```
(***************************************************************************)
(*                     Arithmétique et cryptographie                        *)
(***************************************************************************)

#open "big_int";;
let pprinter b = format__print_string (string_of_big_int b);;
install_printer "pprinter";;


(***************************************************************************)
(*                          Algorithme  d'Euclide                           *)
(***************************************************************************)

(* euclide *)

let rec euclide u v =
  if eq_big_int zero_big_int v then
    u
  else
    let m = mod_big_int u v in
    euclide v m
;;

(* bezout *)

let bezout u v =
  let vu = [| unit_big_int ; zero_big_int ; u |]
  and vv = [| zero_big_int ; unit_big_int ; v |] in
  while not (eq_big_int vv.(2) zero_big_int) do
    let q = div_big_int vu.(2) vv.(2) in
    for i = 0 to 2 do
      let t = sub_big_int vu.(i) (mult_big_int q vv.(i)) in
      vu.(i) <- vv.(i) ;
      vv.(i) <- t
    done
  done ;
  (vu.(0) , vu.(1) , vu.(2))
;;

(* division modulo m *)

let div_mod u v m =
  let (x,y,_) = bezout v m in
  mod_big_int (mult_big_int u x) m
;;


(***************************************************************************)
(*                   Décomposition en facteurs premiers                     *)
(***************************************************************************)

(* méthode par divisions *)

let rec decomp n = function
    []                   -> []
  | (dk::reste) as dkl ->
      if eq_big_int unit_big_int n then
        []
      else
        let q,r = quomod_big_int n dk in
        if eq_big_int zero_big_int r then
          dk :: (decomp q dkl)
        else if gt_big_int q dk then
```



```
          decomp n reste
        else
          [n]
;;

let dk n =
  let rec dk_rec inc head tail =
    if ge_big_int (square_big_int head) n then
      head::tail
    else
      let suiv = add_big_int (big_int_of_int inc) head in
      dk_rec (if inc=2 then 4 else 2) suiv (head::tail)
  in
  let tail = (big_int_of_int 3)::(big_int_of_int 2)::[] in
  rev (dk_rec 2 (big_int_of_int 5) tail)
;;


(* méthode de Fermat *)

let deux = big_int_of_int 2;;


(* On définit la racine carrée sur les grands entiers, avec une méthode
   de dichotomie. *)

let sqrt n =
  let rec dicho a b =
    if le_big_int (sub_big_int b a) unit_big_int then
      a
    else
      let m = div_big_int (add_big_int a b) deux in
      let fm = sub_big_int (square_big_int m) n in
      if eq_big_int fm zero_big_int then
        m
      else if lt_big_int fm zero_big_int then
        dicho m b
      else
        dicho a m
  in
  if lt_big_int n zero_big_int then
    raise (Invalid_argument "sqrt")
  else if eq_big_int n zero_big_int then
    zero_big_int
  else if eq_big_int n unit_big_int then
    unit_big_int
  else
    dicho zero_big_int n
;;

(* méthode de fermat *)

let fermat n =
  let rec fermat_rec x' y' r =
    if eq_big_int zero_big_int r then
      let u = div_big_int (sub_big_int x' y') deux
      and v = div_big_int (sub_big_int (add_big_int x' y') deux) deux in
      u,v
    else if lt_big_int r zero_big_int then
      fermat_rec (add_big_int x' deux) y' (add_big_int r x')
    else
      fermat_rec x' (add_big_int y' deux) (sub_big_int r y')
  in
```

```
  let sqrtn = sqrt n in
  let x' = add_big_int (mult_big_int deux sqrtn) unit_big_int
  and y' = unit_big_int
  and r = sub_big_int (square_big_int sqrtn) n in
  fermat_rec x' y' r
;;

(* et la décomposition qui en découle *)

let decomp2 n =
  let rec decomp_rec l n' =
    let u,v = fermat n' in
    if eq_big_int u unit_big_int then
      v::l
    else
      let lu = decomp_rec l u in
      decomp_rec lu v
  in
  let rec facteur2 l n =
    let q,r = quomod_big_int n deux in
    if eq_big_int r zero_big_int then
      facteur2 (deux::l) q
    else
      l,n
  in
  let l2,n' = facteur2 [] n in
  decomp_rec l2 n'
;;


(***************************************************************************)
(*                                   RSA                                   *)
(***************************************************************************)

(* On a la clé publique : *)

let N = big_int_of_string "49808911";;
let p = big_int_of_string "5685669";;

(* et le message codé : *)

let cM = map big_int_of_string [ "41021358" ; "1169160" ; "31809925" ;
       "17484908" ; "32136910" ; "30925474" ; "22255997" ; "48565640"
       ; "37770004" ; "48144453" ; "6582428" ; "39919725" ] ;;


(* On factorise N : *)

decomp2 N;;

(* on trouve [17669; 2819] qui sont x et y : *)

let x = big_int_of_string "17669";;
let y = big_int_of_string "2819";;

(* on trouve s en inversant p modulo (x-1)(y-1) : *)

let m = mult_big_int (pred_big_int x) (pred_big_int y);;

let s = div_mod unit_big_int p m;;


(* on décode alors le message en élevant à la puissance s :
```



```
 *
 * là il faut faire attention d'écrire quelque chose d'efficace pour
 * l'élévation à la puissance sinon on part dans les choux.
 *
 * on utilise le fait que l'on travaille modulo N, et les deux égalités :
 *
 *   power(x,2n)   = square(power(x,n))
 *   power(x,2n+1) = square(power(x,n)) * x
 *)

let rec power_mod N x n =
 let rec power x n =
  if eq_big_int n zero_big_int then
    unit_big_int
  else if eq_big_int n unit_big_int then
    mod_big_int x N
  else
    let q,r = quomod_big_int n deux in
    let xq = mod_big_int (square_big_int (power x q)) N in
    if eq_big_int r zero_big_int then
      xq
    else
      mod_big_int (mult_big_int x xq) N

 in power x n
;;

(* le message décodé est alors : *)

let dM = map (fun x -> mod_big_int (power_mod N x s) N) cM
;;

(* on trouve :
 * [676577; 763269; 838432; 868265; 737769; 788432; 857832; 838580; 698232;
 *  766578; 716571; 694632]
 *
 * et on l'affiche : *)

let affiche m =
  let affiche_morceau s =
    for i = 0 to 2 do
      let a = int_of_string (sub_string s (2*i) 2) in
      print_char (char_of_int a)
    done

  in do_list (fun b -> affiche_morceau (string_of_big_int b)) m
;;

affiche dM;;

(* on obtient : "CAML EST VRAIMENT UN SUPER LANGAGE. ". *)
```