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

Assignment 5

Due by Friday, March 19, at 5pm
Submission Via Gradescope

Fill in the two gaps in the following EasyCrypt file, `Assignment5.ec`, which is available on the course website. Make sure EasyCrypt is able to check your proofs.

```
(* ASSIGNMENT 5

   Due on Gradescope by 5pm on Friday, March 19 *)

prover quorum=2 ["Alt-Ergo" "Z3"].

timeout 2.  (* smt timeout in seconds - can increase *)

require import AllCore Distr.

(* a type mybool with elements tt standing for "true" and ff standing
   for "false", and an exclusive or operator ^^: *)

type mybool = [tt | ff].

(* because tt is an alias for the value () : unit, you'll sometimes
   see Top.tt in goals for the value tt of mybool *)

op (^^) : mybool -> mybool -> mybool.

(* we can write x ^^ y instead of (^^) x y

   ^^ is left associatve, so x ^^ y ^^ z means (x ^^ y) ^^ z *)

(* axioms defining exclusive or (nosmt means these axioms aren't
   available to the SMT provers): *)

axiom nosmt xor_tt_tt : tt ^^ tt = ff.
axiom nosmt xor_tt_ff : tt ^^ ff = tt.
axiom nosmt xor_ff_tt : ff ^^ tt = tt.
axiom nosmt xor_ff_ff : ff ^^ ff = ff.

(* lemmas for exclusive or: *)
```

```
(* false on the right *)
lemma xor_ff (x : mybool) : x ^^ ff = x.
proof.
case x; [apply xor_tt_ff | apply xor_ff_ff].
qed.

(* cancelling *)
lemma xorK (x : mybool) : x ^^ x = ff.
proof.
case x; [apply xor_tt_tt | apply xor_ff_ff].
qed.

(* commutativity *)
lemma xorC (x y : mybool) : x ^^ y = y ^^ x.
case x.
case y => //.
by rewrite xor_tt_ff xor_ff_tt.
case y => //.
by rewrite xor_ff_tt xor_tt_ff.
qed.

(* associativity *)
lemma xorA (x y z : mybool) : (x ^^ y) ^^ z = x ^^ (y ^^ z).
proof.
case x.
case y.
rewrite xor_tt_tt.
case z.
by rewrite xor_ff_tt xor_tt_tt xor_tt_ff.
by rewrite xor_ff_ff xor_tt_ff xor_tt_tt.
case z.
by rewrite xor_tt_ff xor_tt_tt xor_ff_tt xor_tt_tt.
by rewrite xor_tt_ff xor_tt_ff xor_ff_ff xor_tt_ff.
case y.
rewrite xor_ff_tt.
case z.
by rewrite xor_tt_tt xor_ff_ff.
by rewrite xor_tt_ff xor_ff_tt.
case z.
by rewrite xor_ff_ff xor_ff_tt xor_ff_tt.
by rewrite xor_ff_ff xor_ff_ff.
qed.
```

```
(* a sub-distribution dmybool on mybool - this means that the sum of
   the values of tt and ff in dmybool may be less than 1 *)

op dmybool : mybool distr.

(* dmybool is a distribution, i.e., the sum of the values of tt and ff
   in dmybool is 1%r *)

axiom dmybool_ll : is_lossless dmybool.

(* if d is a sub-distribution on type 'a, and E is an event
   (predicate) on 'a (E : 'a -> bool), then mu d E is the probability
   that choosing a value from d will satisfy E

   if d is a sub-distribution on type 'a, and x : 'a, then mu1 d x is
   the probablity that choosing a value from d will result in x, i.e.,
   is the value of x in d

   mu1 is defined by: mu1 d x = mu d (pred1 x), where pred1 x is the
   predicate that returns true iff its argument is x *)

(* both tt and ff have value one-half in dmybool: *)

axiom dmybool1E (b : mybool) :
  mu1 dmybool b = 1%r / 2%r.

(* then we can prove that dmybool is full, i.e., that its support is
   all of mybool, i.e., that both tt and ff have non-zero values in
   dmybool: *)

lemma dmybool_fu : is_full dmybool.
proof.
rewrite /is_full => x.
rewrite /support dmybool1E StdOrder.RealOrder.divr_gt0 //.
qed.

(* QUESTION 1 (40 Points) *)

module M1 = {
  var x, y : mybool  (* private *)
  var z : mybool     (* public *)

  proc f() : unit = {
    var b : mybool;
```

```
    b <$ dmybool;
    z <- x ^^ y ^^ b;
  }
}.

lemma lem1 :
  equiv[M1.f ~ M1.f : ={M1.z} ==> ={M1.z}].
proof.
(* BEGIN FILL IN *)

(* END FILL IN *)
qed.

(* QUESTION 2 (60 Points) *)

module M2 = {
  var x, y : mybool  (* private *)
  var z    : mybool  (* public *)

  proc f() : unit = {
    var u, v : mybool;
    if (x = y) {
      u <$ dmybool;
      z <- u ^^ x;
    }
    else {
      u <$ dmybool;
      v <$ dmybool;
      z <- u ^^ v ^^ x ^^ y;
    }
  }
}.

lemma lem2 :
  equiv[M2.f ~ M2.f : ={M2.z} ==> ={M2.z}].
proof.
(* BEGIN FILL IN *)

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