CS 591 G1—Formal Methods in Security and Privacy—Spring 2021

Assignment 1

Due by Friday, February 12, at 5pm
Submission Via Gradescope

Fill in the five gaps in the following EasyCrypt file, `Assignment1.ec`, which is available on the course website. Make sure EasyCrypt is able to check your proofs.

```
(* ASSIGNMENT 1

   Due on Gradescope by 5pm on Friday, February 12 *)

require import AllCore.

(* QUESTION 1

   prove the following lemma without using the tactics ‘case‘ or
   ‘smt‘, and without using any lemmas from the EasyCrypt Library: *)

lemma sub (a b c : bool) :
  (a => b => c) => (a => b) => a => c.
proof.
(* BEGIN FILL IN *)

(* END FILL IN *)
qed.

(* QUESTION 2

   prove the following lemma without using the tactics ‘case‘ or
   ‘smt‘, and without using any lemmas from the EasyCrypt Library: *)

lemma peirce (a b : bool) :
  (a \/ !a) => ((a => b) => a) => a.
proof.
(* BEGIN FILL IN *)

(* END FILL IN *)
qed.

(* QUESTION 3
```

```
   prove the following lemma without using the `smt` tactic, and
   without using any lemmas from the EasyCrypt Library (you may use
   the `case` tactic): *)

lemma not_exists (P : 'a -> bool) :
  (! exists (x : 'a), P x) <=> (forall (x : 'a), ! (P x)).
proof.
(* BEGIN FILL IN *)

(* END FILL IN *)
qed.

(* QUESTION 4

   This question is about proving the equivalence of two definitions
   of when an integer is prime. *)

require import IntDiv StdOrder. (* lemmas for integer mod and div *)
import IntOrder.                (* lemmas about <, <= on int *)

(* n %/ x is the integer division of n by x, discarding any remainder

   n %% x is the remainder of integer division of n by x

   x %| n tests whether x divides n, i.e., n %% x = 0

   %/ and %% are actually abbreviations, defined in terms of edivz;
   consequently, when using `search` to look for lemmas involving
   these abbreviations, one must search for `edivz` instead. But we've
   provided (below) the lemma that needs such facts *)

(* here are two ways of defining when an integer is prime, which
   you will prove are equivalent: *)

op is_prime1 (n : int) : bool =
  2 <= n /\
  ! exists (x : int),
    x %| n /\ 1 < x /\ x < n.

op is_prime2 (n : int) : bool =
   2 <= n /\
   forall (x : int),
   x %| n => x <= 1 \/ x = n.
```

```
(* you may use the following lemma in your proof (it should probably
   be provided by IntDiv): *)

lemma div_le (x n : int) :
  1 <= x => 1 <= n => x %| n => x <= n.
proof.
move => ge1_x ge1_n x_div_n.
have n_eq : n = (n %/ x) * x.
  by rewrite eq_sym -dvdz_eq.
rewrite n_eq -{1}mul1z ler_pmul // (lez_trans 1) //.
case (1 <= n %/ x) => //.
rewrite -ltrNge => lt1_n_div_x.
have le0_n : n <= 0.
  by rewrite n_eq mulr_le0_ge0 1:-ltzS // (lez_trans 1).
have // : 1 <= 0.
  by apply (lez_trans n).
qed.

(* When proving the following lemma, this lemma from the EasyCrypt
   Library will be helpful:

lemma forall_iff (P P' : 'a -> bool) :
 (forall x, P x <=> P' x) =>
 (forall (x : 'a), P x) <=> (forall (x : 'a), P' x).
*)

(* prove the following lemma without using the `smt` tactic; you
   may use the `case` tactic as well as any lemmas from the EasyCrypt
   Library

   hint: you can use your solution to QUESTION 3, and you can use
   `search` to look for needed lemmas in the EasyCrypt Library. E.g.,

     search [!] (/\)

   searches for lemmas involving negation and conjunction

   for *partial credit*, you can use `smt` or even `admit` for parts
   of your proof *)

lemma prime_equiv_ge2 (n : int) :
  2 <= n =>
  (! (exists (x : int), x %| n /\ 1 < x /\ x < n) <=>
```

```
   (forall (x : int), x %| n => x <= 1 \/ x = n)).
proof.
(* BEGIN FILL IN *)

(* END FILL IN *)
qed.

(* use prime_equiv_ge2 (but not the `smt` tactic) to prove the
   following lemma asserting the equivalence of the two definitions of
   primality (you won't need `case` or lemmas from the EasyCrypt
   Library, but you may use them) *)

lemma prime_equiv (n : int) :
 is_prime1 n <=> is_prime2 n.
proof.
(* BEGIN FILL IN *)

(* END FILL IN *)
qed.
```