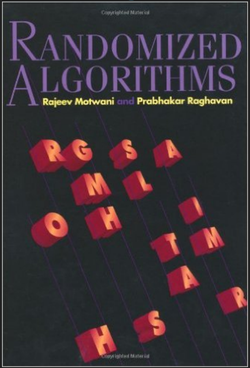
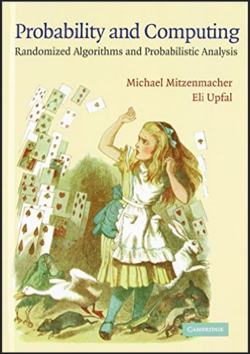
An Assertion-Based Program Logic for Probabilistic Programs

Gilles Barthe, Thomas Espitau, Marco Gaboardi,
Benjamin Grégoire, Justin Hsu, and Pierre-Yves Strub

Randomized algorithms are everywhere!





Complex programs

Algorithm 1 Joint Differentially Private Convex Solver: $\mathsf{PrivDude}(\mathcal{O}, \sigma, \tau, w, \varepsilon, \delta, \beta)$

Input: Convex problem $\mathcal{O} = (S, v, c, b)$ with n agents and k coupling constraints, gradient sensitivity bounded by σ, a dual bound τ, width bounded by w, and privacy parameters $\varepsilon > 0, \delta \in (0,1)$, confidence parameter $\beta \in (0,1)$.

Initialize:

$$\lambda_j^{(1)} := 0 \text{ for } j \in [k], \qquad T := w^2, \qquad \varepsilon' := \frac{\varepsilon\sigma}{\sqrt{8T\ln(2/\delta)}}, \qquad \delta' := \frac{\delta}{2T},$$

$$\eta := \frac{2\tau}{\sqrt{T}\left(w + \frac{1}{\varepsilon'}\log\left(\frac{Tk}{\beta}\right)\right)}, \qquad \Lambda := \{\lambda \in \mathbb{R}_+^k \mid \|\lambda\|_\infty \leq 2\tau\}.$$

for iteration $t = 1 \dots T$

for each agent $i = 0 \dots n$

Compute personal best response:

$$x_t^{(i)} := \operatorname*{argmax}_{x \in S^{(i)}} v^{(i)}(x) - \sum_{j=1}^{k} \lambda_j^{(t)} c^{(i)}(x).$$

for each constraint $j = 1 \dots k$

Compute noisy gradient:

$$\widehat{\ell}_j^{(t)} := \left(\sum_{i=0}^{n} c^{(i)}(x_t^{(i)})\right) - b_j + \mathcal{N}\left(0, \frac{2\sigma^2 \log(1.25/\delta')}{\varepsilon'^2}\right),$$

Do gradient descent update:

Complex proofs

Proof. Let ν_t denote the noise vector we have in round t, we can decompose the regret into several parts

$$
\begin{aligned}
\mathcal{R}_T &= \frac{1}{T}\sum_{t=1}^{T}\langle p_t, x_t\rangle - \frac{1}{T}\min_{p\in\mathcal{P}}\sum_{t=1}^{T}\langle p, x_t\rangle \\
&= \frac{1}{T}\sum_{t=1}^{T}\langle p_t, \widehat{x}_t\rangle - \frac{1}{T}\sum_{t=1}^{T}\langle p_t, \nu_t\rangle - \frac{1}{T}\left[\min_{p\in\mathcal{P}}\sum_{t=1}^{T}\langle p, x_t\rangle - \min_{\widehat{p}\in\mathcal{P}}\sum_{t=1}^{T}\langle \widehat{p}, \widehat{x}_t\rangle\right] - \frac{1}{T}\min_{\widehat{p}\in\mathcal{P}}\sum_{t=1}^{T}\langle \widehat{p}, \widehat{x}_t\rangle \\
&= \left[\frac{1}{T}\sum_{t=1}^{T}\langle p_t, \widehat{x}_t\rangle - \frac{1}{T}\min_{\widehat{p}\in\mathcal{P}}\sum_{t=1}^{T}\langle \widehat{p}, \widehat{x}_t\rangle\right] - \frac{1}{T}\sum_{t=1}^{T}\langle p_t, \nu_t\rangle - \frac{1}{T}\left[\min_{p\in\mathcal{P}}\sum_{t=1}^{T}\langle p, x_t\rangle - \min_{\widehat{p}\in\mathcal{P}}\sum_{t=1}^{T}\langle \widehat{p}, \widehat{x}_t\rangle\right] \\
&= \widehat{\mathcal{R}}_T - \frac{1}{T}\sum_{t=1}^{T}\langle p_t, \nu_t\rangle - \frac{1}{T}\left[\min_{p\in\mathcal{P}}\sum_{t=1}^{T}\langle p, x_t\rangle - \min_{\widehat{p}\in\mathcal{P}}\sum_{t=1}^{T}\langle \widehat{p}, \widehat{x}_t\rangle\right] \\
&\leq \widehat{\mathcal{R}}_T - \frac{1}{T}\min_{p\in\mathcal{P}}\sum_{t=1}^{T}\langle p, \nu_t\rangle - \frac{1}{T}\left[\min_{p\in\mathcal{P}}\sum_{t=1}^{T}\langle p, x_t\rangle - \min_{\widehat{p}\in\mathcal{P}}\sum_{t=1}^{T}\langle \widehat{p}, \widehat{x}_t\rangle\right].
\end{aligned}
$$

We will bound the three terms separately. By the no-regret guarantee of online gradient descent in Lemma 13, we have the following the regret guarantee w.r.t the noisy losses if we set $\eta = \frac{\|\mathcal{P}\|}{\sqrt{T}\|\widehat{\mathcal{X}}\|}$

$$
\widehat{\mathcal{R}}_T = \frac{1}{T}\sum_{t=1}^{T}\langle p_t, \widehat{x}_t\rangle - \min_{p\in\mathcal{P}}\frac{1}{T}\sum_{t=1}^{T}\langle p, \widehat{x}_t\rangle \leq \frac{\|\mathcal{P}\|^2}{2\eta T} + \frac{\eta\|\widehat{\mathcal{X}}\|^2}{2} = \frac{\|\mathcal{P}\|\|\widehat{\mathcal{X}}\|}{\sqrt{T}},
$$

where $\|\mathcal{P}\|$ and $\|\widehat{\mathcal{X}}\|$ denote the bound on the ℓ_2 norm of the vectors $\{p_t\}$ and $\{\widehat{x}_t\}$ respectively.

Recall that for any random variable Y sampled from the Gaussian distribution $\mathcal{N}(0, \sigma^2)$, we

A simple randomized algorithm and property

Noisy sum

$sum \leftarrow 0;$
for $i = 1, \dots, n$ do
  $toss \xleftarrow{\$} \mathit{flip}(p);$
  $sum \leftarrow sum + toss;$
return(sum)

A simple randomized algorithm and property

Noisy sum

$sum \leftarrow 0;$
for $i = 1, \ldots, n$ do
  $toss \xleftarrow{\$} \mathit{flip}(p);$
  $sum \leftarrow sum + toss;$
return(sum)

To show: sum not too small

$$\boxed{\begin{array}{c}\Pr[sum \leq n \cdot p - 4\sqrt{n \cdot p}] \\ \text{is at most } 0.0005\end{array}}$$

A simple randomized algorithm and property

Noisy sum

$sum \leftarrow 0;$
for $i = 1, \ldots, n$ do
  $toss \xleftarrow{\$} \mathit{flip}(p);$
  $sum \leftarrow sum + toss;$
return(sum)

To show: sum not too small

$$\boxed{\begin{array}{c}\Pr[sum \leq n \cdot p - 4\sqrt{n \cdot p}] \\ \text{is at most } 0.0005\end{array}}$$

Proof of correctness, on paper?

1. sum is sum of n independent p-biased coins.
2. Apply standard concentration bound, done.

Deductive verification? Not so easy.

Expectation-based approaches

- Rules manipulate single expected value/probability
- Can't directly express properties like independence
- Kozen's PPDL (1985); Morgan, McIver, Seidel's PGCL (1996)

Deductive verification? Not so easy.

Expectation-based approaches

- Rules manipulate single expected value/probability
- Can't directly express properties like independence
- Kozen's PPDL (1985); Morgan, McIver, Seidel's PGCL (1996)

Program logic (assertion-based) approaches

- Use general boolean assertions on distributions
- Complex loop rules, more limited programming languages
- Chadha et al. (2007); Rand and Zdancewic (2015)

Overall goal: Narrow this gap

Work with higher-level properties as much as possible

- Minimize reasoning about single probabilities

Avoid reasoning at level of program semantics

- Side-conditions should be easy to check

Incorporate proof methods from paper proofs

- Structure the proof, abstract away unimportant details

More concretely: Our contributions

- A new program logic for probabilistic programs
- Embeddings of several specialized proof techniques
- Implementation and formalized examples

The Ellora Framework:
A Lightning Tour

The core: A program logic for probabilistic programs

The pWhile imperative language

$$c ::= x \leftarrow e \mid x \xleftarrow{\$} d \mid c; c \mid \text{if } e \text{ then } c \text{ else } c \mid \text{while } e \text{ do } c$$

The core: A program logic for probabilistic programs

The pWhile imperative language

$$c ::= x \leftarrow e \mid \boxed{x \xleftarrow{\$} d} \mid c; c \mid \text{if } e \text{ then } c \text{ else } c \mid \text{while } e \text{ do } c$$

Sample from primitive distributions

- Biased coin flips, uniform distribution, …
- Geometric distribution, Laplace distribution, …

The core: A program logic for probabilistic programs

The pWhile imperative language

$$c ::= x \leftarrow e \mid x \xleftarrow{\$} d \mid c; c \mid \text{if } e \text{ then } c \text{ else } c \mid \text{while } e \text{ do } c$$

Sample from primitive distributions

- Biased coin flips, uniform distribution, …
- Geometric distribution, Laplace distribution, …

Commands transform (sub-)distributions over memories

- Distribution over inputs $\mapsto$ Distribution over outputs

Assertion language: two layers

State assertions: model memories

$$\phi, \psi \quad ::= \quad e = e' \mid e < e' \mid \ldots$$

Assertion language: two layers

State assertions: model memories

$$\phi, \psi \quad ::= \quad e = e' \mid e < e' \mid \dots$$

Distribution assertions: model distributions

$$\Phi, \Psi \quad ::= \quad \mathbb{E}[e] = \mathbb{E}[e'] \mid \mathbb{E}[e] < \mathbb{E}[e'] \mid \dots$$

Assertion language: two layers

State assertions: model memories

$$\phi, \psi \quad ::= \quad e = e' \mid e < e' \mid \ldots$$

Distribution assertions: model distributions

$$\Phi, \Psi \quad ::= \quad \mathbb{E}[e] = \mathbb{E}[e'] \mid \mathbb{E}[e] < \mathbb{E}[e'] \mid \ldots$$

Examples: defined notation

$$\mathbb{P}[\phi] \triangleq \mathbb{E}[1_\phi] \qquad\qquad \Box\phi \triangleq \mathbb{P}[\phi] = 1$$

Proof system

Typical program logic judgment

$$\{\Phi\}\ c\ \{\Psi\}$$

Proof system

Typical program logic judgment

$$\{\Phi\}\ c\ \{\Psi\}$$

System rules

$$\frac{\eta_0 \Rightarrow \eta_1 \qquad \{\eta_1\}\ s\ \{\eta_2\} \qquad \eta_2 \Rightarrow \eta_3}{\{\eta_0\}\ s\ \{\eta_3\}}\ [\textsc{Conseq}] \qquad \frac{\forall x : T.\ \{\eta\}\ s\ \{\eta'\}}{\{\exists x : T.\ \eta\}\ s\ \{\eta'\}}\ [\textsc{Exists}]$$

$$\frac{}{\{\eta\}\ \textbf{abort}\ \{\Box\bot\}}\ [\textsc{Abort}] \qquad \frac{\eta' \triangleq \eta[\![x \leftarrow e]\!]}{\{\eta'\}\ x \leftarrow e\ \{\eta\}}\ [\textsc{Assgn}] \qquad \frac{}{\{\eta\}\ \textbf{skip}\ \{\eta\}}\ [\textsc{Skip}]$$

$$\frac{\eta' \triangleq \eta[\![x \xleftarrow{\$} g]\!]}{\{\eta'\}\ x \xleftarrow{\$} g\ \{\eta\}}\ [\textsc{Sample}] \qquad \frac{\{\eta_0\}\ s_1\ \{\eta_1\} \qquad \{\eta_1\}\ s_2\ \{\eta_2\}}{\{\eta_0\}\ s_1; s_2\ \{\eta_2\}}\ [\textsc{Seq}]$$

$$\frac{\{\eta_1 \wedge \Box e\}\ s_1\ \{\eta_1'\} \qquad \{\eta_2 \wedge \Box\neg e\}\ s_2\ \{\eta_2'\}}{\{(\eta_1 \wedge \Box e) \oplus (\eta_2 \wedge \Box\neg e)\}\ \textbf{if}\ e\ \textbf{then}\ s_1\ \textbf{else}\ s_2\ \{\eta_1' \oplus \eta_2'\}}\ [\textsc{Cond}]$$

$$\frac{\{\eta_1\}\ s\ \{\eta_1'\} \qquad \{\eta_2\}\ s\ \{\eta_2'\}}{\{\eta_1 \oplus \eta_2\}\ s\ \{\eta_1' \oplus \eta_2'\}}\ [\textsc{Split}]$$

How to reason about loops?

Well-known pitfall: naive rule unsound!

- Always have:

$$\{\mathbb{P}[\top] = 1\} \text{ skip } \{\mathbb{P}[\top] = 1\}$$

- But not:

$$\{\mathbb{P}[\top] = 1\} \text{ while true do skip } \{\mathbb{P}[\top] = 1\}$$

How to reason about loops?

Well-known pitfall: naive rule unsound!

- Always have:

$$\{\mathbb{P}[\top] = 1\} \text{ skip } \{\mathbb{P}[\top] = 1\}$$

- But not:

$$\{\mathbb{P}[\top] = 1\} \text{ while true do skip } \{\mathbb{P}[\top] = 1\}$$

Tradeoff

Generality of invariants/allowed termination behavior

Our solution: A family of loop rules

$$\frac{\{\Phi \wedge \Box b\}\ c\ \{\Phi\}}{\{\Phi\}\ \mathsf{while}\ b\ \mathsf{do}\ c\ \{\Phi \wedge \Box \neg b\}}$$

Our solution: A family of loop rules

$$\frac{\{\Phi \wedge \Box b\}\ c\ \{\Phi\}}{\{\Phi\}\ \textsf{while}\ b\ \textsf{do}\ c\ \{\Phi \wedge \Box\neg b\}}$$

Loop: Bounded number of iterations ("for-loops")

- Invariant Φ: arbitrary predicate

Our solution: A family of loop rules

$$\frac{\{\Phi \land \Box b\}\ c\ \{\Phi\}}{\{\Phi\}\ \text{while}\ b\ \text{do}\ c\ \{\Phi \land \Box \neg b\}}$$

Loop: Bounded number of iterations (“for-loops”)

- Invariant Φ: arbitrary predicate

Loop: Terminates with probability 1

- Invariant Φ: “topologically closed” (e.g., $\mathbb{P}[\phi] = 1/2$)

Our solution: A family of loop rules

$$\frac{\{\Phi \wedge \Box b\}\ c\ \{\Phi\}}{\{\Phi\}\ \text{while } b \text{ do } c\ \{\Phi \wedge \Box \neg b\}}$$

Loop: Bounded number of iterations (“for-loops”)

- Invariant Φ: arbitrary predicate

Loop: Terminates with probability 1

- Invariant Φ: “topologically closed” (e.g., $\mathbb{P}[\phi] = 1/2$)

Loop: Arbitrary termination

- Invariant Φ: “downwards closed” (e.g., $\mathbb{P}[\phi] < 1/2$)

Adding to the Toolbox:
Specialized Proof Techniques

Two common properties in paper proofs

Probabilistic independence

- In our assertions:

$$e \# e' \triangleq \forall a, b.\, \mathbb{P}[e = a \land e' = b] = \mathbb{P}[e = a] \cdot \mathbb{P}[e' = b]$$

Two common properties in paper proofs

Probabilistic independence

- In our assertions:

$$e \# e' \triangleq \forall a, b.\, \mathbb{P}[e = a \wedge e' = b] = \mathbb{P}[e = a] \cdot \mathbb{P}[e' = b]$$

Distribution laws

- In our assertions:

$$e \sim \textsf{Unif}(A) \triangleq \forall a \in A.\, \mathbb{P}[e = a] = 1/|A|$$

Reasoning about independence and distribution laws

Useful facts about independence

$$(e_1, e_2) \# e_3 \implies (e_1 \# e_3) \wedge (e_2 \# e_3)$$

Combining independence and uniformity

$$e \sim \mathsf{Unif}(A) \wedge e' \sim \mathsf{Unif}(A') \wedge (e \# e') \implies (e, e') \sim \mathsf{Unif}(A \times A')$$

Incorporating this reasoning in Ellora

Build a program logic IL around these assertions,
soundness by embedding into core program logic.

Other tools available in Ellora

Prior work: union bound logic [ICALP 2016]

- Designed for proving proeprties of the form $\mathbb{P}[\phi] < \beta$

Precondition calculus

- Similar to Morgan and McIver's weakest pre-expectations
- Defined on syntax of assertions

Implementation and Formalized Examples

Implementation

Part of the EasyCrypt system

- Tactic-based proofs, SMT support

Formalization of basic discrete probability theory

- Definitions: independence, basic distributions, ...
- Theorems: Markov inequality, Chernoff bound, ...

Examples: Nine verified algorithms

Name	Lines of Code	Lines of Proof
`hypercube`	100	1140
`coupon`	27	184
`vertex-cover`	30	61
`pairwise-indep`	30	231
`private-sums`	22	80
`poly-id-test`	22	32
`random-walk`	16	42
`dice-sampling`	10	64
`matrix-prod-test`	20	75

Examples: Nine verified algorithms

Name	Lines of Code	Lines of Proof
`hypercube`	100	1140
`coupon`	27	184
`vertex-cover`	30	61
`pairwise-indep`	30	231
`private-sums`	22	80
`poly-id-test`	22	32
`random-walk`	16	42
`dice-sampling`	10	64
`matrix-prod-test`	20	75

A classic example: Valiant's hypercube routing

Hypergraph network

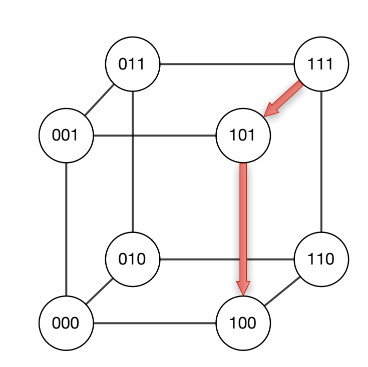
- Nodes: $\{0, 1\}^d$
- Given: permutation π
- Edge capacity 1
- Goal: route i to $\pi(i)$

A classic example: Valiant's hypercube routing

Hypergraph network

- Nodes: $\{0, 1\}^d$
- Given: permutation π
- Edge capacity 1
- Goal: route i to $\pi(i)$

Routing 111 to 100 $(d = 3)$



A classic example: Valiant's hypercube routing

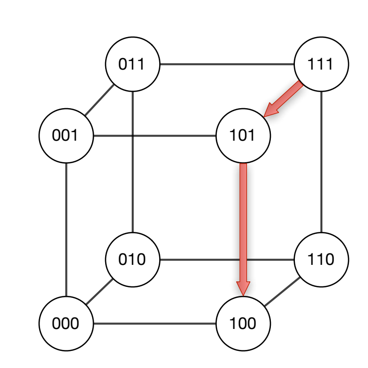
Hypergraph network

- Nodes: $\{0,1\}^d$
- Given: permutation π
- Edge capacity 1
- Goal: route i to $\pi(i)$

Valiant's routing plan

- Uniformly random $\rho(i)$
- Route: $i \mapsto \rho(i) \mapsto \pi(i)$

Routing 111 to 100 $(d = 3)$



A classic example: Valiant's hypercube routing

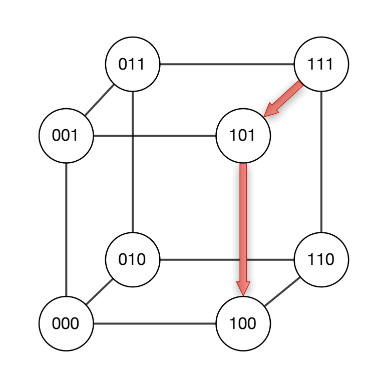
Hypergraph network

- Nodes: $\{0,1\}^d$
- Given: permutation π
- Edge capacity 1
- Goal: route i to $\pi(i)$

Valiant's routing plan

- Uniformly random $\rho(i)$
- Route: $i \mapsto \rho(i) \mapsto \pi(i)$

Routing 111 to 100 $(d = 3)$



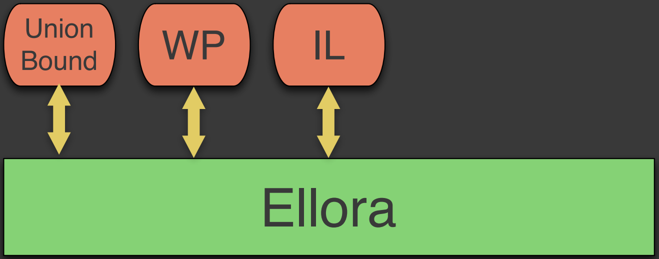
Show: with high probability, routes all 2^d packets in $O(d)$ steps

Future Directions and Open Design Questions

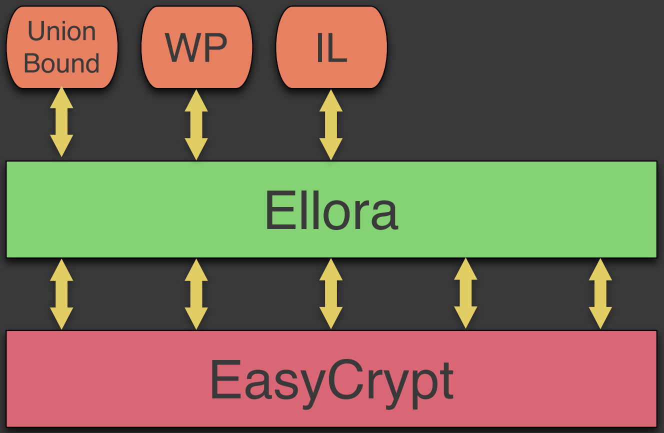
The story so far

Ellora

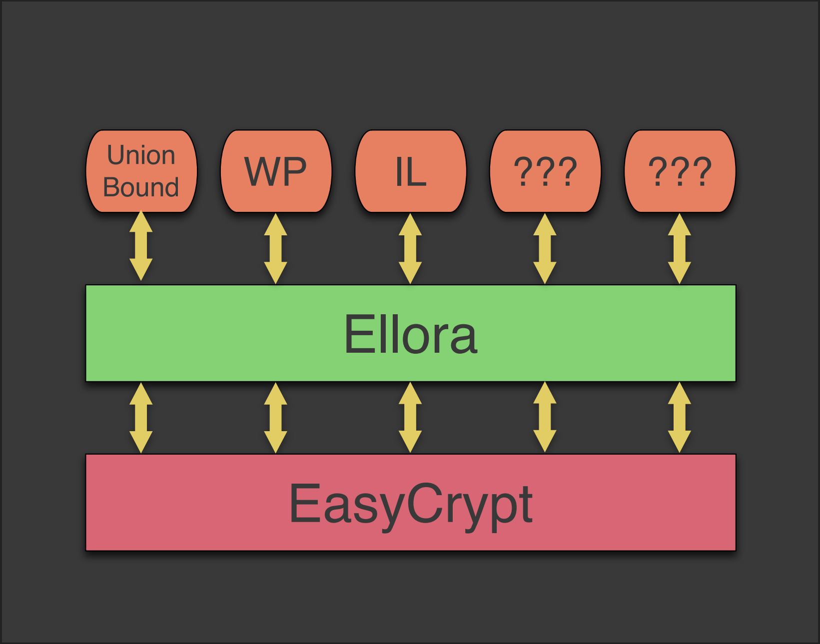
The story so far



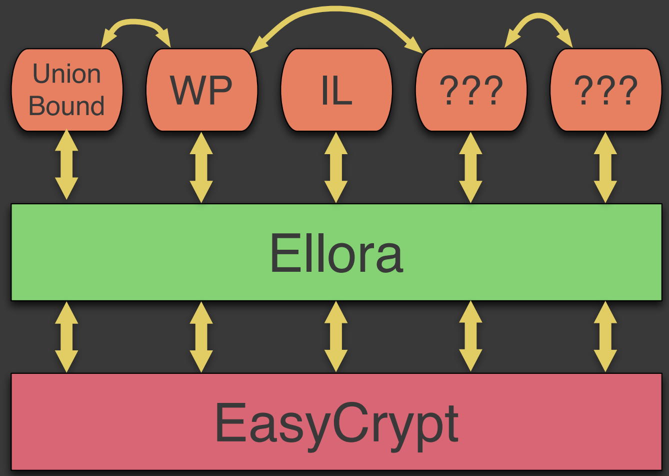
Next steps?



Next steps?



Next steps?



Open design questions

How to structure the assertion language?

- Need help managing large assertions and invariants

Open design questions

How to structure the assertion language?

- Need help managing large assertions and invariants

Deterministic inputs or distribution over inputs?

- Deterministic gives simpler but less flexible pre-conditions

Open design questions

How to structure the assertion language?

- Need help managing large assertions and invariants

Deterministic inputs or distribution over inputs?

- Deterministic gives simpler but less flexible pre-conditions

How to combine different proof techniques?

- Want to support many tools, but not all can be freely mixed

Open design questions

How to structure the assertion language?

- Need help managing large assertions and invariants

Deterministic inputs or distribution over inputs?

- Deterministic gives simpler but less flexible pre-conditions

How to combine different proof techniques?

- Want to support many tools, but not all can be freely mixed

Should reasoning be code-directed?

- Maybe easier: lift random sampling instructions out

An Assertion-Based Program Logic for Probabilistic Programs

Gilles Barthe, Thomas Espitau, Marco Gaboardi,
Benjamin Grégoire, Justin Hsu, and Pierre-Yves Strub