CS 599: Formal Methods in Security and Privacy
RHL and 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 information do not flow in what is considered nonconfidential.
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 Logic - RHL

\[ c_1 \sim c_2 : P \Rightarrow Q \]
More Relational Hoare Logic
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
What do we do if our two programs have different forms? There are three pairs of one-sided rules.

One pair for if.
Assignment — left

\[ \vdash x := e \sim \text{skip:} \]
\[ P[e<1>/x<1>] \Rightarrow P \]
Assignment — right

⊢ skip ~ x := e:

\[ P[e<2>/x<2>] \Rightarrow P \]

Also pair of one-sided rules for while — we’ll ignore for now
Program Equivalence Rule

\[ \vdash P : c_1 \equiv c_1 \]

\[ \vdash P : c_2 \equiv c_2 \quad \vdash c_1' \sim c_2' : P \Rightarrow Q \]

\[ \vdash c_1 \sim c_2 : P \Rightarrow Q \]

\[ \vdash P : c_1 \equiv c_2 \text{ means } \{ c_1 \}_m = \{ c_2 \}_m \text{ for all } m \text{ such that } P(m) \]
Program Equivalences

\[ \models P : \text{skip};c \equiv c \]

\[ \models P : c;\text{skip} \equiv c \]

\[ \models P : (c_1;c_2);c_3 \equiv c_1;(c_2;c_3) \]

...
Rules of Relational Hoare Logic
Combining Composition and Equivalence

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

\[ \frac{\vdash c_1; c_2 \sim c_1' : P \implies R}{\vdash c_3 \sim c_2' ; c_3' : R \implies Q} \]
\[ \vdash c_1; c_2; c_3 \sim c_1' ; c_2' ; c_3' : P \implies Q \]
Rules of Relational Hoare Logic
Combining Composition and Equivalence

\[ \vdash c_1 \sim \text{skip} \colon P \Rightarrow R \]
\[ \vdash c_2 \sim c_1' \colon R \Rightarrow Q \]
\[ \vdash c_1; c_2 \sim \text{skip}; c_1' \colon P \Rightarrow Q \]
\[ \vdash c_1; c_2 \sim c_1' \colon P \Rightarrow Q \]
Rules of Relational Hoare Logic
Combining Composition and Equivalence

\[ \vdash c_1 \sim c_1' : P \Rightarrow R \]
\[ \vdash c_2 \sim \text{skip} : R \Rightarrow Q \]

\[ \vdash c_1 ; c_2 \sim c_1' ; \text{skip} : P \Rightarrow Q \]

\[ \vdash c_1 ; c_2 \sim c_1' : P \Rightarrow Q \]
Relational Hoare Logic in EasyCrypt

- EasyCrypt’s implementation of Relational Hoare Logic has much in common with its implementation of Hoare Logic.

- Look for the pRHL tactics in Section 3.4 of the EasyCrypt Reference Manual (the “p” stands for “probabilistic”, but ignore that for now).
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) $\{c_1\}_{m_1} = \bot$ iff $\{c_2\}_{m_2} = \bot$

2) $\{c_1\}_{m_1} = m_1 \, \prime$ and $\{c_2\}_{m_2} = m_2 \, \prime$ implies $Q(m_1 \, \prime, m_2 \, \prime)$.

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

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

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

d₁, d₂, ... 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

- \( \text{uniform}({0,1}^n) \)
- \( \text{gaussian}(k, \sigma) \)
- \( \text{laplace}(k, b) \)
Semantics of Probabilistic Expressions

We would like to define it on the structure:

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

but is the result just a value?
A discrete subdistribution over a set $A$ is a function $\mu : A \rightarrow [0, 1]$ 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

$$
P_\mu [E] = \sum_{a \in E} \mu (a)$$
Probabilistic Subdistributions

Let’s consider $\mu \in D(A)$, and $E \subseteq A$, we have the following properties

\[ P_\mu[\emptyset] = 0 \]
\[ P_\mu[A] \leq 1 \]
\[ 0 \leq P_\mu[E] \leq 1 \]

$E \subseteq F \subseteq A$ implies $P_\mu[E] \leq P_\mu[F]$

$E \subseteq A$ and $F \subseteq A$ implies $P_\mu[E \cup F] \leq P_\mu[E] + P_\mu[F] - P_\mu[E \cap F]$

We will denote by $O$ 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 $\text{unit}: A \to D(A)$
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 \( \text{let } a = \mu \text{ in } M\ a \) defined as:

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

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

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

What is the meaning of the following command?

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

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

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

We will denote this relation as:

\[ \{ c \}_{\text{m}=\mu} \]
Semantics of Commands

This is defined on the structure of commands:

\[
\begin{align*}
\{\text{abort}\}_m &= \emptyset \\
\{\text{skip}\}_m &= \text{unit}(m) \\
\{x:=e\}_m &= \text{unit}(m[x\leftarrow\{e\}_m]) \\
\{x:=d\}_m &= \text{let } a=\{d\}_m \text{ in } \text{unit}(m[x\leftarrow a]) \\
\{c;c'\}_m &= \text{let } m'=\{c\}_m \text{ in } \{c'\}_m' \\
\{\text{if } e \text{ then } c_t \text{ else } c_f\}_m &= \{c_t\}_m \text{ if } \{e\}_m = \text{true} \\
\{\text{if } e \text{ then } c_t \text{ else } c_f\}_m &= \{c_f\}_m \text{ if } \{e\}_m = \text{false}
\end{align*}
\]
Revisiting the example

OneTimePad\((m : \text{private msg}) : \text{public msg}\)
key := Uniform\(\{0,1\}^n\);
cipher := msg \text{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 $m_1 \sim_{\text{low}} m_2 : \{c\}_{m_1}=\mu_1$ and $\{c\}_{m_2}=\mu_2$ implies $\mu_1 \sim_{\text{low}} \mu_2$
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?