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

Assignment 4

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

Fill in the three gaps in the following EasyCrypt file, `Assignment4.ec`, which is available on the course website. Make sure EasyCrypt is able to check your proofs.

```
(* ASSIGNMENT 4

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

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

require import AllCore List.

(* QUESTION 1 (33 Points) *)

module M1 = {
  var x, y, i : int   (* public *)
  var bs : bool list  (* private *)

  proc f() : unit = {
    while (x <> y) {
      if (x < y) {
        x <- x + 1;
        i <- 0;
        while (i < x) {
          bs <- rcons bs true;
          i <- i + 1;
        }
      }
      else {
        y <- y + 1;
        i <- 0;
        while (i < y) {
          bs <- rcons bs false;
          i <- i + 1;
        }
      }
    }
```

```
  }
}.

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

(* END FILL IN *)
qed.

(* QUESTION 2 (33 Points) *)

module M2 = {
  var x, i, j : int  (* public *)
  var y, z : int     (* private *)

  proc f() : unit = {
    while (0 <= i) {
      if (i = y) {
        j <- 0;
        while (j < 10) {
          x <- x * i * y;
          j <- j + 1;
        }
      }
      else {
        if (i = z) {
          j <- 0;
          while (j < 10) {
            x <- x * i * z;
            j <- j + 1;
          }
        }
        else {
          j <- 0;
          while (j < 10) {
            x <- x * i;
            j <- j + 1;
          }
        }
      }
      i <- i - 1;
    }
```

```
  }
}.

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

(* END FILL IN *)
qed.

(* QUESTION 3 (34 Points) *)

(* require but don't import IntDiv, as we want to stop smt from
   directly using its operators (to avoid being confused): *)

require IntDiv.

(* we make smt treat the following three operators as black boxes

   in goals, EasyCrypt will print

     (odd i)%top

   instead of just

     odd i

   because there is also an `odd` in the EasyCrypt Library *)

op nosmt even (k : int) : bool = IntDiv.(%%) k 2 = 0.  (* test if even *)
op nosmt odd  (k : int) : bool = ! even k.             (* test if odd *)

(* integer division - we can still write x %/ y *)

op nosmt (%/) (x y : int) : int = IntDiv.(%/) x 2.

lemma even0 : even 0.
proof. rewrite /even /#. qed.

lemma odd_plus1 (k : int) : odd (k + 1) <=> even k.
proof. rewrite /odd /even /#. qed.

lemma even_plus1 (k : int) : even (k + 1) <=> odd k.
```

```
proof. rewrite /odd /even /#. qed.

lemma even_minus1 (k : int) : even (k - 1) = odd k.
proof. rewrite /odd /even /#. qed.

lemma odd_minus1 (k : int) : odd (k - 1) = even k.
proof. rewrite /odd /even /#. qed.

lemma not_odd (k : int) : ! odd k <=> even k.
proof. by rewrite /odd. qed.

lemma not_even (k : int) : ! even k <=> odd k.
proof. by rewrite /odd. qed.

lemma zero_div2 : 0 %/ 2 = 0.
proof. rewrite /(%/) /#. qed.

lemma even_plus1_div2 (k : int) :
  even k => (k + 1) %/ 2 = k %/ 2.
proof. rewrite /even /(%/) /#. qed.

lemma odd_plus1_div2 (k : int) :
  odd k => (k + 1) %/ 2 = k %/ 2 + 1.
proof. rewrite /odd /even /(%/) /#. qed.

lemma even_minus1_div2 (k : int) :
  even k => (k - 1) %/ 2 = k %/ 2 - 1.
proof. rewrite /even /(%/) /#. qed.

lemma odd_minus1_div2 (k : int) :
  odd k => (k - 1) %/ 2 = k %/ 2.
proof. rewrite /odd /even /(%/) /#. qed.

module M3 = {
  proc f(n  : int,  (* public *)
         x  : int,  (* public *)
         y  : int,  (* private *)
         z  : int)  (* private *)
         : int = {  (* public result *)
    var i : int;    (* adversary can't observe *)
    var w : int;    (* final value is returned, so made public *)
    (* note that y and z aren't changed, below *)
    i <- 0;
    w <- x;
```

```
    if (0 <= n) {
      while (i < n) {
        if (odd i) {
          w <- w + x + y;
        }
        else {
          w <- w + x + z;
        }
        i <- i + 1;
      }
      while (0 < i) {
        if (odd i) {
          w <- w - z;
        }
        else {
          w <- w - y;
        }
        i <- i - 1;
      }
    }
    return w;
  }
}.

lemma lem3 :
  equiv [M3.f ~ M3.f : ={n, x} ==> ={res}].
proof.
(* BEGIN FILL IN *)

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