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

Assignment 6

Due by Friday, April 16, at 5pm
Submission Via Gradescope

Fill in the the gaps in the following EasyCrypt file, `Assignment6.ec`, which is available on the course website. Replace the `fill_in` constant by the corresponding value of ϵ for each exercise and complete the proof. Make sure EasyCrypt is able to check your proofs.

```
(* ASSIGNMENT 6

Due on Gradescope by 5pm on Friday, April 16 *)


require import AllCore Ring.
require import Distr List Aprhl StdRing StdOrder StdBigop.
  (*---*) import IntID IntOrder RField RealOrder.

op eps: { real | 0%r <= eps } as ge0_eps.
op del: { real | 0%r <= del } as ge0_delta.

op gs: { int | 0 <= gs } as ge0_gs.

op fill_in : { real | fill_in = 0%r } as fill_in0.

hint exact : ge0_eps.
hint exact : ge0_delta.
hint exact : ge0_gs.
hint exact : fill_in0.
(* used by trivial only if exact *)

(*
   EX 1

  Please, replace the fill_in with good values for epsilon and delta
  and complete the proof.
*)

module M1 = {
   proc f (x:int): int = {
     var r;
```

```
     x<- x * 2;
     r <$ lap eps x;
    return r;
  }
}.

lemma ex1 : aequiv [ [fill_in & 0%r]
  M1.f ~ M1.f
 :  (`|x{1} - x{2}|<= 2)
==> res{2} = res{1} ].
proof.
admit.
qed.



(*
   EX 2

  Please, replace the fill_in with good values for epsilon and delta
  and complete the proof.
*)

module M2 = {
   proc f (x:int): int = {
     var r;
     x<- x ^ 2;
     r <$ lap eps x;
    return r;
  }
}.



lemma ex2 : aequiv [ [fill_in &  0%r]
  M2.f ~ M2.f
 :  x{1} = x{2}
==> res{2} = res{1} ].
proof.
admit.
qed.


(*
```

```
   EX 3

  Please, replace the fill_in with good values for epsilon and delta
  and complete the proof.
*)

module M3 = {
  proc f (d:int list , q1 q2 :int list -> int): int = {
      var r;
      var s;
      r <$ lap (eps) (q1 d);
      s <$ lap (eps) (q2 d);
  return r;
  }
}.

pred adj (ms ns : int list) =
  size ms = size ns /\
  (exists (i : int), forall (j : int),
  !(i=j)=> nth 0 ms j = nth 0 ns j).



lemma ex3 : aequiv [ [fill_in & 0%r]
  M3.f ~ M3.f
    :  (adj d{1} d{2} /\ ={q1} /\  ={q2} /\
        forall d d', (adj d d') => (`|(q1{1} d)-(q1{2} d')| <= 3) /\
        forall d d', (adj d d') => (`|(q2{1} d)-(q2{2} d')| <= 3)
       )
==> res{2} = res{1} ].
proof.
  admit.
qed.


(*
   EX 4

  Please, replace the fill_in with good values for epsilon and delta
  and complete the proof.
*)

module M4 = {
  proc f (d:int list , q :int list -> int): int = {
```

```
      var r;
      var s;
      r <$ lap (eps/gs%r) (q d);
      s <$ lap eps r;
  return r;
  }
}.


lemma ex4 : aequiv [ [fill_in & 0%r]
  M4.f ~ M4.f
    :  (adj d{1} d{2} /\ ={q} /\
        forall d d', (adj d d') => (`|(q{1} d)-(q{2} d')| <= gs)
       )
==> res{2} = res{1} ].
proof.
admit.
qed.

(*
   Auxiliary definitions and lemmas
   The definition of Adjacent_e has changed wrt the
   one we used in lab. Please look at it carefully.
*)

pred eq_in_range (ms ns : int list, i j : int) =
  forall (k : int),
  i <= k <= j => nth 0 ms k = nth 0 ns k.

lemma eq_in_range_succ (ms ns : int list, i j : int) :
  eq_in_range ms ns i j => eq_in_range ms ns (i + 1) j.
proof.
move => eir_i.
rewrite /eq_in_range => k le_iplus1_k_j.
rewrite eir_i /#.
qed.


pred adjacent_e (ms ns : int list) (i:int)=
  size ms = size ns /\
   0 <= i < size ms /\
   eq_in_range ms ns 0 (i - 1) /\
   nth 0 ms i <> nth 0 ns i /\  (* this is different from the one from lab*)
   eq_in_range ms ns (i + 1) (size ms - 1).
```

```
pred adjacent (ms ns : int list) = exists i, adjacent_e ms ns i.

lemma size_eq_adjacent (ms ns : int list) :
  adjacent ms ns => size ms = size ns.
proof.
rewrite /adjacent. rewrite /adjacent_e => [#].
smt().
qed.

lemma adjacent_sub_abs_bound (ms ns : int list, i : int) :
  adjacent_e ms ns i => 0 <= i < size ms =>
  nth 0 ms i <> nth 0 ns i.
proof.
  rewrite /adjacent_e. smt().
qed.

lemma adjacent_ne_sub_eq_after (ms ns : int list, i : int) :
  adjacent_e ms ns i => 0 <= i < size ms => nth 0 ms i <> nth 0 ns i =>
  eq_in_range ms ns (i + 1) (size ms - 1).
proof.
rewrite /adjacent_e. smt().
qed.

lemma bigops_help : forall n r,
0<=n => r * n%r = bigi predT (fun (_ : int) => r) 0 n.
proof.
move => n r H. rewrite sumr_const count_predT size_range ler_maxr.
     smt().
  rewrite intmulr; auto.
qed.

(*
  End auxiliary definitions and lemmas
*)

(*
  Ex 5

  Please, replace the fill_in with good values for epsilon and delta
  and complete the proof.
*)


module M5 = {
```

```
  proc f (ls : int list, k : int) : int = {
    var s : int <- 0;
    var z : int;
    var i : int <- 0;
    while (i < size ls ) {
      if (nth 0 ls i <= k) {
        s <- s + 1;
      }
      i <- i + 1;
    }
    z <$ lap (2%r*eps) s;
    return z;
  }
}.


lemma ex5 :
  aequiv
  [[ fill_in & 0%r]
   M5.f ~ M5.f : adjacent ls{1} ls{2} /\ ={k} ==> res{1} = res{2}].
proof.
  proc.
  (*
   Hint: You may want to take inspiration from the lem6 for the
   example sum we looked at in lab. Especially, the invariant
   you need is almost identical to the one from lab.
  *)
  admit.
qed.



(*
  Ex  6

  Please, replace all the fill_in with good values for epsilon and delta
  and complete the proof.
*)


module M6 = {
  proc f (ls : int list, k:int): int list = {
    var s :int <- 0;
    var output : int list;
```

```
    var z : int;
    var i :int <- 0;
    output <- [];
    while (i < size ls ) {
          if (nth 0 ls i <= k) {
             s <- s + 2;
      }
      i <- i + 1;
      z <$ lap eps s;
      output <- z :: output;
    }
    return output;
  }
}.


lemma ex6 n : 0<=n => aequiv [ [ fill_in & 0%r]
  M6.f ~ M6.f
 :  (adjacent ls{1} ls{2} /\  ={k} /\ n = size ls{1})
      ==> res{2} = res{1} ].
    proof.
    move => H.
    proc.
    seq 3 3: (adjacent ls{1} ls{2} /\ ={i, s, k, output} /\ i{1} = 0
              /\ s{1} = 0  /\ 0<= n /\ n = size ls{1} ).
      toequiv; auto.
    awhile  [ (fun _ => fill_in) & (fun _ => 0%r) ] n [n-i-1]
    (adjacent ls{1} ls{2} /\ ={i, output, k} /\ 0 <= i{1} <= n /\
     (! ={s} => `|s{1} - s{2}| <= 2 /\
     eq_in_range ls{1} ls{2} i{1} (n - 1)) /\
     0<= n /\ n = size ls{1}).
       auto; smt(ge0_eps fill_in0).
       auto; smt().
       auto; smt().
     (* the next step will not go through if you don't remove the all the
        fill_in above or if you fill them in with terms of the wrong shape
     *)
     apply bigops_help. trivial.
     rewrite /sumr sumr_const intmulr;auto.
     move => v.
     seq 2 2:
     (adjacent ls{1} ls{2} /\ ={i, output, k} /\ 0 <= i{1} <= size ls{1} /\
     (! ={s} => `|s{1} - s{2}| <= 2 /\
     eq_in_range ls{1} ls{2} i{1} (size ls{1} - 1)) /\
```

```
     0<= n /\ n = size ls{1} /\ v=n-i{1}).
     (* complete the proof *)
     admit.
qed.
```