Cryptic

P. Baillehache

February 24, 2020

Contents

Introduction

Cryptic library is a C library to cypher and decypher data.

It provides an implementation of the Feistel cypher scheme.

It uses the `PBErr` and `GSet` libraries.

1 Interface

```
// *************** CRYPTIC.H ***************
#ifndef CRYPTIC_H
#define CRYPTIC_H

// ================= Include ==================
#include <stdlib.h>
```

```
#include <stdio.h>
#include <stdbool.h>
#include "pberr.h"
#include "gset.h"

// ================= Define ==================

// ================ Data structures ==================

// Structure for the Feistel cypher
typedef struct FeistelCyphering {

  // GSet of null terminated strings, all the same size,
  // which are the keys to cypher/decypher
  GSetStr* keys;

  // Function to be used during cyphering
  void (*fun)(
    unsigned char*,
    unsigned char*,
    unsigned char*,
    unsigned long);

} FeistelCyphering;

// ================ Functions declaration ====================

// Static constructor for a Feistel cypher,
// 'keys' is a GSet of null terminated strings, all the same size
// 'fun' is the cyphering function of the form
// void (*fun)(char* src, char* dest, char* key, unsigned long len)
// 'src', 'dest' have same length 'len'
// 'key' may be of any length
#if BUILDMODE != 0
static inline
#endif
FeistelCyphering FeistelCypheringCreateStatic(
  GSetStr* keys,
  void (*fun)(
    unsigned char*,
    unsigned char*,
    unsigned char*,
    unsigned long));

// Function to free the memory used by the static FeistelCyphering
void FeistelCypheringFreeStatic(
  FeistelCyphering* that);

// Function to cypher the message 'msg' with the FeistelCyphering 'that'
// The message length 'lenMsg' must be a multiple of the length of
// the keys
// Return a new string containing the cyphered message
unsigned char* FeistelCypheringCypher(
  FeistelCyphering* that,
  unsigned char* msg,
  unsigned long lenMsg);

// Function to decypher the message 'msg' with the FeistelCyphering
// 'that'
// The message length 'lenMsg' must be a multiple of the length of
// the keys
// Return a new string containing the decyphered message
```

```
unsigned char* FeistelCypheringDecypher(
  FeistelCyphering* that,
  unsigned char* msg,
  unsigned long lenMsg);

// ================ inliner ====================

#if BUILDMODE != 0
#include "cryptic-inline.c"
#endif

#endif
```

2 Code

2.1 cryptic.c

```
// *************** CRYPTIC.C ***************

// ================= Include ==================
#include "cryptic.h"
#if BUILDMODE == 0
#include "cryptic-inline.c"
#endif

// ================ Functions implementation ==================

// Function to free the memory used by the static FeistelCyphering
void FeistelCypheringFreeStatic(
  FeistelCyphering* that) {

#if BUILDMODE == 0

  if (that == NULL) {

    CrypticErr->_type = PBErrTypeNullPointer;
    sprintf(
      CrypticErr->_msg,
      "'that' is null");
    PBErrCatch(CrypticErr);

  }

#endif

  // Reset pointers
  that->keys = NULL;
  that->fun = NULL;

}

// Function to cypher the message 'msg' with the FeistelCyphering 'that'
// The message length 'lenMsg' must be a multiple of the length of
// the keys
// Return a new string containing the cyphered message
unsigned char* FeistelCypheringCypher(
  FeistelCyphering* that,
  unsigned char* msg,
```

```
  unsigned long lenMsg) {

#if BUILDMODE == 0

  if (that == NULL) {

    CrypticErr->_type = PBErrTypeNullPointer;
    sprintf(
      CrypticErr->_msg,
      "'keys' is null");
    PBErrCatch(CrypticErr);

  }

  if (msg == NULL) {

    CrypticErr->_type = PBErrTypeNullPointer;
    sprintf(
      CrypticErr->_msg,
      "'msg' is null");
    PBErrCatch(CrypticErr);

  }

  if (lenMsg % 2 != 0) {

    CrypticErr->_type = PBErrTypeInvalidArg;
    sprintf(
      CrypticErr->_msg,
      "'lenMsg' is not multiple of 2 (%lu)",
      lenMsg);
    PBErrCatch(CrypticErr);

  }

#endif

  // Allocate memory for the cyphered message
  unsigned char* cypheredMsg =
    PBErrMalloc(
      CrypticErr,
      lenMsg * sizeof(unsigned char));

  // Initialized the cyphered message with the initial message
  memcpy(
    cypheredMsg,
    msg,
    lenMsg);

  // Declare a variable to memorize the helf length of the message
  unsigned long halfLenMsg = lenMsg / 2;

  // Allocate memory for the cyphering function
  unsigned char* str =
    PBErrMalloc(
      CrypticErr,
      halfLenMsg * sizeof(unsigned char));

  // Loop on keys
  GSetIterForward iter = GSetIterForwardCreateStatic(that->keys);
  do {
```

```
    // Get the key
    unsigned char* key = GSetIterGet(&iter);

    // Copy right half of the current cyphered message into the left
    // of the temporary string
    memcpy(
      str,
      cypheredMsg + halfLenMsg,
      halfLenMsg);

    // Cypher the right half and store it into the right of the
    // temporary string
    (that->fun)(
      cypheredMsg + halfLenMsg,
      str + halfLenMsg,
      key,
      halfLenMsg);

    // Apply the XOR operator on the half right of the temporary
    // string with the left half of the cyphered message
    for (
      int iChar = halfLenMsg;
      iChar--;) {

      str[halfLenMsg + iChar] =
        str[halfLenMsg + iChar] ^
        cypheredMsg[iChar];

    }

    // Copy the temporary string into the cyphered message
    memcpy(
      cypheredMsg,
      str,
      lenMsg);

  } while (GSetIterStep(&iter));

  // Exchange the two halves of the cyphered message
  for (
    int iChar = halfLenMsg;
    iChar--;) {

    str[halfLenMsg + iChar] = cypheredMsg[iChar];
    str[iChar] = cypheredMsg[halfLenMsg + iChar];

  }

  memcpy(
    cypheredMsg,
    str,
    lenMsg);

  // Free memory
  free(str);

  // Return the cyphered message
  return cypheredMsg;

}

// Function to decypher the message 'msg' with the FeistelCyphering
```

```
// 'that'
// The message length 'lenMsg' must be a multiple of the length of
// the keys
// Return a new string containing the decyphered message
unsigned char* FeistelCypheringDecypher(
  FeistelCyphering* that,
  unsigned char* msg,
  unsigned long lenMsg) {

#if BUILDMODE == 0

  if (that == NULL) {

    CrypticErr->_type = PBErrTypeNullPointer;
    sprintf(
      CrypticErr->_msg,
      "'keys' is null");
    PBErrCatch(CrypticErr);

  }

  if (msg == NULL) {

    CrypticErr->_type = PBErrTypeNullPointer;
    sprintf(
      CrypticErr->_msg,
      "'msg' is null");
    PBErrCatch(CrypticErr);

  }

  if (lenMsg % 2 != 0) {

    CrypticErr->_type = PBErrTypeInvalidArg;
    sprintf(
      CrypticErr->_msg,
      "'lenMsg' is not multiple of 2 (%lu)",
      lenMsg);
    PBErrCatch(CrypticErr);

  }

#endif

  // Allocate memory for the cyphered message
  unsigned char* cypheredMsg =
    PBErrMalloc(
      CrypticErr,
      lenMsg * sizeof(unsigned char));

  // Initialized the cyphered message with the initial message
  memcpy(
    cypheredMsg,
    msg,
    lenMsg);

  // Declare a variable to memorize the helf length of the message
  unsigned long halfLenMsg = lenMsg / 2;

  // Allocate memory for the cyphering function
  unsigned char* str =
    PBErrMalloc(
```

```
    CrypticErr,
    halfLenMsg * sizeof(unsigned char));

// Loop on keys
GSetIterBackward iter = GSetIterBackwardCreateStatic(that->keys);
do {

  // Get the key
  unsigned char* key = GSetIterGet(&iter);

  // Copy right half of the current cyphered message into the left
  // of the temporary string
  memcpy(
    str,
    cypheredMsg + halfLenMsg,
    halfLenMsg);

  // Cypher the right half and store it into the right of the
  // temporary string
  (that->fun)(
    cypheredMsg + halfLenMsg,
    str + halfLenMsg,
    key,
    halfLenMsg);

  // Apply the XOR operator on the half right of the temporary
  // string with the left half of the cyphered message
  for (
    int iChar = halfLenMsg;
    iChar--;) {

    str[halfLenMsg + iChar] =
      str[halfLenMsg + iChar] ^
      cypheredMsg[iChar];

  }

  // Copy the temporary string into the cyphered message
  memcpy(
    cypheredMsg,
    str,
    lenMsg);

} while (GSetIterStep(&iter));

// Exchange the two halves of the cyphered message
for (
  int iChar = halfLenMsg;
  iChar--;) {

  str[halfLenMsg + iChar] = cypheredMsg[iChar];
  str[iChar] = cypheredMsg[halfLenMsg + iChar];

}

memcpy(
  cypheredMsg,
  str,
  lenMsg);

// Free memory
free(str);
```

```
  // Return the cyphered message
  return cypheredMsg;

}
```

2.2 cryptic-inline.c

```
// ************** CRYPTIC-INLINE.C ***************

// ================ Functions implementation ====================

// Static constructor for a Feistel cypher,
// 'keys' is a GSet of null terminated strings, all the same size
// 'fun' is the cyphering function of the form
// void (*fun)(char* src, char* dest, char* key, unsigned long len)
// 'src', 'dest' have same length 'len'
// 'key' may be of any length
#if BUILDMODE != 0
static inline
#endif
FeistelCyphering FeistelCypheringCreateStatic(
  GSetStr* keys,
  void (*fun)(
    unsigned char*,
    unsigned char*,
    unsigned char*,
    unsigned long)) {

#if BUILDMODE == 0

  if (keys == NULL) {

    CrypticErr->_type = PBErrTypeNullPointer;
    sprintf(
      CrypticErr->_msg,
      "'keys' is null");
    PBErrCatch(CrypticErr);

  }

  if (fun == NULL) {

    CrypticErr->_type = PBErrTypeNullPointer;
    sprintf(
      CrypticErr->_msg,
      "'fun' is null");
    PBErrCatch(CrypticErr);

  }

#endif

  // Declare a FeistelCyphering and set the properties
  FeistelCyphering c = {

    .keys = keys,
    .fun = fun

  };
```

```
  // Return the FeistelCyphering
  return c;

}
```

3 Makefile

```
# Build mode
# 0: development (max safety, no optimisation)
# 1: release (min safety, optimisation)
# 2: fast and furious (no safety, optimisation)
BUILD_MODE?=0

all: pbmake_wget main

# Automatic installation of the repository PBMake in the parent folder
pbmake_wget:
if [ ! -d ../PBMake ]; then wget https://github.com/BayashiPascal/PBMake/archive/master.zip; unzip master.zip; rm -f

# Check code style
style:
cbo *.h *.c

# Makefile definitions
MAKEFILE_INC=../PBMake/Makefile.inc
include $(MAKEFILE_INC)

# Rules to make the executable
repo=cryptic
$($(repo)_EXENAME): \
$($(repo)_EXENAME).o \
$($(repo)_EXE_DEP) \
$($(repo)_DEP)
$(COMPILER) `echo "$($(repo)_EXE_DEP) $($(repo)_EXENAME).o" | tr ' ' '\n' | sort -u` $(LINK_ARG) $($(repo)_LINK_ARG)

$($(repo)_EXENAME).o: \
$($(repo)_DIR)/$($(repo)_EXENAME).c \
$($(repo)_INC_H_EXE) \
$($(repo)_EXE_DEP)
$(COMPILER) $(BUILD_ARG) $($(repo)_BUILD_ARG) `echo "$($(repo)_INC_DIR)" | tr ' ' '\n' | sort -u` -c $($(repo)_DIR)/$
```

4 Unit tests

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include "cryptic.h"

void CypheringFun(
  unsigned char* src,
  unsigned char* dest,
  unsigned char* key,
  unsigned long len) {

  unsigned long lenKey = strlen((char*)key);
```

```
  for (
    unsigned int iChar = 0;
    iChar < len;
    ++iChar) {

    dest[iChar] = src[iChar] + key[iChar % lenKey];

  }

}

void UnitTestFeistelCyphering() {

  GSetStr keys = GSetStrCreateStatic();
  unsigned char keyA[] = "123456";
  unsigned char keyB[] = "abcdef";
  GSetAppend(
    &keys,
    (char*)keyA);
  GSetAppend(
    &keys,
    (char*)keyB);
  unsigned char msg[] = "Hello World.";
  printf("Message:          ");
  for (
    unsigned int iChar = 0;
    iChar < strlen((char*)msg);
    ++iChar) {

    printf(
      "%03u,",
      msg[iChar]);

  }

  printf("\n");
  FeistelCyphering cypher =
    FeistelCypheringCreateStatic(
      &keys,
      &CypheringFun);
  unsigned char* cypheredMsg =
    FeistelCypheringCypher(
      &cypher,
      msg,
      strlen((char*)msg));
  printf("Cyphered message: ");
  for (
    unsigned int iChar = 0;
    iChar < strlen((char*)msg);
    ++iChar) {

    printf(
      "%03u,",
      cypheredMsg[iChar]);

  }

  printf("\n");
  unsigned char* decypheredMsg =
    FeistelCypheringDecypher(
      &cypher,
      cypheredMsg,
```

```
      strlen((char*)msg));
  int ret =
    strcmp(
      (char*)msg,
      (char*)decypheredMsg);
  if (ret != 0) {

    CrypticErr->_type = PBErrTypeUnitTestFailed;
    sprintf(
      CrypticErr->_msg,
      "FeistelCypheringCypher/FeistelCypheringDecypher NOK");
    PBErrCatch(CrypticErr);

  }

  printf(
    "%s\n",
    decypheredMsg);

  FeistelCypheringFreeStatic(&cypher);
  GSetFlush(&keys);
  free(cypheredMsg);
  free(decypheredMsg);
  printf("UnitTestFeistelCyphering OK\n");

}

void UnitTestAll() {

  UnitTestFeistelCyphering();
  printf("UnitTestAll OK\n");

}

int main() {

  UnitTestAll();

  // Return success code
  return 0;

}
```

5 Unit tests output

```
Message:          072,101,108,108,111,032,087,111,114,108,100,046,
Cyphered message: 118,073,094,092,063,132,192,196,201,204,246,068,
Hello World.
UnitTestFeistelCyphering OK
UnitTestAll OK
```