Language-based Cryptographic Proofs in Coq

or

Coq for Probabilistic Programs

FEDERICO OLMEDO
UNIVERSITY OF CHILE

ICSEC KICK-OFF WORKSHOP
SANTIAGO, CHILE — MARCH 2018
Motivation
Rigor crisis in the cryptographic community

In our opinion, many proofs in cryptography have become essentially unverifiable. Our field may be approaching a crisis of rigor.

Bellare & Rogaway (2006)

Do we have a problem with cryptographic proofs?
Yes, we do. The problem is that as a community, we generate more proofs than we carefully verify (and as a consequence some of our published proofs are incorrect).

Halevi (2005)
The case of OAEP encryption scheme

Introduction and security proof

Since 1994

Worldwide industrial standard

- PKCS#1 v2
- IEEE P1363
- ISO 18033-2
The rigor crisis of the cryptographic community

The case of OAEP encryption scheme

1994

Introduction and security proof

Since 1994

Worldwide industrial standard

Standard

PKCS#1 v2
IEEE P1363
ISO 18033-2

2001

Security proof is flawed

And 7 years later...

There appears to be a non-trivial gap in the OAEP security proof [and] this gap cannot be filled.

Shoup (2001)
The rigorous crisis of the cryptographic community

The case of BONEH-FRANKLIN encryption scheme

- Introduction and security proof
- Used as subcomponent of several cryptographic protocols
The rigor crisis of the cryptographic community

The case of BONEH-FRANKLIN encryption scheme

- **2001**: Introduction and security proof
- **Since 2001**: Used as subcomponent of several cryptographic protocols
- **2005**: Security proof is flawed

*This is just another example in which a well-known and widely used construction turns out to have an unnoticed flawed security reduction.*

Galindo (2005)
CertiCrypt:
Framework for constructing certified cryptographic proofs in Coq

http://certicrypt.gforge.inria.fr/
CertiCrypt:
Framework for constructing certified cryptographic proofs in Coq

http://certicrypt.gforge.inria.fr/

Substantial effort
- 30,000 lines
- 4-6 years
- 6 people
CertiCrypt: Framework for constructing certified cryptographic proofs in Coq

http://certicrypt.gforge.inria.fr/

Substantial effort
- 30,000 lines
- 4-6 years
- 6 people

High impact
- Formalization of several encryption schemes, digital signatures, hash functions, zero-knowledge protocols, etc
- 12 publications
Basics about CertiCrypt
What is a secure cryptographic scheme?
What is a secure cryptographic scheme?

A cryptographic scheme is **secure** if an efficient adversary can break it only with negligible probability.
What is a secure cryptographic scheme?

A cryptographic scheme is secure if an efficient adversary can break it only with negligible probability.
What is a secure cryptographic scheme?

A cryptographic scheme is **secure** if an efficient adversary can break it only with negligible *probability*

Cryptographic schemes *must* be probabilistic (Goldwasser & Micali, ’82)
What is a secure cryptographic scheme?

A cryptographic scheme is secure if an efficient adversary can break it only with negligible probability.

Cryptographic schemes must be probabilistic (Goldwasser & Micali, ’82).
What is a secure cryptographic scheme?

A cryptographic scheme is secure if an efficient adversary can break it only with negligible probability.

- Cryptographic schemes must be probabilistic (Goldwasser & Micali, ’82)
- Adversaries should run in probabilistic polynomial time (PPT)
What is a secure cryptographic scheme?

A cryptographic scheme is secure if an efficient adversary can break it only with negligible probability.

- Cryptographic schemes must be probabilistic (Goldwasser & Micali, ’82)
- Adversaries should run in probabilistic polynomial time (PPT)
What is a secure cryptographic scheme?

A cryptographic scheme is secure if an efficient adversary can break it only with negligible probability.

- Cryptographic schemes must be probabilistic (Goldwasser & Micali, ’82)

- Adversaries should run in probabilistic polynomial time (PPT)

- There exists a standard security notion for each kind of cryptographic scheme
What is a secure cryptographic scheme?

A cryptographic scheme is **secure** if an **efficient** adversary can **break** it only with negligible **probability**

- Cryptographic schemes **must** be probabilistic (Goldwasser & Micali, ’82)
- Adversaries should run in **probabilistic polynomial time** (PPT)
- There exists a **standard security notion** for each kind of cryptographic scheme

Attack game
What is a secure cryptographic scheme?

A cryptographic scheme is secure if an efficient adversary can break it only with negligible probability.

- Cryptographic schemes must be probabilistic (Goldwasser & Micali, ’82)
- Adversaries should run in probabilistic polynomial time (PPT)
- There exists a standard security notion for each kind of cryptographic scheme

\[ \Pr[\text{\textit{A} breaks the scheme}] \leq \epsilon \]
How do security proof proceed?

By stepwise transformation of the attack game, towards a “simpler” game.
How do security proof proceed?

By stepwise transformation of the attack game, towards a “simpler” game

\[
\Pr_{G_0}[E_0] \leq f_1 \left( \Pr_{G_1}[E_1] \right)
\]

Probability of breaking the scheme
How do security proof proceed?

By stepwise transformation of the attack game, towards a “simpler” game

Probability of breaking the scheme
How do security proof proceed?

By stepwise transformation of the attack game, towards a “simpler” game

Probability of breaking the scheme

\[ \Pr_{G_0}[E_0] \leq f_1(\Pr_{G_1}[E_1]) \leq \cdots \leq f_n(\Pr_{G_n}[E_n]) \]
How do security proof proceed?

By stepwise transformation of the attack game, towards a “simpler” game

Attack game

\[ G_0 \rightarrow G_1 \rightarrow \ldots \rightarrow G_n \]

Final game

\[ \Pr_{G_0}[E_0] \leq f_1(\Pr_{G_1}[E_1]) \leq \ldots \leq f_n(\Pr_{G_n}[E_n]) \]

How do we represent games?

Probability of breaking the scheme

\[ \Pr_{G_0}[E_0] \leq f(\Pr_{G_n}[E_n]) \leq \epsilon \]
How do security proof proceed?

By stepwise transformation of the attack game, towards a “simpler” game

\[
\Pr_{G_0}[E_0] \leq f_1(\Pr_{G_1}[E_1]) \leq \cdots \leq f_n(\Pr_{G_n}[E_n])
\]

How do we relate the probabilities of events between consecutive games?
Language-based cryptographic proofs
Language-based cryptographic proofs

Games $\implies$ (probabilistic) programs
Language-based cryptographic proofs

- Games $\implies$ (probabilistic) programs
- Probability space $\implies$
- Probability of event $\implies$
- Game transformations $\implies$
- Generic adversary $\implies$
Language-based cryptographic proofs

Games $\implies$ (probabilistic) programs

Probability space $\implies$ program denotation

Probability of event $\implies$

Game transformations $\implies$

Generic adversary $\implies$
## Language-based cryptographic proofs

<table>
<thead>
<tr>
<th>Concept</th>
<th>Implication</th>
</tr>
</thead>
<tbody>
<tr>
<td>Games</td>
<td>$\implies$  (probabilistic) programs</td>
</tr>
<tr>
<td>Probability space</td>
<td>$\implies$  program denotation</td>
</tr>
<tr>
<td>Probability of event</td>
<td>$\implies$  probability of postcondition</td>
</tr>
<tr>
<td>Game transformations</td>
<td>$\implies$</td>
</tr>
<tr>
<td>Generic adversary</td>
<td>$\implies$</td>
</tr>
</tbody>
</table>
Language-based cryptographic proofs

Games $\implies$ (probabilistic) programs

Probability space $\implies$ program denotation

Probability of event $\implies$ probability of postcondition

Game transformations $\implies$ program transformations

Generic adversary $\implies$
Language-based cryptographic proofs

<table>
<thead>
<tr>
<th>Concept</th>
<th>Transformation</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Games</td>
<td>⟹</td>
<td>(probabilistic) programs</td>
</tr>
<tr>
<td>Probability space</td>
<td>⟹</td>
<td>program denotation</td>
</tr>
<tr>
<td>Probability of event</td>
<td>⟹</td>
<td>probability of postcondition</td>
</tr>
<tr>
<td>Game transformations</td>
<td>⟹</td>
<td>program transformations</td>
</tr>
<tr>
<td>Generic adversary</td>
<td>⟹</td>
<td>unspecified procedure</td>
</tr>
</tbody>
</table>
The probabilistic language

\[ C ::= \begin{array}{l}
\text{skip} & \text{nop} \\
C; C & \text{sequence} \\
V \leftarrow E & \text{assignment} \\
V \leftarrow \mathcal{D}E & \text{random sampling} \\
\text{if } E \text{ then } C \text{ else } C & \text{conditional} \\
\text{while } E \text{ do } C & \text{while loop} \\
V \leftarrow \mathcal{P}(E, \ldots, E) & \text{procedure call}
\end{array} \]
The probabilistic language

\[ \mathcal{C} ::= \begin{array}{ll}
\text{skip} & \text{nop} \\
\mathcal{C}; \mathcal{C} & \text{sequence} \\
V \leftarrow \mathcal{E} & \text{assignment} \\
V \leftarrow \mathcal{D}\mathcal{E} & \text{random sampling} \\
\text{if } \mathcal{E} \text{ then } \mathcal{C} \text{ else } \mathcal{C} & \text{conditional} \\
\text{while } \mathcal{E} \text{ do } \mathcal{C} & \text{while loop} \\
V \leftarrow \mathcal{P} (\mathcal{E}, \ldots, \mathcal{E}) & \text{procedure call}
\end{array} \]

\[ [c] : \mathcal{S} \rightarrow \mathcal{D}(\mathcal{S}) \]
The probabilistic language

\[ C ::= \begin{align*}
  & \text{skip} \quad & \text{nop} \\
| & C ; C \quad & \text{sequence} \\
| & V \leftarrow \mathcal{E} \quad & \text{assignment} \\
| & V \leftarrow \mathcal{D}\mathcal{E} \quad & \text{random sampling} \\
| & \text{if } \mathcal{E} \text{ then } C \text{ else } C \quad & \text{conditional} \\
| & \text{while } \mathcal{E} \text{ do } C \quad & \text{while loop} \\
| & V \leftarrow \mathcal{P}(\mathcal{E}, \ldots, \mathcal{E}) \quad & \text{procedure call}
\end{align*} \]

\[ [c] : \forall (k : \mathbb{N}). \mathbb{S}_k \rightarrow \mathcal{D}(\mathbb{S}_k) \]
How do we relate the probability of program?
How do we relate the probability of program?

We need to prove claims of the form

$$\Pr_{c_1(s_1)}[E_1] \leq f(\Pr_{c_2(s_2)}[E_2])$$
How do we relate the probability of program?

We need to prove claims of the form

$$Pr_{c_1(s_1)}[E_1] \leq f(Pr_{c_2(s_2)}[E_2])$$

But usually, it suffices proving claims of the form

$$Pr_{c_1(s_1)}[E] = Pr_{c_2(s_2)}[E]$$
How do we relate the probability of program?

We need to prove claims of the form

\[ \Pr_{c_1(s_1)}[E_1] \leq f(\Pr_{c_2(s_2)}[E_2]) \]

But usually, it suffices proving claims of the form

\[ \Pr_{c_1(s_1)}[E] = \Pr_{c_2(s_2)}[E] \]

for which we can rely on **observational equivalence** between programs:

\[ \{I\} \ c_1 \sim c_2 \ \{O\} \]
How do we relate the probability of program?

We need to prove claims of the form

$$\Pr_{c_1(s_1)}[E_1] \leq f(\Pr_{c_2(s_2)}[E_2])$$

But usually, it suffices proving claims of the form

$$\Pr_{c_1(s_1)}[E] = \Pr_{c_2(s_2)}[E]$$

for which we can rely on **observational equivalence** between programs:

{\{I\} c_1 \sim c_2 \{O\}}
How do we relate the probability of program?

We need to prove claims of the form

$$\Pr_{c_1(s_1)}[E_1] \leq f(\Pr_{c_2(s_2)}[E_2])$$

But usually, it suffices proving claims of the form

$$\Pr_{c_1(s_1)}[E] = \Pr_{c_2(s_2)}[E]$$

for which we can rely on observational equivalence between programs:
How do we relate the probability of program?

We need to prove claims of the form

$$\Pr_{c_1(s_1)}[E_1] \leq f(\Pr_{c_2(s_2)}[E_2])$$

But usually, it suffices proving claims of the form

$$\Pr_{c_1(s_1)}[E] = \Pr_{c_2(s_2)}[E]$$

for which we can rely on **observational equivalence** between programs:
How do we relate the probability of program?

We need to prove claims of the form

\[
\Pr_{c_1(s_1)}[E_1] \leq f(\Pr_{c_2(s_2)}[E_2])
\]

But usually, it suffices proving claims of the form

\[
\Pr_{c_1(s_1)}[E] = \Pr_{c_2(s_2)}[E]
\]

for which we can rely on **observational equivalence** between programs:

\[
f_v(E) \subseteq O \quad \{I\} \quad c_1 \sim c_2 \quad \{O\}
\]

\[
\Pr_{c_1(s_1)}[E] = \Pr_{c_2(s_2)}[E]
\]
How do we relate the probability of program?

We need to prove claims of the form

\[ \Pr_{c_1(s_1)}[E_1] \leq f(\Pr_{c_2(s_2)}[E_2]) \]

But usually, it suffices proving claims of the form

\[ \Pr_{c_1(s_1)}[E] = \Pr_{c_2(s_2)}[E] \]

for which we can rely on observational equivalence between programs:

\[
\begin{align*}
fv(E) \subseteq O \\
\{l\} c_1 \sim c_2 \{O\} \\
Pr_{c_1(s_1)}[E] = Pr_{c_2(s_2)}[E] \\
s_1 =_l s_2
\end{align*}
\]
Proving observational equivalence

CertiCrypt provides several *mechanised program transformations* for establishing observational equivalence.
CertiCrypt provides several *mechanised program transformations* for establishing observational equivalence

**Program Transformation:**

\[ T(c_1, c_2, l, O) = (c'_1, c'_2, l', O') \]
CertiCrypt provides several *mechanised program transformations* for establishing observational equivalence

**Program Transformation:**

\[ \mathcal{T}(c_1, c_2, I, O) = (c'_1, c'_2, I', O') \]

**Soundness Result:**
Proving observational equivalence

CertiCrypt provides several *mechanised program transformations* for establishing observational equivalence

**Program Transformation:** \( \mathcal{T}(c_1, c_2, l, O) = (c'_1, c'_2, l', O') \)

**Soundness Result:**

\( \{l\} \ c_1 \sim c_2 \ \{O\} \)
Proving observational equivalence

CertiCrypt provides several *mechanised program transformations* for establishing observational equivalence

**Program Transformation:**

\[ T(c_1, c_2, I, O) = (c'_1, c'_2, I', O') \]

**Soundness Result:**

\[
\frac{T(c_1, c_2, I, O) = (c'_1, c'_2, I', O')}{\{I\} \ c_1 \sim c_2 \ {O}}
\]
Proving observational equivalence

CertiCrypt provides several *mechanised program transformations* for establishing observational equivalence

**Program Transformation:**

\[ T(c_1, c_2, I, O) = (c'_1, c'_2, I', O') \]

**Soundness Result:**

\[ T(c_1, c_2, I, O) = (c'_1, c'_2, I', O') \]

\[ \{ I' \} c'_1 \sim c'_2 \{ O' \} \]

\[ \{ I \} c_1 \sim c_2 \{ O \} \]
Proving observational equivalence

CertiCrypt provides several *mechanised program transformations* for establishing observational equivalence

**Program Transformation:**

\[ \mathcal{T}(c_1, c_2, I, O) = (c'_1, c'_2, I', O') \]

**Soundness Result:**

\[
\begin{align*}
\mathcal{T}(c_1, c_2, I, O) &= (c'_1, c'_2, I', O') \\
\{I\} c_1 &\sim c_2 \{O\}
\end{align*}
\]

**Some Instances:**

- Deadcode elimination
- Constant propagation
- Procedure call inlining
- Common prefix/suffix elimination
CertiCrypt provides an (incomplete) tactic for proving *self-equivalence*

Does \( \{I\} \ c \sim c \ \{O\} \) hold?
Proving observational equivalence

CertiCrypt provides an (incomplete) tactic for proving *self-equivalence*

Does \( \{I\} c \sim c \{O\} \) hold?

- Analyse dependencies to compute \( I' \) such that \( \{I'\} c \sim c \{O\} \)
CertiCrypt provides an (incomplete) tactic for proving \textit{self-equivalence}

Does \( \{I\} \ c \sim c \{O\} \) hold?

- Analyse dependencies to compute \( I' \) such that \( \{I'\} \ c \sim c \{O\} \)
- Check that \( I' \subseteq I \)
Security proof of ElGamal encryption scheme

**Game ElGamal**: 
\[(x, \alpha) \leftarrow \text{KG}();\]  
\[(m_0, m_1) \leftarrow \mathcal{A}(\alpha);\]  
\[b \leftarrow \{0, 1\};\]  
\[(\beta, \zeta) \leftarrow \text{Enc}(\alpha, m_b);\]  
\[b' \leftarrow \mathcal{A}'(\alpha, \beta, \zeta);\]  
\[d \leftarrow b = b'.\]

**Game ElGamal_0**: 
\[x \leftarrow \mathbb{Z}_q; y \leftarrow \mathbb{Z}_q;\]  
\[(m_0, m_1) \leftarrow \mathcal{A}(g^x);\]  
\[b \leftarrow \{0, 1\};\]  
\[\zeta \leftarrow g^{xy} \times m_b;\]  
\[b' \leftarrow \mathcal{A}'(g^x, g^y, \zeta);\]  
\[d \leftarrow b = b'.\]

**Game ElGamal_1**: 
\[x \leftarrow \mathbb{Z}_q; y \leftarrow \mathbb{Z}_q;\]  
\[(m_0, m_1) \leftarrow \mathcal{A}(g^x);\]  
\[z \leftarrow \mathbb{Z}_q; \zeta \leftarrow g^{z};\]  
\[b' \leftarrow \mathcal{A}'(g^x, g^y, \zeta);\]  
\[d \leftarrow b = b'.\]

**Game ElGamal_2**: 
\[x \leftarrow \mathbb{Z}_q; y \leftarrow \mathbb{Z}_q;\]  
\[(m_0, m_1) \leftarrow \mathcal{A}(g^x);\]  
\[z \leftarrow \mathbb{Z}_q; \zeta \leftarrow g^{z};\]  
\[b' \leftarrow \mathcal{A}'(g^x, g^y, \zeta);\]  
\[d \leftarrow b = b'.\]

**Game DDH_0**: 
\[x \leftarrow \mathbb{Z}_q;\]  
\[y \leftarrow \mathbb{Z}_q;\]  
\[d \leftarrow \mathcal{B}(g^x, g^y, g^{xy})\]

**Game DDH_1**: 
\[x \leftarrow \mathbb{Z}_q;\]  
\[y \leftarrow \mathbb{Z}_q;\]  
\[d \leftarrow \mathcal{B}(g^x, g^y, g^z)\]

**Adversary \(B(\alpha, \beta, \gamma)\)**:  
\[(m_0, m_1) \leftarrow \mathcal{A}(\alpha);\]  
\[b \leftarrow \{0, 1\};\]  
\[b' \leftarrow \mathcal{A}'(\alpha, \beta \times m_b);\]  
\[\text{return } b = b'.\]

**Lemma B.PPT**: PPT B. Proof. PPT_tac. Qed.

**Lemma B_wf**: WFAdv B. Proof. ... Qed.
Security proof of ElGamal encryption scheme

\textbf{Game ElGamal}_2:
\begin{align*}
x & \leftarrow \mathbb{Z}_q; \quad y \leftarrow \mathbb{Z}_q; \\
(m_0, m_1) & \leftarrow A(g^x); \\
z & \leftarrow \mathbb{Z}_q; \quad \zeta \leftarrow g^z; \\
b' & \leftarrow A'(g^x, g^y, \zeta); \\
b & \leftarrow \{0, 1\}; \\
d & \leftarrow b = b'\end{align*}

\textbf{Game ElGamal}_1:
\begin{align*}
x & \leftarrow \mathbb{Z}_q; \quad y \leftarrow \mathbb{Z}_q; \\
(m_0, m_1) & \leftarrow A(g^x); \\
b & \leftarrow \{0, 1\}; \\
z & \leftarrow \mathbb{Z}_q; \quad \zeta \leftarrow g^z \times m_b; \\
b' & \leftarrow A'(g^x, g^y, \zeta); \\
d & \leftarrow b = b'\end{align*}

\text{swap. eqobs\_hd 4. eqobs\_tl 2. apply mult\_pad.}
Observational equivalence is not enough
Observational equivalence is not enough

\[
\{x\} \quad \text{if} \ (x=0) \ \text{then} \ y\leftarrow x \ \text{else} \ y\leftarrow 1 \ \sim \ \text{if} \ (x=0) \ \text{then} \ y\leftarrow 0 \ \text{else} \ y\leftarrow 1 \quad \{x, y\}
\]
Observational equivalence is not enough

- Establishing observational equivalence may require additional contextual information

\[
\{x\} \quad \text{if } (x=0) \quad \text{then } y \leftarrow x \quad \text{else } y \leftarrow 1 \quad \sim \quad \text{if } (x=0) \quad \text{then } y \leftarrow 0 \quad \text{else } y \leftarrow 1 \quad \{x, y\}
\]
Observational equivalence is not enough

- Establishing observational equivalence may require additional contextual information

\[
\{x\} \quad \text{if} \ (x=0) \quad \text{then} \quad y \leftarrow x \quad \text{else} \quad y \leftarrow 1 \quad \sim \quad \text{if} \ (x=0) \quad \text{then} \quad y \leftarrow 0 \quad \text{else} \quad y \leftarrow 1 \quad \{x, y\}
\]

- Cryptographic proofs may involve weaker relationships between consecutive games, e.g.

\[
\Pr_{c_1(s_1)}[E_1] \leq \Pr_{c_2(s_2)}[E_2]
\]
Relational Hoare logic
Relational Hoare logic

Standard Hoare Logic (HL)

\( \{ P \} c \{ Q \} \)
Relational Hoare logic

Standard Hoare Logic (HL)

\[ \{ P \} \ c \ \{ Q \} \]

\[ s \rightarrow P(s) \]

\[ c \]

\[ s' \rightarrow Q(s') \]
Relational Hoare logic

Standard Hoare Logic (HL)

\[ \{ P \} \ c \ { Q \} \]

\[
\begin{align*}
  s & \quad P(s) \\
  c & \\
  s' & \quad Q(s')
\end{align*}
\]

Relational Hoare Logic (RHL)

\[ \{ P \} \ c_1 \sim c_2 \ { Q \} \]
Relational Hoare logic

Standard Hoare Logic (HL)

\( \{ P \} \; c \; \{ Q \} \)

\( s \quad P(s) \)

\( \downarrow \quad c \quad \uparrow \)

\( s' \quad Q(s') \)

Relational Hoare Logic (RHL)

\( \{ P \} \; c_1 \sim c_2 \; \{ Q \} \)

\( s_1 \quad P \quad s_2 \)

\( \downarrow \quad c_1 \quad \uparrow \quad c_2 \)

\( s_1' \quad Q \quad s_2' \)
Relational Hoare logic

Standard Hoare Logic (HL)

\[ \{P\} c \{Q\} \]

\[ s \xrightarrow{P(s)} s' \]

\[ s \xrightarrow{c} s' \xrightarrow{Q(s')} \]

Relational Hoare Logic (RHL)

\[ \{P\} c_1 \sim c_2 \{Q\} \]

\[ s_1 \xrightarrow{P} s_2 \]

\[ s_1 \xleftarrow{c_1} P \xrightarrow{c_2} s_2 \]

\[ s_1' \xrightarrow{Q} s_2' \]

\[ s_1' \xleftarrow{Q} s_2' \]
Relational Hoare logic

Standard Hoare Logic (HL)

\[ \{ P \} \ c \ \{ Q \} \]
\[ s \rightarrow P(s) \]
\[ c \rightarrow c \]
\[ s' \leftarrow Q(s') \]

Relational Hoare Logic (RHL)

probabilistic programs

\[ \{ P \} \ c_1 \sim c_2 \ \{ Q \} \]
\[ s_1 \leftarrow P \rightarrow s_2 \]
\[ c_1 \rightarrow c_1 \]
\[ c_2 \rightarrow c_2 \]
\[ \mu_1 \leftarrow Q \rightarrow \mu_2 \]
distributions over states
Relational Hoare logic

Standard Hoare Logic (HL)

\[
\{ P \} \ c \ \{ Q \}
\]

\[
s \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad

Relational Hoare logic — Judgment examples

$z := y+1 \sim z := x$
Relational Hoare logic — Judgment examples

\[ \models \{ y(1)+1 = x(2) \} \quad z := y+1 \sim z := x \]
Relational Hoare logic — Judgment examples

\[ \models \{ y_1 + 1 = x_2 \} \quad z := y + 1 \sim z := x \quad \{ z_1 = z_2 \} \]
Relational Hoare logic — Judgment examples

\[ = \{y(1)+1 = x(2)\} \quad z := y+1 \sim z := x \quad \{z(1) = z(2)\} \]

\[ \text{if } b \text{ then } x := 0 \quad \sim \quad \text{if } b \text{ then } x := 1 \]
\[ \text{else } x := 1 \quad \sim \quad \text{else } x := 0 \]
Relational Hoare logic — Judgment examples

\[ \models \{ y(1) + 1 = x(2) \} \quad z := y + 1 \sim z := x \quad \{ z(1) = z(2) \} \]

\[ \models \{ b(1) = b(2) \} \quad \text{if } b \text{ then } x := 0 \quad \text{else } x := 1 \sim \text{if } b \text{ then } x := 1 \quad \text{else } x := 0 \]
Relational Hoare logic — Judgment examples

\[\models \{y_1 + 1 = x_2\} \quad z := y + 1 \sim z := x \quad \{z_1 = z_2\}\]

\[\models \{b_1 = b_2\} \quad \text{if } b \text{ then } x := 0 \quad \text{else } x := 1 \sim \text{if } b \text{ then } x := 1 \quad \text{else } x := 0 \quad \{x_1 = 1 - x_2\}\]
Proof system
Proof system

- Most rules are direct adaptations of traditional HL rules
Proof system

Most rules are direct adaptations of traditional HL rules

(\texttt{-\{P\} skip \{P\}})
Proof system

Most rules are direct adaptations of traditional HL rules

⊢ \{P\} \text{skip} ~ \text{skip} \{P\}

(⊢ \{P\} \text{skip} \{P\})
Proof system

Most rules are direct adaptations of traditional HL rules

\[
\vdash \{ P \} \text{skip} \sim \text{skip} \{ P \} \quad \quad (\vdash \{ P \} \text{skip} \{ P \})
\]

\[
\begin{align*}
&\vdash \{ P \} \ c \{ Q' \} \quad \vdash \{ Q' \} \ c' \{ Q \} \\
&\Rightarrow \vdash \{ P \} \ c; c' \{ Q \}
\end{align*}
\]
Most rules are direct adaptations of traditional HL rules

\[\vdash \{P\} \text{skip} \sim \text{skip} \{P\}\]

\[\vdash \{P\} c_1 \sim c_2 \{Q'\} \quad \vdash \{Q'\} c'_1 \sim c'_2 \{Q\}\]

\[\vdash \{P\} c_1; c'_1 \sim c_2; c'_2 \{Q\}\]
Proof system

Most rules are direct adaptations of traditional HL rules

\[ \vdash \{P\} \text{skip} \sim \text{skip} \{P\} \]

\[ \vdash \{P\} c_1 \sim c_2 \{Q'\} \quad \vdash \{Q'\} c'_1 \sim c'_2 \{Q\} \]

\[ \vdash \{P\} c_1; c'_1 \sim c_2; c'_2 \{Q\} \]

\[ \vdash \{P\} \text{skip} \{P\} \]

\[ \vdash \{P\} \text{skip} \{P\} \]

\[ \vdash \{P\} c \{Q'\} \quad \vdash \{Q'\} c' \{Q\} \]

\[ \vdash \{P\} c; c' \{Q\} \]

Requires programs to execute lockstep
Proof system

- Most rules are direct adaptations of traditional HL rules

\[ \vdash \{ P \} \text{skip} \sim \text{skip} \{ P \} \]  
\[ \vdash \{ P \} c_1 \sim c_2 \{ Q' \} \quad \vdash \{ Q' \} c'_1 \sim c'_2 \{ Q \} \]  
\[ \vdash \{ P \} c_1; c'_1 \sim c_2; c'_2 \{ Q \} \]  
\[ (\vdash \{ P \} \text{skip} \{ P \}) \]

\[ \vdash \{ P \} c \{ Q' \} \vdash \{ Q' \} c' \{ Q \} \]  
\[ \vdash \{ P \} c; c' \{ Q \} \]  

- Requires programs to execute lockstep

\[ \vdash \{ I \land G_{1\langle 1 \rangle} \} c_1 \sim c_2 \{ I \} \quad \models (I \Rightarrow G_{1\langle 1 \rangle} = G_{2\langle 2 \rangle}) \]  
\[ \vdash \{ I \} \text{while } G_1 \text{ do } c_1 \sim \text{while } G_2 \text{ do } c_2 \{ I \land \neg G_{1\langle 1 \rangle} \} \]  
[while]
Proof system

- Most rules are direct adaptations of traditional HL rules

\[
\begin{align*}
\vdash \{P\} \text{skip} \sim \text{skip} \{P\} \quad \vdash \{P\} \ c_1 \sim c_2 \ \{Q'\} \quad \vdash \{Q'\} \ c'_1 \sim c'_2 \ \{Q\} \\
\vdash \{P\} \ c_1; c'_1 \sim c_2; c'_2 \ \{Q\} \\
\end{align*}
\]

\[
\vdash \{P\} \ c \ {Q'} \vdash \{Q'\} \ c' \ \{Q\}
\]

- Requires programs to execute lockstep

\[
\begin{align*}
\vdash \{I \land G_1(1)\} \ c_1 \sim c_2 \ \{I\} \\
\vdash \{I\} \ \text{while} \ G_1 \ \text{do} \ c_1 \sim \text{while} \ G_2 \ \text{do} \ c_2 \ \{I \land \neg G_1(1)\}
\end{align*}
\]

\[
\vdash \{I\} \quad \vdash \{I \implies G_1(1) = G_2(2)\}
\]

- (The classic fragment) only relates programs that are structurally equal.
Proof system

- Most rules are direct adaptations of traditional HL rules

\[ \vdash \{ P \} \text{skip} \sim \text{skip} \{ P \} \quad (\vdash \{ P \} \text{skip} \{ P \}) \]
\[ \vdash \{ P \} c_1 \sim c_2 \{ Q' \} \quad \vdash \{ Q' \} c'_1 \sim c'_2 \{ Q \} \]
\[ \vdash \{ P \} c_1; c'_1 \sim c_2; c'_2 \{ Q \} \]
\[ (\vdash \{ P \} c_1; c'_1 \sim c_2; c'_2 \{ Q \}) \]

- Requires programs to execute lockstep

\[ \vdash \{ l \land G_{1(1)} \} c_1 \sim c_2 \{ l \} \quad \models \left( l \implies G_{1(1)} = G_{2(2)} \right) \]
\[ \vdash \{ l \} \text{while } G_1 \text{ do } c_1 \sim \text{while } G_2 \text{ do } c_2 \{ l \land \neg G_{1(1)} \} \]
\[ \text{[while]} \]

- The classic fragment only relates programs that are structurally equal.
  But the logic can be extended with “one-sided” rules, e.g.
Proof system

- Most rules are direct adaptations of traditional HL rules

\[
\vdash \{P\} \text{skip} \sim \text{skip} \{P\} \\
\vdash \{P\} c_1 \sim c_2 \{Q\} \quad \vdash \{Q\} c'_1 \sim c'_2 \{Q\} \\
\hline
\vdash \{P\} c_1; c'_1 \sim c_2; c'_2 \{Q\}
\]

(\vdash \{P\} \text{skip} \{P\})

(\vdash \{P\} c \{Q\} \quad \vdash \{Q\} c' \{Q\})

(\vdash \{P\} c; c' \{Q\})

- Requires programs to execute lockstep

\[
\vdash \{I \land G_{1(1)}\} c_1 \sim c_2 \{I\} \\
\hline
\vdash \{I\} \text{while } G_1 \text{ do } c_1 \sim \text{while } G_2 \text{ do } c_2 \{I \land \neg G_{1(1)}\}
\]

\(\vdash \{I \implies G_{1(1)} = G_{2(2)}\}\) [while]

- (The classic fragment) only relates programs that are structurally equal. But the logic can be extended with “one-sided” rules, e.g.

\[
\vdash \{P\} \text{if } G \text{ then } c_1 \text{ else } c'_1 \sim c_2 \{Q\}
\]

[c-branch]
Proof system

- Most rules are direct adaptations of traditional HL rules

\[ \vdash \{ P \} \text{skip} \sim \text{skip} \{ P \} \quad (\vdash \{ P \} \text{skip} \{ P \}) \]

\[ \vdash \{ P \} c_1 \sim c_2 \{ Q' \} \quad \vdash \{ Q' \} c_1' \sim c_2' \{ Q \} \]

\[ \vdash \{ P \} c_1; c_1' \sim c_2; c_2' \{ Q \} \quad (\vdash \{ P \} c \{ Q' \} \quad \vdash \{ Q' \} c' \{ Q \}) \quad (\vdash \{ P \} c; c' \{ Q \}) \]

- Requires programs to execute lockstep

\[ \vdash \{ l \land G_{1(1)} \} c_1 \sim c_2 \{ l \} \quad \models (l \implies G_{1(1)} = G_{2(2)}) \]

\[ \vdash \{ l \} \text{while } G_1 \text{ do } c_1 \sim \text{while } G_2 \text{ do } c_2 \{ l \land \neg G_{1(1)} \} \quad \text{[while]} \]

- (The classic fragment) only relates programs that are structurally equal. But the logic can be extended with “one-sided” rules, e.g.

\[ \vdash \{ P \land G_{(1)} \} c_1 \sim c_2 \{ Q \} \quad \vdash \{ P \land \neg G_{(1)} \} c_1' \sim c_2 \{ Q \} \]

\[ \vdash \{ P \} \text{if } G \text{ then } c_1 \text{ else } c_1' \sim c_2 \{ Q \} \quad \text{[c-branch]} \]
From the logic to probability claims
From the logic to probability claims

\[
\frac{\Pr[c_1(s_1) : A]}{\Pr[c_2(s_2) : B]} \quad \text{[Pr-Eq]}
\]
From the logic to probability claims

\[ \models \{P\} c_1 \sim c_2 \{Q\} \]

\[ \text{Pr}[c_1(s_1) : A] = \text{Pr}[c_2(s_2) : B] \]  

[Pr-Eq]
From the logic to probability claims

\[
\begin{align*}
\models \{P\} c_1 \sim c_2 \{Q\} & \quad Q \implies (A \iff B) \\
\Pr[c_1(s_1) : A] &= \Pr[c_2(s_2) : B] \\
\end{align*}
\]

[Pr-Eq]
From the logic to probability claims

\[
\begin{align*}
\text{Pr}\left[ c_1(s_1) : A \right] &= \text{Pr}\left[ c_2(s_2) : B \right] \\
\text{Pr-Eq} &
\end{align*}
\]
From the logic to probability claims

\[
\begin{align*}
\frac{s_1 P s_2 \models \{P\} c_1 \sim c_2 \{Q\} \quad Q \implies (A(1) \iff B(2))}{\Pr[c_1(s_1) : A] = \Pr[c_2(s_2) : B]} & \text{[Pr-Eq]} \\
\frac{\Pr[c_1(s_1) : A] \leq \Pr[c_2(s_2) : B]} & \text{[Pr-Le]}
\end{align*}
\]
From the logic to probability claims

\[
\begin{align*}
\frac{s_1 \ P \ s_2 \quad \models \ {\{P\} \ c_1 \sim c_2 \ \{Q\} \quad Q \implies (A^{(1)} \iff B^{(2)})}}{\Pr[c_1(s_1) : A] = \Pr[c_2(s_2) : B]} & \quad [\text{Pr-Eq}] \\
\frac{s_1 \ P \ s_2 \quad \models \ {\{P\} \ c_1 \sim c_2 \ \{Q\} \quad Q \implies (A^{(1)} \implies B^{(2)})}}{\Pr[c_1(s_1) : A] \leq \Pr[c_2(s_2) : B]} & \quad [\text{Pr-Le}]
\end{align*}
\]
Wrapping up
Conclusion
Conclusion

Successful application of machine-checked proofs to the field of cryptography
Conclusion

Successful application of machine-checked proofs to the field of cryptography

- Formal semantics of probabilistic language
- A probabilistic relational Hoare logic
- Mechanised program transformations
- Formalization of emblematic schemes: OAEP, ElGamal, FDH, etc.
Conclusion

Successful application of machine-checked proofs to the field of cryptography

- Formal semantics of probabilistic language
- A probabilistic relational Hoare logic
- Mechanised program transformations
- Formalization of emblematic schemes: OAEP, ElGamal, FDH, etc.

**Key Insight:**

View cryptographic proofs as a problem of (relational) probabilistic program verification
Successful application of machine-checked proofs to the field of cryptography

- Formal semantics of probabilistic language
- A probabilistic relational Hoare logic
- Mechanised program transformations
- Formalization of emblematic schemes: OAEP, ElGamal, FDH, etc.

Key Insight:
View cryptographic proofs as a problem of (relational) probabilistic program verification

Thanks!
Backup Slides
Language semantics

\[
\begin{align*}
\llbracket \text{skip} \rrbracket m &= \text{unit } m \\
\llbracket c; c' \rrbracket m &= \text{bind } (\llbracket c \rrbracket m) \llbracket c' \rrbracket \\
\llbracket x \leftarrow e \rrbracket m &= \text{unit } (m \{\{e\} m / x\}) \\
\llbracket x \triangleleft d \rrbracket m &= \text{bind } (\llbracket d \rrbracket D E m) (\lambda v. \text{unit } (m \{v / x\})) \\
\llbracket \text{assert } e \rrbracket m &= \text{if } (\llbracket e \rrbracket E m = \text{true}) \text{ then } (\text{unit } m) \text{ else } \mu_0 \\
\llbracket \text{if } e \text{ then } c_1 \text{ else } c_2 \rrbracket m &= \text{if } (\llbracket e \rrbracket E m = \text{true}) \text{ then } (\llbracket c_1 \rrbracket m) \text{ else } (\llbracket c_2 \rrbracket m) \\
\llbracket \text{while } e \text{ do } c \rrbracket m &= \lambda f. \text{ lub } (\lambda n. (\llbracket \text{while } e \text{ do } c \rrbracket n m)(f)) \\
\text{where} & \quad \llbracket \text{while } e \text{ do } c \rrbracket_0 = \text{assert } \neg e \\
& \quad \llbracket \text{while } e \text{ do } c \rrbracket_{n+1} = \text{if } e \text{ then } c; \llbracket \text{while } e \text{ do } c \rrbracket_n
\end{align*}
\]
The measure monad (ALEA library)

\[ D(A) \triangleq (A \to [0, 1]) \to [0, 1] \]

\[ \mu(f) = \text{"expected value of } f \text{ wrt } \mu\]

\[
\begin{align*}
\text{unit} & : A \to D(A) \\
& \overset{\text{def}}{=} \lambda x. \lambda f. f(x) \\
\text{bind} & : D(A) \to (A \to D(B)) \to D(B) \\
& \overset{\text{def}}{=} \lambda \mu. \lambda M. \lambda f. \mu(\lambda x. M(x)(f)).
\end{align*}
\]

Example

\[
\begin{align*}
[ b_1 \overset{\$}{\leftarrow} \{ t, f \}; b_2 \overset{\$}{\leftarrow} \{ t, f \}] s &= \lambda f. \frac{1}{4} f(s[b_1, b_2/t, t]) + \frac{1}{4} f(s[b_1, b_2/t, f]) \\
& \hspace{1cm} + \frac{1}{4} f(s[b_1, b_2/f, t]) + \frac{1}{4} f(s[b_1, b_2/f, f])
\end{align*}
\]
Lifting relations to distributions via couplings
Lifting relations to distributions via couplings

\[ (\mu_1, \mu_2) \models Q^\# \triangleq \exists \mu \in \mathcal{D}(S \times S) . \begin{cases} \pi_1(\mu) = \mu_1 \land \pi_2(\mu) = \mu_2, \\
\Pr_{\mu}[\neg Q] = 0 \end{cases} \]

* See *Logical, Metric, and Algorithmic Characterisations of Probabilistic Bisimulation*, Deng & Du.
Proof system (two-sided rules)

\[
\begin{align*}
\vdash \{P\} \text{ skip} & \sim \text{ skip } \{P\} \quad \text{[skip]} \\
\vdash \{P\} c_1 \sim c_2 \ \{Q\} & \vdash \{Q\} c_1' \sim c_2' \ \{Q\} \quad \text{[seq]} \\
\vdash \{\text{true}\} \text{ abort} & \sim \text{ abort } \{Q\} \quad \text{[abort]} \\
\vdash \{Q[x_1/A(1), y_2/B(2)]\} \ x := A \sim y := B \ \{Q\} & \quad \text{[assign]} \\
\vdash \{Q \implies P'\} \quad \vdash \{P'\} c_1 \sim c_2 \ \{Q\} & \vdash \{Q' \implies Q\} \quad \text{[cons]} \\
\vdash \{P \implies G_{1(1)} = G_{2(2)}\} \quad \vdash \{P \land G_{1(1)}\} c_1 \sim c_2 \ \{Q\} & \vdash \{P \land \neg G_{1(1)}\} c'_1 \sim c'_2 \ \{Q\} \quad \text{[if]} \\
\vdash \{I \land G_{1(1)}\} c_1 \sim c_2 \ \{I\} \quad \vdash \{I \implies G_{1(1)} = G_{2(2)}\} \quad \text{[while]} \\
\vdash \{P^{-1}\} c_2 \sim c_1 \ \{Q^{-1}\} \quad \vdash \{P\} c_1 \sim c_2 \ \{Q\} \quad \vdash \{P'\} c_2 \sim c_3 \ \{Q'\} \quad \text{[comp]} \\
\vdash \{\mu_1 \triangleright \lambda \nu \cdot \eta_{s_1[x_1/\nu]}\} \mathcal{L}(Q) (\mu_2 \triangleright \lambda \nu \cdot \eta_{s_2[x_2/\nu]}) \quad \vdash \{P\} x_1 := \mu_1 \sim x_2 := \mu_2 \ \{Q\} \quad \text{[rand]} \\
\end{align*}
\]
Proof system (one-sided rules)

\[
\vdash \{\text{false}\} c_1 \leadsto c_2 \{Q\} \quad [\text{contr}]
\]

\[
\vdash \{Q[x_{(1)}/A_{(1)}]\} x := A \leadsto \text{skip} \{Q\} \quad [\text{d-assgn}]
\]

\[
\vdash \{P \land G_{(1)}\} c_1 \leadsto c_2 \{Q\} \quad \vdash \{P \land \neg G_{(1)}\} c'_1 \leadsto c_2 \{Q\} \\
\vdash \{P\} \text{if } G \text{ then } c_1 \text{ else } c'_1 \leadsto c_2 \{Q\} \quad [\text{c-branch}]
\]

\[
\vdash \{P \land \neg G_{(1)}\} \text{while } G \text{ do } c \leadsto \text{skip} \{P \land \neg G_{(1)}\} \quad [\text{d-while}]
\]