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 \( m_1 \sim_{\text{low}} m_2 \):

1) \( \{c\}_{m_1} = \bot \) iff \( \{c\}_{m_2} = \bot \)

2) \( \{c\}_{m_1} = m_1' \) and \( \{c\}_{m_2} = m_2' \) implies \( m_1' \sim_{\text{low}} m_2' \)
Relational Hoare Quadruples

Precondition

Program_1 \sim \text{Program}_2

Postcondition

\begin{align*}
\frac{c_1 \sim c_2 : P \Rightarrow Q}{\text{Program}}\end{align*}
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.
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 \( c_1 \sim c_2 : P \Rightarrow Q \) through the rules of the logic.
Soundness and completeness with respect to Hoare Logic

\( \vdash_{\text{RHL}} c_1 \sim c_2 : P \Rightarrow Q \)

iff

\( \vdash_{\text{HL}} c_1 ; c_2 : P \Rightarrow Q \)
Soundness and completeness with respect to Hoare Logic

\[ \vdash_{\text{RHL}} c_1 \sim c_2 : P \Rightarrow Q \]

iff

\[ \vdash_{\text{HL}} c_1; c_2 : P \Rightarrow Q \]

Under the assumption that we can partition the memory adequately, and that we have termination.
Today: Probabilistic Language
An example

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

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, ... probabilistic expressions
A discrete subdistribution over a set $A$ is a function $\mu : A \to [0, 1]$ such that the mass of $\mu$, 
$$|\mu| = \sum_{a \in A} \mu(a)$$
verifies $|\mu| \leq 1$.

The support of a discrete subdistribution $\mu$, $\text{supp}(\mu) = \{a \in A \mid \mu(a) > 0\}$ 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 $\mu$ is of type $D(A)$ denoted $\mu : D(A)$ 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 $\mu \in \mathcal{D}(A)$, and $E \subseteq A$, we have the following properties:

- $\mathbb{P}_\mu[\emptyset] = 0$
- $\mathbb{P}_\mu[A] \leq 1$
- $0 \leq \mathbb{P}_\mu[E] \leq 1$

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

$E \subseteq A$ and $F \subseteq A$ 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 $\mathcal{O}$ the subdistribution $\mu$ defined as constant 0.
Operations over Probabilistic Subdistributions

Let’s consider an arbitrary \( a \in A \), we will often use the distribution \( \text{unit}(a) \) defined as:

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

We can think about \( \text{unit} \) as a function of type \( \text{unit}:A \rightarrow D(A) \).
Let’s consider a distribution $\mu \in \mathcal{D}(A)$, and a function $M: A \rightarrow \mathcal{D}(B)$ then we can define their composition by means of an expression $\text{let } a = \mu \text{ in } M\ a \text{ defined as:}$

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

\[ \{ \text{uniform}([0,1]^n) \}_m \in D([0,1]^n) \]

\[ \{ \text{gaussian}(k, \sigma) \}_m \in D(\text{Real}) \]
Today:
Probabilistic Language
Semantics of PWhile Commands

What is the meaning of the following command?

\[ k := \$ \text{ uniform}(\{0,1\}^n); \quad z := x \mod k; \]
Semantics of PWhile Commands

What is the meaning of the following command?

\[ k := \$ \text{uniform}(\{0,1\}^n); \ x := x \mod k; \]

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

\[ \text{Cmd} \times \text{Mem} \rightarrow D(\text{Mem}) \]

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:

\[ \{ \text{abort} \}_m = 0 \]
Semantics of Commands

This is defined on the structure of commands:

\[
\{\text{abort}\}_m = \emptyset
\]

\[
\{\text{skip}\}_m = \text{unit}(m)
\]
Semantics of Commands

This is defined on the structure of commands:

\[
\{\text{abort}\}_m = O
\]

\[
\{\text{skip}\}_m = \text{unit}(m)
\]

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

This is defined on the structure of commands:

\[
\begin{align*}
\{\text{abort}\}_m &= \emptyset \\
\{\text{skip}\}_m &= \text{unit}(m) \\
\{x:=e\}_m &= \text{unit}(m[x\leftarrow\{e\}_m])
\end{align*}
\]

\[
\{c;c'\}_m = \text{let } m' = \{c\}_m \text{ in } \{c'\}_{m'}
\]
Semantics of Commands

This is defined on the structure of commands:

\[
\begin{align*}
\{\text{abort}\}_m &= 0 \\
\{\text{skip}\}_m &= \text{unit}(m) \\
\{x:=e\}_m &= \text{unit}(m[x \leftarrow \{e\}_m]) \\
\{c; c'\}_m &= \text{let } m' = \{c\}_m \text{ in } \{c'\}_m' \\
\{\text{if } e \text{ then } c_t \text{ else } c_f\}_m &= \{c_t\}_m \text{ if } \{e\}_m = \text{true}
\end{align*}
\]
Semantics of Commands

This is defined on the structure of commands:

\{\text{abort}\}_m = \text{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{if } e \text{ then } c_t \text{ else } c_f\}_m = \{c_t\}_m \text{ if } \{e\}_m = \text{true}

\{\text{if } e \text{ then } c_t \text{ else } c_f\}_m = \{c_f\}_m \text{ if } \{e\}_m = \text{false}
Semantics of Commands

This is defined on the structure of commands:

\[ \{ \text{abort} \}_m = O \]

\[ \{ \text{skip} \}_m = \text{unit}(m) \]

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

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

\[ \{ c; c' \}_m = \text{let } m'=\{ c \}_m \text{ in } \{ c' \}_m' \]

\[ \{ \text{if } e \text{ then } c_t \text{ else } c_f \}_m = \{ c_t \}_m \text{ if } \{ e \}_m = \text{true} \]

\[ \{ \text{if } e \text{ then } c_t \text{ else } c_f \}_m = \{ c_f \}_m \text{ if } \{ e \}_m = \text{false} \]
Semantics of While

What about while

How did we handle the deterministic case?
Semantics of While

What about while

$$\{\text{while } e \text{ do } c\}_m = ???$$

How did we handle the deterministic case?
Semantics of While

We defined it as

\[ \{\text{while } e \text{ do } c\}_m = \sup_{n \in \mathbb{N}} \mu_n \]

Where

\[ \mu_n = \]

let \( m' = \{\text{while}^n e \text{ do } c\}_m \) in \{if e then abort\}_m'
Semantics of While

We defined it as

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

Where

$$\mu_n =$$

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

Is this well defined?
Semantics of Commands

This is defined on the structure of commands:

\[
\{\text{while } e \text{ do } c\}_m = \sup_{n \in \text{Nat}} \mu_n \\
\mu_n = \\
\text{let } m' = \{(\text{while}^n e \text{ do } c)\}_m \text{ in } \{\text{if } e \text{ then abort}\}_m'
\]
Semantics of Commands

This is defined on the structure of commands:

\[
\{\text{abort}\}_m = 0
\]

\[
\{\text{while } e \text{ do } c\}_m = \sup_{n \in \mathbb{Nat}} \mu_n
\]

\[
\mu_n = \text{let } m' = \{(\text{while}_n e \text{ do } c)\}_m \text{ in } \{\text{if } e \text{ then abort}\}_m'
\]
Semantics of Commands

This is defined on the structure of commands:

\[
\{\text{abort}\}_m = 0
\]

\[
\{\text{skip}\}_m = \text{unit}(m)
\]

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

\[
\mu_n = \begin{cases} 
\text{let } m' = \{\text{while }^n e \text{ do } c\}_m \text{ in } \{\text{if } e \text{ then abort}\}_m', & n > 0 \\
0, & n = 0 
\end{cases}
\]
Semantics of Commands

This is defined on the structure of commands:

\[
\begin{align*}
\{\text{abort}\}_m &= 0 \\
\{\text{skip}\}_m &= \text{unit}(m) \\
\{x := e\}_m &= \text{unit}(m[x \leftarrow \{e\}_m])
\end{align*}
\]

\[
\{\text{while } e \text{ do } c\}_m = \sup_{n \in \text{Nat}} \mu_n \\
\mu_n = \\
\text{let } m' = \{(\text{while}_n e \text{ do } c)\}_m \text{ in } \{\text{if } e \text{ then abort}\}_m'
\]
Semantics of Commands

This is defined on the structure of commands:

\[
\begin{align*}
\{\text{abort}\}_m &= \emptyset \\
\{\text{skip}\}_m &= \text{unit}(m) \\
\{x:=e\}_m &= \text{unit}(m[x\leftarrow\{e\}_m]) \\
\{c;c'\}_m &= \text{let } m' = \{c\}_m \text{ in } \{c'\}_{m'} \\
\{\text{while } e \text{ do } c\}_m &= \sup_{n \in \text{Nat}} \mu_n \\
\mu_n &= \text{let } m' = \{(\text{while}_n e \text{ do } c)\}_m \text{ in } \{\text{if } e \text{ then abort}\}_{m'}
\end{align*}
\]
Semantics of Commands

This is defined on the structure of commands:

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

\[ \{c;c'\}_m = \text{let } m' = \{c\}_m \text{ in } \{c'\}_m' \]
\[ \{\text{if } e \text{ then } c_t \text{ else } c_f\}_m = \{c_t\}_m \text{ if } \{e\}_m = \text{true} \]

\[ \{\text{while } e \text{ do } c\}_m = \sup_{n \in \text{Nat}} \mu_n \]
\[ \mu_n = \text{let } m' = \{(\text{while }^n e \text{ do } c)\}_m \text{ in } \{\text{if } e \text{ then } \text{abort}\}_m' \]
Semantics of Commands

This is defined on the structure of commands:

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

\[
\{c;c'\}_m = \text{let } m' = \{c\}_m \text{ in } \{c'\}_m' \\
\{\text{if } e \text{ then } c_t \text{ else } c_f\}_m = \{c_t\}_m \text{ if } \{e\}_m = \text{true} \\
\{\text{if } e \text{ then } c_t \text{ else } c_f\}_m = \{c_f\}_m \text{ if } \{e\}_m = \text{false} \\
\{\text{while } e \text{ do } c\}_m = \sup_{n \in \mathbb{Nat}} \mu_n \\
\mu_n = \text{let } m' = \{(\text{while}_n e \text{ do } c)\}_m \text{ in } \{\text{if } e \text{ then abort}\}_m'
\]
Semantics of Commands

This is defined on the structure of commands:

\[
\{\text{abort}\}_m = 0
\]
\[
\{\text{skip}\}_m = \text{unit}(m)
\]
\[
\{x:=e\}_m = \text{unit}(m[x\leftarrow\{e\}_m])
\]
\[
\{x:=$d\}_m = \text{let } a=\{d\}_m \text{ in } \text{unit}(m[x\leftarrow a])
\]
\[
\{c;\ c'\}_m = \text{let } m'=\{c\}_m \text{ in } \{c'\}_m'
\]
\[
\{\text{if } e \text{ then } c_t \text{ else } c_f\}_m = \{c_t\}_m \text{ if } \{e\}_m=\text{true}
\]
\[
\{\text{if } e \text{ then } c_t \text{ else } c_f\}_m = \{c_f\}_m \text{ if } \{e\}_m=\text{false}
\]
\[
\{\text{while } e \text{ do } c\}_m = \sup_{n\in\text{Nat}} \mu_n
\]
\[
\mu_n = \text{let } m'=(\text{while}_n e \text{ do } c)_m \text{ in } \{\text{if } e \text{ then } \text{abort}\}_m'
\]
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

\texttt{OneTimePad}(m : private msg) : public msg

key := $ \text{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 $\text{prog}$ 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 \( m_1 \sim_{\text{low}} m_2 \):

\[
\{c\}_{m_1} = \mu_1 \quad \text{and} \quad \{c\}_{m_2} = \mu_2 \quad \text{implies} \quad \mu_1 \sim_{\text{low}} \mu_2
\]
Revisiting the example

OneTimePad(m : private msg) : public msg
key := Unifrom({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

\textbf{OneTimePad}(m : private msg) : public msg

\hspace{1em} key := Uniform(\{0,1\}^n);

\hspace{1em} cipher := msg \text{ xor} key;

\hspace{1em} 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 := $\text{Uniform}\left(\{0,1\}^n\right)$;
cipher := msg xor key;
return cipher

\[
\begin{align*}
m_1 & \quad \oplus \quad m_2 \\
m_1 \oplus k & \\
\end{align*}
\]
Revisiting the example

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

Suppose we can now chose the key for \(m_2\). What could we we choose?
Revisiting the example

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

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

\[ \text{OneTimePad}(m : \text{private msg}) : \text{public msg} \]

\[
\begin{align*}
\text{key} & : = \$ \text{Uniform}\{0,1\}^n; \\
\text{cipher} & : = m \text{ xor} \text{ key}; \\
\text{return} & \\text{cipher}
\end{align*}
\]

Applying the property above

\[
\begin{align*}
m_1 \oplus k & \quad m_2 \\
\text{m}_1 \oplus k & \quad \text{m}_1 \oplus k
\end{align*}
\]

Applying the property above
Revisiting the example

\textbf{OneTimePad}(m : private msg) : public msg
key ::= Uniform(\{0,1\}^n);
cipher ::= msg \text{ xor} key;
return cipher
Coupling

$\mu_1$

$\mu_2$
Coupling

$\mu_1$

$\mu_2$
Example of Our Coupling

\[ k = 10 \oplus k \oplus 00 \]
Example of Our Coupling

\[ k = 10 \oplus k \oplus 00 \]
Coupling formally

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

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