Lecture Notes: EasyCrypt in Cryptography and Mathematics

Author Name

January 24, 2025

1 Introduction to EasyCrypt

EasyCrypt is a formal verification tool primarily used to prove security properties of cryptographic protocols and mathematical theories. This tool enables users to express and reason about various types, operators, and proofs systematically, offering a higher degree of confidence in the correctness of their cryptographic systems.

The key features of EasyCrypt include its support for a wide range of mathematical structures, its powerful proof tactics, and its ability to handle both concrete and abstract data types.

In this lecture, we will cover:

- Types and operators in EasyCrypt.
- Proof techniques and tactics.
- Example proofs.
- Exercises to reinforce learning.

2 Basic Types and Operators in EasyCrypt

2.1 Types in EasyCrypt

EasyCrypt provides a variety of basic types for constructing mathematical expressions. These include:

- `unit`: A type that has a single element, denoted as `()`.
- `int`: The integer type.
- `bool`: The boolean type.
- `real`: The real number type.

- `t1 * t2 * ... * tn`: Product types, allowing the grouping of multiple types together.
- `t1 -> t2`: Function types, which represent functions that take an argument of type `t1` and return a result of type `t2`.

Operator precedence is crucial:

- `*` has higher precedence than `->`.
- `->` is right-associative.

Thus, `t1 * t2 -> t3 -> t4` is interpreted as `(t1 * t2) -> (t3 -> t4)`.

2.2 Operators in EasyCrypt

EasyCrypt allows defining operators (functions) with specific types. Here's how operators are defined and used:

```
op f (x y : int) = if 0 < x then x - 2 * y else 1.
op g (a b : bool) = !(a /\ b) /\ (a \/ b).
```

These operators are functions that perform conditional evaluations, logical negation, and conjunction/disjunction.

For example:

- The operator `f` is defined to subtract $2 \times y$ from x when $x > 0$, and return 1 otherwise.
- The operator `g` performs logical negation on a conjunction, and then applies disjunction.

2.3 Operator Usage Example

Consider the following example using a defined operator `f`:

```
op p : int -> int = f 4.
op y : int = p 5.
```

Here, `p` is the function obtained by applying `f` to 4, and `y` is the result of applying `p` to 5. The value of `y` will be calculated as:

$$y = f(4)(5) = (4 - 2 \times 5) = -6.$$

3 Proof Techniques in EasyCrypt

3.1 Theorem and Lemma Definitions

In EasyCrypt, you can define axioms and prove theorems (lemmas). For instance, consider the following axiom:

```
axiom h_ax (x : int) : x <> 0 => h x.
```

This axiom states that for any non-zero integer x, applying h to x yields a true result.

You can also define and prove lemmas, such as:

```
lemma not_or (a b : bool) : !(a \/ b) => !a /\ !b.
proof.
...
qed.
```

This lemma proves that the negation of the disjunction of a and b implies the conjunction of their negations.

3.2 Basic Proof Tactics

EasyCrypt supports various proof tactics that help simplify and break down the goals into smaller subgoals. Here are some essential tactics:

3.2.1 The `split` Tactic

The `split` tactic is used to break down conjunctions. For example, if we have a goal like $a \wedge b$, we can split it into two subgoals: a and b.

Example:

```
lemma not_or (a b : bool) : !(a \/ b) => !a /\ !b.
proof.
split.
- admit.
- admit.
qed.
```

In this case, the `split` tactic divides the goal into two subgoals. The `admit` tactic is used temporarily to accept the subgoals without providing a proof.

3.2.2 The `simplify` and `trivial` Tactics

The `simplify` tactic is used to reduce logical formulas to simpler forms. The `trivial` tactic applies basic logical rules to solve goals.

Example:

```
simplify.
```

This simplifies expressions like $\neg \text{true} \vee (x < y)$ to just $x < y$.

3.2.3 The `admit` Tactic

The `admit` tactic is used when you want to accept a goal without proving it immediately. It's useful when developing proofs incrementally. However, using `admit` makes the proof incomplete.

Example:

```
admit.
```

3.3 Applying Axioms and Lemmas

To apply an already-proven axiom or lemma, use the `apply` tactic. For example, applying the lemma `not_or_imp` proves the goal:

```
apply not_or_imp.
```

This will apply the lemma, reducing the goal.

4 Advanced Proof Techniques

4.1 Using Induction

Induction is a crucial technique in mathematical proofs, especially when dealing with recursive or structural definitions. EasyCrypt provides induction principles for types like integers and trees.

Example: For proving properties over integers:

```
elim /intind.
```

This tactic applies mathematical induction to the integer type.

4.2 Rewriting Equations

EasyCrypt supports rewriting equations to simplify expressions:

```
rewrite (f_eq x).
```

This rewrites the goal using the equation $f(x) = x + 1$.

4.3 Abstract and Concrete Data Types

EasyCrypt allows defining both abstract and concrete data types. For example, a concrete datatype for binary trees might be defined as:

```
type ('a, 'b) tree = [
| Leaf of 'a
| Node of 'b * ('a, 'b) tree * ('a, 'b) tree
].
```

You can then define recursive operators and prove properties about these data types using induction.

5 Exercises

5.1 Exercise 1: Prove the Negation of a Disjunction

Prove the lemma:

$$\neg(a \lor b) \implies (\neg a \land \neg b)$$

using the `split` and `simplify` tactics.

5.2 Exercise 2: Define a Recursive Function and Prove Its Properties

Define a recursive function to compute the size of a binary tree and prove that the size is always greater than or equal to 0.

5.3 Exercise 3: Apply a Previously Proved Lemma

Use the lemma `not_or_imp` to simplify the following goal:

$$\neg(a \lor b) \implies \neg a \land \neg b$$

6 Conclusion

In this lecture, we have explored how to use EasyCrypt to define types, operators, and tactics for proving cryptographic and mathematical properties. The system provides a powerful and flexible way to model and verify cryptographic protocols.