EHN 410 - Group 7

0.1

Generated by Doxygen 1.8.13

Contents

Chapter 1

EHN 410 - Group 7

Group members

- Mohamed Ameen Omar (u16055323)
- Douglas Healy (u16018100)
- Llewellyn Moyse (u15100708)

RSA Key Generation

1. Open a Linux Terminal.
2. Navigate to the root directory containing the *rsakeygen* source code.
3. Run the ∗"make rsakeygen"∗ command.
4. An executable called *rsakeygen* will be created.
5. Use ∗"./rsakeygen"∗ to run the RSA Key Generation program (if no input parameters are specified, a help menu will be displayed)
6. A list of input parameters and respective default values can be seen below:

Parameter	Description	Default Value
-h	Prints out the help menu	
-b	Specifies the number of bits for the public/private keys to be generated	None
-KU	Specifies the file to which the **public** key will be written	None
-KR	Specifies the file to which the **private** key will be written	None
-key	Specifies the key for the initialisation of the RNG (in hex by default)	None
-kf	Specifies the path to the key for the initialisation of the RNG (hex by default)	None
-ascii	Specifies that the key used is in ascii instead of hex	Hex

RSA Key Generation Usage Example

```
./rsakeygen -b bits -KU public_key_file -KR private_key_file -key key
```

RSA Encryption

1. Open a Linux Terminal.
2. Navigate to the root directory containing the *rsaencrypt* source code.
3. Run the ∗"make rsaencrypt"∗ command.
4. An executable called *rsaencrypt* will be created.
5. Use ∗"./rsaencrypt"∗ to run the RSA Encryption program (if no input parameters are specified, a help menu will be displayed)
6. A list of input parameters and respective default values can be seen below:

Parameter	Description	Default Value
-h	Prints out the help menu	
-key	Specifies the RC4 key to encrypt/decrypt	None
-fo	Specifies the file to write the encrypted result to	None
-KU	Specifies the RSA public key file to use for encryption	None
-kf	Specifies the path to the RC4 key to encrypt/decrypt	None
-hex	Specifies that the key used is in hex instead of ascii	ascii

RSA Encryption Usage Example

```
./rsaencrypt -key key -fo outputfile -KU public_key_file
```

RSA Decryption

1. Open a Linux Terminal.
2. Navigate to the root directory containing the *rsadecrypt* source code.
3. Run the ∗"make rsadecrypt"∗ command.
4. An executable called *rsadecrypt* will be created.
5. Use ∗"./rsadecrypt"∗ to run the RSA Decryption program (if no input parameters are specified, a help menu will be displayed)
6. A list of input parameters and respective default values can be seen below:

Parameter	Description	Default Value
-h	Prints out the help menu	
-fi	Specifies the path to the key to decrypt	None
-key	Specifies the key to decrypt	None
-KR	Specifies the RSA private key file to use for decryption	None
-fo	Specifies the file to write the decrypted result to	None

RSA Decryption Usage Example

```
./rsadecrypt -fi inputfile -KR private_key_file -fo outputfile
```

RC4

1. Open a Linux Terminal.
2. Navigate to the root directory containing the *rc4* source code.
3. Run the ∗"make rc4"∗ command.
4. An executable called *rc4* will be created.
5. Use ∗"./rc4"∗ to run the RC4 Encryption/Decryption Program (if no input parameters are specified, a help menu will be displayed)
6. A list of input parameters and respective default values can be seen below:

Parameter	Description	Default Value
-h	Prints out the help menu	
-fi	Specifies the file to encrypt/decrypt	None
-fo	Specifies the file to write the encrypted/decrypted result to	None
-kf	Specifies the path to the encryption key for the initialisation of the RNG (ascii by default)	None
-hex	Specifies that the key used is in hex instead of ascii	ascii

>∗ If no key is specified by the command line parameters then the user will be prompted for a key at runtime. >∗ All keys must have a maximum size of 16 bytes.

RC4 Encryption/Decryption Usage Example

```
./rc4 -fi inputfile -fo outputfile -kf keyfile
```

Chapter 2

Namespace Index

2.1 Namespace List

Here is a list of all namespaces with brief descriptions:

Chapter 3

Data Structure Index

3.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 4

File Index

4.1 File List

Here is a list of all files with brief descriptions:

Chapter 5

Namespace Documentation

5.1 mod Namespace Reference

Variables

- int x = 495
- int y = 145
- int z = 841

5.1.1 Variable Documentation

5.1.1.1 x

```
int x = 495
```

Definition at line 1 of file mod.py.

5.1.1.2 y

```
int y = 145
```

Definition at line 2 of file mod.py.

5.1.1.3 z

```
int z = 841
```

Definition at line 3 of file mod.py.

5.2 rc4Test Namespace Reference

Variables

- string inputfileName = "temp"
- string outputFileName = "temp2"
- string keyFile = ""
- temp = subprocess.call(["./rc4","-fi", inputfileName, "-fo", outputFileName])

5.2.1 Variable Documentation

5.2.1.1 inputfileName

```
string inputfileName = "temp"
```

Definition at line 10 of file rc4Test.py.

5.2.1.2 keyFile

```
string keyFile = ""
```

Definition at line 12 of file rc4Test.py.

5.2.1.3 outputFileName

```
string outputFileName = "temp2"
```

Definition at line 11 of file rc4Test.py.

5.2.1.4 temp

```
temp = subprocess.call(["./rc4","-fi", inputfileName, "-fo", outputFileName])
```

Definition at line 14 of file rc4Test.py.

5.3 rsaKeyGenTester Namespace Reference

Variables

- string bits = "55"
- string publickeyfile = "temp"
- string privatekeyfile = "temp2"
- string keyFile = ""
- string key = ""
- temp = subprocess.call(["./rsakeygen", "-b", bits, "-KU", publickeyfile, "-KR" , privatekeyfile , "-key" , key , "-kf" , keyFile])

5.3.1 Variable Documentation

5.3.1.1 bits

```
string bits = "55"
```

Definition at line 7 of file rsaKeyGenTester.py.

5.3.1.2 key

```
string key = ""
```

Definition at line 11 of file rsaKeyGenTester.py.

5.3.1.3 keyFile

```
string keyFile = ""
```

Definition at line 10 of file rsaKeyGenTester.py.

5.3.1.4 privatekeyfile

```
string privatekeyfile = "temp2"
```

Definition at line 9 of file rsaKeyGenTester.py.

5.3.1.5 publickeyfile

```
string publickeyfile = "temp"
```

Definition at line 8 of file rsaKeyGenTester.py.

5.3.1.6 temp

```
temp = subprocess.call(["./rsakeygen", "-b", bits, "-KU", publickeyfile, "-KR" , privatekeyfile
, "-key" , key , "-kf" , keyFile])
```

Definition at line 16 of file rsaKeyGenTester.py.

5.4 test Namespace Reference

Variables

- int x = 88003541953488079566952319866978440447
- int y = 49236241020712442839828388720313042593
- int n = 166501317222622877548769568260396763403

5.4.1 Variable Documentation

5.4.1.1 n

```
int n = 166501317222622877548769568260396763403
```

Definition at line 4 of file test.py.

5.4.1.2 x

```
int x = 88003541953488079566952319866978440447
```

Definition at line 2 of file test.py.

5.4.1.3 y

```
int y = 49236241020712442839828388720313042593
```

Definition at line 3 of file test.py.

Chapter 6

Data Structure Documentation

6.1 rc4ctx_t Struct Reference

struct rc4ctx_t - Structure used to retain the context of the RC4 key stream generator.

```
#include <rc4Lib.h>
```

Data Fields

- U8 state [RC4_STATE_SIZE]
- uint16_t index1
- uint16_t index2

6.1.1 Detailed Description

struct rc4ctx_t - Structure used to retain the context of the RC4 key stream generator.

Definition at line 38 of file rc4Lib.h.

6.1.2 Field Documentation

6.1.2.1 index1

```
uint16_t index1
```

Definition at line 40 of file rc4Lib.h.

6.1.2.2 index2

```
uint16_t index2
```

Definition at line 41 of file rc4Lib.h.

6.1.2.3 state

```
U8 state[RC4_STATE_SIZE]
```

Definition at line 39 of file rc4Lib.h.

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

- rc4Lib.h

6.2 rsactx_t Struct Reference

```
#include <rsa.h>
```

Data Fields

- mpz_t p
- mpz_t q
- mpz_t e
- mpz_t d
- mpz_t qn
- mpz_t n
- uint16_t numBits
- unsigned char ∗ initKey
- uint8_t initKeyLength
- mpz_t KU [2]
- mpz_t KR [2]

6.2.1 Detailed Description

Definition at line 20 of file rsa.h.

6.2.2 Field Documentation

6.2.2.1 d

```
mpz_t d
```

Definition at line 24 of file rsa.h.

6.2.2.2 e

```
mpz_t e
```

Definition at line 23 of file rsa.h.

6.2.2.3 initKey

```
unsigned char* initKey
```

Definition at line 28 of file rsa.h.

6.2.2.4 initKeyLength

```
uint8_t initKeyLength
```

Definition at line 29 of file rsa.h.

6.2.2.5 KR

```
mpz_t KR[2]
```

Definition at line 31 of file rsa.h.

6.2.2.6 KU

```
mpz_t KU[2]
```

Definition at line 30 of file rsa.h.

6.2.2.7 n

```
mpz_t n
```

Definition at line 26 of file rsa.h.

6.2.2.8 numBits

```
uint16_t numBits
```

Definition at line 27 of file rsa.h.

6.2.2.9 p

```
mpz_t p
```

Definition at line 21 of file rsa.h.

6.2.2.10 q

```
mpz_t q
```

Definition at line 22 of file rsa.h.

6.2.2.11 qn

```
mpz_t qn
```

Definition at line 25 of file rsa.h.

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

- rsa.h

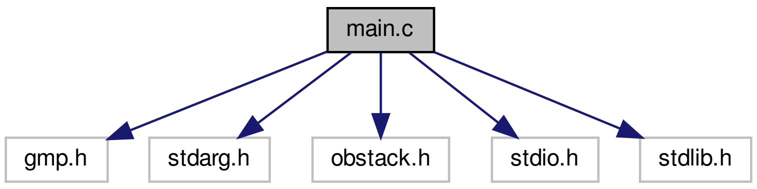
Chapter 7

File Documentation

7.1 main.c File Reference

```
#include <gmp.h>
#include <stdarg.h>
#include <obstack.h>
#include "stdio.h"
#include "stdlib.h"
```

Include dependency graph for main.c:



Functions

- int main (int argc, char ∗argv[ ])

7.1.1 Function Documentation

7.1.1.1 main()

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

Definition at line 37 of file main.c.

7.2 mod.py File Reference

Namespaces

- mod

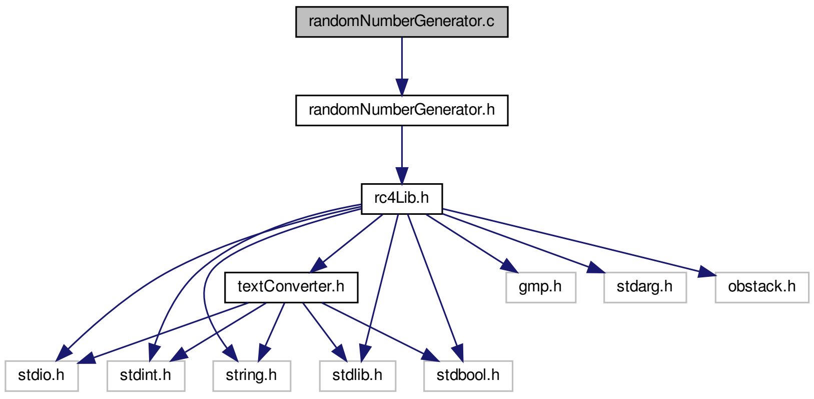
Variables

- int x = 495
- int y = 145
- int z = 841

7.3 randomNumberGenerator.c File Reference

Random number generator implementation file. This file is used to generate random numbers to be used in the RSA generation. The random number generator takes in a specified seed value. rseed is used to initialise. The rrand is used to retrieve a single byte from the random number generator. After completion, the destroyRNG function must be executed.

```
#include "randomNumberGenerator.h"
```
Include dependency graph for randomNumberGenerator.c:



Functions

- void rseed (U8 ∗key, int keylen, int isKeyHex)

  rseed - Function used to create a random number generator object and set the seed for the random number generator.
- U8 rrand ()

  rrand - Function used to generate a random number of a single byte long.
- void destroyRNG ()

  destroyRNG - Function used to deallocate all memory allocated for the random number generator, specifically the RC4 key stream state context structure.

Variables

- rc4ctx_t ∗ rngContext = NULL

  var - rngContext - The RC4 context state used for random number generation.

7.3.1 Detailed Description

Random number generator implementation file. This file is used to generate random numbers to be used in the RSA generation. The random number generator takes in a specified seed value. rseed is used to initialise. The rrand is used to retrieve a single byte from the random number generator. After completion, the destroyRNG function must be executed.

Authors

Mohamed Ameen Omar (u16055323)
Douglas Healy (u16018100)
Llewellyn Moyse (u15100708)

Version

0.1

Date

2019-05-22

Copyright

Copyright (c) 2019

7.3.2 Function Documentation

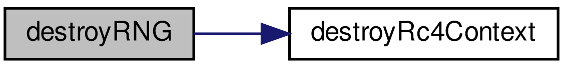
7.3.2.1 destroyRNG()

```
void destroyRNG ( )
```

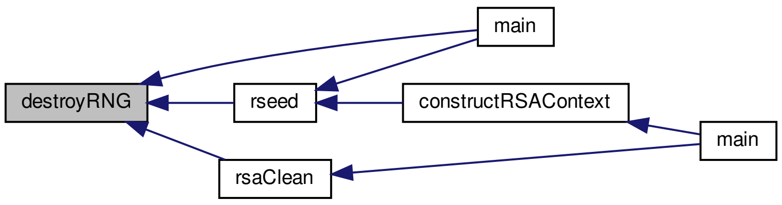
destroyRNG - Function used to deallocate all memory allocated for the random number generator, specifically the RC4 key stream state context structure.

Definition at line 61 of file randomNumberGenerator.c.

Here is the call graph for this function:



Here is the caller graph for this function:



7.3.2.2 rrand()

```
U8 rrand ( )
```

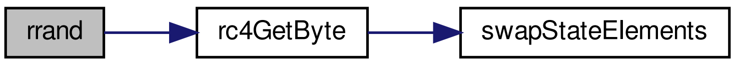
rrand - Function used to generate a random number of a single byte long.

Returns

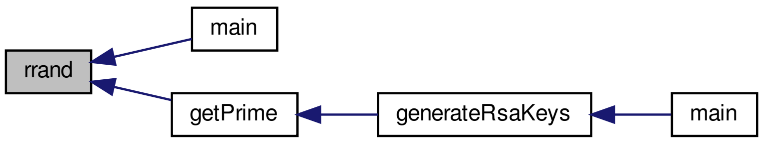
U8 - A random number of 8 bits long.

Definition at line 45 of file randomNumberGenerator.c.

Here is the call graph for this function:



Here is the caller graph for this function:



7.3.2.3 rseed()

```
void rseed (
            U8 * key,
            int keylen,
            int isKeyHex )
```

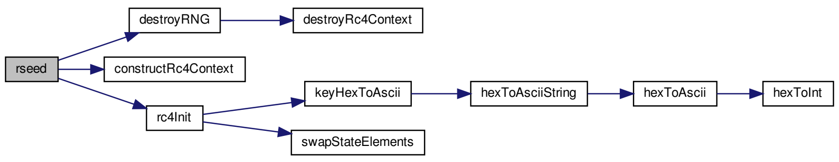
rseed - Function used to create a random number generator object and set the seed for the random number generator.

Parameters

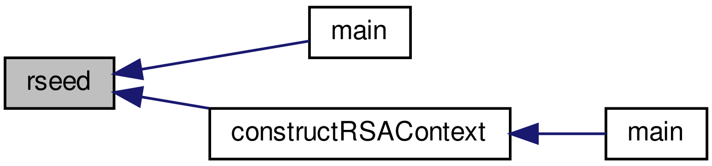
key	- uint8_t∗ - The key used to seed the random number generator.
keylen	- The length of the key
key.	
isKeyHex	- flag used to indicate if the key
key	is hex or ascii encoded.

Definition at line 32 of file randomNumberGenerator.c.

Here is the call graph for this function:



Here is the caller graph for this function:



7.3.3 Variable Documentation

7.3.3.1 rngContext

```
rc4ctx_t* rngContext = NULL
```

var - rngContext - The RC4 context state used for random number generation.

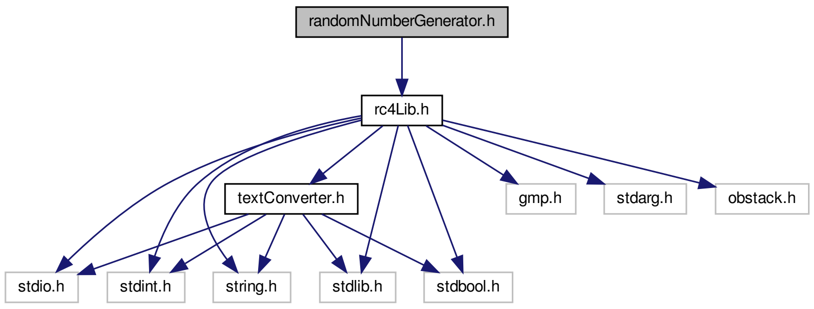
Definition at line 22 of file randomNumberGenerator.c.

7.4 randomNumberGenerator.h File Reference

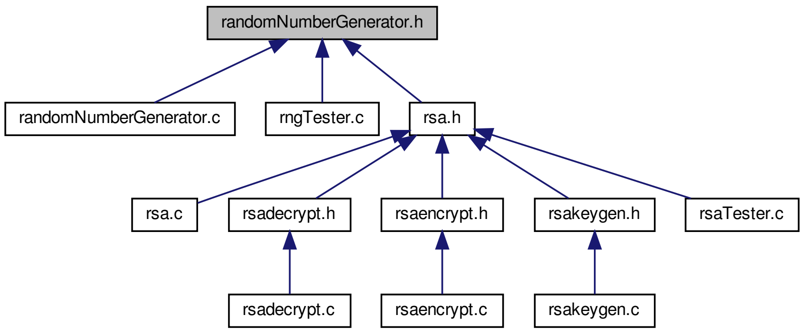
Random number generator function prototype file. This file is used to generate random numbers to be used in the RSA generation. The random number generator takes in a specified seed value. rseed is used to initialise. The rrand is used to retrieve a single byte from the random number generator. After completion, the destroyRNG function must be executed.

```
#include "rc4Lib.h"
```

Include dependency graph for randomNumberGenerator.h:



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



Functions

- void rseed (U8 ∗key, int keylen, int isKeyHex)

  rseed - Function used to create a random number generator object and set the seed for the random number generator.
- U8 rrand ()

  rrand - Function used to generate a random number of a single byte long.
- void destroyRNG ()

  destroyRNG - Function used to deallocate all memory allocated for the random number generator, specifically the RC4 key stream state context structure.

Variables

- rc4ctx_t ∗ rngContext

  var - rngContext - The RC4 context state used for random number generation.

7.4.1 Detailed Description

Random number generator function prototype file. This file is used to generate random numbers to be used in the RSA generation. The random number generator takes in a specified seed value. rseed is used to initialise. The rrand is used to retrieve a single byte from the random number generator. After completion, the destroyRNG function must be executed.

Authors

Mohamed Ameen Omar (u16055323)
Douglas Healy (u16018100)
Llewellyn Moyse (u15100708)

Version

0.1

Date

2019-05-22

Copyright

Copyright (c) 2019

7.4.2 Function Documentation

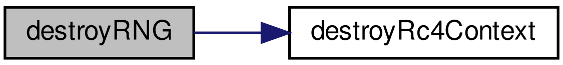
7.4.2.1 destroyRNG()

```
void destroyRNG ( )
```

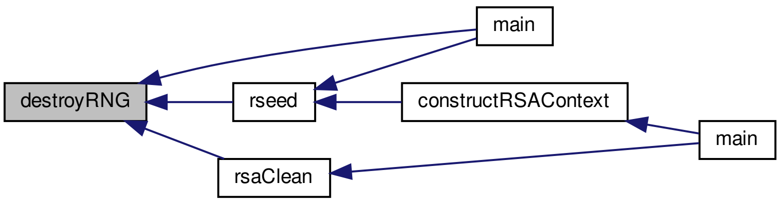
destroyRNG - Function used to deallocate all memory allocated for the random number generator, specifically the RC4 key stream state context structure.

Definition at line 61 of file randomNumberGenerator.c.

Here is the call graph for this function:



Here is the caller graph for this function:



7.4.2.2 rrand()

```
U8 rrand ( )
```

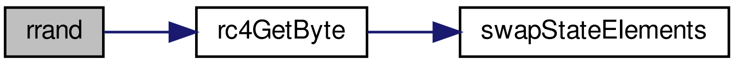
rrand - Function used to generate a random number of a single byte long.

Returns

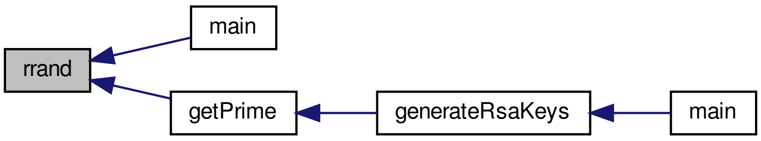
U8 - A random number of 8 bits long.

Definition at line 45 of file randomNumberGenerator.c.

Here is the call graph for this function:



Here is the caller graph for this function:



7.4.2.3 rseed()

```
void rseed (
              U8 * key,
              int keylen,
              int isKeyHex )
```

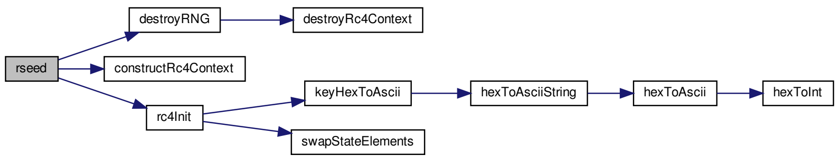
rseed - Function used to create a random number generator object and set the seed for the random number generator.

Parameters

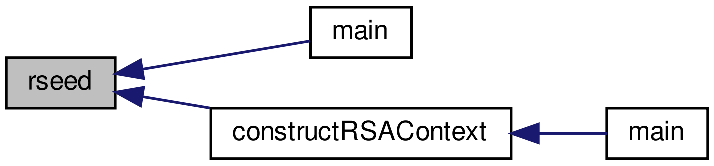
key	- uint8_t∗ - The key used to seed the random number generator.
keylen	- The length of the key
key.	
isKeyHex	- flag used to indicate if the key
key	is hex or ascii encoded.

Definition at line 32 of file randomNumberGenerator.c.

Here is the call graph for this function:



Here is the caller graph for this function:



7.4.3 Variable Documentation

7.4.3.1 rngContext

```
rc4ctx_t* rngContext
```

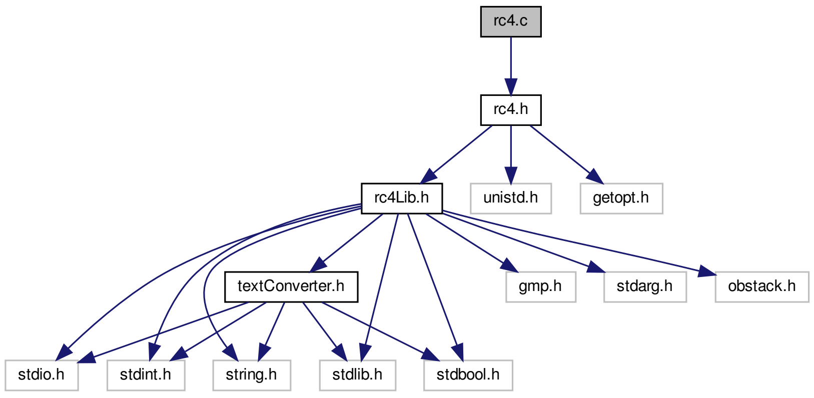
var - rngContext - The RC4 context state used for random number generation.

Definition at line 22 of file randomNumberGenerator.h.

7.5 rc4.c File Reference

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

Include dependency graph for rc4.c:



Functions

- int main (int argc, char ∗argv[ ])
- void printHelp ()

  printHelp - Function used to print the help menu for the rc4 utility
- void clearMemory (unsigned char ∗inputFileName, unsigned char ∗outputFileName, unsigned char ∗keyFile, unsigned char ∗key)

  clearMemory - Function used to deallocate all memory allocated for the rc4 utility.
- void verifyArgument (size_t argCounter, size_t argc, char ∗parameter)

  verifyArgument - Function used to verify if a paramter has an argument or not.

Variables

- const size_t RC4KEYLENGTH = 16

7.5.1 Detailed Description

Authors

Mohamed Ameen Omar (u16055323)
Douglas Healy (u16018100)
Llewellyn Moyse (u15100708)

Version

0.1

Date

2019-05-22

Copyright

Copyright (c) 2019

7.5.2 Function Documentation

7.5.2.1 clearMemory()

```
void clearMemory (
            unsigned char * inputFileName,
            unsigned char * outputFileName,
            unsigned char * keyFile,
            unsigned char * key )
```

clearMemory - Function used to deallocate all memory allocated for the rc4 utility.

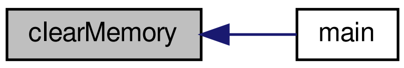
clearMemory - Function used to deallocate all memory allocated for the rsa keygen utility.

clearMemory - Function used to deallocate all memory allocated for the rsa encryption utility.

clearMemory - Function used to deallocate all memory allocated for the rsa decryption utility.

Definition at line 165 of file rc4.c.

Here is the caller graph for this function:

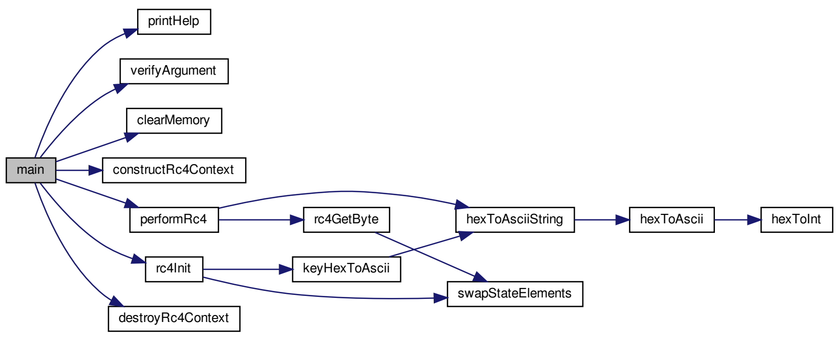


7.5.2.2 main()

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

Definition at line 18 of file rc4.c.

Here is the call graph for this function:



7.5.2.3 printHelp()

```
void printHelp ( )
```

printHelp - Function used to print the help menu for the rc4 utility

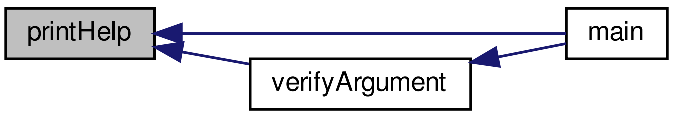
printHelp - Function used to print the help menu for the rsa kegen utility

printHelp - Function used to print the help menu for the rsa encrypt utility

printHelp - Function used to print the help menu for the rsa decrypt utility

Definition at line 148 of file rc4.c.

Here is the caller graph for this function:



7.5.2.4 verifyArgument()

```
void verifyArgument (
             size_t argCounter,
             size_t argc,
             char * parameter )
```

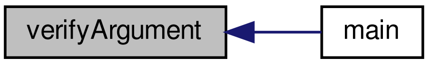
verifyArgument - Function used to verify if a paramter has an argument or not.

Parameters

argCounter	- The current index being verified for the commandline paramters.
argc	- The total number of commandline arguments.
parameter	- The parameter whose argument is being verified.

Definition at line 194 of file rc4.c.

Here is the caller graph for this function:



7.5.3 Variable Documentation

7.5.3.1 RC4KEYLENGTH

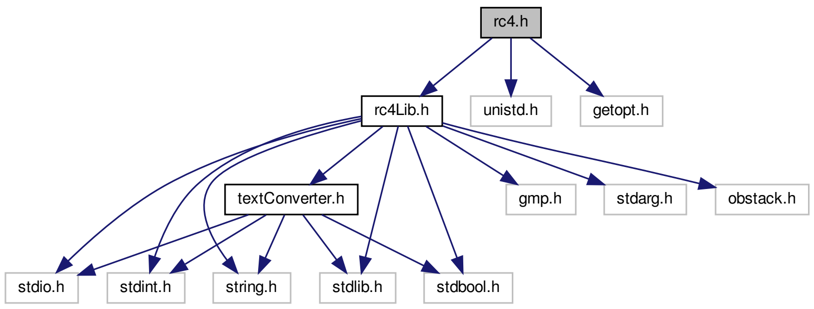
```
const size_t RC4KEYLENGTH = 16
```

Definition at line 16 of file rc4.c.

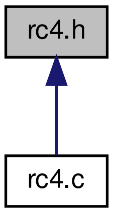
7.6 rc4.h File Reference

```
#include "rc4Lib.h"
#include <unistd.h>
#include <getopt.h>
```

Include dependency graph for rc4.h:



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



Functions

- void printHelp ()

  printHelp - Function used to print the help menu for the rc4 utility
- void clearMemory (unsigned char ∗inputFileName, unsigned char ∗outputFileName, unsigned char ∗keyFile, unsigned char ∗key)

  clearMemory - Function used to deallocate all memory allocated for the rc4 utility.
- void verifyArgument (size_t argCounter, size_t argc, char ∗parameter)

  verifyArgument - Function used to verify if a paramter has an argument or not.

Variables

- const size_t RC4KEYLENGTH

7.6.1 Detailed Description

Authors

Mohamed Ameen Omar (u16055323)
Douglas Healy (u16018100)
Llewellyn Moyse (u15100708)

Version

0.1

Date

2019-05-22

Copyright

Copyright (c) 2019

7.6.2 Function Documentation

7.6.2.1 clearMemory()

```
void clearMemory (
             unsigned char * publickeyfile,
             unsigned char * privatekeyfile,
             unsigned char * keyFile,
             unsigned char * key )
```

clearMemory - Function used to deallocate all memory allocated for the rc4 utility.

clearMemory - Function used to deallocate all memory allocated for the rc4 utility.

Definition at line 165 of file rc4.c.

7.6.2.2 printHelp()

```
void printHelp ( )
```

printHelp - Function used to print the help menu for the rc4 utility

printHelp - Function used to print the help menu for the rc4 utility

Definition at line 148 of file rc4.c.

7.6.2.3 verifyArgument()

```
void verifyArgument (
             size_t argCounter,
             size_t argc,
             char * parameter )
```

verifyArgument - Function used to verify if a paramter has an argument or not.

Parameters

argCounter	- The current index being verified for the commandline paramters.
argc	- The total number of commandline arguments.
parameter	- The parameter whose argument is being verified.

Definition at line 194 of file rc4.c.

7.6.3 Variable Documentation

7.6.3.1 RC4KEYLENGTH

```
const size_t RC4KEYLENGTH
```

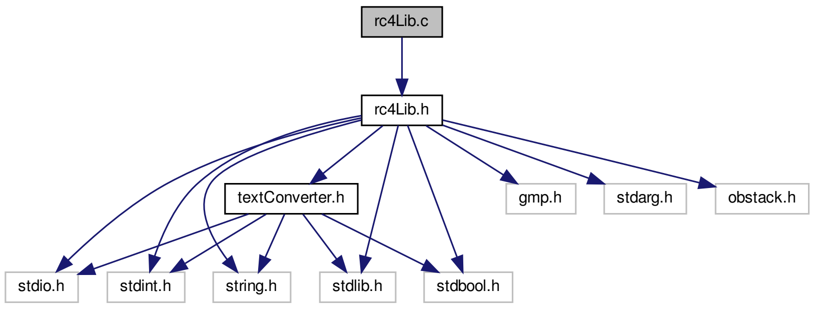
Definition at line 16 of file rc4.c.

7.7 rc4Lib.c File Reference

RC4 library implementation file. This file is used to perform encryption/decryption using the rc4 stream cipher. First an RC4 context is created using the constructRc4Context function. rc4 is then initialised with the rc4Init - by passing in the init key + key length and indicating if the key is hex or not. A byte of the key stream is received using the rc4GetByte (with the rc4 context) or encrypted/decrypted by using the appropriate function with the input and output file passed in.

```
#include "rc4Lib.h"
```

Include dependency graph for rc4Lib.c:



Functions

- rc4ctx_t ∗ constructRc4Context ()

  constructRc4Context - Function to construct the RC4 context structure and construct the state vector used for RC4 byte generation. The user must use the function in order to initialize the RC4 state. The caller must also ensure that they use the member function destroyRc4Context in order to deallocate all memory once the RC4 structure is no longer needed.
- void destroyRc4Context (rc4ctx_t ∗rc4Ctx)

  destroyRc4Context - Function used to deallocate all memory allocated for the rc4 context structure passed in as
- void rc4Init (rc4ctx_t ∗rc4Ctx, U8 ∗key, int keylen, uint8_t isKeyHex)

  rc4Init - Function used to initialize the state of the RC4 context and the state vector used for the RC4 byte generate. Requires the use of the function constructRc4Context in order to generate a RC4 context. If the initialization key provided in
- uint8_t rc4GetByte (rc4ctx_t ∗rc4Ctx)

  rc4GetByte - Function used to generate a single byte of the RC4 key stream for the RC4 context passed in as
- void swapStateElements (U8 ∗val1, U8 ∗val2)

  swapStateElements - Function to swap the contents of the uint8_t pointers passed in as parameters
- void performRc4 (unsigned char ∗inputFileName, unsigned char ∗outputFileName, rc4ctx_t ∗rc4Ctx, int is↩ TextHex)

  performRc4 - Function used to encrypt or decrypt the contents of the file

7.7.1 Detailed Description

RC4 library implementation file. This file is used to perform encryption/decryption using the rc4 stream cipher. First an RC4 context is created using the constructRc4Context function. rc4 is then initialised with the rc4Init - by passing in the init key + key length and indicating if the key is hex or not. A byte of the key stream is received using the rc4GetByte (with the rc4 context) or encrypted/decrypted by using the appropriate function with the input and output file passed in.

Authors

Mohamed Ameen Omar (u16055323)
Douglas Healy (u16018100)
Llewellyn Moyse (u15100708)

Version

0.1

Date

2019-05-22

Copyright

Copyright (c) 2019

7.7.2 Function Documentation

7.7.2.1 constructRc4Context()

`rc4ctx_t* constructRc4Context ( )`

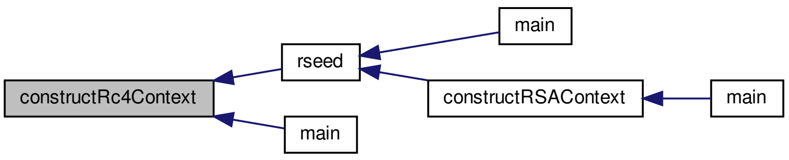
constructRc4Context - Function to construct the RC4 context structure and construct the state vector used for RC4 byte generation. The user must use the function in order to initialize the RC4 state. The caller must also ensure that they use the member function destroyRc4Context in order to deallocate all memory once the RC4 structure is no longer needed.

Returns

rc4ctx_t∗ - a pointer to the rc4 context object created.

Definition at line 28 of file rc4Lib.c.

Here is the caller graph for this function:



7.7.2.2 destroyRc4Context()

```
void destroyRc4Context (
            rc4ctx_t * rc4Ctx )
```

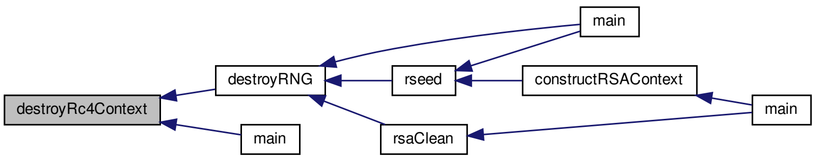
destroyRc4Context - Function used to deallocate all memory allocated for the rc4 context structure passed in as

Parameters

rc4Ctx.	Sets the parameter
rc4Ctx	to NULL.
rc4Ctx	- The RC4 context structure to deallocate and clean.

Definition at line 49 of file rc4Lib.c.

Here is the caller graph for this function:



7.7.2.3 performRc4()

```
void performRc4 (
            unsigned char * inputFileName,
            unsigned char * outputFileName,
            rc4ctx_t * rc4Ctx,
            int isTextHex )
```

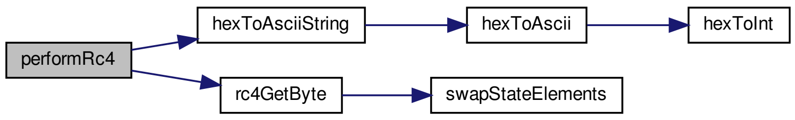
performRc4 - Function used to encrypt or decrypt the contents of the file

Parameters

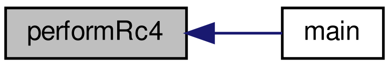
inputFileName	and write the result to the file
outputFileName.	The encryption or decryption is done using RC4 encryption or decryption. Each character in the input file is read in and a single byte for the RC4 key stream is generated. A single byte for the plaintext or Ciphertext and XOR'ed with a single byte from the RC4 key stream to generate a the corresponding ciphertext or plaintext. The bytes used during the RC4 encryption or decryption are generated from the RC4 context passed in as
rc4Ctx.	RC4 encryption
inputFileName	- unsigned char∗ - pointer to a string containing the path to the input file.
outputFileName	- unsigned char∗ - pointer to a string containing the path to the output file.
rc4Ctx	- rc4ctx_t∗ - a pointer to the RC4 context to use during encryption or decryption.
isTextHex	- flag used to determine if the input is encoded using ascii or hex encoding.

Definition at line 140 of file rc4Lib.c.

Here is the call graph for this function:



Here is the caller graph for this function:



7.7.2.4 rc4GetByte()

```
uint8_t rc4GetByte (
            rc4ctx_t * rc4Ctx )
```

rc4GetByte - Function used to generate a single byte of the RC4 key stream for the RC4 context passed in as

Parameters

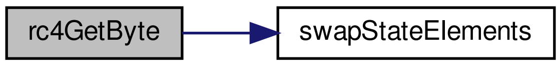
rc4Ctx.	Returns the single byte as a single uint8_t.
rc4Ctx	- rc4ctx_t∗ - a pointer to the RC4 context from which to generate the single byte.

Returns

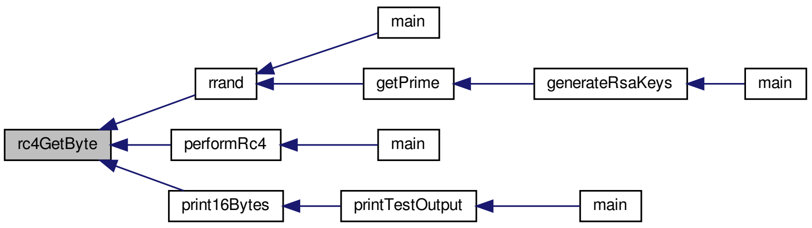
uint8_t - A single byte in the RC4 key stream.

Definition at line 105 of file rc4Lib.c.

Here is the call graph for this function:



Here is the caller graph for this function:



7.7.2.5 rc4Init()

```
void rc4Init (
             rc4ctx_t * rc4Ctx,
             U8 * key,
             int keylen,
             uint8_t isKeyHex )
```

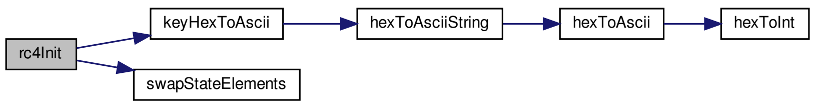
rc4Init - Function used to initialize the state of the RC4 context and the state vector used for the RC4 byte generate. Requires the use of the function constructRc4Context in order to generate a RC4 context. If the initialization key provided in

Parameters

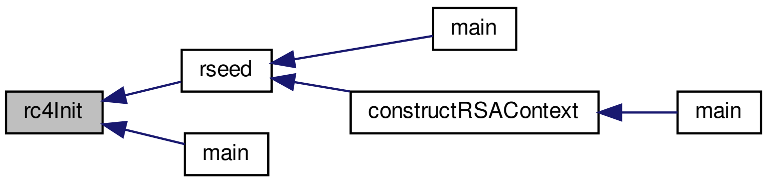
key	is a hex string, it is converted to ascii string before initialization.
rc4Ctx	- rc4ctx_t∗ - a pointer to the RC4 context structure to initialize.
key	- uint8_t∗ - a pointer to the initialization key to be used.
keylen	- int - the length of the key provided
key.	
isKeyHex	- uint8_t - a flag to determine if the key passed in as
key	is a hex string or ascii string.

Definition at line 68 of file rc4Lib.c.

Here is the call graph for this function:



Here is the caller graph for this function:



7.7.2.6 swapStateElements()

```
void swapStateElements (
            U8 * val1,
            U8 * val2 )
```

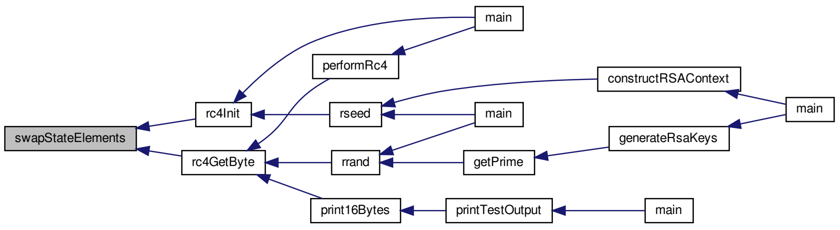
swapStateElements - Function to swap the contents of the uint8_t pointers passed in as parameters

Parameters

val1	and
val2.	
val1	- uint8_t∗ - pointer to a uint8_t variable whose contents to switch with
val2.	
val2	- uint8_t∗ - pointer to a uint8_t variable whose contents to switch with
val1.	

Definition at line 120 of file rc4Lib.c.

Here is the caller graph for this function:

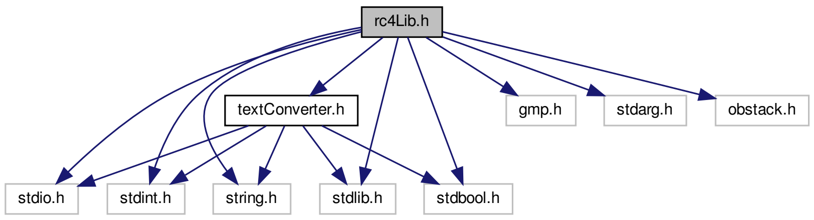


7.8 rc4Lib.h File Reference

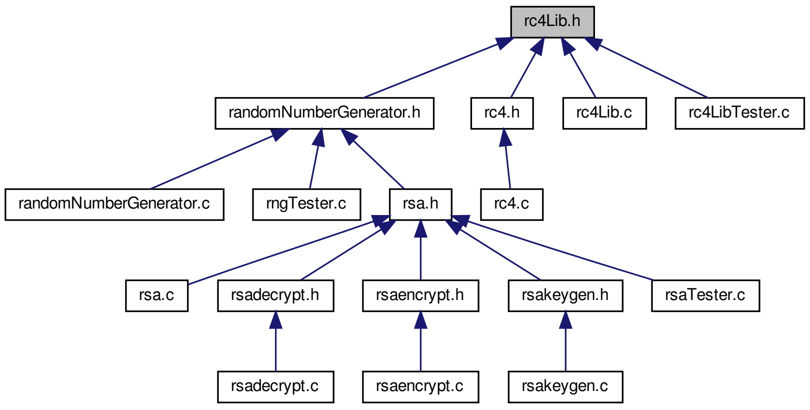
RC4 library function prototype file. This file is used to perform encryption/decryption using the rc4 stream cipher. First an RC4 context is created using the constructRc4Context function. rc4 is then initialised with the rc4Init - by passing in the init key + key length and indicating if the key is hex or not. A byte of the key stream is received using the rc4GetByte (with the rc4 context) or encrypted/decrypted by using the appropriate function with the input and output file passed in.

```
#include <stdio.h>
#include <gmp.h>
#include <stdarg.h>
#include <obstack.h>
#include <stdint.h>
#include <string.h>
#include <stdlib.h>
#include <stdbool.h>
#include "textConverter.h"
```

Include dependency graph for rc4Lib.h:



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



Data Structures

- struct rc4ctx_t

    struct rc4ctx_t - Structure used to retain the context of the RC4 key stream generator.

Macros

- #define RC4_STATE_SIZE 256

Typedefs

- typedef unsigned char U8

Functions

- rc4ctx_t ∗ constructRc4Context ()

    constructRc4Context - Function to construct the RC4 context structure and construct the state vector used for RC4 byte generation. The user must use the function in order to initialize the RC4 state. The caller must also ensure that they use the member function destroyRc4Context in order to deallocate all memory once the RC4 structure is no longer needed.
- void destroyRc4Context (rc4ctx_t ∗rc4Ctx)

    destroyRc4Context - Function used to deallocate all memory allocated for the rc4 context structure passed in as
- void rc4Init (rc4ctx_t ∗rc4Ctx, U8 ∗key, int keylen, uint8_t isKeyHex)

    rc4Init - Function used to initialize the state of the RC4 context and the state vector used for the RC4 byte generate. Requires the use of the function constructRc4Context in order to generate a RC4 context. If the initialization key provided in
- U8 rc4GetByte (rc4ctx_t ∗rc4Ctx)

    rc4GetByte - Function used to generate a single byte of the RC4 key stream for the RC4 context passed in as
- void swapStateElements (U8 ∗val1, U8 ∗val2)

    swapStateElements - Function to swap the contents of the uint8_t pointers passed in as parameters
- void performRc4 (unsigned char ∗inputFileName, unsigned char ∗outputFileName, rc4ctx_t ∗rc4Ctx, int is↩ TextHex)

    performRc4 - Function used to encrypt or decrypt the contents of the file

7.8.1 Detailed Description

RC4 library function prototype file. This file is used to perform encryption/decryption using the rc4 stream cipher. First an RC4 context is created using the constructRc4Context function. rc4 is then initialised with the rc4Init - by passing in the init key + key length and indicating if the key is hex or not. A byte of the key stream is received using the rc4GetByte (with the rc4 context) or encrypted/decrypted by using the appropriate function with the input and output file passed in.

Authors

Mohamed Ameen Omar (u16055323)
Douglas Healy (u16018100)
Llewellyn Moyse (u15100708)

Version

0.1

Date

2019-05-22

Copyright

Copyright (c) 2019

7.8.2 Macro Definition Documentation

7.8.2.1 RC4_STATE_SIZE

```
#define RC4_STATE_SIZE 256
```

Definition at line 30 of file rc4Lib.h.

7.8.3 Typedef Documentation

7.8.3.1 U8

```
typedef unsigned char U8
```

Definition at line 32 of file rc4Lib.h.

7.8.4 Function Documentation

7.8.4.1 constructRc4Context()

```
rc4ctx_t* constructRc4Context ( )
```

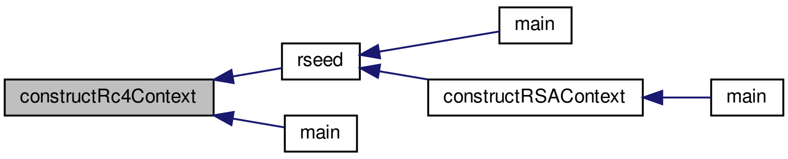
constructRc4Context - Function to construct the RC4 context structure and construct the state vector used for RC4 byte generation. The user must use the function in order to initialize the RC4 state. The caller must also ensure that they use the member function destroyRc4Context in order to deallocate all memory once the RC4 structure is no longer needed.

Returns

rc4ctx_t∗ - a pointer to the rc4 context object created.

Definition at line 28 of file rc4Lib.c.

Here is the caller graph for this function:



7.8.4.2 destroyRc4Context()

```
void destroyRc4Context (
            rc4ctx_t * rc4Ctx )
```

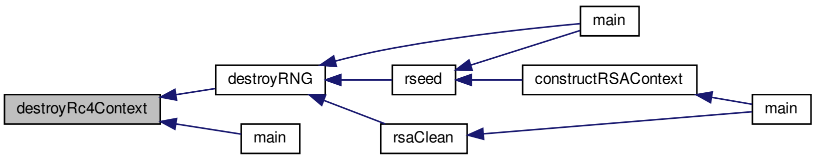
destroyRc4Context - Function used to deallocate all memory allocated for the rc4 context structure passed in as

Parameters

rc4Ctx.	Sets the parameter
rc4Ctx	to NULL.
rc4Ctx	- The RC4 context structure to deallocate and clean.

Definition at line 49 of file rc4Lib.c.

Here is the caller graph for this function:



7.8.4.3 performRc4()

```
void performRc4 (
            unsigned char * inputFileName,
            unsigned char * outputFileName,
            rc4ctx_t * rc4Ctx,
            int isTextHex )
```

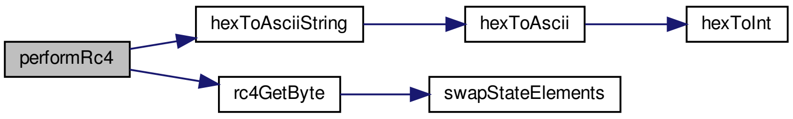
performRc4 - Function used to encrypt or decrypt the contents of the file

Parameters

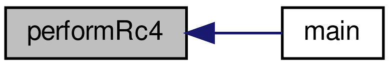
inputFileName	and write the result to the file
outputFileName.	The encryption or decryption is done using RC4 encryption or decryption. Each character in the input file is read in and a single byte for the RC4 key stream is generated. A single byte for the plaintext or Ciphertext and XOR'ed with a single byte from the RC4 key stream to generate a the corresponding ciphertext or plaintext. The bytes used during the RC4 encryption or decryption are generated from the RC4 context passed in as
rc4Ctx.	RC4 encryption
inputFileName	- unsigned char* - pointer to a string containing the path to the input file.
outputFileName	- unsigned char* - pointer to a string containing the path to the output file.
rc4Ctx	- rc4ctx_t* - a pointer to the RC4 context to use during encryption or decryption.
isTextHex	- flag used to determine if the input is encoded using ascii or hex encoding.

Definition at line 140 of file rc4Lib.c.

Here is the call graph for this function:



Here is the caller graph for this function:



7.8.4.4 rc4GetByte()

```
U8 rc4GetByte (
             rc4ctx_t * rc4Ctx )
```

rc4GetByte - Function used to generate a single byte of the RC4 key stream for the RC4 context passed in as

Parameters

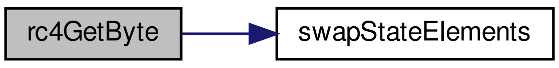
rc4Ctx.	Returns the single byte as a single uint8_t.
rc4Ctx	- rc4ctx_t∗ - a pointer to the RC4 context from which to generate the single byte.

Returns

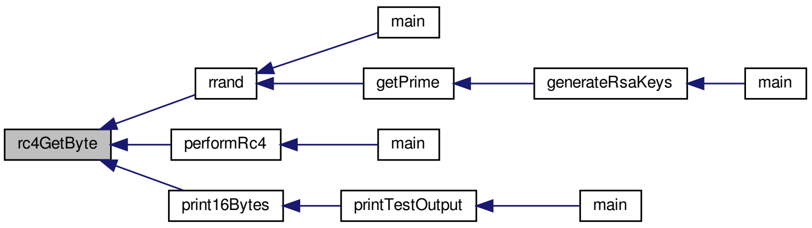
uint8_t - A single byte in the RC4 key stream.

Definition at line 105 of file rc4Lib.c.

Here is the call graph for this function:



Here is the caller graph for this function:



7.8.4.5 rc4Init()

```
void rc4Init (
            rc4ctx_t * rc4Ctx,
            U8 * key,
            int keylen,
            uint8_t isKeyHex )
```

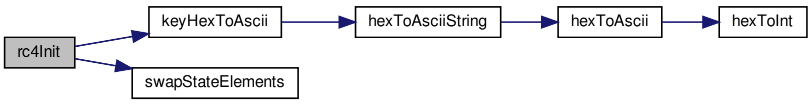
rc4Init - Function used to initialize the state of the RC4 context and the state vector used for the RC4 byte generate. Requires the use of the function constructRc4Context in order to generate a RC4 context. If the initialization key provided in

Parameters

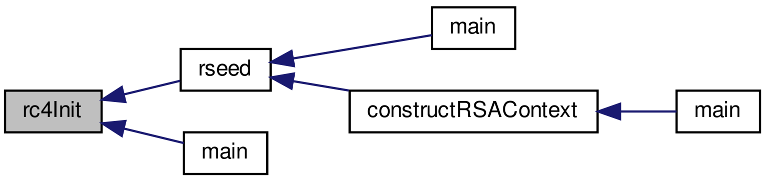
key	is a hex string, it is converted to ascii string before initialization.
rc4Ctx	- rc4ctx_t∗ - a pointer to the RC4 context structure to initialize.
key	- uint8_t∗ - a pointer to the initialization key to be used.
keylen	- int - the length of the key provided
key.	
isKeyHex	- uint8_t - a flag to determine if the key passed in as
key	is a hex string or ascii string.

Definition at line 68 of file rc4Lib.c.

Here is the call graph for this function:



Here is the caller graph for this function:



7.8.4.6 swapStateElements()

```
void swapStateElements (
            U8 * val1,
            U8 * val2 )
```

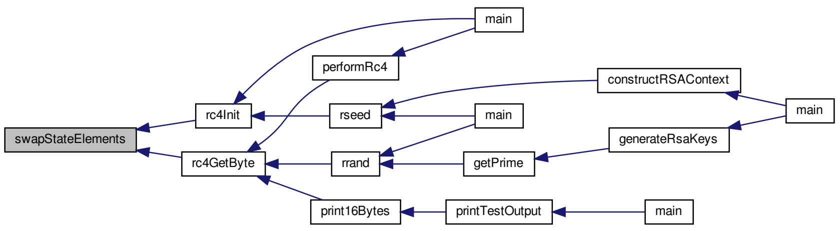
swapStateElements - Function to swap the contents of the uint8_t pointers passed in as parameters

Parameters

val1	and
val2.	
val1	- uint8_t∗ - pointer to a uint8_t variable whose contents to switch with
val2.	
val2	- uint8_t∗ - pointer to a uint8_t variable whose contents to switch with
val1.	

Definition at line 120 of file rc4Lib.c.

Here is the caller graph for this function:

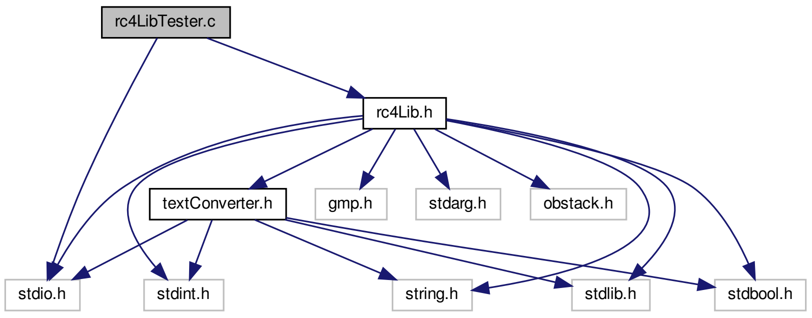


7.9 rc4LibTester.c File Reference

```
#include "rc4Lib.h"
```

`#include "stdio.h"`

Include dependency graph for rc4LibTester.c:



Functions

- void print16Bytes (rc4ctx_t ∗rc4Ctx)
- void printTestOutput (rc4ctx_t ∗rc4Ctx)
- int main (int argc, char ∗argv[ ])

7.9.1 Detailed Description

Authors

Mohamed Ameen Omar (u16055323)
Douglas Healy (u16018100)
Llewellyn Moyse (u15100708)

Version

0.1

Date

2019-05-23

Copyright

Copyright (c) 2019

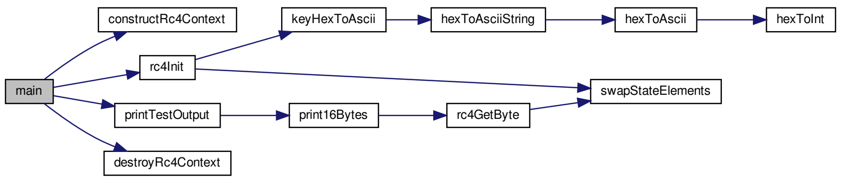
7.9.2 Function Documentation

7.9.2.1 main()

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

Definition at line 39 of file rc4LibTester.c.

Here is the call graph for this function:

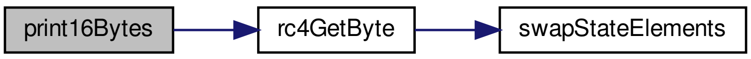


7.9.2.2 print16Bytes()

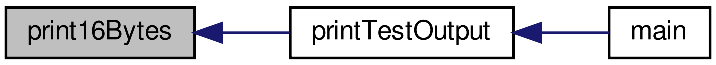
```
void print16Bytes (
            rc4ctx_t * rc4Ctx )
```

Definition at line 16 of file rc4LibTester.c.

Here is the call graph for this function:



Here is the caller graph for this function:

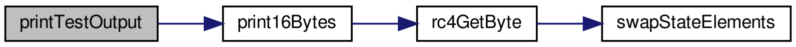


7.9.2.3 printTestOutput()

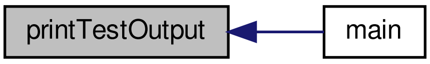
```
void printTestOutput (
             rc4ctx_t * rc4Ctx )
```

Definition at line 26 of file rc4LibTester.c.

Here is the call graph for this function:



Here is the caller graph for this function:



7.10 rc4Test.py File Reference

Namespaces

- rc4Test

Variables

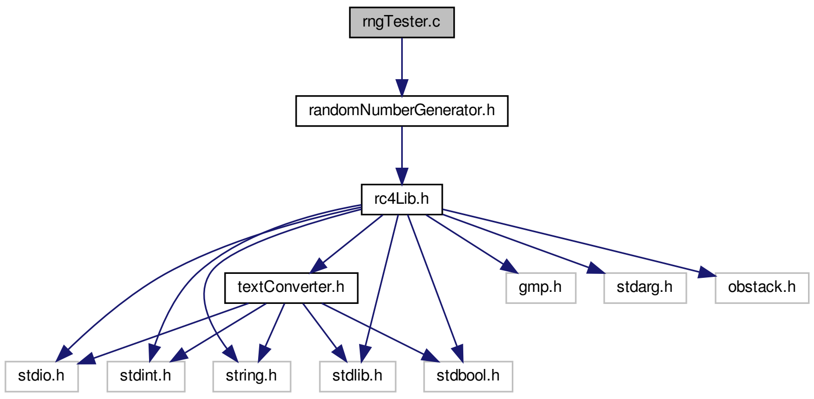
- string inputfileName = "temp"
- string outputFileName = "temp2"
- string keyFile = ""
- temp = subprocess.call(["./rc4","-fi", inputfileName, "-fo", outputFileName])

7.11 README.md File Reference

7.12 rngTester.c File Reference

```
#include "randomNumberGenerator.h"
```

Include dependency graph for rngTester.c:



Functions

- int main (int argc, char ∗argv[ ])

7.12.1 Detailed Description

Authors

Mohamed Ameen Omar (u16055323)
Douglas Healy (u16018100)
Llewellyn Moyse (u15100708)

Version

0.1

Date

2019-05-23

Copyright

Copyright (c) 2019

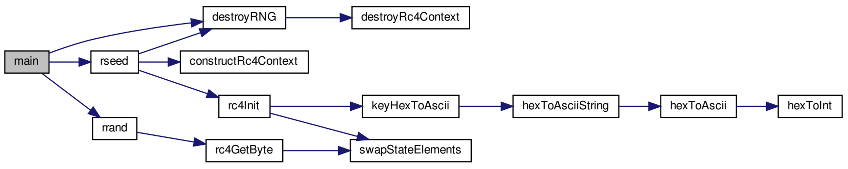
7.12.2 Function Documentation

7.12.2.1 main()

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

Definition at line 17 of file rngTester.c.

Here is the call graph for this function:

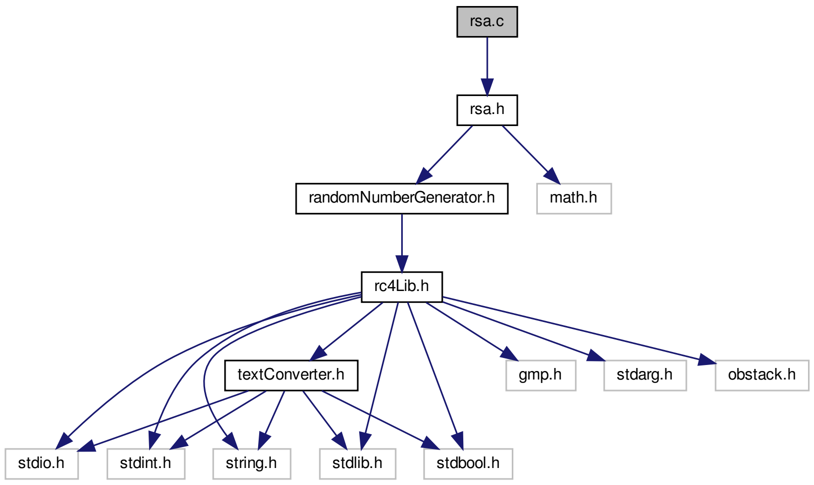


7.13 rsa.c File Reference

RSA library implementation file. This file contains the necessary functionality to perform RSA key generation as well as encryption/decryption. The functions in this file consist of RSA key generation, RSA encryption, RSA decryption, Getting prime numbers and writing the keys to a file.

```
#include "rsa.h"
```

Include dependency graph for rsa.c:



Functions

- rsactx_t ∗ constructRSAContext (unsigned char ∗initKey, uint8_t initKeyLength, int isKeyHex, int numBits)

  constructRSAContext - Function used to construct the RSA context structure used for RSA key-pair generation. The function allocates all memory required. In addition, the function initializes the random number generator (RC4 key stream) using the key passed in as
- void rsaInit (rsactx_t ∗rsaCtx)

  rsaInit - Function used to initialize the RSA context state structure passed in as
- void generateRsaKeys (rsactx_t ∗rsaCtx)

  generateRsaKeys - Function used to generate the RSA public and private key-pair according to the specifications within the RSA state passed in as
- void getPrime (mpz_t p, int bits)

  getPrime - Function used to generate a prime number of length
- void rsaEncrypt (unsigned char ∗outputFile, unsigned char ∗publicKeyFile, unsigned char ∗plainText, size_t isPlaintextHex)

  rsaEncrypt - Function used to encrypt the plaintext passed in as
- void rsaDecrypt (unsigned char ∗outputFile, unsigned char ∗privateKeyFile, unsigned char ∗cipherText)

  rsaDecrypt - Function used to decrypt the ciphertext passed in as
- void rsaWriteKeysToFile (rsactx_t ∗rsaCtx, unsigned char ∗publicKeyFileName, unsigned char ∗privateKey↩ FileName)

  rsaWriteKeysToFile - Function used to write the public and private keys store in the RSA Context passed in as
- void rsaClean (rsactx_t ∗rsaCtx)

  rsaClean - Function used to deallocate all memory allocated for the RSA context state structure in

Variables

- const int64_t CONSTANTE = 65537

7.13.1 Detailed Description

RSA library implementation file. This file contains the necessary functionality to perform RSA key generation as well as encryption/decryption. The functions in this file consist of RSA key generation, RSA encryption, RSA decryption, Getting prime numbers and writing the keys to a file.

Authors

Mohamed Ameen Omar (u16055323)
Douglas Healy (u16018100)
Llewellyn Moyse (u15100708)

Version

0.1

Date

2019-05-22

Copyright

Copyright (c) 2019

7.13.2 Function Documentation

7.13.2.1 constructRSAContext()

```
rsactx_t* constructRSAContext (
             unsigned char * initKey,
             uint8_t initKeyLength,
             int isKeyHex,
             int numBits )
```

constructRSAContext - Function used to construct the RSA context structure used for RSA key-pair generation. The function allocates all memory required. In addition, the function initializes the random number generator (RC4 key stream) using the key passed in as

Parameters

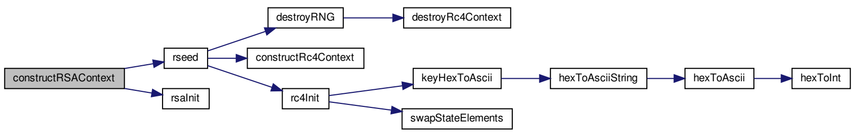
initKey,used	for the generation of the RSA key-pairs. The function calls rsaInit function to aid in initialization. The caller must use cleanRSA in order to deallocate all memory once the key-pair has been generated.
unsigned	char∗ - initKey - The key used to initialize the random number generator.
initKeyLength	- uint8_t - The length of the key passed in as
initKey.	
isKeyHex	- int - a flag indicating whether the key
initKey	is a hex string or ascii encoded string.
numBits	- int - the number of bits required for the public and private RSA key pair to be generated.

Returns

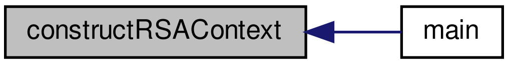
rsactx_t∗ - A pointer to the RSA context structure used to store the state of the RSA key generation.

Definition at line 35 of file rsa.c.

Here is the call graph for this function:



Here is the caller graph for this function:



7.13.2.2 generateRsaKeys()

```
void generateRsaKeys (
            rsactx_t * rsaCtx )
```

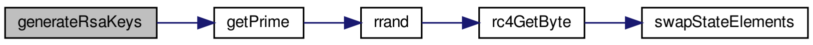
generateRsaKeys - Function used to generate the RSA public and private key-pair according to the specifications within the RSA state passed in as

Parameters

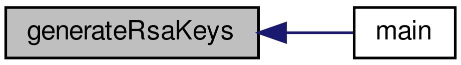
rsaCtx.	Makes use of the mpz libraries in order to compute the prime numbers used for the p and q variables used during RSA key generation. The function does check for negative values for the "d" parameter as a result of under and overflows and makes the required adjustments. Stores the RSA key pair and the parameters used during RSA key generation in the RSA state structure.
rsaCtx	- rsactx_t∗ - The RSA context state to use for the RSA key-pair generation.

Definition at line 77 of file rsa.c.

Here is the call graph for this function:



Here is the caller graph for this function:



7.13.2.3 getPrime()

```
void getPrime (
            mpz_t p,
            int bits )
```

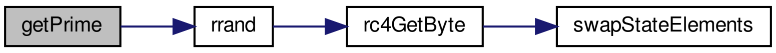
getPrime - Function used to generate a prime number of length

Parameters

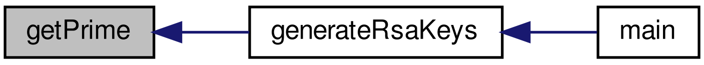
bits	and store it in
p.	Used for RSA key-pair generation.
bits	should be half the length of the total length of the key required for the RSA keys. The function generates
bits	- 1 random numbers using the RC4 random key stream generator and uses the LSB of each random number generated as a bit in the in the final prime number. Once the random number of bits length is generated, the mpz_nextprime is used to get the closest prime number to the random number generated and store it in
p.	
p	
bits	

Definition at line 133 of file rsa.c.

Here is the call graph for this function:



Here is the caller graph for this function:



7.13.2.4 rsaClean()

```
void rsaClean (
            rsactx_t * rsaCtx )
```

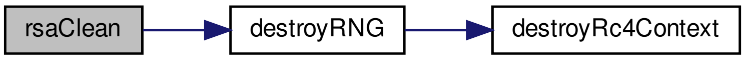
rsaClean - Function used to deallocate all memory allocated for the RSA context state structure in

Parameters

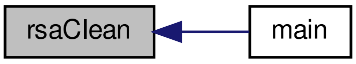
rsaCtx.	In addition deallocates all memory used for the random number generator.
rsaCtx	- rsactx_t∗ - The RSA context state to deallocate.

Definition at line 382 of file rsa.c.

Here is the call graph for this function:


$$\begin{tikzcd}\boxed{\text{rsaClean}} \arrow[r] & \boxed{\text{destroyRNG}} \arrow[r] & \boxed{\text{destroyRc4Context}}\end{tikzcd}$$

Here is the caller graph for this function:


$$\begin{tikzcd}\boxed{\text{rsaClean}} & \boxed{\text{main}} \arrow[l]\end{tikzcd}$$

7.13.2.5 rsaDecrypt()

```
void rsaDecrypt (
            unsigned char ∗ outputFile,
            unsigned char ∗ privateKeyFile,
            unsigned char ∗ cipherText )
```

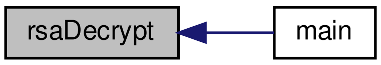
rsaDecrypt - Function used to decrypt the ciphertext passed in as

Parameters

cipherText	using RSA decryption and write the resulting plaintext to the file
outputFile	as a string using ascii plaintext encoding. Function treats the entire cipherText as the a single decimal value and performs the RSA decryption. The function reads in the private key and writes the result to the output file.
outputFile	- unsigned char∗ - File to write the plaintext to.
privateKeyFile	- unsigned char∗ - The file containing the private key to use during RSA decryption. The "n" paramter should be on the first line, followed by the newline character thereafter the "d" paramter should be placed in the private key file.
cipherText	- unsigned char∗ - The ciphertext to decrypt.

Definition at line 256 of file rsa.c.

Here is the caller graph for this function:



7.13.2.6 rsaEncrypt()

```
void rsaEncrypt (
            unsigned char * outputFile,
            unsigned char * publicKeyFile,
            unsigned char * plainText,
            size_t isPlaintextHex )
```

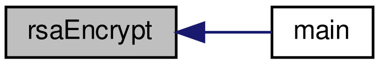
rsaEncrypt - Function used to encrypt the plaintext passed in as

Parameters

plainText	using RSA encryption and write the resulting ciphertext to the file
outputFile	in decimal. Function treats the entire plainText as the a single decimal value and performs the RSA encryption. The function reads in the public key and writes the result to the output file.
outputFile	- unsigned char∗ - File to write the ciphertext to.
publicKeyFile	- unsigned char∗ - The file containing the public key to use during RSA encryption. The "n" paramter should be on the first line, followed by the newline character thereafter the "e" paramter should be placed in the public key file.
plainText	- unsigned char∗ - The plaintext to encrypt.
isPlaintextHex	- size_t - a flag used to indicate if the plaintext is encoded using ascii or hex encoding.

Definition at line 175 of file rsa.c.

Here is the caller graph for this function:



7.13.2.7 rsaInit()

```
void rsaInit (
            rsactx_t * rsaCtx )
```

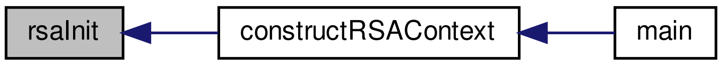
rsaInit - Function used to initialize the RSA context state structure passed in as

Parameters

rsaCtx.	Used as a helper function for the constructRSAContext function. Does not need to be explicitly called by the user. Initializes all mpz library variables used as required.
rsaCtx	- rsactx_t∗ - A pointer to the RSA context state structure to initialize.

Definition at line 54 of file rsa.c.

Here is the caller graph for this function:



7.13.2.8 rsaWriteKeysToFile()

```
void rsaWriteKeysToFile (
            rsactx_t * rsaCtx,
            unsigned char * publicKeyFileName,
            unsigned char * privateKeyFileName )
```

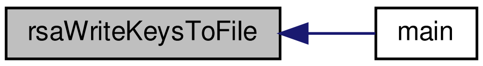
rsaWriteKeysToFile - Function used to write the public and private keys store in the RSA Context passed in as

Parameters

rsaCtx,to	the
publicKeyFileName	and
privateKeyFileName	respectively. The RSA private and public keys are written to the files in accordance with the practical specification. With the n paramter followed by a newline character, followed by d/e and finally a newline character.
rsaCtx	- rsactx_t∗ - The RSA context containing the public and private key pair to be written.
publicKeyFileName	- unsigned char∗ - The file to write the RSA public key to.
privateKeyFileName	- unsigned char∗ - The file to write the RSA private key to.

Definition at line 345 of file rsa.c.

Here is the caller graph for this function:



7.13.3 Variable Documentation

7.13.3.1 CONSTANTE

```
const int64_t CONSTANTE = 65537
```

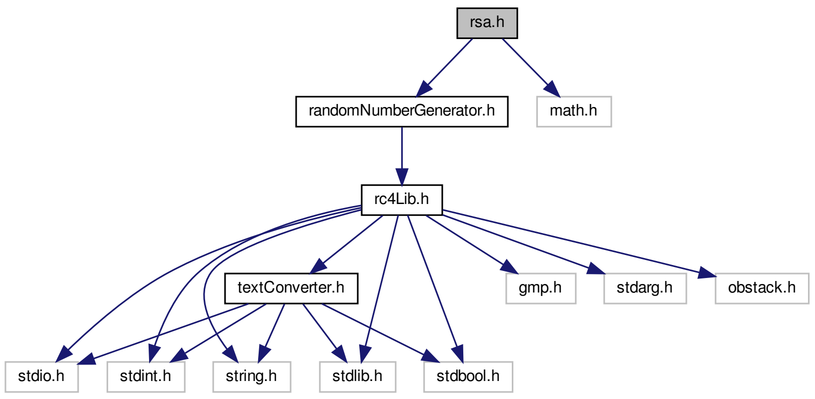
Definition at line 21 of file rsa.c.

7.14 rsa.h File Reference

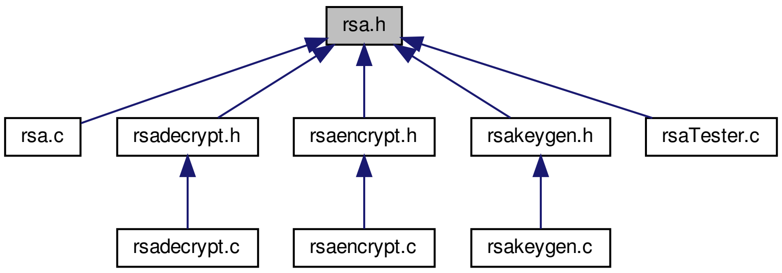
RSA library function prototype file. This file contains the necessary functionality to perform RSA key generation as well as encryption/decryption. The functions in this file consist of RSA key generation, RSA encryption, RSA decryption, Getting prime numbers and writing the keys to a file.

```
#include "randomNumberGenerator.h"
#include <math.h>
```

Include dependency graph for rsa.h:



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



Data Structures

- struct rsactx_t

Functions

- void rsaInit (rsactx_t ∗rsaCtx)

  rsaInit - Function used to initialize the RSA context state structure passed in as
- rsactx_t ∗ constructRSAContext (unsigned char ∗initKey, uint8_t initKeyLength, int isKeyHex, int numBits)

  constructRSAContext - Function used to construct the RSA context structure used for RSA key-pair generation. The function allocates all memory required. In addition, the function initializes the random number generator (RC4 key stream) using the key passed in as
- void rsaWriteKeysToFile (rsactx_t ∗rsaCtx, unsigned char ∗publicKeyFileName, unsigned char ∗privateKey←FileName)

  rsaWriteKeysToFile - Function used to write the public and private keys store in the RSA Context passed in as
- void generateRsaKeys (rsactx_t ∗rsaCtx)

  generateRsaKeys - Function used to generate the RSA public and private key-pair according to the specifications within the RSA state passed in as
- void rsaEncrypt (unsigned char ∗outputFile, unsigned char ∗publicKeyFile, unsigned char ∗plainText, size_t isPlaintextHex)

  rsaEncrypt - Function used to encrypt the plaintext passed in as
- void rsaDecrypt (unsigned char ∗outputFile, unsigned char ∗privateKeyFile, unsigned char ∗cipherText)

  rsaDecrypt - Function used to decrypt the ciphertext passed in as
- void getPrime (mpz_t p, int bits)

  getPrime - Function used to generate a prime number of length
- void rsaClean (rsactx_t ∗rsaCtx)

  rsaClean - Function used to deallocate all memory allocated for the RSA context state structure in

7.14.1 Detailed Description

RSA library function prototype file. This file contains the necessary functionality to perform RSA key generation as well as encryption/decryption. The functions in this file consist of RSA key generation, RSA encryption, RSA decryption, Getting prime numbers and writing the keys to a file.

Authors

Mohamed Ameen Omar (u16055323)
Douglas Healy (u16018100)
Llewellyn Moyse (u15100708)

Version

0.1

Date

2019-05-22

Copyright

Copyright (c) 2019

7.14.2 Function Documentation

7.14.2.1 constructRSAContext()

```
rsactx_t* constructRSAContext (
            unsigned char * initKey,
            uint8_t initKeyLength,
            int isKeyHex,
            int numBits )
```

constructRSAContext - Function used to construct the RSA context structure used for RSA key-pair generation. The function allocates all memory required. In addition, the function initializes the random number generator (RC4 key stream) using the key passed in as

Parameters

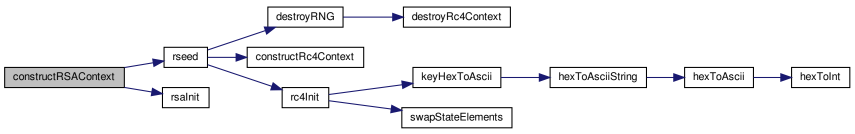
initKey,used	for the generation of the RSA key-pairs. The function calls rsaInit function to aid in initialization. The caller must use cleanRSA in order to deallocate all memory once the key-pair has been generated.
unsigned	char∗ - initKey - The key used to initialize the random number generator.
initKeyLength	- uint8_t - The length of the key passed in as
initKey.	
isKeyHex	- int - a flag indicating whether the key
initKey	is a hex string or ascii encoded string.
numBits	- int - the number of bits required for the public and private RSA key pair to be generated.

Returns

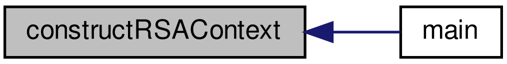
rsactx_t∗ - A pointer to the RSA context structure used to store the state of the RSA key generation.

Definition at line 35 of file rsa.c.

Here is the call graph for this function:



Here is the caller graph for this function:



7.14.2.2 generateRsaKeys()

```
void generateRsaKeys (
             rsactx_t * rsaCtx )
```

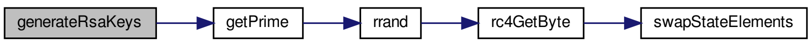
generateRsaKeys - Function used to generate the RSA public and private key-pair according to the specifications within the RSA state passed in as

Parameters

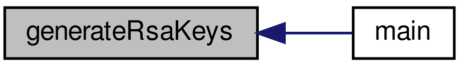
rsaCtx.	Makes use of the mpz libraries in order to compute the prime numbers used for the p and q variables used during RSA key generation. The function does check for negative values for the "d" parameter as a result of under and overflows and makes the required adjustments. Stores the RSA key pair and the parameters used during RSA key generation in the RSA state structure.
rsaCtx	- rsactx_t∗ - The RSA context state to use for the RSA key-pair generation.

Definition at line 77 of file rsa.c.

Here is the call graph for this function:



Here is the caller graph for this function:



7.14.2.3 getPrime()

```
void getPrime (
             mpz_t p,
             int bits )
```

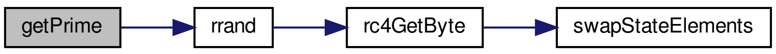
getPrime - Function used to generate a prime number of length

Parameters

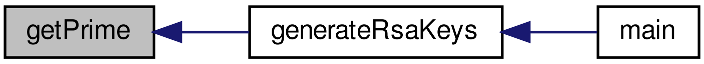
bits	and store it in
p.	Used for RSA key-pair generation.
bits	should be half the length of the total length of the key required for the RSA keys. The function generates
bits	- 1 random numbers using the RC4 random key stream generator and uses the LSB of each random number generated as a bit in the in the final prime number. Once the random number of bits length is generated, the mpz_nextprime is used to get the closest prime number to the random number generated and store it in
p.	
p	
bits	

Definition at line 133 of file rsa.c.

Here is the call graph for this function:



Here is the caller graph for this function:



7.14.2.4 rsaClean()

```
void rsaClean (
            rsactx_t * rsaCtx )
```

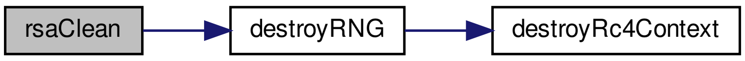
rsaClean - Function used to deallocate all memory allocated for the RSA context state structure in

Parameters

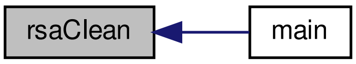
rsaCtx.	In addition deallocates all memory used for the random number generator.
rsaCtx	- rsactx_t* - The RSA context state to deallocate.

Definition at line 382 of file rsa.c.

Here is the call graph for this function:



Here is the caller graph for this function:



7.14.2.5 rsaDecrypt()

```
void rsaDecrypt (
             unsigned char * outputFile,
             unsigned char * privateKeyFile,
             unsigned char * cipherText )
```

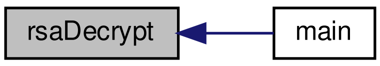
rsaDecrypt - Function used to decrypt the ciphertext passed in as

Parameters

cipherText	using RSA decryption and write the resulting plaintext to the file
outputFile	as a string using ascii plaintext encoding. Function treats the entire cipherText as the a single decimal value and performs the RSA decryption. The function reads in the private key and writes the result to the output file.
outputFile	- unsigned char∗ - File to write the plaintext to.
privateKeyFile	- unsigned char∗ - The file containing the private key to use during RSA decryption. The "n" paramter should be on the first line, followed by the newline character thereafter the "d" paramter should be placed in the private key file.
cipherText	- unsigned char∗ - The ciphertext to decrypt.

Definition at line 256 of file rsa.c.

Here is the caller graph for this function:



7.14.2.6 rsaEncrypt()

```
void rsaEncrypt (
             unsigned char * outputFile,
             unsigned char * publicKeyFile,
             unsigned char * plainText,
             size_t isPlaintextHex )
```

rsaEncrypt - Function used to encrypt the plaintext passed in as

Parameters

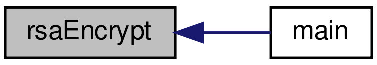
plainText	using RSA encryption and write the resulting ciphertext to the file
outputFile	in decimal. Function treats the entire plainText as the a single decimal value and performs the RSA encryption. The function reads in the public key and writes the result to the output file.

Parameters

outputFile	- unsigned char∗ - File to write the ciphertext to.
publicKeyFile	- unsigned char∗ - The file containing the public key to use during RSA encryption. The "n" paramter should be on the first line, followed by the newline character thereafter the "e" paramter should be placed in the public key file.
plainText	- unsigned char∗ - The plaintext to encrypt.
isPlaintextHex	- size_t - a flag used to indicate if the plaintext is encoded using ascii or hex encoding.

Definition at line 175 of file rsa.c.

Here is the caller graph for this function:



7.14.2.7 rsaInit()

```
void rsaInit (
            rsactx_t * rsaCtx )
```

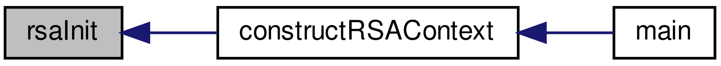
rsaInit - Function used to initialize the RSA context state structure passed in as

Parameters

rsaCtx.	Used as a helper function for the constructRSAContext function. Does not need to be explicitly called by the user. Initializes all mpz library variables used as required.
rsaCtx	- rsactx_t∗ - A pointer to the RSA context state structure to initialize.

Definition at line 54 of file rsa.c.

Here is the caller graph for this function:



7.14.2.8 rsaWriteKeysToFile()

```
void rsaWriteKeysToFile (
            rsactx_t * rsaCtx,
            unsigned char * publicKeyFileName,
            unsigned char * privateKeyFileName )
```

rsaWriteKeysToFile - Function used to write the public and private keys store in the RSA Context passed in as

Parameters

rsaCtx,to	the
publicKeyFileName	and
privateKeyFileName	respectively. The RSA private and public keys are written to the files in accordance with the practical specification. With the n paramter followed by a newline character, followed by d/e and finally a newline character.
rsaCtx	- rsactx_t∗ - The RSA context containing the public and private key pair to be written.
publicKeyFileName	- unsigned char∗ - The file to write the RSA public key to.
privateKeyFileName	- unsigned char∗ - The file to write the RSA private key to.

Definition at line 345 of file rsa.c.

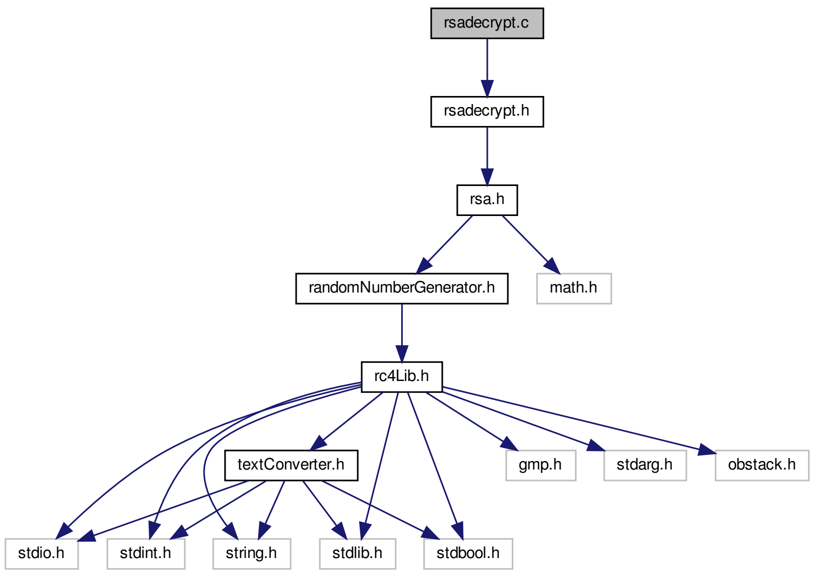
Here is the caller graph for this function:

rsaWriteKeysToFile
main

7.15 rsadecrypt.c File Reference

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

Include dependency graph for rsadecrypt.c:



Functions

- int main (int argc, char ∗argv[ ])
- void printHelp ()

  printHelp - Function used to print the help menu for the rsa decrypt utility
- void clearMemory (unsigned char ∗privateKeyFile, unsigned char ∗outputfileName, unsigned char ∗keyFile, unsigned char ∗key)

  clearMemory - Function used to deallocate all memory allocated for the rsa decryption utility.
- void verifyArgument (size_t argCounter, size_t argc, char ∗parameter)

  verifyArgument - Function used to verify if a paramter has an argument or not.

7.15.1 Detailed Description

Authors

Mohamed Ameen Omar (u16055323)
Douglas Healy (u16018100)
Llewellyn Moyse (u15100708)

Version

0.1

Date

2019-05-22

Copyright

Copyright (c) 2019

7.15.2 Function Documentation

7.15.2.1 clearMemory()

```
void clearMemory (
            unsigned char * privateKeyFile,
            unsigned char * outputfileName,
            unsigned char * keyFile,
            unsigned char * key )
```

clearMemory - Function used to deallocate all memory allocated for the rsa decryption utility.

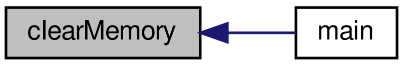
clearMemory - Function used to deallocate all memory allocated for the rsa keygen utility.

clearMemory - Function used to deallocate all memory allocated for the rsa encryption utility.

clearMemory - Function used to deallocate all memory allocated for the rc4 utility.

Definition at line 170 of file rsadecrypt.c.

Here is the caller graph for this function:

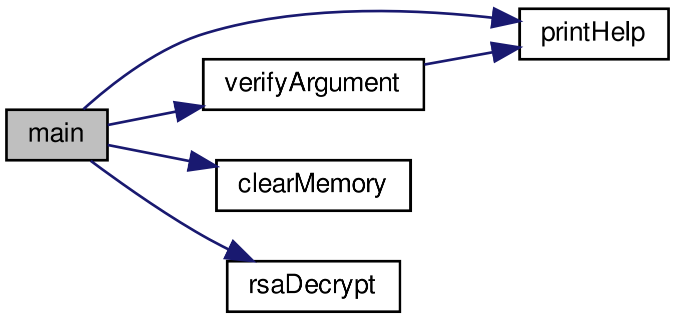


7.15.2.2 main()

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

Definition at line 16 of file rsadecrypt.c.

Here is the call graph for this function:



7.15.2.3 printHelp()

```
void printHelp ( )
```

printHelp - Function used to print the help menu for the rsa decrypt utility

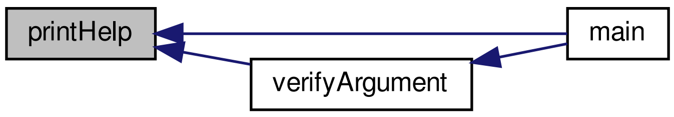
printHelp - Function used to print the help menu for the rsa kegen utility

printHelp - Function used to print the help menu for the rsa encrypt utility

printHelp - Function used to print the help menu for the rc4 utility

Definition at line 156 of file rsadecrypt.c.

Here is the caller graph for this function:



7.15.2.4 verifyArgument()

```
void verifyArgument (
            size_t argCounter,
            size_t argc,
            char * parameter )
```

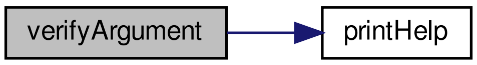
verifyArgument - Function used to verify if a paramter has an argument or not.

Parameters

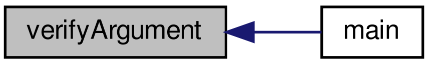
argCounter	- The current index being verified for the commandline paramters.
argc	- The total number of commandline arguments.
parameter	- The parameter whose argument is being verified.

Definition at line 200 of file rsadecrypt.c.

Here is the call graph for this function:



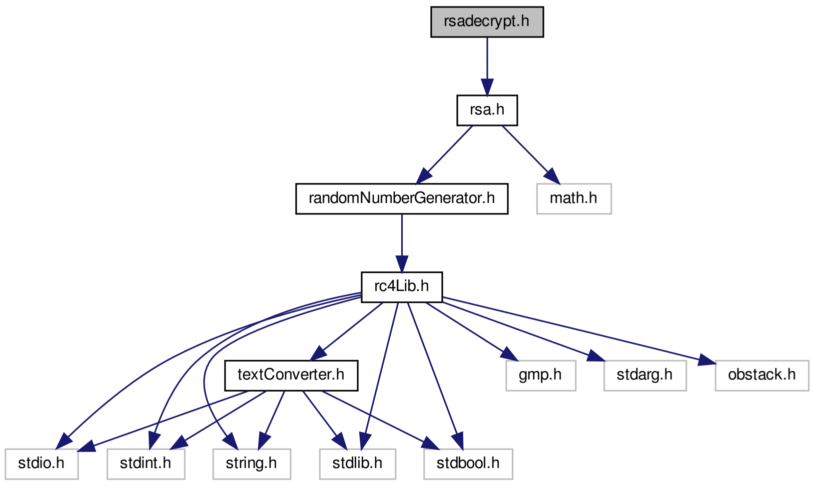
Here is the caller graph for this function:



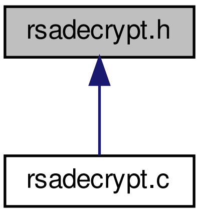
7.16 rsadecrypt.h File Reference

```
#include "rsa.h"
```

Include dependency graph for rsadecrypt.h:



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



Functions

- void printHelp ()

  printHelp - Function used to print the help menu for the rsa decrypt utility
- void clearMemory (unsigned char ∗privateKeyFile, unsigned char ∗outputfileName, unsigned char ∗keyFile, unsigned char ∗key)

  clearMemory - Function used to deallocate all memory allocated for the rsa decryption utility.
- void verifyArgument (size_t argCounter, size_t argc, char ∗parameter)

  verifyArgument - Function used to verify if a paramter has an argument or not.

7.16.1 Detailed Description

Authors

Mohamed Ameen Omar (u16055323)
Douglas Healy (u16018100)
Llewellyn Moyse (u15100708)

Version

0.1

Date

2019-05-22

Copyright

Copyright (c) 2019

7.16.2 Function Documentation

7.16.2.1 clearMemory()

```
void clearMemory (
            unsigned char * publickeyfile,
            unsigned char * privatekeyfile,
            unsigned char * keyFile,
            unsigned char * key )
```

clearMemory - Function used to deallocate all memory allocated for the rsa decryption utility.

clearMemory - Function used to deallocate all memory allocated for the rsa decryption utility.

clearMemory - Function used to deallocate all memory allocated for the rc4 utility.

clearMemory - Function used to deallocate all memory allocated for the rsa decryption utility.

clearMemory - Function used to deallocate all memory allocated for the rc4 utility.

Definition at line 165 of file rc4.c.

7.16.2.2 printHelp()

```
void printHelp ( )
```

printHelp - Function used to print the help menu for the rsa decrypt utility

printHelp - Function used to print the help menu for the rsa decrypt utility

printHelp - Function used to print the help menu for the rc4 utility

printHelp - Function used to print the help menu for the rsa decrypt utility

printHelp - Function used to print the help menu for the rc4 utility

Definition at line 148 of file rc4.c.

7.16.2.3 verifyArgument()

```
void verifyArgument (
             size_t argCounter,
             size_t argc,
             char * parameter )
```

verifyArgument - Function used to verify if a paramter has an argument or not.

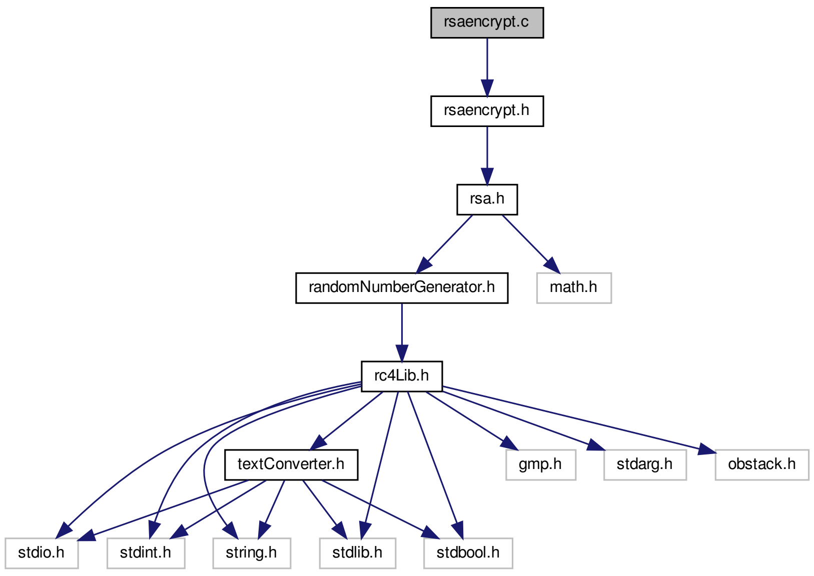
Parameters

argCounter	- The current index being verified for the commandline paramters.
argc	- The total number of commandline arguments.
parameter	- The parameter whose argument is being verified.

Definition at line 194 of file rc4.c.

7.17 rsaencrypt.c File Reference

`#include "rsaencrypt.h"`
Include dependency graph for rsaencrypt.c:



Functions

- int main (int argc, char ∗argv[ ])
- void printHelp ()

  printHelp - Function used to print the help menu for the rsa encrypt utility

- void clearMemory (unsigned char ∗publickeyfile, unsigned char ∗outputfileName, unsigned char ∗keyFile, unsigned char ∗key)

  clearMemory - Function used to deallocate all memory allocated for the rsa encryption utility.
- void verifyArgument (size_t argCounter, size_t argc, char ∗parameter)

  verifyArgument - Function used to verify if a paramter has an argument or not.

7.17.1 Detailed Description

Authors

Mohamed Ameen Omar (u16055323)
Douglas Healy (u16018100)
Llewellyn Moyse (u15100708)

Version

0.1

Date

2019-05-22

Copyright

Copyright (c) 2019

7.17.2 Function Documentation

7.17.2.1 clearMemory()

```
void clearMemory (
             unsigned char * publickeyfile,
             unsigned char * outputfileName,
             unsigned char * keyFile,
             unsigned char * key )
```

clearMemory - Function used to deallocate all memory allocated for the rsa encryption utility.

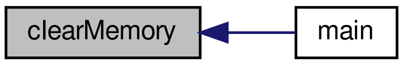
clearMemory - Function used to deallocate all memory allocated for the rsa keygen utility.

clearMemory - Function used to deallocate all memory allocated for the rsa decryption utility.

clearMemory - Function used to deallocate all memory allocated for the rc4 utility.

Definition at line 181 of file rsaencrypt.c.

Here is the caller graph for this function:

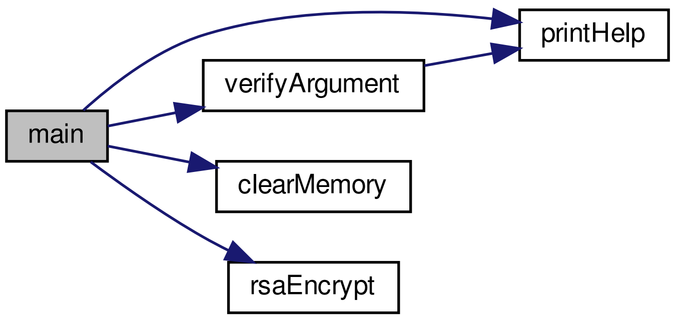


7.17.2.2 main()

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

Definition at line 17 of file rsaencrypt.c.

Here is the call graph for this function:



7.17.2.3 printHelp()

```
void printHelp ( )
```

printHelp - Function used to print the help menu for the rsa encrypt utility

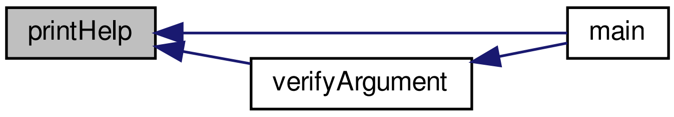
printHelp - Function used to print the help menu for the rsa kegen utility

printHelp - Function used to print the help menu for the rsa decrypt utility

printHelp - Function used to print the help menu for the rc4 utility

Definition at line 164 of file rsaencrypt.c.

Here is the caller graph for this function:



7.17.2.4 verifyArgument()

```
void verifyArgument (
            size_t argCounter,
            size_t argc,
            char * parameter )
```

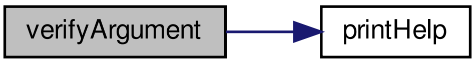
verifyArgument - Function used to verify if a paramter has an argument or not.

Parameters

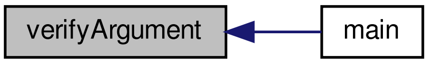
argCounter	- The current index being verified for the commandline paramters.
argc	- The total number of commandline arguments.
parameter	- The parameter whose argument is being verified.

Definition at line 211 of file rsaencrypt.c.

Here is the call graph for this function:



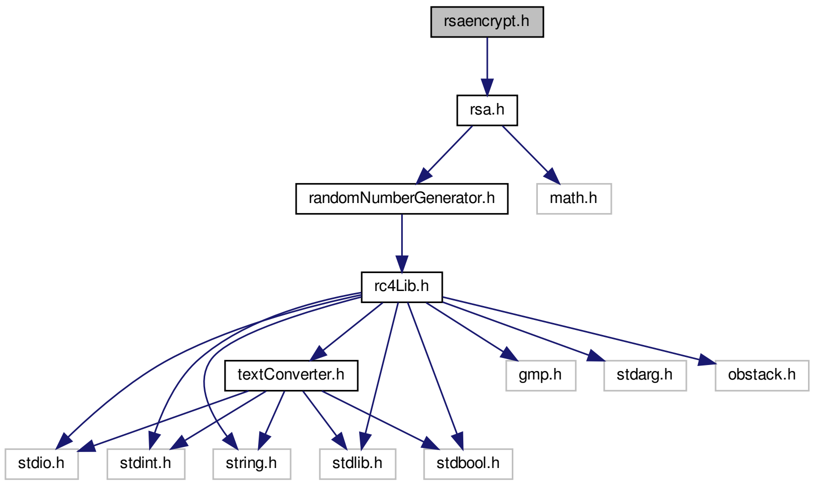
Here is the caller graph for this function:



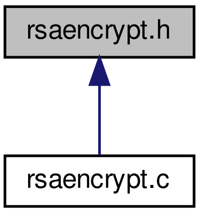
7.18 rsaencrypt.h File Reference

```
#include "rsa.h"
```

Include dependency graph for rsaencrypt.h:



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



Functions

- void printHelp ()

  printHelp - Function used to print the help menu for the rsa encrypt utility
- void clearMemory (unsigned char ∗publickeyfile, unsigned char ∗outputfileName, unsigned char ∗keyFile, unsigned char ∗key)

  clearMemory - Function used to deallocate all memory allocated for the rsa encryption utility.
- void verifyArgument (size_t argCounter, size_t argc, char ∗parameter)

  verifyArgument - Function used to verify if a paramter has an argument or not.

7.18.1 Detailed Description

Authors

Mohamed Ameen Omar (u16055323)
Douglas Healy (u16018100)
Llewellyn Moyse (u15100708)

Version

0.1

Date

2019-05-22

Copyright

Copyright (c) 2019

7.18.2 Function Documentation

7.18.2.1 clearMemory()

```
void clearMemory (
            unsigned char * publickeyfile,
            unsigned char * privatekeyfile,
            unsigned char * keyFile,
            unsigned char * key )
```

clearMemory - Function used to deallocate all memory allocated for the rsa encryption utility.

clearMemory - Function used to deallocate all memory allocated for the rsa encryption utility.

clearMemory - Function used to deallocate all memory allocated for the rsa decryption utility.

clearMemory - Function used to deallocate all memory allocated for the rsa encryption utility.

clearMemory - Function used to deallocate all memory allocated for the rc4 utility.

clearMemory - Function used to deallocate all memory allocated for the rsa decryption utility.

clearMemory - Function used to deallocate all memory allocated for the rc4 utility.

clearMemory - Function used to deallocate all memory allocated for the rsa encryption utility.

clearMemory - Function used to deallocate all memory allocated for the rsa decryption utility.

clearMemory - Function used to deallocate all memory allocated for the rc4 utility.

Definition at line 165 of file rc4.c.

7.18.2.2 printHelp()

```
void printHelp ( )
```

printHelp - Function used to print the help menu for the rsa encrypt utility

printHelp - Function used to print the help menu for the rsa encrypt utility

printHelp - Function used to print the help menu for the rsa decrypt utility

printHelp - Function used to print the help menu for the rsa encrypt utility

printHelp - Function used to print the help menu for the rc4 utility

printHelp - Function used to print the help menu for the rsa decrypt utility

printHelp - Function used to print the help menu for the rc4 utility

printHelp - Function used to print the help menu for the rsa encrypt utility

printHelp - Function used to print the help menu for the rsa decrypt utility

printHelp - Function used to print the help menu for the rc4 utility

Definition at line 148 of file rc4.c.

7.18.2.3 verifyArgument()

```
void verifyArgument (
             size_t argCounter,
             size_t argc,
             char * parameter )
```

verifyArgument - Function used to verify if a paramter has an argument or not.

Parameters

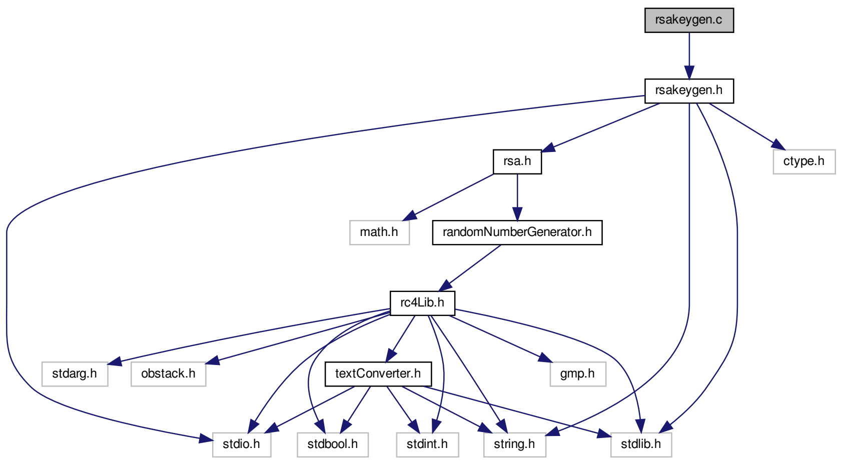
argCounter	- The current index being verified for the commandline paramters.
argc	- The total number of commandline arguments.
parameter	- The parameter whose argument is being verified.

Definition at line 194 of file rc4.c.

7.19 rsakeygen.c File Reference

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

Include dependency graph for rsakeygen.c:



Functions

- int main (int argc, char ∗argv[ ])
- void printHelp ()

    printHelp - Function used to print the help menu for the rsa kegen utility
- void clearMemory (unsigned char ∗publickeyfile, unsigned char ∗privatekeyfile, unsigned char ∗keyFile, unsigned char ∗key)

    clearMemory - Function used to deallocate all memory allocated for the rsa keygen utility.
- void verifyArgument (size_t argCounter, size_t argc, char ∗parameter)

    verifyArgument - Function used to verify if a paramter has an argument or not.

7.19.1 Detailed Description

Authors

Mohamed Ameen Omar (u16055323)
Douglas Healy (u16018100)
Llewellyn Moyse (u15100708)

Version

0.1

Date

2019-05-22

Copyright

Copyright (c) 2019

7.19.2 Function Documentation

7.19.2.1 clearMemory()

```
void clearMemory (
            unsigned char * publickeyfile,
            unsigned char * privatekeyfile,
            unsigned char * keyFile,
            unsigned char * key )
```

clearMemory - Function used to deallocate all memory allocated for the rsa keygen utility.

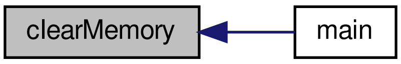
clearMemory - Function used to deallocate all memory allocated for the rsa encryption utility.

clearMemory - Function used to deallocate all memory allocated for the rsa decryption utility.

clearMemory - Function used to deallocate all memory allocated for the rc4 utility.

Definition at line 205 of file rsakeygen.c.

Here is the caller graph for this function:

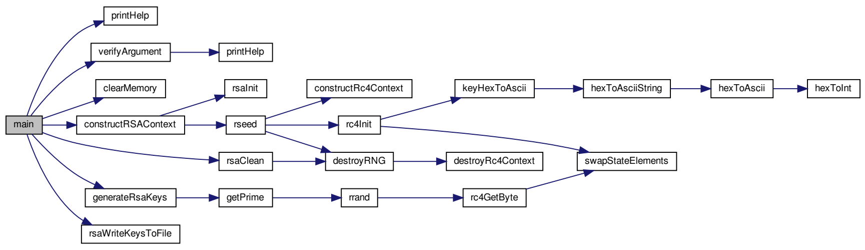


7.19.2.2 main()

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

Definition at line 15 of file rsakeygen.c.

Here is the call graph for this function:



7.19.2.3 printHelp()

```
void printHelp ( )
```

printHelp - Function used to print the help menu for the rsa kegen utility

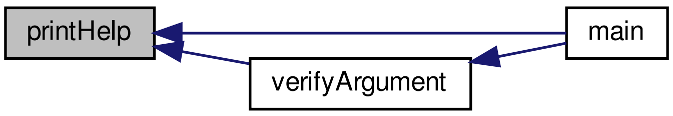
printHelp - Function used to print the help menu for the rsa encrypt utility

printHelp - Function used to print the help menu for the rsa decrypt utility

printHelp - Function used to print the help menu for the rc4 utility

Definition at line 187 of file rsakeygen.c.

Here is the caller graph for this function:



7.19.2.4 verifyArgument()

```
void verifyArgument (
             size_t argCounter,
             size_t argc,
             char * parameter )
```

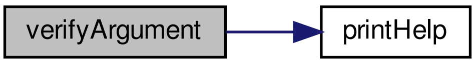
verifyArgument - Function used to verify if a paramter has an argument or not.

Parameters

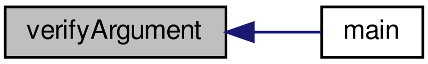
argCounter	- The current index being verified for the commandline paramters.
argc	- The total number of commandline arguments.
parameter	- The parameter whose argument is being verified.

Definition at line 235 of file rsakeygen.c.

Here is the call graph for this function:



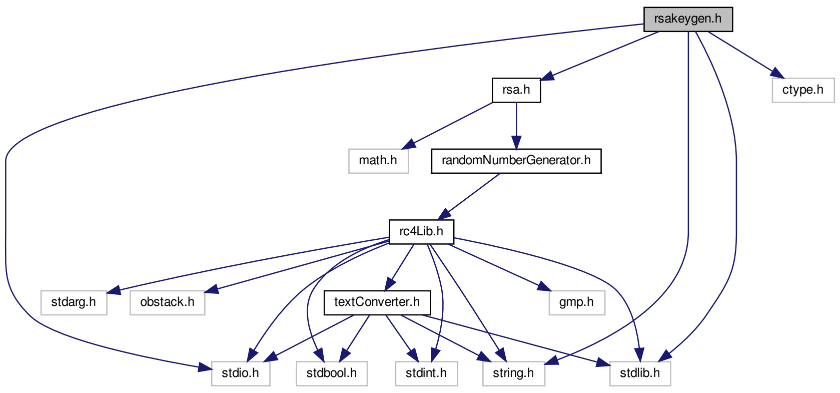
Here is the caller graph for this function:



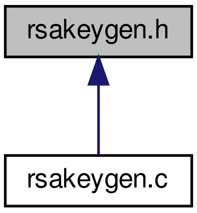
7.20 rsakeygen.h File Reference

```
#include "rsa.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
```

Include dependency graph for rsakeygen.h:



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



Functions

- void printHelp ()

    printHelp - Function used to print the help menu for the rsa kegen utility
- void clearMemory (unsigned char ∗publickeyfile, unsigned char ∗privatekeyfile, unsigned char ∗keyFile, unsigned char ∗key)

    clearMemory - Function used to deallocate all memory allocated for the rsa keygen utility.
- void verifyArgument (size_t argCounter, size_t argc, char ∗parameter)

    verifyArgument - Function used to verify if a paramter has an argument or not.

7.20.1 Detailed Description

Authors

Mohamed Ameen Omar (u16055323)
Douglas Healy (u16018100)
Llewellyn Moyse (u15100708)

Version

0.1

Date

2019-05-22

Copyright

Copyright (c) 2019

7.20.2 Function Documentation

7.20.2.1 clearMemory()

```
void clearMemory (
            unsigned char * publickeyfile,
            unsigned char * privatekeyfile,
            unsigned char * keyFile,
            unsigned char * key )
```

clearMemory - Function used to deallocate all memory allocated for the rsa keygen utility.

clearMemory - Function used to deallocate all memory allocated for the rsa keygen utility.

clearMemory - Function used to deallocate all memory allocated for the rsa encryption utility.

clearMemory - Function used to deallocate all memory allocated for the rsa decryption utility.

clearMemory - Function used to deallocate all memory allocated for the rsa keygen utility.

clearMemory - Function used to deallocate all memory allocated for the rsa encryption utility.

clearMemory - Function used to deallocate all memory allocated for the rc4 utility.

clearMemory - Function used to deallocate all memory allocated for the rsa keygen utility.

clearMemory - Function used to deallocate all memory allocated for the rsa decryption utility.

clearMemory - Function used to deallocate all memory allocated for the rc4 utility.

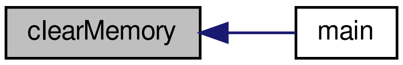
clearMemory - Function used to deallocate all memory allocated for the rsa encryption utility.

clearMemory - Function used to deallocate all memory allocated for the rsa decryption utility.

clearMemory - Function used to deallocate all memory allocated for the rc4 utility.

Definition at line 165 of file rc4.c.

Here is the caller graph for this function:



7.20.2.2 printHelp()

```
void printHelp ( )
```

printHelp - Function used to print the help menu for the rsa kegen utility

printHelp - Function used to print the help menu for the rsa kegen utility

printHelp - Function used to print the help menu for the rsa encrypt utility

printHelp - Function used to print the help menu for the rsa decrypt utility

printHelp - Function used to print the help menu for the rsa kegen utility

printHelp - Function used to print the help menu for the rsa encrypt utility

printHelp - Function used to print the help menu for the rc4 utility

printHelp - Function used to print the help menu for the rsa kegen utility

printHelp - Function used to print the help menu for the rsa decrypt utility

printHelp - Function used to print the help menu for the rc4 utility

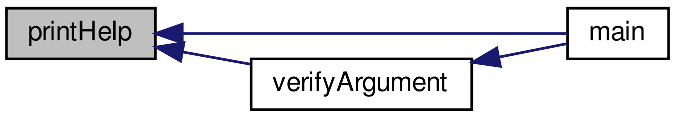
printHelp - Function used to print the help menu for the rsa encrypt utility

printHelp - Function used to print the help menu for the rsa decrypt utility

printHelp - Function used to print the help menu for the rc4 utility

Definition at line 148 of file rc4.c.

Here is the caller graph for this function:



7.20.2.3 verifyArgument()

```
void verifyArgument (
             size_t argCounter,
             size_t argc,
             char * parameter )
```

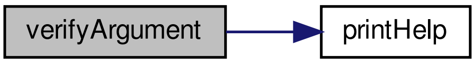
verifyArgument - Function used to verify if a paramter has an argument or not.

Parameters

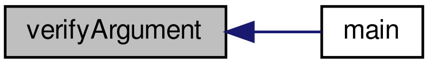
argCounter	- The current index being verified for the commandline paramters.
argc	- The total number of commandline arguments.
parameter	- The parameter whose argument is being verified.

Definition at line 194 of file rc4.c.

Here is the call graph for this function:



Here is the caller graph for this function:



7.21 rsaKeyGenTester.py File Reference

Namespaces

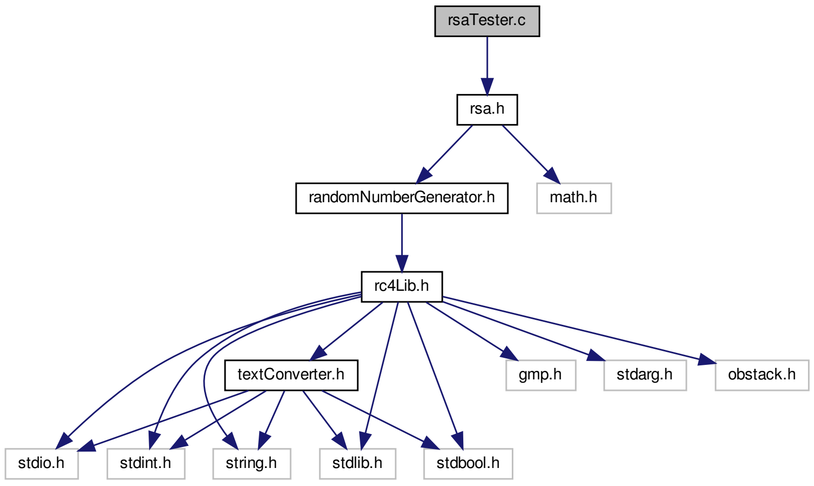
- rsaKeyGenTester

Variables

- string bits = "55"
- string publickeyfile = "temp"
- string privatekeyfile = "temp2"
- string keyFile = ""
- string key = ""
- temp = subprocess.call(["./rsakeygen", "-b", bits, "-KU", publickeyfile, "-KR" , privatekeyfile , "-key" , key , "-kf" , keyFile])

7.22 rsaTester.c File Reference

```
#include "rsa.h"
```
Include dependency graph for rsaTester.c:



Functions

- int main (int argc, char ∗argv[ ])

7.22.1 Detailed Description

Authors

Mohamed Ameen Omar (u16055323)
Douglas Healy (u16018100)
Llewellyn Moyse (u15100708)

Version

0.1

Date

2019-05-22

Copyright

Copyright (c) 2019

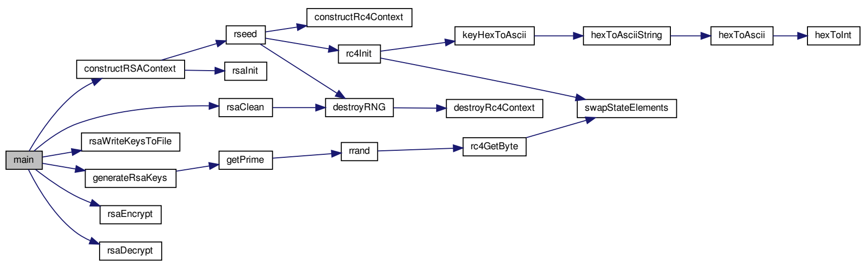
7.22.2 Function Documentation

7.22.2.1 main()

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

Definition at line 15 of file rsaTester.c.

Here is the call graph for this function:



7.23 test.py File Reference

Namespaces

- test

Variables

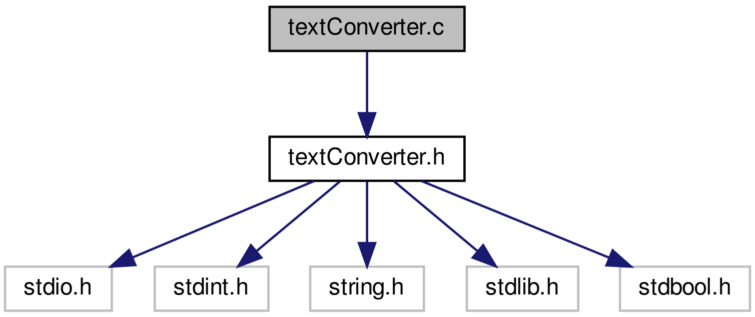
- int x = 88003541953488079566952319866978440447
- int y = 49236241020712442839828388720313042593
- int n = 166501317222622877548769568260396763403

7.24 textConverter.c File Reference

The text converter libary implementation file. This file contains functions used to convert between different bases of text. Such as conversion from ascii to hex, hex to ascii, hex to int. This is used for encryption when a certian base is required, different from the one provided.

```
#include "textConverter.h"
```

Include dependency graph for textConverter.c:



Functions

- uint8_t hexToInt (char ch)

  hexToInt - Function that converts a given hex value into an integer.
- uint8_t hexToAscii (char ch1, char ch2)

  hexToAscii - Function that converts a given hex value to its ASCII equivalent.
- void hexToAsciiString (char ∗hexString, char ∗asciiString, int hexStringLength)

  hexToAsciiString - Function that converts a given string of hex values into its ASCII equivalent. A hex string contains hex chars and is "encoded" in ascii In order to encrypt it, it must be converted to the equivalent ascii plain text string plaintext string is half the size of hex, since two hex chars = 1 ascii char if hex string is "4A" it will be converted to "J" in ascii which will have a hex representation of "4a" The original hex string converted to hex staright or printed in hex straight rather will print or have the value "0x34", "0x31" BASICALLY THE HEX STRING FF IS INTERPRETED AS THE CHARS FF, whereas when using this function we intend it to be "J", ie the char "J"
- unsigned char ∗ asciiToHexString (unsigned char ∗asciiString, unsigned char ∗hexString, size_t asciiString←Len)

  Function name: asciiToHexString - convert an ascii String to an ascii string.
- unsigned char ∗ keyHexToAscii (unsigned char ∗hexKey, int keyLength)

  keyHexToAscii - Function to convert a hex encoded key to an ascii string. The caller must ensure they deallocate the memory allocated for the returned ascii encoded string.

7.24.1 Detailed Description

The text converter libary implementation file. This file contains functions used to convert between different bases of text. Such as conversion from ascii to hex, hex to ascii, hex to int. This is used for encryption when a certian base is required, different from the one provided.

Authors

Mohamed Ameen Omar (u16055323)
Douglas Healy (u16018100)
Llewellyn Moyse (u15100708)

Version

0.1

Date

2019-05-22

Copyright

Copyright (c) 2019

7.24.2 Function Documentation

7.24.2.1 asciiToHexString()

```
unsigned char* asciiToHexString (
            unsigned char * asciiString,
            unsigned char * hexString,
            size_t asciiStringLen )
```

Function name: asciiToHexString - convert an ascii String to an ascii string.

Parameters

asciiString	- unsigned char∗ pointing to the ASCII String to be converted.
hexString	- unsigned char∗ pointing to a memory where the converted Hex string should be stored.
asciiStringLen	- size_t containing the length of the ASCII String to be converted.

Returns

unsigned char∗ asciiToHexString - pointer to the converted Hex String, pointing to the same memory location as

Parameters

hexString.	

Definition at line 84 of file textConverter.c.

7.24.2.2 hexToAscii()

```
uint8_t hexToAscii (
            char ch1,
            char ch2 )
```

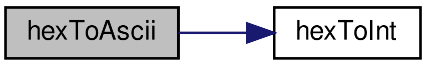
hexToAscii - Function that converts a given hex value to its ASCII equivalent.

Parameters

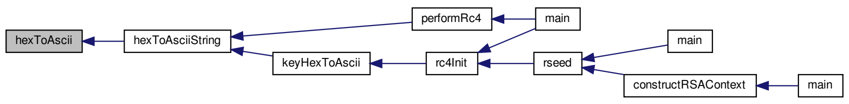
ch1	- char value of the first hex value.
ch2	- char value of the second hex value.

Definition at line 42 of file textConverter.c.

Here is the call graph for this function:



Here is the caller graph for this function:



7.24.2.3 hexToAsciiString()

```
void hexToAsciiString (
             char * hexString,
             char * asciiString,
             int hexStringLength )
```

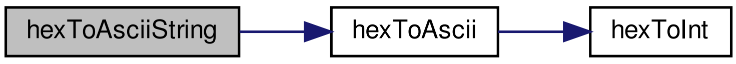
hexToAsciiString - Function that converts a given string of hex values into its ASCII equivalent. A hex string contains hex chars and is "encoded" in ascii In order to encrypt it, it must be converted to the equivalent ascii plain text string plaintext string is half the size of hex, since two hex chars = 1 ascii char if hex string is "4A" it will be converted to "J" in ascii which will have a hex representation of "4a" The original hex string converted to hex staright or printed in hex straight rather will print or have the value "0x34", "0x31" BASICALLY THE HEX STRING FF IS INTERPRETED AS THE CHARS FF, whereas when using this function we intend it to be "J", ie the char "J"

Parameters

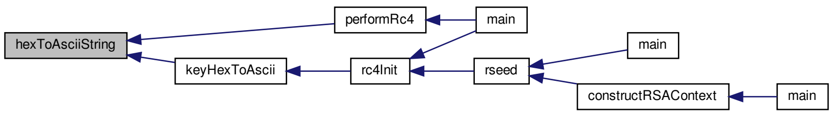
char∗	hexString - The string of hex values to be converted.
char∗	asciiString - The output of the converted hex string.
int	hexStringLength - The length of parameter hexString.

Definition at line 61 of file textConverter.c.

Here is the call graph for this function:



Here is the caller graph for this function:



7.24.2.4 hexToInt()

```
uint8_t hexToInt (
            char ch )
```

hexToInt - Function that converts a given hex value into an integer.

Parameters

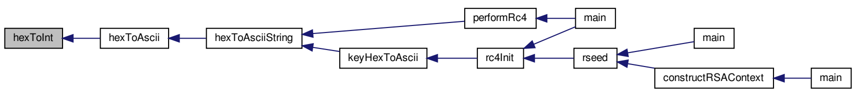
ch	- hex value that wil be converted to int.

Returns

uint8_t the converted int value.

Definition at line 23 of file textConverter.c.

Here is the caller graph for this function:



7.24.2.5 keyHexToAscii()

```
unsigned char* keyHexToAscii (
             unsigned char * hexKey,
             int keyLength )
```

keyHexToAscii - Function to convert a hex encoded key to an ascii string. The caller must ensure they deallocate the memory allocated for the returned ascii encoded string.

Parameters

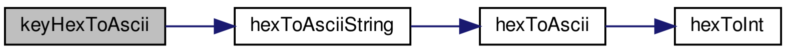
hexKey	- unsigned char* - the hexadecimal encoded key to convert.
keyLength	- int - the length of the key
hexKey.	

Returns

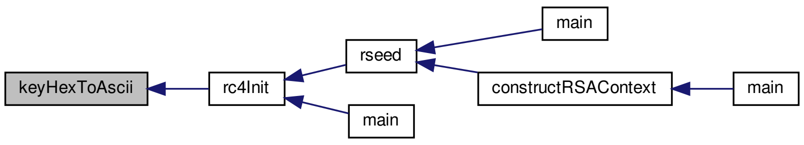
unsigned char* - The resulting ascii encoded string.

Definition at line 104 of file textConverter.c.

Here is the call graph for this function:



Here is the caller graph for this function:



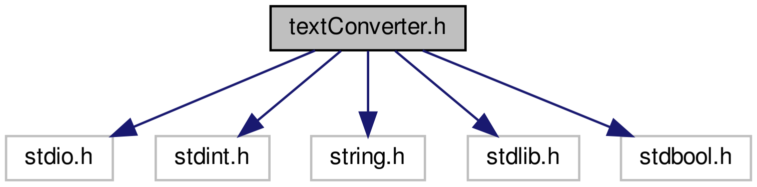
7.25 textConverter.h File Reference

The text converter libary function prototype file. This file contains functions used to convert between different bases of text. Such as conversion from ascii to hex, hex to ascii, hex to int. This is used for encryption when a certian base is required, different from the one provided.

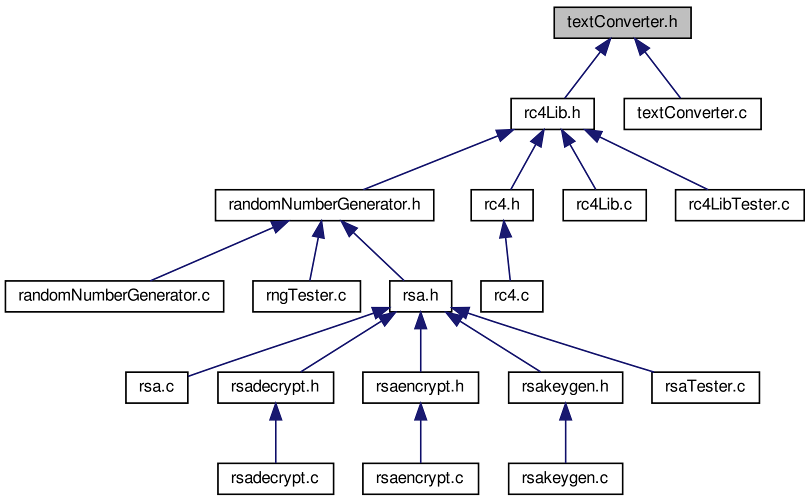
```
#include <stdio.h>
#include <stdint.h>
```

```
#include <string.h>
#include <stdlib.h>
#include <stdbool.h>
```

Include dependency graph for textConverter.h:



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



Functions

- uint8_t hexToInt (char ch)

  hexToInt - Function that converts a given hex value into an integer.
- uint8_t hexToAscii (char ch1, char ch2)

  hexToAscii - Function that converts a given hex value to its ASCII equivalent.
- void hexToAsciiString (char ∗hexString, char ∗asciiString, int hexStringLength)

  hexToAsciiString - Function that converts a given string of hex values into its ASCII equivalent. A hex string contains hex chars and is "encoded" in ascii In order to encrypt it, it must be converted to the equivalent ascii plain text string plaintext string is half the size of hex, since two hex chars = 1 ascii char if hex string is "4A" it will be converted to "J" in ascii which will have a hex representation of "4a" The original hex string converted to hex staright or printed in hex straight rather will print or have the value "0x34", "0x31" BASICALLY THE HEX STRING FF IS INTERPRETED AS THE CHARS FF, whereas when using this function we intend it to be "J", ie the char "J"

- unsigned char ∗ asciiToHexString (unsigned char ∗asciiString, unsigned char ∗hexString, size_t asciiString←↩ Len)

  Function name: asciiToHexString - convert an ascii String to an ascii string.
- unsigned char ∗ keyHexToAscii (unsigned char ∗hexKey, int keyLength)

  keyHexToAscii - Function to convert a hex encoded key to an ascii string. The caller must ensure they deallocate the memory allocated for the returned ascii encoded string.

7.25.1 Detailed Description

The text converter libary function prototype file. This file contains functions used to convert between different bases of text. Such as conversion from ascii to hex, hex to ascii, hex to int. This is used for encryption when a certian base is required, different from the one provided.

Authors

Mohamed Ameen Omar (u16055323)
Douglas Healy (u16018100)
Llewellyn Moyse (u15100708)

Version

0.1

Date

2019-05-22

Copyright

Copyright (c) 2019

7.25.2 Function Documentation

7.25.2.1 asciiToHexString()

```
unsigned char* asciiToHexString (
            unsigned char * asciiString,
            unsigned char * hexString,
            size_t asciiStringLen )
```

Function name: asciiToHexString - convert an ascii String to an ascii string.

Parameters

asciiString	- unsigned char∗ pointing to the ASCII String to be converted.
hexString	- unsigned char∗ pointing to a memory where the converted Hex string should be stored.
asciiStringLen	- size_t containing the length of the ASCII String to be converted.

Returns

unsigned char∗ asciiToHexString - pointer to the converted Hex String, pointing to the same memory location as

Parameters

hexString.	

Definition at line 84 of file textConverter.c.

7.25.2.2 hexToAscii()

```
uint8_t hexToAscii (
            char ch1,
            char ch2 )
```

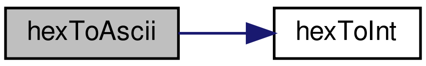
hexToAscii - Function that converts a given hex value to its ASCII equivalent.

Parameters

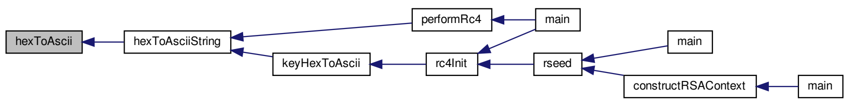
ch1	- char value of the first hex value.
ch2	- char value of the second hex value.

Definition at line 42 of file textConverter.c.

Here is the call graph for this function:



Here is the caller graph for this function:



7.25.2.3 hexToAsciiString()

```
void hexToAsciiString (
            char * hexString,
            char * asciiString,
            int hexStringLength )
```

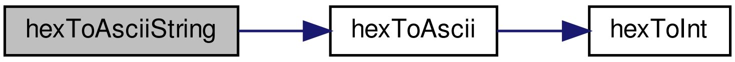
hexToAsciiString - Function that converts a given string of hex values into its ASCII equivalent. A hex string contains hex chars and is "encoded" in ascii In order to encrypt it, it must be converted to the equivalent ascii plain text string plaintext string is half the size of hex, since two hex chars = 1 ascii char if hex string is "4A" it will be converted to "J" in ascii which will have a hex representation of "4a" The original hex string converted to hex staright or printed in hex straight rather will print or have the value "0x34", "0x31" BASICALLY THE HEX STRING FF IS INTERPRETED AS THE CHARS FF, whereas when using this function we intend it to be "J", ie the char "J"

Parameters

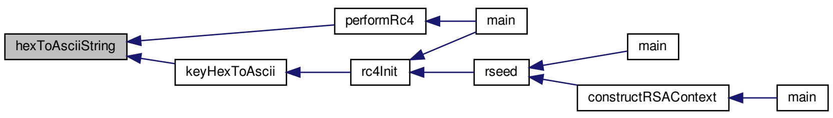
char∗	hexString - The string of hex values to be converted.
char∗	asciiString - The output of the converted hex string.
int	hexStringLength - The length of parameter hexString.

Definition at line 61 of file textConverter.c.

Here is the call graph for this function:



Here is the caller graph for this function:



7.25.2.4 hexToInt()

```
uint8_t hexToInt (
            char ch )
```

hexToInt - Function that converts a given hex value into an integer.

Parameters

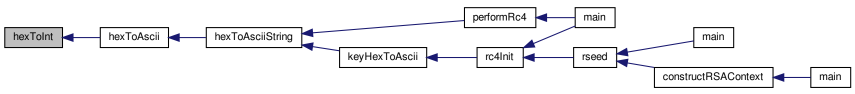
ch	- hex value that wil be converted to int.

Returns

uint8_t the converted int value.

Definition at line 23 of file textConverter.c.

Here is the caller graph for this function:



7.25.2.5 keyHexToAscii()

```
unsigned char* keyHexToAscii (
              unsigned char * hexKey,
              int keyLength )
```

keyHexToAscii - Function to convert a hex encoded key to an ascii string. The caller must ensure they deallocate the memory allocated for the returned ascii encoded string.

Parameters

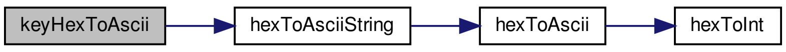
hexKey	- unsigned char∗ - the hexadecimal encoded key to convert.
keyLength	- int - the length of the key
hexKey.	

Returns

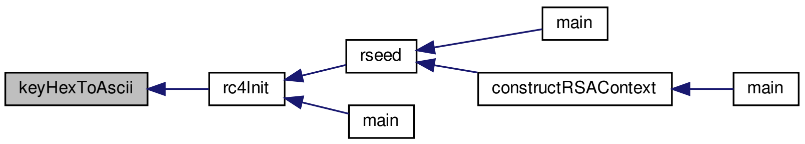
unsigned char∗ - The resulting ascii encoded string.

Definition at line 104 of file textConverter.c.

Here is the call graph for this function:



Here is the caller graph for this function:



Index