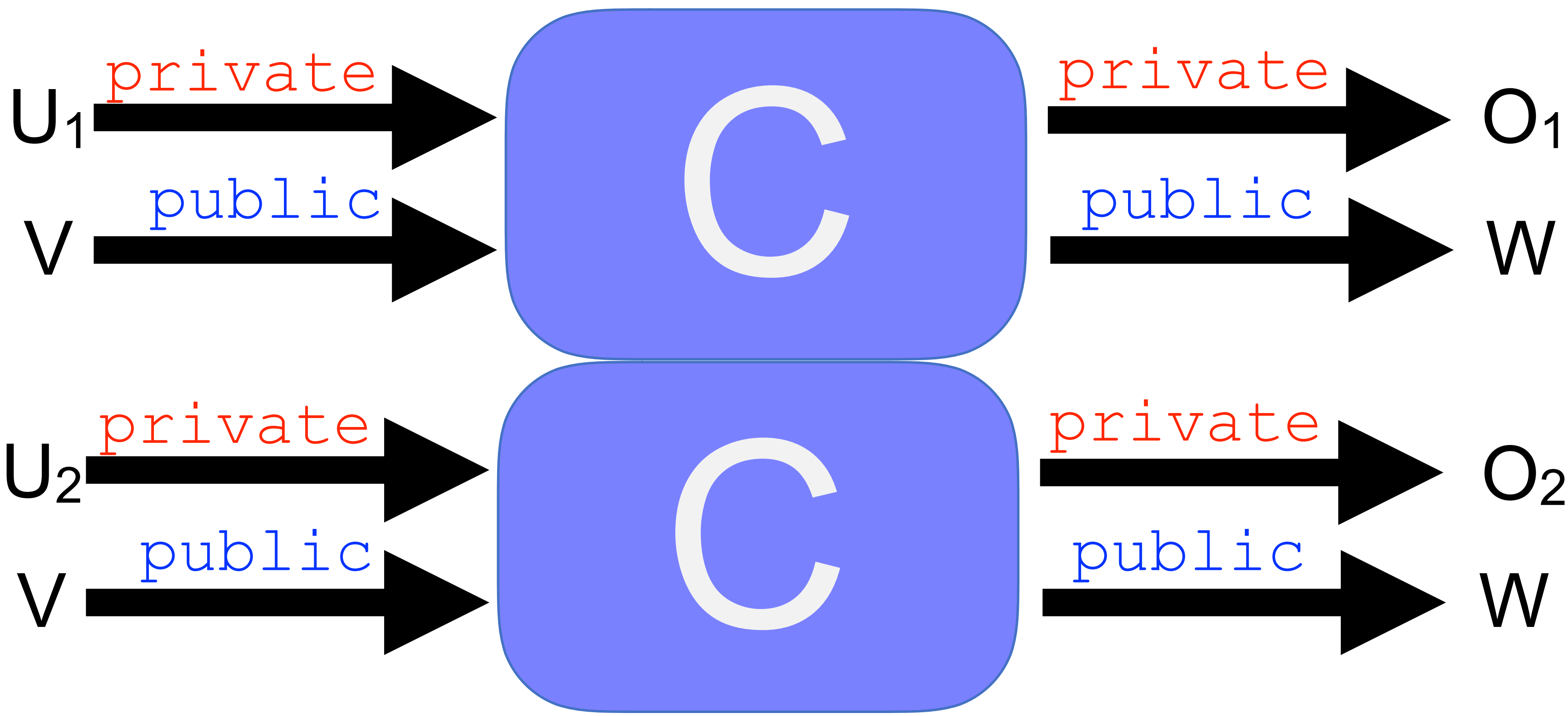
Probabilistic Noninterference

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?

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
```

m_1 m_2

Revisiting the example

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

m_1 $\qquad\qquad$ m_2

$$\downarrow$$

$m_1 \oplus k$

Revisiting the example

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

m_1 $\qquad\qquad$ m_2

$$m_1 \downarrow$$

$$m_1 \oplus k$$

Suppose we can now chose the key for m_2. What could we choose?

Revisiting the example

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

$$m_1 \longrightarrow m_1 \oplus k \qquad\qquad m_2 \longrightarrow m_2 \oplus (m_1 \oplus k \oplus m_2)$$

Suppose we can now chose the key for m_2. What could we choose?

Properties of xor

$$c \oplus (a \oplus c) = a$$

Properties of xor

$$c \oplus (a \oplus c) = a$$

Example:

$$100 \oplus (101 \oplus 100) =$$

$$100 \oplus 001 = 101$$

Revisiting the example

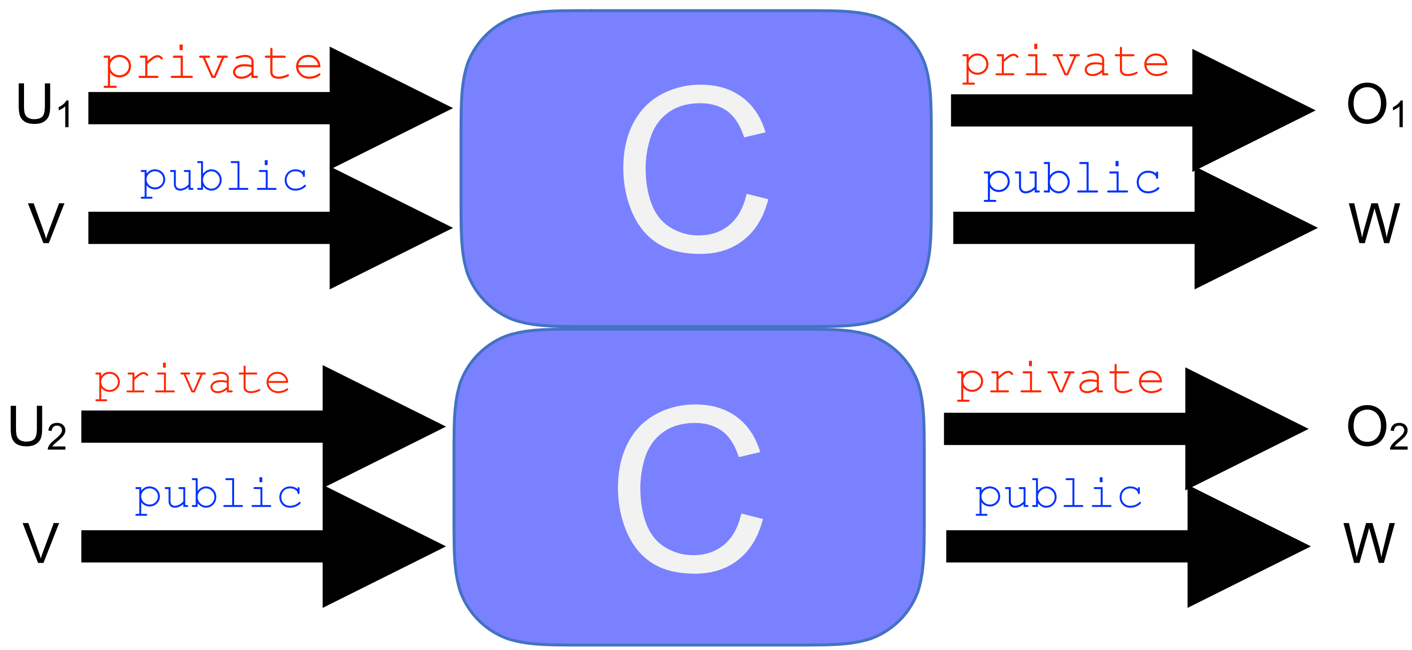
```
OneTimePad(m : private msg) : public msg
   key :=$ Uniform({0,1}^n);
   cipher := msg xor key;
   return cipher
```

$$m_1 \downarrow m_1 \oplus k \qquad\qquad m_2 \downarrow m_1 \oplus k$$

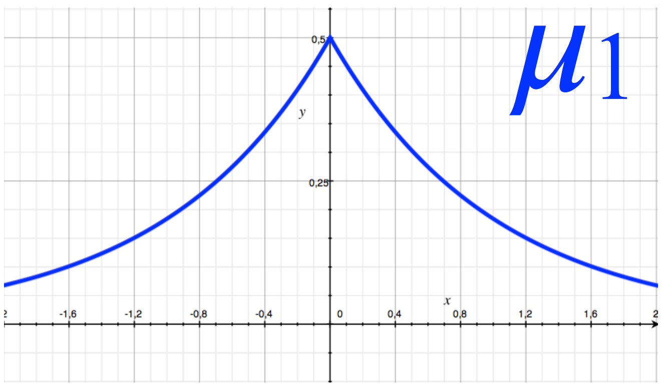
Applying the property above

Revisiting the example

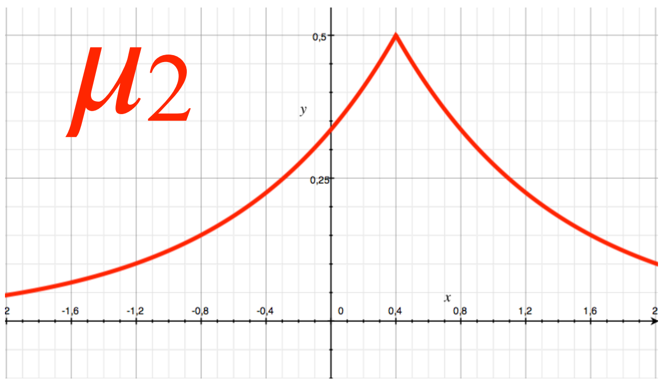
```
OneTimePad(m : private msg) : public msg
   key :=$ Uniform({0,1}^n);
   cipher := msg xor key;
   return cipher
```



Coupling

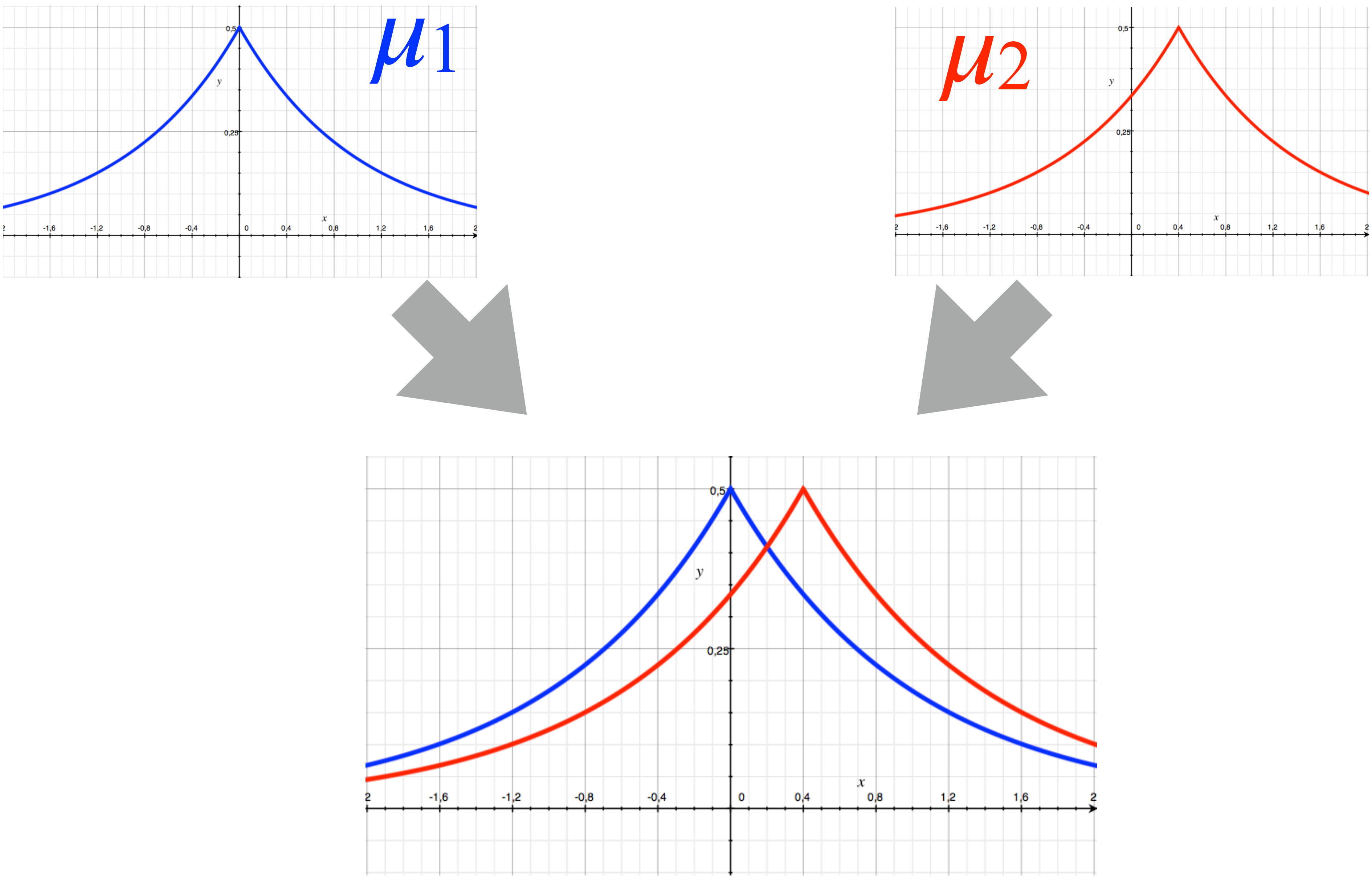


μ_1
0,5
0,25
y
x
-1,6
-1,2
-0,8
-0,4
0
0,4
0,8
1,2
1,6
2



μ_2
0,5
0,25
y
x
-1,6
-1,2
-0,8
-0,4
0
0,4
0,8
1,2
1,6
2

Coupling



Example of Our Coupling

00	0.25
01	0.25
10	0.25
11	0.25

$$\texttt{k} = \texttt{10} \oplus \texttt{k} \oplus \texttt{00}$$

00	0.25
01	0.25
10	0.25
11	0.25

Example of Our Coupling

00	0.25
01	0.25
10	0.25
11	0.25

$$\mathtt{k} = \mathtt{10} \oplus \mathtt{k} \oplus \mathtt{00}$$

00	0.25
01	0.25
10	0.25
11	0.25

	00	01	10	11
00			0.25	
01				0.25
10	0.25			
11		0.25		

Coupling formally

Given two distributions $\mu_1 \in D(A)$, and $\mu_2 \in D(B)$, a coupling between them is a joint distribution $\mu \in D(A x B)$ whose marginal distributions are μ_1 and μ_2, respectively.

$$\pi_1(\mu)(a) = \sum_b \mu(a, b) \qquad \pi_2(\mu)(b) = \sum_a \mu(a, b)$$