RSA 0.9

Short User Manual

Contents

1 • Licence & Copyright *2*

2 • Description & Usage

a) Create new keys *2*

b) RSA-encode plaintext to code-list *3*

c) RSA-decode code-list to plaintext *3*

3 • Source code listings

»Isqrt« – calculate integer part of square root of large integers *4*

»Irand« – calculate pseudo-random integer within two limits *5*

»TtoVl« – translate ASCII text to unique integer *6*

»VltoT« – retrieve ASCII text from unique integer *7*

»crKYs« – create 2 public and 1 secret keys on stack *8*

»→RSA« – RSA-encode plain text using pub_E and pub_N *9*

»RSA→« – Decode code-list into plaintext using sec_D and pub_N *10*

4 • Contact & Thanks *11*

1 • Licence & Copyright

2 • Description & Usage

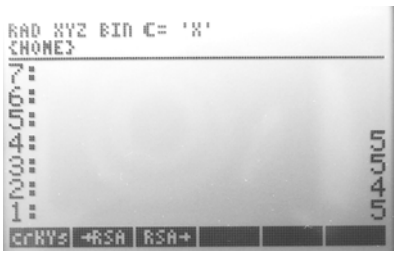
This library is a software product for the Hewlett Packard hp49g+ pocket calculator. It may or may not work on other hp models, but it has been tested only on a hp49g+.

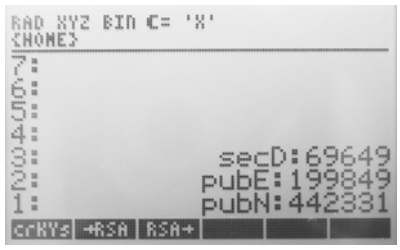
Installing the library:
To install it, transfer the file to your hp49g+ home directory. Store it in port 0,1, or 2 then. Press ON-F3 to make the RSA 0.9 library appear in the LIB menu. That's it.

The library offers three functions:

a) Generate a new set of 2 public and 1 secret key. The software tags them as »pub_E« (public exponent for modular arithmetic), »pub_N« (public modul), and »sec_D« (secret decode key). You can publish the pub_E and pub_N keys, so that others can encode a text addressed to you using these keys. The sec_D key you should never, under no circumstances, give away, as it is your key to retrieving the original message. If you give it away, anyone in possession of this key will be able to decrypt messages encoded with your pub_E and pub_N keys. This function of the library is dubbed »crKYs«, meaning: create keys on the stack.

Try this quick example. After installation, change into the RSA 0.9 libray folder. Put 4 integers on the stack, say 5 5 4 and 5 (these are small, insecure parameters, just for testing), see screenshot to the right. Press the softkey associated with »crKYs«, F1. After a few seconds, the three generated keys appear tagged on the stack, as you can see on the next screenshot.



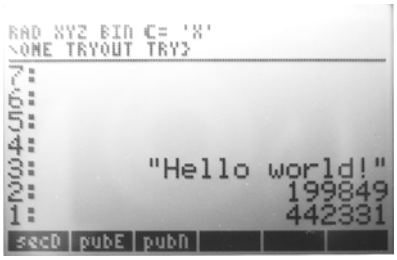


pub_N and pub_E constitute the public part of your key set, give them to whoever you want to send you an RSA encrypted message. Keep sec_D to yourself only. It's probably a good idea to store the keys in variables (you must change the directory first, as you cannot store variables in a library directory).

For low security, choose the second and the forth number around 50. The first and the third are only factors to 10 ^ second number and 10 ^ forth number from which the square roots will be calculated. For medium security, choose the second and fourth number around 100, and be a bit more patient with the User RPL program that calculates your keys. You only have to do this calculation once. For higher security, choose the second and forth number from somewhere around 500. And please prepare to be very patient!

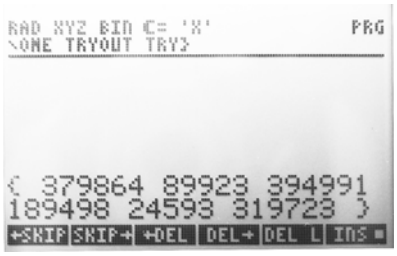
b) RSA encode a plaintext using the pub_E and pub_N keys of the intended recipient.

Now we're ready to RSA encrypt a text message. Put the text message (a string, so put it in quotes) on the stack. Then put the pub_e key part of your recipient on the stack, then the pub_N part of her/his key. See screenshot to the right.



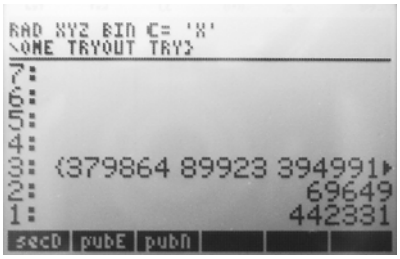
Then press the softkey associated with encoding to RSA, F2. On the stack a list containing several numbers will appear. This is the code-list. It constitutes the encrypted form of your message. See screenshot.

The list contains several more numbers which you can view if you press the arrow-down-key. You should not change the codes as it will scramble parts of the message. Press ENTER to return to the stack after inspecting the code-list. Now you can transfer this list to the intended recipient of your message, and s/he will use her/his secret key to make it plaintext again.



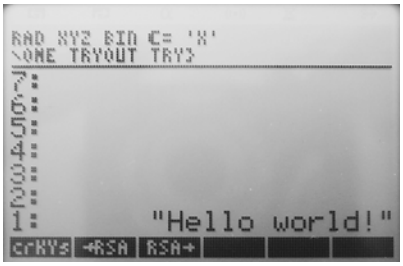
c) Decode RSA code-list using your sec_D and pub_N keys.

Let's decrypt such a code-list. Pretend you are the recipient, and use your secret sec_D key to decode the code-list into plaintext again. Put the code-list on the stack, put your sec_D key on the stack and then put the pub_N key on the stack.



Then press the softkey associated with the decryption of RSA code, F3. On the stack appears the original text messages sent to you. That's it.

Future versions of the library could contain a way to sign a message so that the recipient can be more assured that it was really you who sent the message to her/him.



```
Isqrt: Integer Square Root, User RPL
       by Michael Kuyumcu, info@noemanetz.de
       v0.9 beta – 2006-07-21


Stack diagram:
(Radikand -- integer_part_of_square_root_of_radikand)


Program in User RPL:

<< → R

  << R →STR                          ; convert radikand to string
     DUP                             ; since we will extract substrings from this string later, we'll need a working copy of it
     SIZE 2 / .5 + IP                ; We'll use Heron's algorithm and need a first guess.
     1 SWAP SUB                      ; As a first guess, we'll take the first half of the radikand string as an integer

     STR→ → Xn                       ; saved in the local variable Xn.

     <<
       DO                            ; Until the end criterion is met, apply Heron's scheme:
         R Xn / Xn + 2 /             ; Xn+1:= (R / Xn + Xn) / 2
         EVAL FLOOR                  ; evaluate as a fraction in exact mode and round down if necessary
         DUP DUP                     ; For the end criterion, we need to check two conditions:
         Xn ==                       ; A) Is the new approximation exactly the same as the last?
         SWAP Xn - 1 NEG ==          ; B) Is the new approximation one less than the last?
         OR                          ; Is A) OR B) true?
         SWAP DUP SQ R ≤             ; C) Third condition: is the new approximation squared equal or less the radikand?
                                     ; Our appromimation must be integer-equal or less than the real number approximation.
         ROT AND                     ; AND (A) OR B)) with C)
                                     ;
    SWAP 'Xn' STO                    ; At any rate, save the latest approximation for possible further use
                                     ;
       UNTIL END                     ; loop until the end criterion is met.
       Xn                            ; Put the approximation Xn on the stack.
   >>
 >>
>>
```

```
Irand: Integer Random, User RPL
       by Michael Kuyumcu, info@noemanetz.de
       v0.91 beta – 2006-07-25


Stack diagram:
(lowerLimit   upperLimit -- pseudo_random_number_between_and_including_lowerLimit_and_upperLimit)



Program in User RPL:

<< → BEGINN ENDE                        ; save upper and lower limit

  << ENDE BEGINN - 1 + → FAKTOR         ; Faktor tells how many possible pseudo random numbers can be produced
                                        ; For example: Irand(8,10) may give 8, 9, or 10, so there's 3 possible outcomes.

    << "" 1 ENDE →STR SIZE              ; Save pseudo random number as empty string

      FOR J RAND 10 * IP R→I + NEXT     ; As many digits as the upper limit has we will pseudo-randomly produce
                                        ; and append it to the pseudo random number collection string
                                        ;

      DUP SIZE SWAP STR→ FAKTOR *       ; Convert pseudo random number to integer and multiply by Faktor.
                                        ; The result is a pseudo random integer which still needs to be divided
                                        ; by the correct power of 10 so that there will be a number from [0...1[
                                        ; as a result.


      SWAP 10 SWAP ^ R→I /              ; The power of 10 is given by the length of the pseudo random number string.


      DUP PROPFRAC DUP →STR             ; Deal with special result cases:

      ROT →STR ==                       ; Is the resulting fraction below 0?
      IF THEN DROP BEGINN               ; If yes, the resuling pseudo random number will be 0.


      ELSE DUP →STR "/" POS             ; If not zero, the resulting pseudo random number may be an integer already.
                                        ; If so, just add it to the lowerLimit, and the pseudo random number is done.


        IF THEN OBJ→                    ; But if it was a real fraction, let's get the integer part of it.
        DROP DROP DROP BEGINN +         ; We don't need the fractional part, and not the connecting operator (+ or /)
        ELSE BEGINN +
        END
      END
    >>
  >>
>>
```

```
TtoVl: Text to Value, User RPL
       by Michael Kuyumcu, info@noemanetz.de
       v0.9 beta – 2006-07-25



Stack diagram:
("Text“ -- TextCode)
Text may contain any extended ASCII characters (codes 0 to 255)



Program in User RPL:

<< 1 → COF                          ; That’s the codeFactor which will be multiplied by 256 again and again
  << 0 SWAP                         ; On stack level 1, there is the current codeValue, 0.
    DUP SIZE 1 SWAP                 ; How many characters are there to code?

    FOR J
      DUP J J SUB                   ; Extract character #j from string

      NUM R→I                       ; Convert to ASCII code and get integer part.

      COF *                         ; multiply with the »digit-like« codeFactor
      3 ROLL + SWAP                 ; Add character value to accumulated TextCode

      COF 256 * 'COF' STO           ; Create next CodeFactor for the next character, * 256
    NEXT

    DROP                            ; Forget the original source text.
  >>
>>
```

```
VltoT: Value to Text, User RPL
       by Michael Kuyumcu, info@noemanetz.de
       v0.9 beta – 2006-07-25



Stack diagram:
(TextCode -- "Text")



Program in User RPL:

<< "" SWAP                                  ; The accumulated text string will reside on level 1
  DO
    DUP                                     ; Split off fractional parts after division by 256 and integer
    256 <                                   ; part. If the fractional part < 256, we have reached the final
                                            ; character to be decoded.
    IF THEN CHR + 1                         ; If so, append character and put a 1 on the stack
                                            ; for the UNTIL loop to finish.
    ELSE
       256 IDIV2 CHR                        ;
       ROT SWAP + SWAP                      ; Append character to string
       0                                    ; 0 means: not finished yet.
    END

  UNTIL END                                 ;
>>
```

```
crKYs: create RSA Keys on the stack, User RPL
       von Michael Kuyumcu, info@noemanetz.de
       v0.91 beta – 2006-07-27
       needs the components »Irand« und »Isqrt«


Stack diagram:
(Faktor_p Exponent_p Faktor_q Exponent_q -- secD pubE pubN)
might take several second if Exponent_p and Exponent_q are large (say, over 50).



Program in User RPL:

<< 0 0 0 → PF PE QF QE pN pE sD

  << PF 10 PE ^ * Isqrt PREVPRIME → P            ; Calculate first prime number, p

    << QF 10 QE ^ * Isqrt PREVPRIME → Q          ; Calculate second prime number, q
      << P Q * 'pN' STO                          ; Calculate pubN as p*q

         P 1 – Q 1 – *                           ; Intermediate value ph = (p-1)*(q-1), always even

         DO
           DUP DUP                               ; Now we are looking for a pseudo random number ph such that
           2 SWAP Irand                          ; ph is within [2...ph] and such that
           DUP ROT GCD 1 ==                      ; GCD(Zufallszahl, ph) = 1
           IF THEN 1
           ELSE DROP 0
           END
         UNTIL END

         'pE' STO                                ; So we've found the exponent part of the public key, pub_E

    DUP                                          ; Duplicate ph
    pE IEGCD                                     ; Euclidian algorithm we need part 3 (the rest)
    UNROT DROP DROP DUP 0 ≤                      ; If the rest is less than zero, then add the modul
         IF THEN + 'sD' STO                      ; in order to make the rest positive (just for aesthetics)
         ELSE 'sD' STO DROP                      ; If rest is positive, keep it and drop ph.
                                                 ; Now we have the secret part of the key, too, sec_D
         END

       sD "secD" →TAG                            ; put the values on the stack, tagged.
       pE "pubE" →TAG
       pN "pubN" →TAG
      >>
    >>
  >>
>>
```

```
->RSA: Encode Text with RSA public key algorithm, User RPL
       by Michael Kuyumcu, info@noemanetz.de
       v0.9 beta – 2006-07-26
       needs the component »TtoVl«


Stack diagram:
("Text" public_E public_N -- {CodeList})



Programm in User RPL:

<<            1  0 0    0 →            ; Vorbelegen einiger zusätzlicher Werte
   T pubE PN PV PE F PADD               ; T stands for "Text", pubE is the public exponent,
                                        ; PN is the public modulo (these three values are on the stack)
                                        ; PV, PE are the split-up points for the source text
                                        ; F is a flag that shows that the code of a text part is not
                                        ; coprime to PN. Then a new text part has to be selected.
                                        ; PADD is the number of characters to be encoded in each run.

  <<PN LOG ->NUM 256 LOG →NUM /         ; Calculate how many characters can be encoded at once (given PN)

    IP R→I 64 MIN 'PADD' STO            ; At most 64 will be used in each run.
    {}                                  ; The empty list is going to collect the RSA code values for the text parts
    PN MODSTO                           ; PN serves as the modul for the modular arithmetic

    DO
      PV PADD + 'PE' STO
      0 'F' STO                         ; So far, we believe the text part to be encodeable.

      DO
        PE 1 - 'PE' STO                 ; Shorten the text substring by one
        T SIZE DUP PE <                 ; If the end of the plaintext is reached, its end will be significant.
        IF THEN 'PE' STO ELSE DROP END
        T PV PE SUB                     ; Extract substring from PV to PE, inclusively
        TtoVl                           ; Translate to unique ASCII TextValue
        DUP PN GCD 1 ≠                  ; Check whether this TextValue has a common denominator with PN (a no-no!)
        IF THEN                         ; If the GCD is not 1, we have a problem.
          DROP PE PV >                  ; Solution approach: shorten substring, if still possible, by one character.
          IF THEN 0
          ELSE 1 'F' STO 1 END          ; 1 for the flag F means: try another, shorter, substring
        ELSE pubE POWMOD                ; This is the core of the algorithm: take the TextValue to the Power of pub_E mod pub_N
          DUP 0 <                       ; If we get a negative result, we add pub_N (for aesthetic reasons only)
          IF THEN PN + END 1
        END
      UNTIL END

      F IF THEN "Text not codeable. Please change it." MSGBOX 1
      ELSE +                            ; If encoding was successful, add the CodeValue to the CodeValue list
        T SIZE PE ≤                     ; Have we already processed the last of the text blocks?
        IF THEN 1                       ; If so, put a 1 on the stack for the UNTIL loop to terminate.
        ELSE PE 1 + 'PV' STO 0          ; Otherwise, determine the start of the next text substring.
        END
      END
    UNTIL END
  >>
>>
```

```
RSA->: Decode RSA-Codes into source text using RSA public key algorithm, User RPL
       by Michael Kuyumcu, info@noemanetz.de
       v0.9 beta – 2006-07-26
       needs the component »VltoT«



Stack diagram:
({CodeList} secret_D public_N -- "Plaintext")


Program in User RPL:

<< MODSTO                           ; pub_N serves as the modul for the modular power arithmetic


  → CO secD                         ; Save CodeList and secret decode key

  << CO OBJ→                        ; Split up CodeList into separate codes
    1 "" UNROT SWAP                 ; Provide an empty string to collect the decoded plaintext characters.

    FOR J
      SWAP                          ; Bring the collecting string on top level of stack
                                    ; and the current CodeValue from the CodeList to level 1 for processing.
      secD POWMOD                   ; That's the decoding core: (CodeValue^sec_D) mod pub_N
      VltoT                         ; Translate the resulting large integer back to a ASCII string
      SWAP +                        ; And prepend to collecting plaintext string.
    NEXT                            ; Process next CodeValue

  >>
>>
```

4 • Contact

In case you have questions or feedback, please do not hesitate to contact the author, Michael Kuyumcu, at info@noemanetz.de

Thanks

I thank Wolfgang Rautenberg for making available to the public his great OT49 (operating tools 49) on *www.hpcalc.org*. This tool allowed me to make a library from my User RPL programs in one swift step.

Thanks go also to Prof. Dr. Dörte Haftendorn for providing on her web pages comprehensible information on cryptographic algorithms and their implementation on computer algebra systems (CAS). Visit *http://www.doerte-haftendorn.de/* if you are interested.

And thanks to Hewlett Packard for making this great pocket calculator, the hp 49g+.