EHN Group 12 Practical 3

Generated by Doxygen 1.8.13

Contents

Chapter 1

EHN 410 Group 12 Practical 3

This software was developed by EHN 410 group 12 and is a toolset that can be used to encrypt or decrypt with the RC4 and RSA algorithms implemented from first principles.

The main features are:

- RC4 file encryption and decryption with a key of up to 128 bits.
- RSA public/private key pair generation for up to 4096 bits.
- RSA text encryption and file decryption for up to 4096 bits.

These features allow for fast and secure public-key cryptography by using RSA in conjunction with RC4. This gives the user the advantages of having a very fast encryption process and public-key cryptography, which are typically in conflict with each other.

Compilation

Use the standard **gcc** compiler available on most builds of **Linux**.
This toolset requires the **GMP** library to have been installed on the system before compilation can occur. Follow the instructions provided on the GMP website for information on how to do this.

There are several tools included in this toolset and they have to be compiled separately.

All of these commands should be run in a terminal window in the same folder as the source files.

The RC4 stream file encryption/decryption tool:

```
$ gcc rc4.c prac3.c -lgmp -o rc4
```

The RSA key generation tool:

```
$ gcc rsakeygen.c prac3.c -lgmp -o rsakeygen
```

The RSA encryption tool:

```
$ gcc rsaencrypt.c prac3.c -lgmp -o rsaencrypt
```

The RSA decryption tool:

```
$ gcc rsadecrypt.c prac3.c -lgmp -o rsadecrypt
```

Compilation was tested on **Linux Ubuntu 18.04.4 LTS**.

Usage

A command in the following format should be run in a **terminal window** in the **same folder** where the compiled executable is located:

```
$ ./executable -arg1 value1 -arg2 value2...
```

The arguments provided depend on the tool which you want to use.

The RC4 stream file encryption/decryption tool:

```
$ ./rc4 -fi <input file> -fo <output file> -key <key file>
```

Arguments:

```
-fi       Specifies the path to the input file to be encrypted or decrypted.
-fo       Specifies the path to the output file where the result will be stored.
-key      Specifies the path to the file that contains the key to be used for the
          operation. If this is not specified, the user will be prompted to enter
          the key manually (in ASCII).
```

The RSA key generation tool:

```
$ ./rsakeygen -bitLen <RSA bit length> -fopub <public key output file> -fopriv <private key output file> -init
```

Arguments:

```
-bitLen   Specifies the number of bits to be used in the generation of the RSA
          public/private key pair (should be between 128 and 4096 inclusive).
-fopub    Specifies the path to the file where the public key will be stored.
-fopriv   Specifies the path to the file where the private key will be stored.
-init     Specifies the seed to be used for the RC4 random number generator
          used to generate the keys (in ASCII).
```

The RSA encryption tool:

```
$ ./rsaencrypt -fo <encrypted output file> -fopub <public key input file> -key <RC4 key to encrypt>
```

Arguments:

```
-fo       Specifies the path to the output file where the result will be stored.
-fopub    Specifies the path to the file where the previously generated public key
          is stored.
-key      Specifies the key (used by the RC4 algorithm for encryption) to be
          encrypted (in ASCII).
```

The RSA decryption tool:

```
$ ./rsadecrypt -fi <encrypted input file> -fopriv <private key input file> -fo <decrypted output file>
```

Arguments:

```
-fi       Specifies the path to the file to be decrypted.
-fopriv   Specifies the path to the file where the previously generated private
          key is stored.
-fo       Specifies the path to the output file where the key (used by the RC4
          algorithm for decryption) will be stored.
```

Attention: please take special note of the following:

- Remember to add **"double quotes"** to arguments if **spaces** are present in the string.
  If this is **not** done, only the **first word** in the string will be processed.
- The expected input length for the **-key** argument is **16** characters for RSA encryption and RC4 if no file is specified (i.e. manual entry of the key).
  If an ASCII string with **less** characters are given, the key will be **padded with zeroes** at the end. If an ASCII string with **more** characters are given, the **trailing characters** will be **discarded**.

Makefile usage

A **Makefile** is provided with the source code to allow for easy setup and demo usage.

Open a **terminal window** in the **same folder** as the Makefile to use it.

The following command will build...

...the RC4 encryption/decryption tool:

```
$ make rc4
```

...the RSA key generator tool:

```
$ make rsakeygen
```

...the RSA encryption tool:

```
$ make rsaencrypt
```

...the RSA decryption tool:

```
$ make rsadecrypt
```

The following command will run the demo:
Note: the key "EHN prac 3 demo" must be entered manually when prompted.

```
$ make demo
```

This demo is explained in the next section (Example usage). Afterwards, the private/public RSA key pairs, the encrypted files and final output can be found in the same folder.

To remove all the executables and generated files, the following command can be run:

```
$ make clean
```

Example usage

Example 1: RC4 Encryption

The following command will **encrypt** a **file** called **"input.txt"** (in the same folder) using **RC4** with the key **"EHN prac 3 demo"** and store the result in the file **"encrypted.enc"**:

```
$ ./rc4 -fi "input.txt" -fo "encrypted.enc"
```

Note: the key "EHN prac 3 demo" must be entered manually when prompted.

The following output is expected:

```
EHN Group 12 Practical 3

Using "input.txt" as the input file
Using "encrypted.enc" as the output file
Please enter the key that should be used to encrypt/decrypt the input file (ASCII):  EHN prac 3 demo
Using "EHN prac 3 demo" as the key.
Operation took 0 ms

Encryption/Decryption complete
```

The file "encrypted.enc" can be found in the same folder as the executable.

Example 2: RSA Key Generation

The following command will **generate** a **public/private RSA key pair** by using **128** bits with the seed **"RNG seed"** and store the results in the files **"pubkey.txt"** and **"privkey.txt"**:

```
$ ./rsakeygen -bitLen 128 -fopub pubkey.txt -fopriv privkey.txt -init "RNG seed"
```

The following output is expected:

```
EHN Group 12 Practical 3

128 bits will be generated
Using "pubkey.txt" as the public key file
Using "privkey.txt" as the private key file
Using "RNG seed" as the RC4 RNG seed.

Done
```

The files "pubkey.txt" and "privkey.txt" can be found in the same folder as the executable.

Example 3: RSA Encryption

The following command will **encrypt** the key **"EHN prac 3 demo"** using the previously generated public key and store the result in the file **"cipher.key"**:

```
$ ./rsaencrypt -fo cipher.key -fopub pubkey.txt -key "EHN prac 3 demo"
```

The following output is expected:

```
EHN Group 12 Practical 3

Using "cipher.key" as the output file
Using "pubkey.txt" as the public RSA key file
Using "EHN prac 3 demo" as the key

Done
```

The file "cipher.key" can be found in the same folder as the executable.

Example 4: RC4 Decryption

The following command will **decrypt** the key in the file **"cipher.key"** using the previously generated private key and store the result in the file **"plain.txt"**:

```
$ ./rsadecrypt -fi cipher.key -fopriv privkey.txt -fo plain.txt
```

The following output is expected:

```
EHN Group 12 Practical 3

Using "cipher.key" as the input file
Using "privkey.txt" as the private RSA key file
Using "plain.txt" as the output file

Done
```

The file "plain.txt" can be found in the same folder as the executable.

Example 5: RC4 Decryption

The following command will **decrypt** the file **"encrypted.enc"** using the decrypted key in the file **"plain.txt"** and store the result in the file **"output.txt"**:

```
$ ./rc4 -fi "encrypted.enc" -fo "output.txt" -key "plain.txt"
```

Note: in this example the key "EHN prac 3 demo" is already in "plain.txt". This will differ if a different key was decrypted.

The following output is expected:

```
EHN Group 12 Practical 3

Using "encrypted.enc" as the input file
Using "output.txt" as the output file
Using "plain.txt" as the key file
Using "EHN prac 3 demo" as the key.
Operation took 0 ms

Encryption/Decryption complete
```

The file "output.txt" can be found in the same folder as the executable.

Chapter 2

Data Structure Index

2.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 3

File Index

3.1 File List

Here is a list of all files with brief descriptions:

Chapter 4

Data Structure Documentation

4.1 rc4ctx_t Struct Reference

The struct used to store the current state of the RC4 encryption algorithm.

```
#include <prac3.h>
```

Data Fields

- int S [256]
- int i
- int j

4.1.1 Detailed Description

The struct used to store the current state of the RC4 encryption algorithm.

4.1.2 Field Documentation

4.1.2.1 i

```
int rc4ctx_t::i
```

4.1.2.2 j

```
int rc4ctx_t::j
```

4.1.2.3 S

```
int rc4ctx_t::S[256]
```

The documentation for this struct was generated from the following file:

- prac3.h

4.2 rsactx_t Struct Reference

The RSA struct to store all the key values.

```
#include <prac3.h>
```

Data Fields

- mpz_t e
- mpz_t d
- mpz_t n
- mpz_t p
- mpz_t q

4.2.1 Detailed Description

The RSA struct to store all the key values.

4.2.2 Field Documentation

4.2.2.1 d

```
mpz_t rsactx_t::d
```

4.2.2.2 e

```
mpz_t rsactx_t::e
```

4.2.2.3 n

```
mpz_t rsactx_t::n
```

4.2.2.4 p

```
mpz_t rsactx_t::p
```

4.2.2.5 q

```
mpz_t rsactx_t::q
```

The documentation for this struct was generated from the following file:

- prac3.h

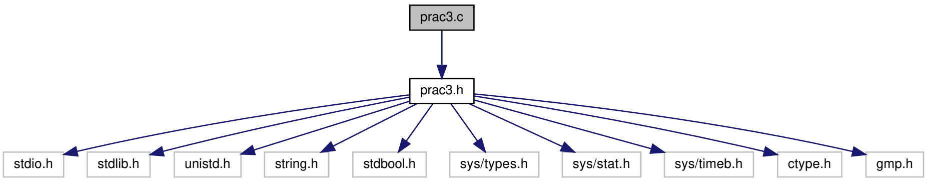
Chapter 5

File Documentation

5.1 prac3.c File Reference

```
#include "prac3.h"
```

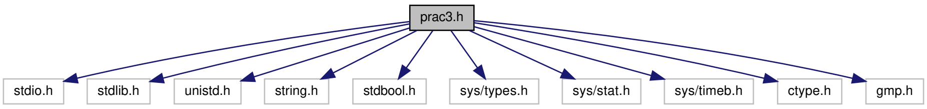
Include dependency graph for prac3.c:



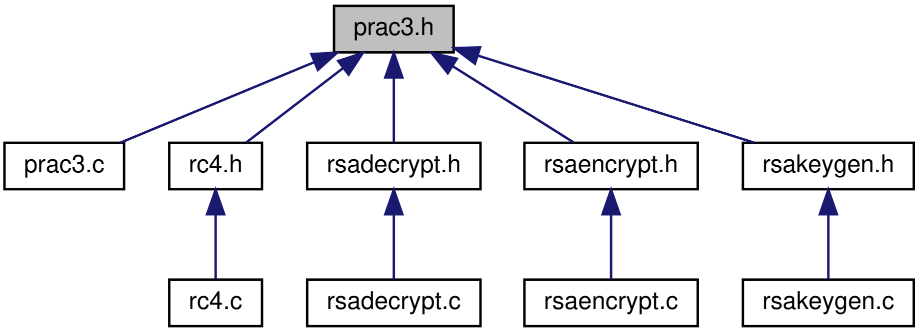
5.2 prac3.h File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <stdbool.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/timeb.h>
#include <ctype.h>
#include <gmp.h>
```

Include dependency graph for prac3.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct rc4ctx_t

  The struct used to store the current state of the RC4 encryption algorithm.
- struct rsactx_t

  The RSA struct to store all the key values.

Macros

- #define U8 unsigned char

Functions

- void rc4_init (struct rc4ctx_t ∗rc4ctx, U8 key[ ], int keylen)
- U8 rc4_getbyte (struct rc4ctx_t ∗rc4ctx)
- void rsa_init (struct rsactx_t ∗rsactx)
- void rsa_clean (struct rsactx_t ∗rsactx)
- void swap (int ∗a, int ∗b)
- int hex_convert (char hex_string[ ], int length)
- void print_hex_string (U8 hex_string[ ], int message_len)

5.2.1 Macro Definition Documentation

5.2.1.1 U8

```
#define U8 unsigned char
```

5.2.2 Function Documentation

5.2.2.1 hex_convert()

```
int hex_convert (
             char hex_string[],
             int length )
```

5.2.2.2 print_hex_string()

```
void print_hex_string (
             U8 hex_string[],
             int message_len )
```

5.2.2.3 rc4_getbyte()

```
U8 rc4_getbyte (
             struct rc4ctx_t * rc4ctx )
```

Returns the next byte of the stream cipher that can be used to encrypt a value and updates the state variables.

Parameters

rc4ctx	A pointer to the struct that will hold all the state variables for the RC4 algorithm.

Returns

The byte generated by the RC4 algorithm.

5.2.2.4 rc4_init()

```
void rc4_init (
             struct rc4ctx_t * rc4ctx,
             U8 key[],
             int keylen )
```

Sets up the RC4 algorithm variables using the key and performs the initial permutation.

Parameters

rc4ctx	A pointer to the struct that will hold all the state variables for the RC4 algorithm.
key	An array of 8-bit values that represent the key.
keylen	The length of the key in bytes.

5.2.2.5 rsa_clean()

```
void rsa_clean (
            struct rsactx_t * rsactx )
```

Frees the memory associated with the RSA context.

Parameters

rsactx	The RSA context struct.

5.2.2.6 rsa_init()

```
void rsa_init (
            struct rsactx_t * rsactx )
```

Initialises the RSA context.

Parameters

rsactx	The RSA context struct.

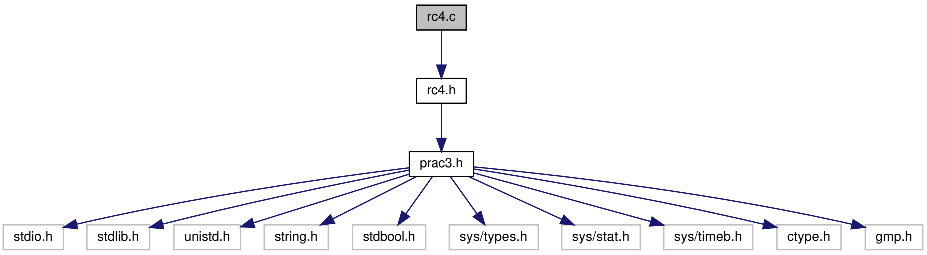
5.2.2.7 swap()

```
void swap (
            int * a,
            int * b )
```

5.3 rc4.c File Reference

```
#include "rc4.h"
```

Include dependency graph for rc4.c:



Functions

- int main (int argc, char ∗argv[ ])

5.3.1 Function Documentation

5.3.1.1 main()

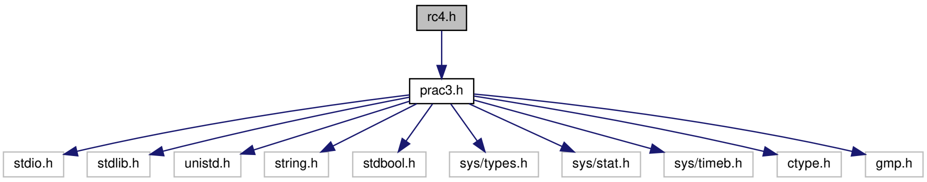
```
int main (
            int argc,
            char * argv[ ] )
```

The main function for the RC4 encryption/decryption utility. Uses the RC4 functions in prac3.h to encrypt/decrypt an input file using a key file, or using a key entered into the terminal.

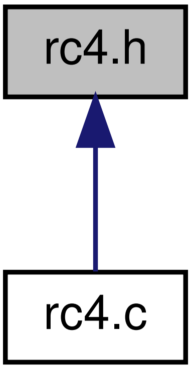
5.4 rc4.h File Reference

```
#include "prac3.h"
```

Include dependency graph for rc4.h:



This graph shows which files directly or indirectly include this file:



Macros

- #define RC4_MAX_KEY_LEN 16

  The maximum length of the key (in bytes) used for the RC4 encryption utility.

5.4.1 Macro Definition Documentation

5.4.1.1 RC4_MAX_KEY_LEN

```
#define RC4_MAX_KEY_LEN 16
```

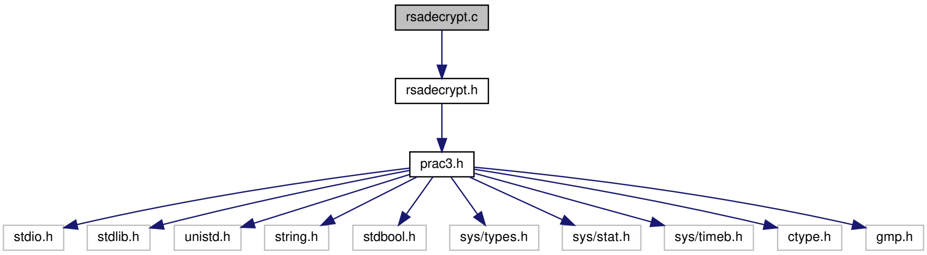
The maximum length of the key (in bytes) used for the RC4 encryption utility.

5.5 README.md File Reference

5.6 rsadecrypt.c File Reference

```
#include "rsadecrypt.h"
```

Include dependency graph for rsadecrypt.c:



Functions

- int main (int argc, char ∗argv[ ])

  This utility decrypts the key used in the RC4 algorithm.

5.6.1 Function Documentation

5.6.1.1 main()

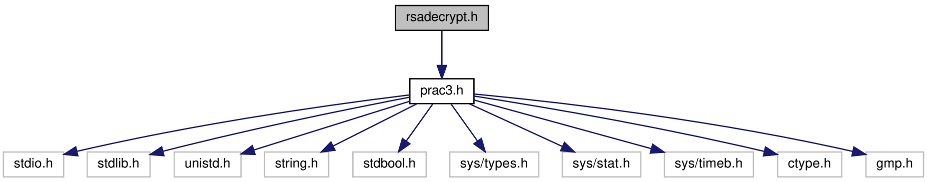
```
int main (
            int argc,
            char ∗ argv[ ] )
```

This utility decrypts the key used in the RC4 algorithm.

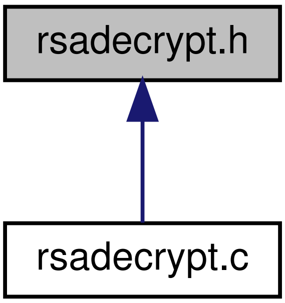
5.7 rsadecrypt.h File Reference

```
#include "prac3.h"
```

Include dependency graph for rsadecrypt.h:



This graph shows which files directly or indirectly include this file:



Functions

- void decrypt_rsa (mpz_t plain, mpz_t d, mpz_t n, mpz_t cipher)

5.7.1 Function Documentation

5.7.1.1 decrypt_rsa()

```
void decrypt_rsa (
            mpz_t plain,
            mpz_t d,
            mpz_t n,
            mpz_t cipher )
```

Uses the GMP power function to decrypt a mpz_t value.

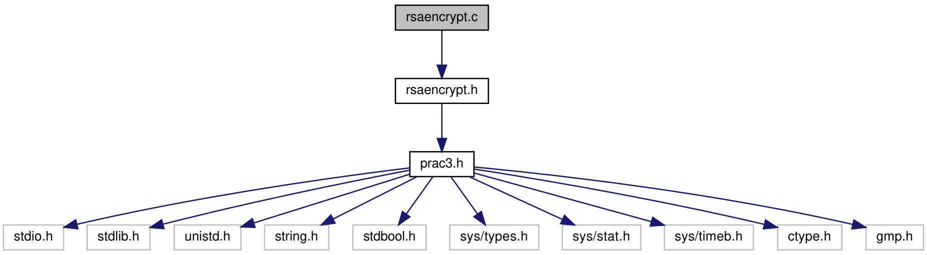
Parameters

plain	The output of the decrypt operation.
d	The secret exponent.
n	The modulus.
cipher	The value to be decrypted.

5.8 rsaencrypt.c File Reference

```
#include "rsaencrypt.h"
```

Include dependency graph for rsaencrypt.c:



Functions

- int main (int argc, char ∗argv[ ])

  This utility encrypts the key used in the RC4 algorithm.

5.8.1 Function Documentation

5.8.1.1 main()

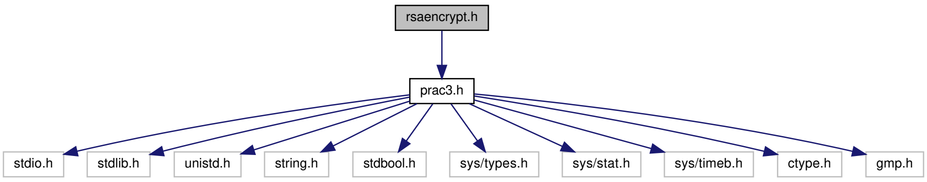
```
int main (
            int argc,
            char ∗ argv[ ] )
```

This utility encrypts the key used in the RC4 algorithm.

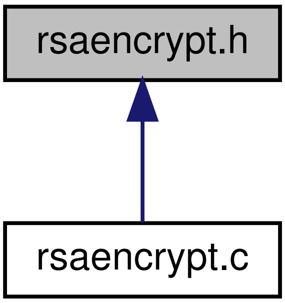
5.9 rsaencrypt.h File Reference

```
#include "prac3.h"
```

Include dependency graph for rsaencrypt.h:



This graph shows which files directly or indirectly include this file:



Functions

- void encrypt_rsa (mpz_t plain, mpz_t e, mpz_t n, mpz_t cipher)

5.9.1 Function Documentation

5.9.1.1 encrypt_rsa()

```
void encrypt_rsa (
            mpz_t plain,
            mpz_t e,
            mpz_t n,
            mpz_t cipher )
```

Uses the GMP power function to encrypt a mpz_t value.

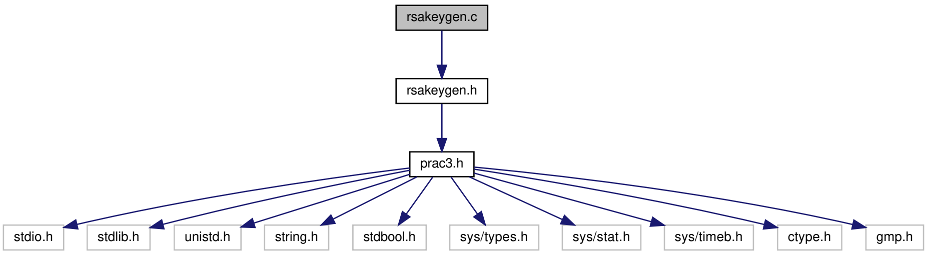
Parameters

plain	The value to be encrypted.
e	The public exponent.
n	The modulus.
cipher	The output of the encrypt operation.

5.10 rsakeygen.c File Reference

```
#include "rsakeygen.h"
```

Include dependency graph for rsakeygen.c:



Functions

- int main (int argc, char ∗argv[ ])

  This utility generates a public/private key pair to be used to encrypt and decrypt the RC4 key.

5.10.1 Function Documentation

5.10.1.1 main()

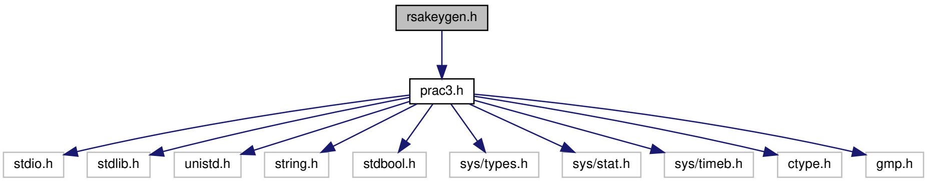
```
int main (
            int argc,
            char ∗ argv[ ] )
```

This utility generates a public/private key pair to be used to encrypt and decrypt the RC4 key.

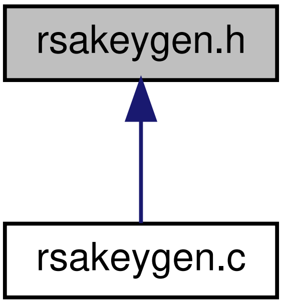
5.11 rsakeygen.h File Reference

```
#include "prac3.h"
```

Include dependency graph for rsakeygen.h:



This graph shows which files directly or indirectly include this file:



Functions

- void getprime (mpz_t prime, int num_bits)
- void getkeys (struct rsactx_t ∗rsactx, int key_len, int e_selection)

Variables

- struct rc4ctx_t rc4ctx

5.11.1 Function Documentation

5.11.1.1 getkeys()

```
void getkeys (
             struct rsactx_t * rsactx,
             int key_len,
             int e_selection )
```

Create the RSA key pair.

Parameters

rsactx	Pointer to the main RSA struct.
key_len	The length of the key that needs to be encrypted in bits.
e_selection	Which common value for e will be chosen.

5.11.1.2 getprime()

```
void getprime (
             mpz_t prime,
             int num_bits )
```

Gets the next prime from a randomly generated value from RC4 RNG.

Parameters

prime	The prime value output.
num_bits	The length of the prime number in bits.

5.11.2 Variable Documentation

5.11.2.1 rc4ctx

```
struct rc4ctx_t rc4ctx
```

Index