Reflection, Rewinding, and Coin-Toss in EasyCrypt

Denis Firsov[1,2] and Dominique Unruh[3]

[1]Guardtime
[2]Tallinn University of Technology
[3]Tartu University

January 17, 2022

Background: EasyCrypt

- EasyCrypt is a theorem prover for verifying cryptographic constructions, where protocols are specified as imperative programs and adversaries are modelled by abstract program modules.
- It has four built-in logics:
  - a probabilistic, relational Hoare logic (pRHL);
  - a probabilistic Hoare logic (pHL) ;
  - an ordinary (possibilistic) Hoare logic (HL);
  - an ambient higher-order logic for proving general mathematical facts and connecting judgments in the other logics.

Notation

We write

$$\Pr[\, r \leftarrow X.p() \,@\, \mathbf{m} : M \,], \text{ where}$$

- X is a program module.
- $p()$ is a procedure of module X.
- $\mathbf{m}$ is an initial memory configuration.
- r stores a result of running $X.p()$ on memory $\mathbf{m}$
- M is a predicate.

Challenging Proof I

Theorem

Let A be a probabilistic program and let $\boldsymbol{m}$ *denote a memory configuration which represents an initial state of A. In this case, the following inequality holds:*

$$\Pr\left[\begin{array}{c} A.init();\ s \leftarrow A.getState(); \\ r_1 \leftarrow A.main();\ A.setState(s); \\ r_2 \leftarrow A.main() \ @\ \boldsymbol{m} : r_1 \wedge r_2 \end{array}\right] \geq \Pr\left[A.init();\ r \leftarrow A.main() \ @\ \boldsymbol{m} : r\right]^2 .$$

Challenging Proof II

$$\Pr\left[\begin{array}{c} A.init();\ s \leftarrow A.getState(); \\ r_1 \leftarrow A.main();\ A.setState(s); \\ r_2 \leftarrow A.main()\ @\ \mathbf{m} : r_1 \wedge r_2 \end{array}\right]$$

$$\stackrel{(1)}{=} \sum_{\mathbf{n}} \mu_1(D_A^{\mathbf{m}}, \mathbf{n}) \cdot \Pr\left[\begin{array}{c} s \leftarrow A.getState(); \\ r_1 \leftarrow A.main(); A.setState(s); \\ r_2 \leftarrow A.main()\ @\ \mathbf{n} : r_1 \wedge r_2 \end{array}\right]$$

$$\stackrel{(2)}{=} \sum_{\mathbf{n}} \mu_1(D_A^{\mathbf{m}}, \mathbf{n}) \cdot \Pr[r \leftarrow A.main()\ @\ \mathbf{n} : r]^2$$

$$\stackrel{(3)}{\geq} \left(\sum_{\mathbf{n}} \mu_1(D_A^{\mathbf{m}}, \mathbf{n}) \cdot \Pr[r \leftarrow A.main()\ @\ \mathbf{n} : r]\right)^2$$

$$\stackrel{(4)}{=} \Pr[A.init();\ r \leftarrow A.main()\ @\ \mathbf{m} : r]^2 .$$

Challenges

- The proof turns the program $A.init()$ into a parameterized distribution of final memories (memories after $A.init()$ terminates).
- EasyCrypt does not have a type “memory”; we cannot define a distribution over memories because we cannot even assign it a type.
- The proof makes use of results about probability distributions (e.g., Jensen’s inequality) which we belive is hard (or even impossible) to prove directly using program logics (e.g., probabilistic Hoare logic).
- How one can generically specify the interface of rewindable programs (modules) which can return their own state.

Our Main Contributions

- We develop a method for reasoning about the probabilistic semantics of programs inside EasyCrypt (we call this probabilistic reflection).
- We design a set of tools to address rewindability in the EasyCrypt framework.
- We validate our results by developing a formal proof of a coin-toss protocol based on rewinding.

Probabilistic Reflection I

- There are no valid types which refer to distribution of final memories.
- However, in EasyCrypt each program has an associated variable. `(glob A){m}` (which we denote by writing $\mathcal{G}_A^{\mathbf{m}}$) which refers to the part of the memory **m** accessible by module A.
- So "effectively", the semantics of a program can be described by looking only at the $\mathcal{G}_A$-part of a memory.
- We define a family of distributions D_A^g which assigns a probability to every possible configuration of memory reachable by A given that execution starts in state g.

Probabilistic Reflection II

Theorem

For all memories $\mathbf{m}$ *and programs A there exists a family of distributions* D_A^g *(with g of type* $\mathcal{G}_A$*) such that for all predicates M on values of type* $\mathcal{G}_A$*:*

$$\Pr\left[\,A.main() \,@\, \boldsymbol{m} : M(\mathcal{G}_A^{\boldsymbol{fin}})\,\right] = \mu(D_A^{\mathcal{G}_A^{\boldsymbol{m}}}, M).$$

- A is a module.
- $A.main$ procedure of A.
- **m** and **fin** are initial and final memory configurations, respectively.
- $\mathcal{G}_A^{\mathbf{m}}$ and $\mathcal{G}_A^{\mathbf{fin}}$ values corresponding to initial and final state of A, respectively.
- D_A a family of distributions describing probabilistic semantics of $A.main$.
- $\mu(D_A^{\mathcal{G}_A^{\mathbf{m}}}, M)$ the total probability of final states reachable from initial state $\mathcal{G}_A^{\mathbf{m}}$ which satisfy M.

Probabilistic Reflection III

- However, being able to reflect the distribution corresponding to a given program is not enough.
- Given a program $A; B$, reflection gives us distributions D_{AB}, D_A, and D_B relating to the semantics of $(A; B)$, A, and B, respectively. However, we do not, a priori, know how D_{AB} is related to D_A and D_B.
- In our formalization, D_{AB} is shown to be the monadic bind of D_A and D_B.

Properties

- Finite probabilistic approximation:

$$\Pr\left[\, r \leftarrow A.run();\ @\,\mathbf{m} : (r, \mathcal{G}_A) \notin L(n) \right] \to 0.$$

- Averaging:

$$\Pr\left[\ x \stackrel{\$}{\leftarrow} d; r \leftarrow A.run(x);\ @\,\mathbf{m} : M(r)\ \right] = \sum_{x \in d} \mu_1(d, x) \cdot \Pr\left[\, r \leftarrow A.run(x);\ @\,\mathbf{m} : M(r) \right].$$

- Jensen's inequality: if X is a distribution, g maps elements of X to reals, and f is convex

$$f(E_X(g)) \leq E_X(f \circ g)$$

Rewinding

Definition

The module A is rewindable if

1. There exists an injective mapping f from the type $\mathcal{G}_A$ to some parameter type *sbits*.
2. The module A must have a terminating procedure *getState*, so that the execution of *A.getState()* in state **m** must return the value $f(\mathcal{G}_A^{\mathbf{m}})$ without changing the state.

$$\Pr\left[r \leftarrow A.getState() \ @ \ \mathbf{m} : \mathcal{G}_A^{\mathbf{fin}} = \mathcal{G}_A^{\mathbf{m}} \wedge r = f(\mathcal{G}_A^{\mathbf{m}})\right] = 1.$$

3. The module A must have a terminating procedure *setState*, so that whenever it gets an argument $x : sbits$ and sets $\mathcal{G}_A^{\mathbf{m}}$ to $f^{-1}(x)$ if $f^{-1}(x)$ is defined. Formally, let g be of type $\mathcal{G}_A$ then

$$\Pr\left[r \leftarrow A.setState(f\,g) \ @ \ \mathbf{m} : \mathcal{G}_A^{\mathbf{fin}} = g\right] = 1.$$

Validation Properties

- Commutativity:

$$\Pr\left[\begin{array}{c} s \leftarrow \mathit{A.getState()};\ r_1 \leftarrow A.run_1();\mathit{A.setState(s)} \\ r_2 \leftarrow A.run_2() @\ \mathbf{m} : M(r_1, r_2) \end{array}\right] = \Pr\left[\begin{array}{c} s \leftarrow \mathit{A.getState()};\ r_2 \leftarrow A.run_2();\mathit{A.setState(s)} \\ r_1 \leftarrow A.run_1() @\ \mathbf{m} : M(r_1, r_2) \end{array}\right].$$

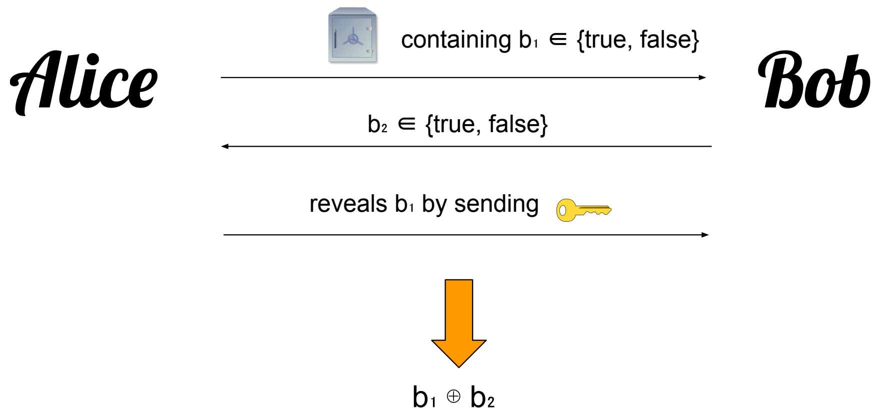
- Multiplication rule:

$$\Pr\left[\begin{array}{c} s \leftarrow \mathit{A.getState()};\ r_1 \leftarrow A.run_1();\mathit{A.setState(s)} \\ r_2 \leftarrow A.run_2() @\ \mathbf{m} : M_1(r_1) \wedge M_2(r_2) \end{array}\right] = \Pr[r \leftarrow A.run_1();\ @\ \mathbf{m} : M_1(r_1)] \cdot \Pr[r \leftarrow A.run_2();\ @\ \mathbf{m} : M_2(r_2)].$$

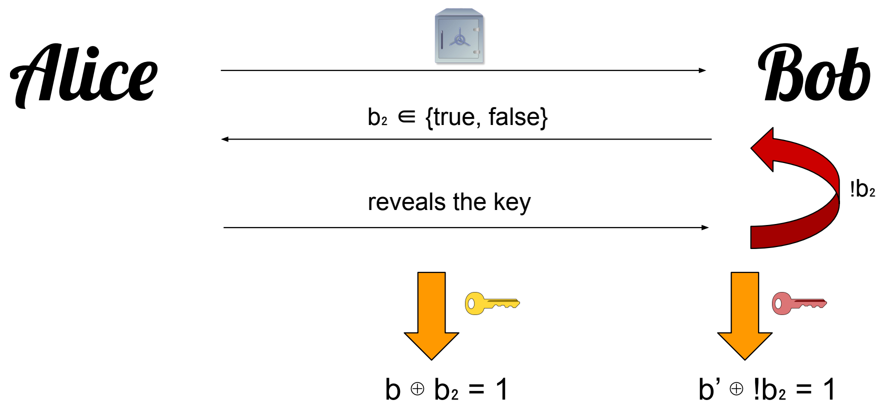
Coin-Toss Protocol (M. Blum, 1983)

Alice and Bob are recently divorced, living in two separate cities, and want to decide who gets to keep the car. To decide, Alice wants to flip a coin over the telephone. However, Bob is concerned that if he were to tell Alice the result of his coin toss, she would adjust hers and automatically tell him that she wins.

Coin-Toss Protocol I



Coin-Toss Protocol II



Thank you!