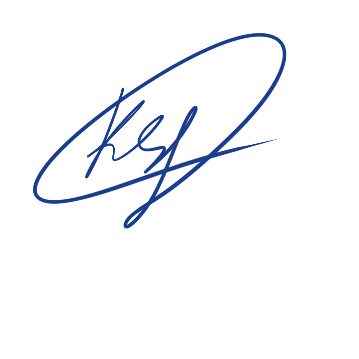
**Федеральное государственное бюджетное образовательное учреждение высшего образования "Белгородский государственный технологический университет им. В.Г. Шухова"**

Кафедра программного обеспечения вычислительной техники и автоматизированных систем.

**Лабораторная работа №1**

Работа с CNG.

Выполнил:

Студент группы КБ-211

Коренев Д.Н.

Принял:

Смакаев А.В.

Оглавление

[Задание 3](#_Toc178204335)

[Разработка программы 3](#_Toc178204336)

[Эксперимент по изменению каждого отдельного бита в открытом тексте 4](#_Toc178204337)

[График зависимости количества измененных в шифротексте битов от позиции измененного бита в открытом тексте 9](#_Toc178204338)

[Вывод 9](#_Toc178204339)

[Приложения 11](#_Toc178204340)

*Цель работы:* ознакомится с криптографическим API операционных систем семейства Windows. Разработать консольное приложение, использующее CNG для генерации ключа и шифрования/дешифрования файлов. Исследовать лавинный эффект при разных режимах сцепления блоков.

Задание

1. Разработать консольное приложение, позволяющее генерировать ключи шифрования и шифровать/дешифровать файлы симметричным шифром AES с использованием криптографического API CNG.

Разработка программы

Исходный код программы см. в приложении 1.

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание  
Рисунок 1. Пример работы программы.

Изображение выглядит как текст, снимок экрана, программное обеспечение, Шрифт

Автоматически созданное описание  
Рисунок 2. --help приложения.

1. Исследовать лавинный эффект при разных режимах сцепления блоков (ECB, CBC, CFB). В ходе исследования нужно
   1. произвести эксперимент по изменению каждого отдельного бита в открытом тексте;

Эксперимент по изменению каждого отдельного бита в открытом тексте

Python: flip-bit.py

import sys

import os

import argparse

def flip\_n\_bit(input\_file, n, output\_file):

    """

    Flip single bit in file at position n (bit numbering starts from 0)

    """

    with open(input\_file, 'rb') as f:

        data = f.read()

    data = bytearray(data)

    byte\_index = n // 8

    bit\_index = n % 8

    data[byte\_index] ^= (1 << bit\_index)

    with open(output\_file, 'wb') as f:

        f.write(data)

def main():

    parser = argparse.ArgumentParser(description='Flip n bits in file')

    parser.add\_argument('input\_file', help='input file')

    parser.add\_argument('n', type=int, help='number of bit to flip')

    parser.add\_argument('output\_file', help='output file')

    args = parser.parse\_args()

    if not os.path.exists(args.input\_file):

        print(f'File {args.input\_file} does not exist')

        sys.exit(1)

    flip\_n\_bit(args.input\_file, args.n, args.output\_file)

if \_\_name\_\_ == '\_\_main\_\_':

    main()

Python: count-bit-diff.py

import sys

import os

import argparse

def bits\_diff(a, b):

    if not isinstance(a, bytes):

        if isinstance(a, int):

            a = bytearray([a])

    if not isinstance(a, bytes):

        if isinstance(a, int):

            b = bytearray([b])

    bits1 = ''.join([f'{a:08b}' for a in a])

    bits2 = ''.join([f'{a:08b}' for a in b])

    len1 = len(bits1)

    len2 = len(bits2)

    max\_len = max(len(bits1), len(bits2))

    bits1 = bits1.ljust(max\_len, '0')

    bits2 = bits2.ljust(max\_len, '0')

    output\_str = ""

    for i in range(max\_len):

        if i < len1 and i < len2:

            if bits1[i] == bits2[i]:

                output\_str += '0'

            else:

                output\_str += '1'

        else:

            output\_str += '1'

    return output\_str.count('1'), output\_str

def print\_bits(a):

    print(f'{a:08b}', end='')

def print\_bits\_array(arr):

    for a in arr:

        print\_bits(a)

def print\_bit\_diff(arr1, arr2):

    diffstr = bits\_diff(arr1, arr2)[1]

    out\_str = ""

    real\_len = min(len(arr1), len(arr2)) \* 8

    for i in range(len(diffstr)):

        if i < real\_len:

            if diffstr[i] == '1':

                out\_str += "\033[38;2;255;16;16m" + '1'

            else:

                out\_str += "\033[38;2;66;166;66m" + '0'

        else:

            out\_str += "\033[38;2;228;128;128m" + '1'

    out\_str += "\033[0m"

    print(out\_str)

def main():

    parser = argparse.ArgumentParser(

        description='Calculate count of bits different between two files')

    parser.add\_argument('file1', help='first file')

    parser.add\_argument('file2', help='second file')

    # optional argument

    parser.add\_argument('--limit', type=int, default=4096,

                        help='limit of comparison size in bytes')

    parser.add\_argument('--log-file', type=str, default=None, help='log file')

    parser.add\_argument('--verbose', action='store\_true',

                        help='print bit representation of files')

    args = parser.parse\_args()

    log\_file\_hand = None

    if args.log\_file is not None:

        log\_file\_hand = open(args.log\_file, 'w')

    if not os.path.exists(args.file1):

        print(f'File {args.file1} does not exist')

        sys.exit(1)

    if not os.path.exists(args.file2):

        print(f'File {args.file2} does not exist')

        sys.exit(1)

    if args.limit <= 0 or args.limit == None:

        file\_size = min(

            os.path.getsize(args.file1),

            os.path.getsize(args.file2))

        args.limit = file\_size

    with open(args.file1, 'rb') as f1, open(args.file2, 'rb') as f2:

        data1 = f1.read()

        data2 = f2.read()

        if args.verbose:

            print('src1:\t', end='')

            print\_bits\_array(data1)

            print()

            print('src2:\t', end='')

            print\_bits\_array(data2)

            print()

            print('diff:\t', end='')

            print\_bit\_diff(data1, data2)

            print(bits\_diff(data1, data2))

        else:

            print(bits\_diff(data1, data2)[0])

if \_\_name\_\_ == '\_\_main\_\_':

    main()

Python: make-graphs.py

import pandas as pd

import matplotlib.pyplot as plt

# Read CSV files without headers and assign column names

ecb\_df = pd.read\_csv('./temp/ecb.log', header=None, names=['index', 'value'])

cfb\_df = pd.read\_csv('./temp/cfb.log', header=None, names=['index', 'value'])

cbc\_df = pd.read\_csv('./temp/cbc.log', header=None, names=['index', 'value'])

# Plot the data

plt.figure(figsize=(10, 6))

plt.plot(ecb\_df['index'], ecb\_df['value'], label='ECB')

plt.plot(cfb\_df['index'], cfb\_df['value'], label='CFB')

plt.plot(cbc\_df['index'], cbc\_df['value'], label='CBC')

# Set y-axis to logarithmic scale

# plt.yscale('log')

# Add labels and title

plt.xlabel('Changed bit in plaintext')

plt.ylabel('Count of changed bits in ciphertext')

plt.title('Avalanche effect of different block cipher modes')

# Add legend

plt.legend()

# Display the plot

# plt.show()

# Save the plot

plt.savefig('./temp/block\_cipher\_modes.png')

PowerShell: win\_test\_avalanche.ps1

$build\_path = ".\build\cifs.exe"

$file\_to\_test = ".\tests\lorem.txt"

$ecb\_log\_file = ".\temp\ecb.log"

$cbc\_log\_file = ".\temp\cbc.log"

$cfb\_log\_file = ".\temp\cfb.log"

$bits\_to\_flip = @(0, 2, 8, 16, 32, 64, 128, 256, 384, 512, 768, 1024, 1536, 2048, 2500, 3084, 3500, 4096, 4500, 5000, 5500, 6000, 6500, 6750, 7000, 7250, 7500, 7750, 8192)  # Define the array with values to test

# copy test file to temp

Copy-Item $file\_to\_test .\temp\aval

$file\_to\_test = ".\temp\aval"

# clear log files

Remove-Item $ecb\_log\_file

Remove-Item $cbc\_log\_file

Remove-Item $cfb\_log\_file

Write-Host "<] Generate key and IV"

& $build\_path -m keygen -g .\temp\key.bin

if ($LASTEXITCODE -ne 0) { exit $LASTEXITCODE }

& $build\_path -m iv -g .\temp\iv.bin

if ($LASTEXITCODE -ne 0) { exit $LASTEXITCODE }

Write-Host "<] Generate reference test file with ECB mode for txt file"

& $build\_path -m enc -k .\temp\key.bin -i .\temp\iv.bin -f $file\_to\_test -c ecb -o "$file\_to\_test.enc"

if ($LASTEXITCODE -ne 0) { exit $LASTEXITCODE }

Write-Host "<] Run tests for ECB mode"

foreach ($bit\_to\_flip in $bits\_to\_flip) {

    & .\tests\flip-bit.py $file\_to\_test $bit\_to\_flip "$file\_to\_test-$bit\_to\_flip.txt"

    if ($LASTEXITCODE -ne 0) { exit $LASTEXITCODE }

    & $build\_path -m enc -k .\temp\key.bin -i .\temp\iv.bin -f "$file\_to\_test-$bit\_to\_flip.txt" -c ecb -o "$file\_to\_test-$bit\_to\_flip.enc"

    if ($LASTEXITCODE -ne 0) { exit $LASTEXITCODE }

    $log\_entry = "$bit\_to\_flip, "

    $count\_bit\_diff\_output = & .\tests\count-bit-diff.py "$file\_to\_test.enc" "$file\_to\_test-$bit\_to\_flip.enc"

    if ($LASTEXITCODE -ne 0) { exit $LASTEXITCODE }

    $log\_entry += $count\_bit\_diff\_output

    $log\_entry >> $ecb\_log\_file

}

Write-Host "<] Generate reference test file with CBC mode for txt file"

& $build\_path -m enc -k .\temp\key.bin -i .\temp\iv.bin -f $file\_to\_test -c cbc -o "$file\_to\_test.enc"

if ($LASTEXITCODE -ne 0) { exit $LASTEXITCODE }

Write-Host "<] Run tests for CBC mode"

foreach ($bit\_to\_flip in $bits\_to\_flip) {

    & .\tests\flip-bit.py $file\_to\_test $bit\_to\_flip "$file\_to\_test-$bit\_to\_flip.txt"

    if ($LASTEXITCODE -ne 0) { exit $LASTEXITCODE }

    & $build\_path -m enc -k .\temp\key.bin -i .\temp\iv.bin -f "$file\_to\_test-$bit\_to\_flip.txt" -c cbc -o "$file\_to\_test-$bit\_to\_flip.enc"

    if ($LASTEXITCODE -ne 0) { exit $LASTEXITCODE }

    $log\_entry = "$bit\_to\_flip, "

    $count\_bit\_diff\_output = & .\tests\count-bit-diff.py "$file\_to\_test.enc" "$file\_to\_test-$bit\_to\_flip.enc"

    if ($LASTEXITCODE -ne 0) { exit $LASTEXITCODE }

    $log\_entry += $count\_bit\_diff\_output

    $log\_entry >> $cbc\_log\_file

}

Write-Host "<] Generate reference test file with CFB mode for txt file"

& $build\_path -m enc -k .\temp\key.bin -i .\temp\iv.bin -f $file\_to\_test -c cfb -o "$file\_to\_test.enc"

if ($LASTEXITCODE -ne 0) { exit $LASTEXITCODE }

Write-Host "<] Run tests for CFB mode"

foreach ($bit\_to\_flip in $bits\_to\_flip) {

    & .\tests\flip-bit.py $file\_to\_test $bit\_to\_flip "$file\_to\_test-$bit\_to\_flip.txt"

    if ($LASTEXITCODE -ne 0) { exit $LASTEXITCODE }

    & $build\_path -m enc -k .\temp\key.bin -i .\temp\iv.bin -f "$file\_to\_test-$bit\_to\_flip.txt" -c cfb -o "$file\_to\_test-$bit\_to\_flip.enc"

    if ($LASTEXITCODE -ne 0) { exit $LASTEXITCODE }

    $log\_entry = "$bit\_to\_flip, "

    $count\_bit\_diff\_output = & .\tests\count-bit-diff.py "$file\_to\_test.enc" "$file\_to\_test-$bit\_to\_flip.enc"

    if ($LASTEXITCODE -ne 0) { exit $LASTEXITCODE }

    $log\_entry += $count\_bit\_diff\_output

    $log\_entry >> $cfb\_log\_file

}

Write-Host "<] Plotting graph"

& .\tests\make-graphs.py

if ($LASTEXITCODE -ne 0) { exit $LASTEXITCODE }

Write-Host "<] Done"

Write-Host "<] Removing temp files"

Remove-Item .\temp\aval\*

* 1. построить график зависимости количества измененных в шифротексте битов от позиции измененного бита в открытом тексте.

График зависимости количества измененных в шифротексте битов от позиции измененного бита в открытом тексте

Изображение выглядит как текст, снимок экрана, График, линия

Автоматически созданное описание  
Рисунок 3. График зависимости количества измененных в шифротексте битов от позиции измененного бита в открытом тексте.

Вывод

В ходе работы мы разработали консольное приложение для генерации ключей и шифрования/дешифрования файлов с использованием симметричного шифра AES через криптографическое API CNG. Программа успешно выполняла операции шифрования и дешифрования файлов, а также предоставляла интерфейс для взаимодействия с пользователем через командную строку.

Целью эксперимента было исследование лавинного эффекта при изменении битов в открытом тексте при различных режимах сцепления блоков (ECB, CBC, CFB). Лавинный эффект проявлялся по-разному в зависимости от режима. В режиме ECB наблюдалось отсутствие сильного лавинного эффекта, так как изменение одного бита в открытом тексте приводило к изменению только соответствующего блока шифротекста. В режиме CFB лавинный эффект был выражен сильнее — изменение одного бита в открытом тексте приводило к изменению всех последующих блоков шифротекста. В режиме CBC лавинный эффект также проявлялся, но его интенсивность зависела от положения измененного бита в открытом тексте.

Мы установили, что лавинный эффект значительно зависит как от выбранного режима сцепления блоков, так и от позиции инвертируемого бита в открытом тексте.

Приложения

Приложение 1. Исходный код программы.

С

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#include <stdint.h>

#include <string.h>

#include <errno.h>

#include <unistd.h>

#define ADD\_EXPORTS

#define IMECLUI\_IMPLEMENTATION

#include "src/imeclui.h"

#define ARGPARSE\_IMPLEMENTATION

#include "src/argparse.h"

#ifdef \_LINUX

#include "kcapi.h"

#endif // \_LINUX

#ifdef \_WIN32

#include <windows.h>

#include <bcrypt.h>

#endif // \_WIN32

// ===--- MACROS ---============================================================

#define \_\_MACROS

// Colors

#define C\_RESET \

    IME\_ESC     \

    IME\_RESET   \

    IME\_ESC\_END

#define C\_RED \

    IME\_ESC   \

    IME\_RED   \

    IME\_ESC\_END

#define C\_GREEN \

    IME\_ESC     \

    IME\_GREEN   \

    IME\_ESC\_END

#define C\_CYAN                 \

    IME\_ESC                    \

    IME\_RGB\_COLOR(0, 200, 180) \

    IME\_ESC\_END

#define C\_DIMM       \

    IME\_ESC          \

    IME\_BRIGHT\_BLACK \

    IME\_ESC\_END

#define C\_HEADER                \

    IME\_ESC                     \

    IME\_RGB\_COLOR(255, 117, 24) \

    IME\_ESC\_END

#define C\_ERROR \

    C\_RED

// Main assert, in a 'slay💅' style

/// @brief Assert a condition

/// @param cond [in] condition to assert

/// @param msg [in] message to print if the condition is false

#define MASSERT(cond, msg)                                              \

    if (!(cond))                                                        \

    {                                                                   \

        printf(C\_ERROR);                                                \

        printf("[ASSERTION FAILED] (%lld) <%s:%d:%s> %s",               \

               (long long)(cond), \_\_FILE\_\_, \_\_LINE\_\_, \_\_func\_\_, msg);   \

        printf(C\_RESET "\n");                                           \

        exit(-1);                                                       \

    }

// Preferrable allocators

/// @brief Allocate memory for an array of type T

/// @param T [in] type of the array

/// @param size [in] size of the array

/// @return pointer to the allocated memory

#define ALLOC(T, size) ((T \*)malloc((size) \* sizeof(T)))

/// @brief Allocate memory for an array of type T and set it to zero

/// @param T [in] type of the array

/// @param size [in] size of the array

/// @return pointer to the allocated memory

#define CALLOC(T, size) ((T \*)calloc((size), sizeof(T)))

/// @brief Reallocate memory for an array of type T

/// @param T [in] type of the array

/// @param ptr [in] pointer to the array

/// @param size [in] size of the array

#define REALLOC(T, ptr, size) ((T \*)realloc(ptr, (size) \* sizeof(T)))

/// @brief Free memory

/// @param ptr [in] pointer to the memory

#define FREE(ptr)    \

    if (ptr != NULL) \

    {                \

        free(ptr);   \

        ptr = NULL;  \

    }

/// @brief Start progress bar

/// @param pbar [in] name of the progress bar

/// @param len [in] length of the progress bar

#define START\_PROGRESS\_BAR(pbar, len) \

    char \_\_##pbar[len] = {0};         \

    memset(\_\_##pbar, '#', sizeof(\_\_##pbar) - 1)

/// @brief Print progress bar

/// @param pbar [in] name of the progress bar

/// @param per [in] percentage of the progress

/// @param fmt [in] format string additional information

/// @param ... [in] additional information for the format string

#define PROGRESS\_BAR\_RUNING(pbar, per, fmt, ...) ({                 \

    float p = per >= 100.0 ? 100.0 : per;                           \

    int left = (p / 100.0) \* (sizeof(\_\_##pbar) - 1);                \

    int right = (sizeof(\_\_##pbar) - 1) - left;                      \

    printf("\r[%.\*s%\*s] %.0f%%" fmt, left, \_\_##pbar, right, "",     \

        (float)per, \_\_VA\_ARGS\_\_);                                   \

})

/// @brief Update progress bar

/// @param pbar [in] name of the progress bar

/// @param i [in] current iteration

/// @param numIterations [in] total number of iterations

#define UPDATE\_PROGRESS\_BAR(pbar, i, numIterations)                     \

    PROGRESS\_BAR\_RUNING(pbar, (float)(i + 1) / numIterations \* 100,     \

        " (%lld/%lld)", (long long)(i + 1), (long long)numIterations);  \

    if (numIterations == i + 1) {                                       \

        printf("\n");                                                   \

    }                                                                   \

// int main(int argc, char \*argv[])

// {

//     char buf[8192];

//     struct kcapi\_handle \*handle;

//     struct iovec iov;

//     ssize\_t ret;

//     int i;

//     (void)argc;

//     (void)argv;

//     iov.iov\_base = buf;

//     ret = kcapi\_cipher\_init(&handle, "cbc(aes)", 0);

//     if (ret)

//             return (int)ret;

//     ret = kcapi\_cipher\_setkey(handle, (unsigned char \*)"0123456789abcdef", 16);

//     if (ret)

//             return (int)ret;

//     ret = kcapi\_cipher\_stream\_init\_enc(handle, (unsigned char \*)"0123456789abcdef", NULL, 0);

//     if (ret < 0)

//             return (int)ret;

//  for (i = 0; i < 100; i++) {

//      //printf("round %d\n", i);

//      iov.iov\_len = 6182;

//      ret = kcapi\_cipher\_stream\_update(handle, &iov, 1);

//      if (ret < 0)

//          return (int)ret;

//      iov.iov\_len = 6182;

//      ret = kcapi\_cipher\_stream\_op(handle, &iov, 1);

//      if (ret < 0)

//          return (int)ret;

//  }

//         kcapi\_cipher\_destroy(handle);

//         return 0;

// }

/// @brief Write byte array to a binary file

/// @param bytes

/// @param size

/// @param file\_name

void fwrite\_bin(const uint8\_t \*bytes, size\_t size, const char \*file\_name)

{

    FILE \*file = fopen(file\_name, "wb");

    MASSERT(file != NULL, "Can't open file for writing");

    fwrite(bytes, 1, size, file);

    fclose(file);

}

/// @brief Read binary file to a byte array

/// @param bytes pointer to the result

/// @param size pointer to the result

/// @param file\_name

/// @warning The result must be freed after usage

void fread\_bin(uint8\_t \*\*bytes, size\_t \*size, const char \*file\_name)

{

    FILE \*file = fopen(file\_name, "rb");

    MASSERT(file != NULL, "Can't open file for reading");

    fseek(file, 0, SEEK\_END);

    \*size = ftell(file);

    fseek(file, 0, SEEK\_SET);

    \*bytes = ALLOC(uint8\_t, \*size);

    MASSERT(\*bytes != NULL, "Memory allocation failed");

    fread(\*bytes, 1, \*size, file);

    fclose(file);

}

/// @brief Print byte as binary number

/// @param byte [in] byte to print

void print\_byte\_bin(uint8\_t byte)

{

    int i;

    for (i = 0; i < 8; i++)

    {

        uint8\_t shift\_byte = 0x01 << (7 - i);

        if (shift\_byte & byte)

        {

            printf("1");

        }

        else

        {

            printf("0");

        }

    }

}

/// @brief Convert array of integers to a null-terminated

/// string of ASCII characters

/// @param data [in] array of integers

/// @param data\_size [in] array size

/// @param str [in] pointer to the result

/// @warning The result must be freed after usage

void convert\_array\_to\_str(unsigned long long int \*data, size\_t data\_size,

                          char \*\*str)

{

    char \*res = CALLOC(char, data\_size);

    MASSERT(res != NULL, "Memory allocation failed");

    for (size\_t i = 0; i < data\_size; i++)

    {

        res[i] = (char)data[i];

    }

    \*str = res;

}

/// @brief Read chunk of data from file

/// @param bytes pointer to the result

/// @param size pointer to the result

/// @param start start position in file

/// @param end end position in file

/// @param file\_name

/// @warning The result must be freed after usage

void read\_bin\_file\_chunk(uint8\_t \*\*bytes, size\_t \*size,

                         size\_t start, size\_t end,

                         const char \*file\_name)

{

    FILE \*file = fopen(file\_name, "rb");

    MASSERT(file != NULL, "Can't open file");

    fseek(file, 0, SEEK\_END);

    size\_t file\_size = ftell(file);

    fseek(file, 0, SEEK\_SET);

    if (end == 0)

    {

        end = file\_size;

    }

    if (end > file\_size)

    {

        end = file\_size;

    }

    \*size = end - start;

    \*bytes = ALLOC(uint8\_t, \*size);

    MASSERT(\*bytes != NULL, "Memory allocation failed");

    fseek(file, start, SEEK\_SET);

    fread(\*bytes, 1, \*size, file);

    fclose(file);

}

/// @brief Write chunk of data to a specified position in file

/// @param bytes data to write

/// @param size data size

/// @param start start position in file

/// @param file\_name file name

void write\_bin\_file\_chunk(const uint8\_t \*bytes, size\_t size,

                          size\_t start, const char \*file\_name)

{

    FILE \*file = fopen(file\_name, "r+b");

    if (file == NULL)

    {

        file = fopen(file\_name, "wb");

    }

    MASSERT(file != NULL, "Can't open file");

    fseek(file, start, SEEK\_SET);

    fwrite(bytes, 1, size, file);

    fclose(file);

}

/// @brief Count chunks in the file

/// @param file\_name

/// @param chunk\_size

/// @return number of chunks in the file

size\_t count\_file\_chunks(const char \*file\_name, size\_t chunk\_size)

{

    FILE \*file = fopen(file\_name, "rb");

    MASSERT(file != NULL, "Can't open file");

    fseek(file, 0, SEEK\_END);

    size\_t file\_size = ftell(file);

    fclose(file);

    return (file\_size + chunk\_size - 1) / chunk\_size;

}

/// @brief Get file size

/// @param file\_name

/// @return file size in bytes

size\_t file\_size(const char \*file\_name)

{

    FILE \*file = fopen(file\_name, "rb");

    MASSERT(file != NULL, "Can't open file");

    fseek(file, 0, SEEK\_END);

    size\_t file\_size = ftell(file);

    fclose(file);

    return file\_size;

}

/// @brief Print byte array as hex string

/// @param array [in] byte array

/// @param size [in] array size

void print\_byte\_array\_hex(uint8\_t \*array, size\_t size) {

    printf("0x");

    for (size\_t i = 0; i < size; i++) {

        printf("%02X", (uint8\_t)array[i]);

    }

    printf("\n");

}

/// @brief Lowercase string

/// @param str [in/out] string to convert

void str\_lower(char \*str) {

    for (int i = 0; str[i]; i++) {

        str[i] = tolower(str[i]);

    }

}

/// @brief Lowercase wide string

/// @param str [in/out] string to convert

void wstr\_lower(wchar\_t \*str) {

    for (int i = 0; str[i]; i++) {

        str[i] = towlower(str[i]);

    }

}

/// @brief Check if substring is in the string

/// @param str [in] string

/// @param substr [in] substring

/// @return true if substring is in the string

bool is\_in\_str(const char \*str, const char \*substr) {

    return strstr(str, substr) != NULL;

}

#ifdef \_LINUX

int test\_sym\_aes(){

    char buf[8192];

    struct kcapi\_handle \*handle;

    // Initialization vector

    struct iovec iov;

    int i;

    iov.iov\_base = buf;

    // struct kcapi\_handle \*handle = NULL;

    const char \*key = "0123456789abcdef"; // 16 bytes key for AES-128

    const char \*iv = "abcdef9876543210";  // 16 bytes IV for AES-CBC

    char \*plaintext\_orig = "This is a test message... cat cat";

    unsigned char ciphertext[128]; // Ensure this is large enough for your data

    int ret;

    char \*plaintext = ALLOC(char, strlen(plaintext\_orig) + 1 + 100);

    strcpy(plaintext, plaintext\_orig);

    char \*text\_out = CALLOC(char, strlen(plaintext\_orig) + 1 + 100);

    printf("Plaintext: %s\n", plaintext);

    // Initialize the cipher handle

    // char \*alg = "cbc(aes)";

    char \*alg = "ecb(aes)";

    ret = kcapi\_cipher\_init(&handle, alg, 0);

    if (ret) {

        fprintf(stderr, "kcapi\_cipher\_init() failed: %d\n", ret);

        fprintf(stderr, "Error: %s\n", strerror(errno));

        return ret;

    }

    printf("Handle: 0x%016llX\n", (long long unsigned int)handle);

    printf("Using: %s\n", alg);

    // Set the encryption key

    ret = kcapi\_cipher\_setkey(handle, (const uint8\_t\*)key, strlen(key));

    if (ret) {

        fprintf(stderr, "kcapi\_cipher\_setkey() failed: %d\n", ret);

        fprintf(stderr, "Error: %s\n", strerror(errno));

        kcapi\_cipher\_destroy(handle);

        return ret;

    }

    // Encrypt the data

    ret = kcapi\_cipher\_stream\_init\_enc(handle, (const uint8\_t\*)iv, NULL, 0);

    if (ret < 0) {

        fprintf(stderr, "kcapi\_cipher\_stream\_init\_enc() failed: %d\n", ret);

        fprintf(stderr, "Error: %s\n", strerror(errno));

        kcapi\_cipher\_destroy(handle);

        return ret;

    }

    for (i = 0; i < strlen(plaintext); i++) {

        iov.iov\_len = 1;

        iov.iov\_base = (void \*)plaintext + i;

        ret = kcapi\_cipher\_stream\_update(handle, &iov, 1);

        if (ret < 0) {

            fprintf(stderr, "kcapi\_cipher\_stream\_update() failed: %d\n", ret);

            fprintf(stderr, "Error: %s\n", strerror(errno));

            kcapi\_cipher\_destroy(handle);

            return ret;

        }

        ciphertext[i] = ((char \*)iov.iov\_base)[0];

    }

    printf("Encoded: ");

    for (int i = 0; i < strlen(plaintext); i++) {

        printf("%02x", ciphertext[i]);

    }

    printf("\n");

    // prepare to decode data

    ret = kcapi\_cipher\_stream\_init\_dec(handle, (const uint8\_t\*)iv, NULL, 0);

    if (ret < 0) {

        fprintf(stderr, "kcapi\_cipher\_stream\_init\_dec() failed: %d\n", ret);

        fprintf(stderr, "Error: %s\n", strerror(errno));

        kcapi\_cipher\_destroy(handle);

        return ret;

    }

    // Decode the data

    for (i = 0; i < strlen(plaintext); i++) {

        iov.iov\_len = 1;

        iov.iov\_base = (void \*)ciphertext + i;

        ret = kcapi\_cipher\_stream\_update(handle, &iov, 1);

        if (ret < 0) {

            fprintf(stderr, "kcapi\_cipher\_stream\_update() failed: %d\n", ret);

            fprintf(stderr, "Error: %s\n", strerror(errno));

            kcapi\_cipher\_destroy(handle);

            return ret;

        }

        text\_out[i] = ((char \*)iov.iov\_base)[0];

    }

    printf("Decoded: %s\n", text\_out);

    printf("Encryption successful. Ciphertext length: %d\n", ret);

    // Clean up

    FREE(text\_out);

    FREE(plaintext)

    kcapi\_cipher\_destroy(handle);

    return 0;

}

void generate\_key(const char \*key\_path)

{

    uint8\_t key[16];

    FILE \*file = fopen(key\_path, "wb");

    MASSERT(file != NULL, "Can't open file for writing");

    for (int i = 0; i < 16; i++)

    {

        key[i] = rand() % 256;

    }

    fwrite(key, 1, 16, file);

    fclose(file);

}

uint8\_t\* kpp\_keygen(const char \*ciphername)

{

    struct kcapi\_handle \*handle = NULL;

    uint8\_t \*outbuf = NULL;

    size\_t outbuflen;

    ssize\_t ret;

    ret = kcapi\_kpp\_init(&handle, "ctr(aes)", 0);

    fprintf(stderr, "kcapi\_kpp\_init() failed: %d\n", ret);

    fprintf(stderr, "Error: %s\n", strerror(errno));

    MASSERT(ret >= 0, "Initialization of cipher failed");

    // ret = kcapi\_kpp\_setkey(handle, cavs\_test->key, cavs\_test->keylen);

    // MASSERT(ret >= 0, "Having kernel generating keys failed\n");

    outbuflen = 128;

    {

        outbuf = CALLOC(uint8\_t, (size\_t)ret);

        MASSERT(outbuf != NULL, "Memory allocation failed");

    }

    ret = kcapi\_kpp\_keygen(handle, outbuf, outbuflen, KCAPI\_ACCESS\_SENDMSG);

    MASSERT(ret >= 0, "Key generation failed");

    kcapi\_kpp\_destroy(handle);

    // kcapi\_aes\_

    return outbuf;

}

#endif // \_LINUX

#ifdef \_WIN32

#define NT\_SUCCESS(Status) (((NTSTATUS)(Status)) >= 0)

#define AES\_KEY\_BYTE\_SIZE 16 // 128 bits

#define AES\_KEY\_BIT\_SIZE AES\_KEY\_BYTE\_SIZE \* 8

// #define CHUNK\_SIZE 1024 \* 1024 \* 1024 // 1024 MB

#define CHUNK\_SIZE 1024 \* 1024 // 1 MB

/// @brief Generate AES key and save it to a file

/// @param key\_path [in] path to the key file

void win\_generate\_aes\_key(char \*key\_path) {

    BCRYPT\_ALG\_HANDLE hAlgorithm = NULL;

    BCRYPT\_KEY\_HANDLE hKey = NULL;

    NTSTATUS status;

    DWORD keyObjectLength = 0, resultLength = 0;

    PBYTE keyObject = NULL;

    PBYTE keyBlob = NULL;

    PBYTE keyMaterial = NULL;

    // Open an algorithm handle

    status = BCryptOpenAlgorithmProvider(&hAlgorithm,

        BCRYPT\_AES\_ALGORITHM, NULL, 0);

    MASSERT(NT\_SUCCESS(status), "BCryptOpenAlgorithmProvider failed");

    // Calculate the size of the buffer to hold the key object

    status = BCryptGetProperty(hAlgorithm, BCRYPT\_OBJECT\_LENGTH,

        (PBYTE)&keyObjectLength, sizeof(DWORD), &resultLength, 0);

    MASSERT(NT\_SUCCESS(status), "BCryptGetProperty failed");

    // Allocate the key object

    keyObject = (PBYTE)HeapAlloc(GetProcessHeap(), 0, keyObjectLength);

    MASSERT(keyObject != NULL, "HeapAlloc failed");

    // Allocate the key material

    keyMaterial = (PBYTE)HeapAlloc(GetProcessHeap(), 0, AES\_KEY\_BYTE\_SIZE);

    MASSERT(keyMaterial != NULL, "HeapAlloc failed");

    // Generate random key material

    status = BCryptGenRandom(NULL, keyMaterial, AES\_KEY\_BYTE\_SIZE,

        BCRYPT\_USE\_SYSTEM\_PREFERRED\_RNG);

    MASSERT(NT\_SUCCESS(status), "BCryptGenRandom failed");

    // Generate the key

    status = BCryptGenerateSymmetricKey(hAlgorithm, &hKey, keyObject,

        keyObjectLength, keyMaterial, AES\_KEY\_BYTE\_SIZE, 0);

    MASSERT(NT\_SUCCESS(status), "BCryptGenerateSymmetricKey failed");

    // Get size of the key and allocate memory

    status = BCryptExportKey(hKey, NULL, BCRYPT\_OPAQUE\_KEY\_BLOB, NULL, 0,

        &resultLength, 0);

    MASSERT(NT\_SUCCESS(status), "BCryptExportKey failed");

    keyBlob = (PBYTE)HeapAlloc(GetProcessHeap(), 0, resultLength);

    MASSERT(keyBlob != NULL, "HeapAlloc failed");

    // Export the key

    // BCRYPT\_KEY\_DATA\_BLOB - key material is exported in plaintext form

    // BCRYPT\_OPAQUE\_KEY\_BLOB - key material is embedded in a BLOB that is

    //      encrypted

    status = BCryptExportKey(hKey, NULL, BCRYPT\_OPAQUE\_KEY\_BLOB, keyBlob,

        resultLength, &resultLength, 0);

    MASSERT(NT\_SUCCESS(status), "BCryptExportKey failed");

    // Save the key to a file or use it as needed

    FILE \*keyFile = fopen(key\_path, "wb");

    if (keyFile) {

        fwrite(keyBlob, 1, resultLength, keyFile);

        fclose(keyFile);

        printf("AES%d key generated and saved to \"%s\"\n",

            AES\_KEY\_BIT\_SIZE, key\_path);

    } else {

        printf("Failed to open file for writing\n");

    }

    if (hKey) BCryptDestroyKey(hKey);

    if (hAlgorithm) BCryptCloseAlgorithmProvider(hAlgorithm, 0);

    if (keyObject) HeapFree(GetProcessHeap(), 0, keyObject);

    if (keyBlob) HeapFree(GetProcessHeap(), 0, keyBlob);

}

/// @brief Read AES key from file

/// @param key\_path [in] path to the key file

/// @param keyBlob [out] pointer to the key

/// @param keySize [out] pointer to the key size

void win\_read\_aes\_key(char \*key\_path, BYTE \*\*keyBlob, DWORD \*keySize) {

    FILE \*keyFile = fopen(key\_path, "rb");

    if (keyFile) {

        fseek(keyFile, 0, SEEK\_END);

        \*keySize = ftell(keyFile);

        fseek(keyFile, 0, SEEK\_SET);

        // keyBlob = (BYTE \*)HeapAlloc(GetProcessHeap(), 0, keySize);

        \*keyBlob = ALLOC(BYTE, (size\_t)\*keySize);

        if (\*keyBlob) {

            fread(\*keyBlob, 1, (size\_t)\*keySize, keyFile);

            fclose(keyFile);

            printf("Key read from file \"%s\"\n", key\_path);

        } else {

            printf("Failed to allocate memory for key\n");

        }

    } else {

        printf("Failed to open file for reading\n");

    }

}

/// @brief Generate AES initialization vector and save it to a file

/// @param iv\_path [in] path to the IV file

void win\_generate\_aes\_iv(char \*iv\_path) {

    BCRYPT\_ALG\_HANDLE hAlgorithm = NULL;

    NTSTATUS status;

    DWORD ivSize = AES\_KEY\_BYTE\_SIZE; // AES block size is 16 bytes

    BYTE iv[AES\_KEY\_BYTE\_SIZE];

    FILE \*ivFile = NULL;

    // Open an algorithm handle

    status = BCryptOpenAlgorithmProvider(&hAlgorithm,

        BCRYPT\_AES\_ALGORITHM, NULL, 0);

    MASSERT(NT\_SUCCESS(status), "BCryptOpenAlgorithmProvider failed");

    // Generate random IV

    status = BCryptGenRandom(NULL, iv, ivSize, BCRYPT\_USE\_SYSTEM\_PREFERRED\_RNG);

    MASSERT(NT\_SUCCESS(status), "BCryptGenRandom failed");

    // Write IV to file

    ivFile = fopen(iv\_path, "wb");

    MASSERT(ivFile != NULL, "Failed to open IV file for writing");

    fwrite(iv, 1, ivSize, ivFile);

    fclose(ivFile);

    printf("IV(%d) generated and saved to \"%s\"\n", AES\_KEY\_BIT\_SIZE, iv\_path);

    // Clean up

    if (hAlgorithm) BCryptCloseAlgorithmProvider(hAlgorithm, 0);

}

/// @brief Read AES initialization vector from file

/// @param iv\_path [in] path to the IV file

/// @param ivBlob [out] pointer to the IV

/// @param ivSize [out] pointer to the IV size

void win\_read\_aes\_iv(char \*iv\_path, BYTE \*\*ivBlob, DWORD \*ivSize) {

    FILE \*ivFile = fopen(iv\_path, "rb");

    MASSERT(ivFile != NULL, "Failed to open IV file");

    fseek(ivFile, 0, SEEK\_END);

    \*ivSize = ftell(ivFile);

    fseek(ivFile, 0, SEEK\_SET);

    \*ivBlob = ALLOC(BYTE, \*ivSize);

    MASSERT(\*ivBlob != NULL, "Memory allocation failed");

    fread(\*ivBlob, 1, \*ivSize, ivFile);

    fclose(ivFile);

    printf("IV read from file \"%s\"\n", iv\_path);

}

/// @brief Encrypt chunk using AES

/// @param chunk [in]

/// @param chunk\_size [in] size of the chunk

/// @param hAlgorithm [in] algorithm handle

/// @param hKey [in] key handle

/// @param iv [in] initialization vector

/// @param ivSize [in] size of the initialization vector

/// @param ciphertext [out] pointer to the ciphertext

/// @param ciphertextSize [out] pointer to the ciphertext size

void win\_enc\_chunk\_aes(BYTE \*chunk, DWORD chunk\_size,

                       BCRYPT\_ALG\_HANDLE hAlgorithm, BCRYPT\_KEY\_HANDLE hKey,

                       BYTE \*iv, DWORD ivSize,

                       BYTE \*\*ciphertext, DWORD \*ciphertextSize)

{

    NTSTATUS status;

    DWORD resultLength = 0;

    // Calculate the buffer size for the ciphertext

    status = BCryptEncrypt(hKey, chunk, chunk\_size, NULL, iv, ivSize, NULL, 0,

                           ciphertextSize, BCRYPT\_BLOCK\_PADDING);

    MASSERT(NT\_SUCCESS(status), "BCryptEncrypt failed");

#ifdef \_DEBUG

    printf("chunk\_size: %lu\n", chunk\_size);

#endif // \_DEBUG

    // Allocate the ciphertext buffer

    \*ciphertext = ALLOC(BYTE, \*ciphertextSize);

    MASSERT(\*ciphertext != NULL, "HeapAlloc failed");

    // Perform the encryption

    status = BCryptEncrypt(hKey, chunk, chunk\_size, NULL, iv, ivSize,

                           \*ciphertext, \*ciphertextSize, &resultLength,

                           BCRYPT\_BLOCK\_PADDING);

    // get chaining mode

    wchar\_t chainingMode[64]; // Buffer to hold the chaining mode

    DWORD chainingModeSize = sizeof(chainingMode);

    ULONG result = 0;

    status = BCryptGetProperty(hAlgorithm, BCRYPT\_CHAINING\_MODE,

                               (PBYTE)chainingMode, chainingModeSize,

                               &result, 0);

    MASSERT(NT\_SUCCESS(status), "BCryptGetProperty failed");

    // if (wcscmp(chainingMode, (wchar\_t \*)BCRYPT\_CHAIN\_MODE\_ECB) == 0) {

    //     \*ciphertextSize = chunk\_size;

    // } else {

    //     \*ciphertextSize = resultLength;

    // }

#ifdef \_DEBUG

    printf("status: %ld\n", status);

    printf("cipherTextSize: %lu\n", \*ciphertextSize);

    printf("resultLength: %lu\n", resultLength);

#endif // \_DEBUG

    MASSERT(NT\_SUCCESS(status), "BCryptEncrypt failed");

}

/// @brief Decrypt chunk using AES

/// @param chunk [in]

/// @param chunk\_size [in] size of the chunk

/// @param hAlgorithm [in] algorithm handle

/// @param hKey [in] key handle

/// @param iv [in] initialization vector

/// @param ivSize [in] size of the initialization vector

/// @param plaintext [out] pointer to the plaintext

/// @param plaintextSize [out] pointer to the plaintext size

void win\_dec\_chunk\_aes(BYTE \*chunk, DWORD chunk\_size,

                       BCRYPT\_ALG\_HANDLE hAlgorithm, BCRYPT\_KEY\_HANDLE hKey,

                       BYTE \*iv, DWORD ivSize,

                       BYTE \*\*plaintext, DWORD \*plaintextSize, bool isLastChunk)

{

    NTSTATUS status;

    DWORD resultLength = 0;

    // Calculate the buffer size for the plaintext

    status = BCryptDecrypt(hKey, chunk, chunk\_size, NULL, iv, ivSize, NULL, 0,

                           plaintextSize, BCRYPT\_BLOCK\_PADDING);

    MASSERT(NT\_SUCCESS(status), "BCryptDecrypt failed");

    // Allocate the plaintext buffer

    \*plaintext = ALLOC(BYTE, \*plaintextSize);

    MASSERT(\*plaintext != NULL, "HeapAlloc failed");

    // Perform the decryption

#ifdef \_DEBUG

    printf("chunk\_size: %lu\n", chunk\_size);

    printf("ivSize: %lu\n", ivSize);

    printf("plaintextSize: %lu\n", \*plaintextSize);

#endif // \_DEBUG

    status = BCryptDecrypt(hKey, chunk, chunk\_size, NULL, iv, ivSize,

                           \*plaintext, \*plaintextSize, &resultLength,

                           BCRYPT\_BLOCK\_PADDING);

    MASSERT(NT\_SUCCESS(status), "BCryptDecrypt failed");

    // get chaining mode

    wchar\_t chainingMode[64]; // Buffer to hold the chaining mode

    DWORD chainingModeSize = sizeof(chainingMode);

    ULONG result = 0;

    status = BCryptGetProperty(hAlgorithm, BCRYPT\_CHAINING\_MODE,

                               (PBYTE)chainingMode, chainingModeSize,

                               &result, 0);

    MASSERT(NT\_SUCCESS(status), "BCryptGetProperty failed");

        \*plaintextSize = resultLength;

    if (wcscmp(chainingMode, (wchar\_t \*)BCRYPT\_CHAIN\_MODE\_ECB) != 0) {

    }

    // \*plaintextSize = resultLength;

    if (isLastChunk) {

        \*plaintextSize = resultLength;

    }

#ifdef \_DEBUG

    if (isLastChunk) printf("isLastChunk: %d\n", isLastChunk);

    printf("status: %ld\n", status);

    printf("resultLength: %lu\n", resultLength);

    printf("plaintextSize: %lu\n", \*plaintextSize);

    printf("hKey: %p\n", hKey);

    printf("iv: ");

    for (DWORD i = 0; i < ivSize; i++) {

        printf("%02x", iv[i]);

    }

    printf("\n");

#endif // \_DEBUG

    // MASSERT(false, "");

}

/// @brief Encrypt file using AES

/// @param key\_path [in] path to the key file

/// @param file\_path [in] path to the file for encryption

/// @param chaining [in] chaining mode

/// @param iv\_path [in] path to the initialization vector

/// @param output\_path [in] path to the encrypted file

/// @param chainingMode [in] chaining mode

void win\_enc\_file\_aes(char \*key\_path, char \*file\_path,

                      char \*chaining, char \*iv\_path, char \*output\_path,

                      wchar\_t\* chainingMode)

{

    FILE \*keyFile = fopen(key\_path, "rb");

    FILE \*inputFile = fopen(file\_path, "rb");

    FILE \*outputFile = fopen(output\_path, "wb");

    MASSERT(keyFile != NULL, "Failed to open key file");

    MASSERT(inputFile != NULL, "Failed to open input file");

    MASSERT(outputFile != NULL, "Failed to open output file");

    DWORD ivSize = 0;

    BYTE \*iv = NULL;

    if (wcscmp(chainingMode, (wchar\_t \*)BCRYPT\_CHAIN\_MODE\_ECB) != 0) {

        FILE \*ivFile = fopen(iv\_path, "rb");

        MASSERT(ivFile != NULL, "Failed to open IV file");

        fseek(ivFile, 0, SEEK\_END);

        ivSize = ftell(ivFile);

        fseek(ivFile, 0, SEEK\_SET);

        iv = ALLOC(BYTE, ivSize);

        fread(iv, 1, ivSize, ivFile);

        fclose(ivFile);

    }

    fseek(keyFile, 0, SEEK\_END);

    DWORD keySize = ftell(keyFile);

    fseek(keyFile, 0, SEEK\_SET);

    BYTE \*keyBlob = ALLOC(BYTE, keySize);

    fread(keyBlob, 1, keySize, keyFile);

    fclose(keyFile);

    fseek(inputFile, 0, SEEK\_END);

    DWORD inputSize = ftell(inputFile);

    fseek(inputFile, 0, SEEK\_SET);

    DWORD chunk\_size = CHUNK\_SIZE;

    if (inputSize < chunk\_size) {

        chunk\_size = inputSize;

    }

    BYTE \*chunk = ALLOC(BYTE, chunk\_size);

    fseek(inputFile, 0, SEEK\_END);

    size\_t fileSize = ftell(inputFile);

    fseek(inputFile, 0, SEEK\_SET);

    BCRYPT\_ALG\_HANDLE hAlgorithm = NULL;

    BCRYPT\_KEY\_HANDLE hKey = NULL;

    NTSTATUS status;

    DWORD resultLength = 0;

    BYTE \*keyObject = NULL;

    DWORD keyObjectSize = 0;

    // Open an algorithm handle

    status = BCryptOpenAlgorithmProvider(&hAlgorithm, BCRYPT\_AES\_ALGORITHM, NULL, 0);

    MASSERT(NT\_SUCCESS(status), "BCryptOpenAlgorithmProvider failed");

    // Set the chaining mode

    status = BCryptSetProperty(hAlgorithm, BCRYPT\_CHAINING\_MODE, (PBYTE)chainingMode, wcslen(chainingMode) + 1, 0);

    MASSERT(NT\_SUCCESS(status), "BCryptSetProperty failed");

#ifdef \_DEBUG

    // Get current chaining mode

    wchar\_t currentChainingMode[64]; // Buffer to hold the chaining mode

    DWORD chainingModeSize = sizeof(currentChainingMode);

    status = BCryptGetProperty(hAlgorithm, BCRYPT\_CHAINING\_MODE, (PBYTE)currentChainingMode, chainingModeSize, &resultLength, 0);

    MASSERT(NT\_SUCCESS(status), "BCryptGetProperty failed");

    wprintf(L"Current chaining mode: %ls\n", currentChainingMode);

#endif // \_DEBUG

    // Calculate the size of the buffer to hold the KeyObject

    status = BCryptGetProperty(hAlgorithm, BCRYPT\_OBJECT\_LENGTH, (PBYTE)&keyObjectSize, sizeof(DWORD), &resultLength, 0);

    MASSERT(NT\_SUCCESS(status), "BCryptGetProperty failed");

    // Allocate the key object

    keyObject = ALLOC(BYTE, keyObjectSize);

    MASSERT(keyObject != NULL, "HeapAlloc failed");

    // Generate the key from the keyBlob

    status = BCryptGenerateSymmetricKey(hAlgorithm, &hKey, NULL, 0, keyBlob, keySize, 0);

    MASSERT(NT\_SUCCESS(status), "BCryptGenerateSymmetricKey failed");

    size\_t numIterations = (fileSize + chunk\_size - 1) / chunk\_size; // Calculate the number of chunks

    START\_PROGRESS\_BAR(pbar, 32);

    for (size\_t i = 0; i < numIterations; i++) {

        size\_t bytesRead = fread(chunk, 1, chunk\_size, inputFile);

        BYTE \*ciphertext = NULL;

        DWORD ciphertextSize = 0;

        win\_enc\_chunk\_aes(chunk, bytesRead, hAlgorithm, hKey, iv, ivSize, &ciphertext, &ciphertextSize);

        fwrite(ciphertext, 1, ciphertextSize, outputFile);

#ifdef \_DEBUG

        printf("written: %lu  bytes\n", ciphertextSize);

#endif // \_DEBUG

        UPDATE\_PROGRESS\_BAR(pbar, i, numIterations);

        FREE(ciphertext);

    }

    printf("File %s encrypted and saved to %s\n", file\_path, output\_path);

    // Clean up

    if (hKey) BCryptDestroyKey(hKey);

    if (hAlgorithm) BCryptCloseAlgorithmProvider(hAlgorithm, 0);

    fclose(inputFile);

    fclose(outputFile);

    FREE(keyObject);

    FREE(keyBlob);

    FREE(iv);

    FREE(chunk);

}

/// @brief Decrypt file using AES

/// @param key\_path [in] path to the key file

/// @param file\_path [in] path to the file for decryption

/// @param chaining [in] chaining mode

/// @param iv\_path [in] path to the initialization vector

/// @param output\_path [in] path to the decrypted file

/// @param chainingMode [in] chaining mode

void win\_dec\_file\_aes(char \*key\_path, char \*file\_path,

                      char \*chaining, char \*iv\_path, char \*output\_path,

                      wchar\_t\* chainingMode)

{

    FILE \*keyFile = fopen(key\_path, "rb");

    FILE \*inputFile = fopen(file\_path, "rb");

    FILE \*outputFile = fopen(output\_path, "wb");

    MASSERT(keyFile != NULL, "Failed to open key file");

    MASSERT(inputFile != NULL, "Failed to open input file");

    MASSERT(outputFile != NULL, "Failed to open output file");

    DWORD ivSize = 0;

    BYTE \*iv = NULL;

    if (wcscmp(chainingMode, (wchar\_t \*)BCRYPT\_CHAIN\_MODE\_ECB) != 0) {

        FILE \*ivFile = fopen(iv\_path, "rb");

        MASSERT(ivFile != NULL, "Failed to open IV file");

        fseek(ivFile, 0, SEEK\_END);

        ivSize = ftell(ivFile);

        fseek(ivFile, 0, SEEK\_SET);

        iv = ALLOC(BYTE, ivSize);

        fread(iv, 1, ivSize, ivFile);

        fclose(ivFile);

    }

    fseek(keyFile, 0, SEEK\_END);

    DWORD keySize = ftell(keyFile);

    fseek(keyFile, 0, SEEK\_SET);

    BYTE \*keyBlob = ALLOC(BYTE, keySize);

    fread(keyBlob, 1, keySize, keyFile);

    fclose(keyFile);

    fseek(inputFile, 0, SEEK\_END);

    DWORD inputSize = ftell(inputFile);

    fseek(inputFile, 0, SEEK\_SET);

    DWORD chunk\_size = CHUNK\_SIZE;

    if (inputSize < chunk\_size) {

        chunk\_size = inputSize;

    }

    // chunk\_size must be multiple of AES block size plus

    // additional bytes (d + 16) & ~15

    // if (wcscmp(chainingMode, BCRYPT\_CHAIN\_MODE\_CBC) == 0 ||

        // wcscmp(chainingMode, BCRYPT\_CHAIN\_MODE\_ECB) == 0) {

    // }

    chunk\_size = (chunk\_size + AES\_KEY\_BYTE\_SIZE) & ~15;

    BYTE \*chunk = ALLOC(BYTE, chunk\_size);

    BCRYPT\_ALG\_HANDLE hAlgorithm = NULL;

    BCRYPT\_KEY\_HANDLE hKey = NULL;

    NTSTATUS status;

    DWORD resultLength = 0;

    BYTE \*keyObject = NULL;

    DWORD keyObjectSize = 0;

    // Open an algorithm handle

    status = BCryptOpenAlgorithmProvider(&hAlgorithm, BCRYPT\_AES\_ALGORITHM,

                                         NULL, 0);

    MASSERT(NT\_SUCCESS(status), "BCryptOpenAlgorithmProvider failed");

    // Set the chaining mode

    status = BCryptSetProperty(hAlgorithm, BCRYPT\_CHAINING\_MODE,

                               (PBYTE)chainingMode, wcslen(chainingMode), 0);

    MASSERT(NT\_SUCCESS(status), "BCryptSetProperty failed");

    // Calculate the size of the buffer to hold the KeyObject

    status = BCryptGetProperty(hAlgorithm, BCRYPT\_OBJECT\_LENGTH,

                               (PBYTE)&keyObjectSize, sizeof(DWORD),

                               &resultLength, 0);

    MASSERT(NT\_SUCCESS(status), "BCryptGetProperty failed");

    // Allocate the key object

    keyObject = ALLOC(BYTE, keyObjectSize);

    MASSERT(keyObject != NULL, "HeapAlloc failed");

    // Generate the key from the keyBlob

    status = BCryptGenerateSymmetricKey(hAlgorithm, &hKey, keyObject,

                                        keyObjectSize, keyBlob, keySize, 0);

    MASSERT(NT\_SUCCESS(status), "BCryptGenerateSymmetricKey failed");

    fseek(inputFile, 0, SEEK\_END);

    size\_t fileSize = ftell(inputFile);

    fseek(inputFile, 0, SEEK\_SET);

    // Calculate the number of chunks

    size\_t numIterations = (fileSize + chunk\_size - 1) / chunk\_size;

    START\_PROGRESS\_BAR(pbar, 32);

    for (size\_t i = 0; i < numIterations; i++) {

        size\_t bytesRead = fread(chunk, 1, chunk\_size, inputFile);

        BYTE \*plaintext = NULL;

        DWORD plaintextSize = 0;

        win\_dec\_chunk\_aes(chunk, bytesRead, hAlgorithm, hKey, iv, ivSize,

                          &plaintext, &plaintextSize, i == numIterations - 1);

#ifdef \_DEBUG

        printf("written: %lu  bytes\n", plaintextSize);

#endif // \_DEBUG

        fwrite(plaintext, 1, plaintextSize, outputFile);

        UPDATE\_PROGRESS\_BAR(pbar, i, numIterations);

        FREE(plaintext);

    }

    printf("File %s decrypted and saved to %s\n", file\_path, output\_path);

    // Clean up

    if (hKey) BCryptDestroyKey(hKey);

    if (hAlgorithm) BCryptCloseAlgorithmProvider(hAlgorithm, 0);

    fclose(inputFile);

    fclose(outputFile);

    FREE(keyObject);

    FREE(keyBlob);

    FREE(iv);

    FREE(chunk);

}

#endif // \_WIN32

int main(int argc, const char \*argv[]) {

    char \*key\_path = NULL;

    char \*file\_path = NULL;

    char \*chaining = NULL;

    char \*iv\_path = NULL;

    char \*output\_path = NULL;

    char \*mode = NULL;

    static const char \*const usages[] = {

        "cifs [options] [[--] args]",

        "cifs [options]",

        "cifs",

        NULL,

    };

    struct argparse\_option options[] = {

        OPT\_HELP(),

        OPT\_STRING('m', "mode", &mode,                          \

                   "mode of operation, "                        \

                   "can be 'keygen', 'ivgen', 'enc' or 'dec'",  \

                   NULL, 0, 0),                                 \

        // для генерации ключа:

        // [out] путь к файлу ключа

        OPT\_GROUP(C\_HEADER "Key generation" C\_RESET " "),

        OPT\_STRING('g', "gen-key", &key\_path,

                   "path to key file", NULL, 0, 0),

        // для генерации инициализирующего вектора:

        // [out] путь к файлу инициализирующего вектора

        OPT\_GROUP(C\_HEADER "IV generation" C\_RESET " "),

        OPT\_STRING('g', "gen-key", NULL,

                   "path to initialization vector", NULL, 0, 0),

        // для шифрования:

        // [in] путь к файлу ключа

        // [in] путь к файлу для шифрования

        // [in] режим сцепления блоков

        // [out] путь к инициализирующему вектору

        // [out] путь к зашифрованному файлу

        OPT\_GROUP(C\_HEADER "Encryption" C\_RESET " "),

        OPT\_STRING('k', "key", &key\_path,

                   "path to key file", NULL, 0, 0),

        OPT\_STRING('i', "iv", &iv\_path,

                   "path to initialization vector", NULL, 0, 0),

        OPT\_STRING('f', "file", &file\_path,

                   "path to file for encryption", NULL, 0, 0),

        OPT\_STRING('c', "chaining", &chaining,

                   "block chaining mode, can be CBC, CFB, ECB", NULL, 0, 0),

        OPT\_STRING('o', "output", &output\_path,

                   "path to output file", NULL, 0, 0),

        // для дешифрования:

        // [in] путь к файлу ключа

        // [in] путь к инициализирующему вектору

        // [in] путь к зашифрованному файлу

        // [in] режим сцепления блоков

        // [out] путь к расшифрованному файлу

        OPT\_GROUP(C\_HEADER "Decryption" C\_RESET " "),

        OPT\_STRING('k', "key", NULL,

                   "path to key file", NULL, 0, 0),

        OPT\_STRING('i', "iv", NULL,

                   "path to initialization vector", NULL, 0, 0),

        OPT\_STRING('f', "file", NULL,

                   "path to encrypted file", NULL, 0, 0),

        OPT\_STRING('c', "chaining", NULL,

                   "block chaining mode, can be CBC, CFB, ECB", NULL, 0, 0),

        OPT\_STRING('o', "output", NULL,

                   "path to output file", NULL, 0, 0),

        OPT\_END(),

    };

    struct argparse argparse;

    argparse\_init(&argparse, options, usages, 0);

    const char \*argparse\_top\_msg = "\n" C\_HEADER "TMP opening msg" C\_RESET " ";

    const char \*argparse\_bottom\_msg = "\n" IME\_ESC IME\_BRIGHT\_BLACK IME\_ESC\_END\

        "For more info go to Git repo: "                                       \

        "https://github.com/Kseen715/xxx" C\_RESET " ";

    argparse\_describe(&argparse, argparse\_top\_msg, argparse\_bottom\_msg);

    if (argc == 0)

    {

        argparse\_usage(&argparse);

        exit(0);

    }

    argc = argparse\_parse(&argparse, argc, argv);

    wchar\_t \*chainingMode = NULL;

    if (mode[0] == 'i'){

        iv\_path = key\_path;

    }

    switch (mode[0])

    {

    case 'k':

        printf("%s", C\_CYAN "Key generation" C\_RESET "\n");

        if (key\_path == NULL)

        {

            printf("%s",

                   C\_ERROR "ERROR: Key path is not specified" C\_RESET "\n");

            argparse\_usage(&argparse);

            exit(1);

        }

#ifdef \_LINUX

        printf("%d\n", kpp\_keygen("ecb(aes)"));

#endif // \_LINUX

#ifdef \_WIN32

        printf(C\_HEADER "Using:" C\_RESET "\n");

        printf("  \\- Key path:\t\t\"%s\"\n", key\_path);

        printf("\n");

        win\_generate\_aes\_key(key\_path);

#ifdef \_DEBUG

        BYTE \*keyBlob = NULL;

        DWORD keySize = 0;

        win\_read\_aes\_key(key\_path, &keyBlob, &keySize);

        printf("keyBlob: %p\n", keyBlob);

        printf("keySize: %ld\n", keySize);

        printf("Key: ");

        print\_byte\_array\_hex(keyBlob, keySize);

        FREE(keyBlob);

        if (keyBlob) HeapFree(GetProcessHeap(), 0, keyBlob);

#endif // \_DEBUG

#endif // \_WIN32

        break;

    case 'i':

        printf("%s", C\_CYAN "IV generation" C\_RESET "\n");

        if (iv\_path == NULL)

        {

            printf("%s",

                   C\_ERROR "ERROR: IV path is not specified" \

                   C\_RESET "\n");

            argparse\_usage(&argparse);

            exit(1);

        }

#ifdef \_LINUX

#endif // \_LINUX

#ifdef \_WIN32

        printf(C\_HEADER "Using:" C\_RESET "\n");

        printf("  \\- IV path:\t\t\"%s\"\n", iv\_path);

        printf("\n");

        win\_generate\_aes\_iv(iv\_path);

#ifdef \_DEBUG

        BYTE \*ivBlob = NULL;

        DWORD ivSize = 0;

        win\_read\_aes\_iv(iv\_path, &ivBlob, &ivSize);

        printf("IV: ");

        print\_byte\_array\_hex(ivBlob, ivSize);

        FREE(ivBlob);

        if (ivBlob) HeapFree(GetProcessHeap(), 0, ivBlob);

#endif // \_DEBUG

#endif // \_WIN32

        break;

    case 'e':

        printf("%s", C\_CYAN "Encryption" C\_RESET "\n");

        if (key\_path == NULL)

        {

            printf("%s",

                   C\_ERROR "ERROR: Key path is not specified" \

                   C\_RESET "\n");

            argparse\_usage(&argparse);

            exit(1);

        }

        if (file\_path == NULL)

        {

            printf("%s",

                   C\_ERROR "ERROR: File path is not specified" \

                   C\_RESET "\n");

            argparse\_usage(&argparse);

            exit(1);

        }

        if (chaining == NULL)

        {

            printf("%s",

                   C\_ERROR "ERROR: Chaining mode is not specified" \

                   C\_RESET "\n");

            argparse\_usage(&argparse);

            exit(1);

        }

        if (iv\_path == NULL)

        {

            printf("%s",

                   C\_ERROR "ERROR: IV path is not specified" \

                   C\_RESET "\n");

            argparse\_usage(&argparse);

            exit(1);

        }

        if (output\_path == NULL)

        {

            printf("%s",

                   C\_ERROR "ERROR: Output path is not specified" \

                   C\_RESET "\n");

            argparse\_usage(&argparse);

            exit(1);

        }

#ifdef \_LINUX

#endif // \_LINUX

#ifdef \_WIN32

        str\_lower(chaining);

        if (is\_in\_str(chaining, "cbc"))

        {

            chainingMode = BCRYPT\_CHAIN\_MODE\_CBC;

        }

        else if (is\_in\_str(chaining, "cfb"))

        {

            chainingMode = BCRYPT\_CHAIN\_MODE\_CFB;

        }

        else if (is\_in\_str(chaining, "ecb"))

        {

            chainingMode = BCRYPT\_CHAIN\_MODE\_ECB;

        }

        else

        {

            printf("%s", C\_ERROR "ERROR: Unknown chaining mode" C\_RESET "\n");

            argparse\_usage(&argparse);

            exit(1);

        }

        printf(C\_HEADER "Using:" C\_RESET "\n");

        printf("  |- Key path:\t\t\"%s\"\n", key\_path);

        printf("  |- IV path:\t\t\"%s\"\n", iv\_path);

        printf("  |- Chaining mode:\t%ls\n", chainingMode);

        printf("  |- Input file:\t\"%s\"\n", file\_path);

        printf("  \\- Output file:\t\"%s\"\n", output\_path);

        printf("\n");

        win\_enc\_file\_aes(key\_path, file\_path, chaining, iv\_path,

            output\_path, chainingMode);

#endif // \_WIN32

        break;

    case 'd':

        printf("%s", C\_CYAN "Decryption" C\_RESET "\n");

        if (key\_path == NULL)

        {

            printf("%s",

                   C\_ERROR "ERROR: Key path is not specified" \

                   C\_RESET "\n");

            argparse\_usage(&argparse);

            exit(1);

        }

        if (iv\_path == NULL)

        {

            printf("%s",

                   C\_ERROR "ERROR: IV path is not specified" \

                   C\_RESET "\n");

            argparse\_usage(&argparse);

            exit(1);

        }

        if (file\_path == NULL)

        {

            printf("%s",

                   C\_ERROR "ERROR: File path is not specified" \

                   C\_RESET "\n");

            argparse\_usage(&argparse);

            exit(1);

        }

        if (chaining == NULL)

        {

            printf("%s",

                   C\_ERROR "ERROR: Chaining mode is not specified" \

                   C\_RESET "\n");

            argparse\_usage(&argparse);

            exit(1);

        }

        if (output\_path == NULL)

        {

            printf("%s",

                   C\_ERROR "ERROR: Output path is not specified" \

                   C\_RESET "\n");

            argparse\_usage(&argparse);

            exit(1);

        }

#ifdef \_LINUX

#endif // \_LINUX

#ifdef \_WIN32

        str\_lower(chaining);

        if (is\_in\_str(chaining, "cbc"))

        {

            chainingMode = BCRYPT\_CHAIN\_MODE\_CBC;

        }

        else if (is\_in\_str(chaining, "cfb"))

        {

            chainingMode = BCRYPT\_CHAIN\_MODE\_CFB;

        }

        else if (is\_in\_str(chaining, "ecb"))

        {

            chainingMode = BCRYPT\_CHAIN\_MODE\_ECB;

        }

        else

        {

            printf("%s", C\_ERROR "ERROR: Unknown chaining mode" C\_RESET "\n");

            argparse\_usage(&argparse);

            exit(1);

        }

        printf(C\_HEADER "Using:" C\_RESET "\n");

        printf("  |- Key path:\t\t\"%s\"\n", key\_path);

        printf("  |- IV path:\t\t\"%s\"\n", iv\_path);

        printf("  |- Chaining mode:\t%ls\n", chainingMode);

        printf("  |- Input file:\t\"%s\"\n", file\_path);

        printf("  \\- Output file:\t\"%s\"\n", output\_path);

        printf("\n");

        win\_dec\_file\_aes(key\_path, file\_path, chaining, iv\_path,

            output\_path, chainingMode);

#endif // \_WIN32

        break;

    default:

        printf("%s", C\_ERROR "ERROR: Unknown mode" C\_RESET "\n");

        argparse\_usage(&argparse);

        break;

    }

    return 0;

}