<Final project>

Bea Jae Kyeong, Esmanur Karaca

Index

The main purpose

The main purpose of our project is to apply of SEI CERT Coding Standards to check if our program is well-coded. This is to make our program's quality higher. If the code of the program doesn't meet the standard of SEI CERT C++ Coding Standard, we modified the code.

We used Cppcheck tool designed for C/C++ for this practice. Its main goal is to detect the kinds of errors that a compiler usually can't detect.

Regulations

We chose 10 regulations from the standards, which are :

1. FIO51-CPP. Close files when they are no longer needed
2. FIO38-C. Do not copy a FILE object
3. FIO45-C. Avoid TOCTOU race conditions while accessing files
4. FIO46-C. Do not access a closed file
5. DCL51-CPP. Do not declare or define a reserved identifier
6. FLP30-C. Do not use floating-point variables as loop counters
7. STR34-C. Cast characters to unsigned char before converting to larger integer sizes
8. ARR30-C. Do not form or use out-of-bounds pointers or array subscripts

9. DCL40-C. Do not create incompatible declarations of the same function or object
10. ERR33-C. Detect and handle standard library errors

FIO is from the Rule 07. Input Output (FIO), DCL from Declarations and Initialization (DCL) Rule 01, MSC from Rule 49. Miscellaneous (MSC), STR from Rule 05. Characters and Strings (STR), CTR from Rule 04. Containers (CTR), DCL from Rule 01. Declarations and Initialization (DCL), ARR from Rule 06. Arrays (ARR) and ERR from Rule 08. Exceptions and Error Handling (ERR). The link shows by clicking regulations above.

Implementations of the code

1. FIO51-CPP. Close files when they are no longer needed

We should use std::fstream::close() to close the file when it is no needed. When it is not closed before std::terminate() is called it can make many risks. It may allow an attacker to exhaust system resources and can increase the risk that data written into in-memory file buffers will not be flushed due to abnormal termination.

```
    fp1 = fopen("plain.bin", "rb");
    if (!fp) {
        perror("key.txt");
        exit(1);
    }
    if (!fp1) {
        perror("plain.txt");
        exit(1);
    }
    /// Printing plain text array
    while (fread((void*)buf, 1, sizeof(buf), fp1)) {
        cout << "PLAIN : ";
        for (i = 0; i < 16; i++) {
            plain[i] = buf[i];
            cout << plain[i] << " ";
        }
        cout << endl;
    }
    /// Printing key array
    while (fread((void*)buf, 1, sizeof(buf), fp)) {
        cout << "KEY : ";
        for (i = 0; i < 16; i++) {
            key[i] = buf[i];
            cout << hex << key[i] << " ";
        }
        cout << endl;
    }
    fclose(fp);
    fclose(fp1);
```

We can see that the program is already closed by calling fclose() after the file is opened. So the program meets this standard.

2. FIO38-C. Do not copy a FILE object

We should use a copy of the pointer instead of using a copy of the FILE object. Using a copy of a FILE object in place of the original may result in a crash, which can be used in a denial-of-service attack.

```
int main(void)
{
    int i = 0;
    int key[32];        /// Array storing key
    int plain[20];      /// Array storing plain text
    int decrypted[20];  /// Array storing decrypted text
    unsigned char buf[16];
    unsigned char buf1[16];
    /// Declaring file pointer
    FILE *fp;
    FILE *fp1;
    FILE *fp2;
    FILE *fp3;

    cout << hex;

    cout << "RC : ";
    for (int i = 0; i < 10; i++)
        cout << RCon(i) << " ";
    cout << endl;

    /// Reading binary file
```

We can see that the original FILE object is not copied when it is used.

3. FIO45-C. Avoid TOCTOU race conditions while accessing files

A TOCTOU (time-of-check, time-of-use) race condition is possible when two or more concurrent processes are operating on a shared file system. TOCTOU conditions can be exploited when a program performs two or more file operations on the same file name or path name. TOCTOU race conditions can result in privilege escalation.

The solution is to use the x mode of fopen(). This mode causes fopen() to fail if the file exists. The x mode provides exclusive access to the file only if the host environment provides this support.

```
/// Writing cipher text and decrypted text in binary file
fp2 = fopen("cipher.bin", "wb");
fp3 = fopen("decrypted.bin", "wb");
if (!fp2) {
    perror("cipher.bin");
    exit(1);
}
if (!fp3) {
    perror("decrypted.bin");
    exit(1);
}
/// Writing cipher file
for (i = 0; i < 16; i++) {
    buf1[i] = cipher[i];
}
fwrite(&buf1, 1, sizeof(buf1), fp2);
```

Our program doesn't matched the solution, so we modified "wb" to "wx" in fopen().

```
/// Writing cipher text and decrypted text in binary file
fp2 = fopen("cipher.bin", "wx");
fp3 = fopen("decrypted.bin", "wx");
if (!fp2) {
    perror("cipher.bin");
    exit(1);
}
if (!fp3) {
    perror("decrypted.bin");
    exit(1);
}
/// Writing cipher file
for (i = 0; i < 16; i++) {
    buf1[i] = cipher[i];
}
fwrite(&buf1, 1, sizeof(buf1), fp2);
```

4. FIO46-C. Do not access a closed file

We shouldn't use stdout after the file is already closed. Using the value of a pointer to a FILE object after the associated file is closed is undefined behavior.

```
    }
    fwrite(&buf1, 1, sizeof(buf1), fp3);

    fclose(fp2);
    fclose(fp3);

    fputs("stdout successfully closed.", stderr);

    system("pause");
    return 0;
}
```

We applied the code, instead of using printf() after the file is closed, we used fputs() to meet the standard.

5. DCL51-CPP. Do not declare or define a reserved identifier

We shouldn't declare or define a reserved identifier. No other identifiers are reserved. Declaring or defining an identifier in a context in which it is reserved results in undefined behavior.

In a user-defined literal, literal suffix identifiers are required to start with an underscore. Literal suffixes without the underscore prefix are reserved for future library implementations.

```
 */
int main(void)
{
    int i = 0;
    int key[32];        /// Array storing key
    int plain[20];      /// Array storing plain text
    int decrypted[20];  /// Array storing decrypted text
    unsigned char buf[16];
    unsigned char buf1[16];
    int operator"" one;
    /// Declaring file pointer
    FILE *fp;
    FILE *fp1;
    FILE *fp2;
    FILE *fp3;

    cout << hex;

    cout << "RC : ":
```

We didn't had the code in our program, so we wrote it. A user-defined literal operator"" one is declared. Literal suffix identifiers are required to start with an underscore for future library implementations.

```
int i = 0;
int key[32];        /// Array storing key
int plain[20];      /// Array storing plain text
int decrypted[20];  /// Array storing decrypted text
unsigned char buf[16];
unsigned char buf1[16];
int operator"" _one;
/// Declaring file pointer
FILE *fp;
FILE *fp1;
FILE *fp2;
```

In order to meet the standard, user-defined literal is named operator"" _one, which is not a reserved identifier.

6. FLP30-C. Do not use floating-point variables as loop counters

Floating-point numbers are subject to representational limitations just as integers are, and binary floating-point numbers cannot represent all real numbers exactly, even if they can be represented in a small number of decimal digits.

```
int InvSbox(int n) {
    int i, j, k, m;

    for (i = 1; i < 256; i++) {
        m = i;
        for (j = 7; j >= 0; j--) {
            b[j] = m / (1 << j);        /// Making 1 to 255 into binary and store b[0] to b[7]
            m %= (1 << j);
        }
        for (k = 0; k < 8; k++) {
            b1[k] = b[(k + 2) % 8] ^ b[(k + 5) % 8] ^ b[(k + 7) % 8] ^ c1[k];   /// XOR after
            m = m + b1[k] * (1 << k);       /// Restore from binary
        }
        invsbox[i] = InvGF[(255 - GF[m])];  /// Inverse
    }
    sbox[0] = 0xf3;                         /// 0 doesn't have inverse
    return invsbox[n];
}
```

In our code, we are not using the float as a counter of the loop, instead we are using integer to prevent the value having float points. So when we shift j times from "1", we want the value to be 0, not having a float point. By doing this, the float is not needed as a counter.

7. ARR30-C. Do not form or use out-of-bounds pointers or array subscripts

Writing to out-of-range pointers or array subscripts can result in a buffer overflow and the execution of arbitrary code with the permissions of the vulnerable process. Reading from out-of-range array subscripts can result in unintended information disclosure.

```
int Sbox(int num) {
    int m, w;
    int a = 1;
    for (int e = 0; e < 256; e++) { /// Fun
        InvGF[e] = a;
        w = a & 0x80;
        a <<= 1;
        if (w == 0x80)
            a ^= 0x69;
        a ^= InvGF[e];
        GF[InvGF[e]] = e;
    }
    InvGF[255] = 0;

    for (int i = 1; i < 256; i++) {
```

In our code, Sbox, InvSbox, and Rcon has index which is an integer. We can

meet this standard by using an unsigned type to avoid having to check for negative values while still rejecting out-of-bounds positive values of index.

```
int Sbox(size_t num) {
    int m, w;
    int a = 1;
    for (int e = 0; e < 256; e++) { /// Function G
        InvGF[e] = a;
        w = a & 0x80;
        a <<= 1;
        if (w == 0x80)
            a ^= 0x69;
        a ^= InvGF[e];
        GF[InvGF[e]] = e;
    }
    InvGF[255] = 0;

    for (int i = 1; i < 256; i++) {
        m = InvGF[(255 - GF[i])];        /// Invers
        for (int j = 7; j >= 0; j--) {
            b[j] = m / (1 << j);         /// Genera
            m %= (1 << j);
        }
```

We can use size_t instead of integer for preventing errors.

8. STR34-C. Cast characters to unsigned char before converting to larger integer sizes

Signed character data must be converted to unsigned char before being assigned or converted to a larger signed type. Conversion of character data resulting in a value in excess of UCHAR_MAX is an often-missed error that can result in a disturbingly broad range of potentially severe vulnerabilities.

```
void SubBytes()
{
    cout << "SB: ";
    for (int i = 0; i < 4; i++)
    {
        for (int j = 0; j < 4; j++)
        {
            state[j][i] = Sbox(state[j][i]);
            cout << state[j][i] << " ";
        }
    }
    cout << endl;
}
```

In our code, state array is an integer value, and Sbox is an unsigned char array. We defined variables as an unsigned char before assigning it to an integer, so we meet the standard.

9. DCL40-C. Do not create incompatible declarations of the same function or object

Two or more incompatible declarations of the same function or object must not appear in the same program because they result in undefined behavior. We should use same declaration of the variable which has the same name.

```
 * @brief Substitute each Byte value of state array by Sbox
 */
void SubBytes()
{
    cout << "SB: ";
    for (int i = 0; i < 4; i++)
    {
        for (int j = 0; j < 4; j++)
        {
            state[j][i] = Sbox(state[j][i]);        /// assigned Sbox as an unsigned char before converted into state a
            cout << state[j][i] << " ";
        }
    }
    cout << endl;
}
/**
```

In our code, we are using same declaration of same named value, for example int for every i, j, and size_t for every num variables. So we are meeting the standard.

10. ERR33-C. Detect and handle standard library errors

We should check for every returned value. Failing to detect error conditions can lead to unpredictable results, including abnormal program termination and denial-of-service attacks or could allow an attacker to run arbitrary code.

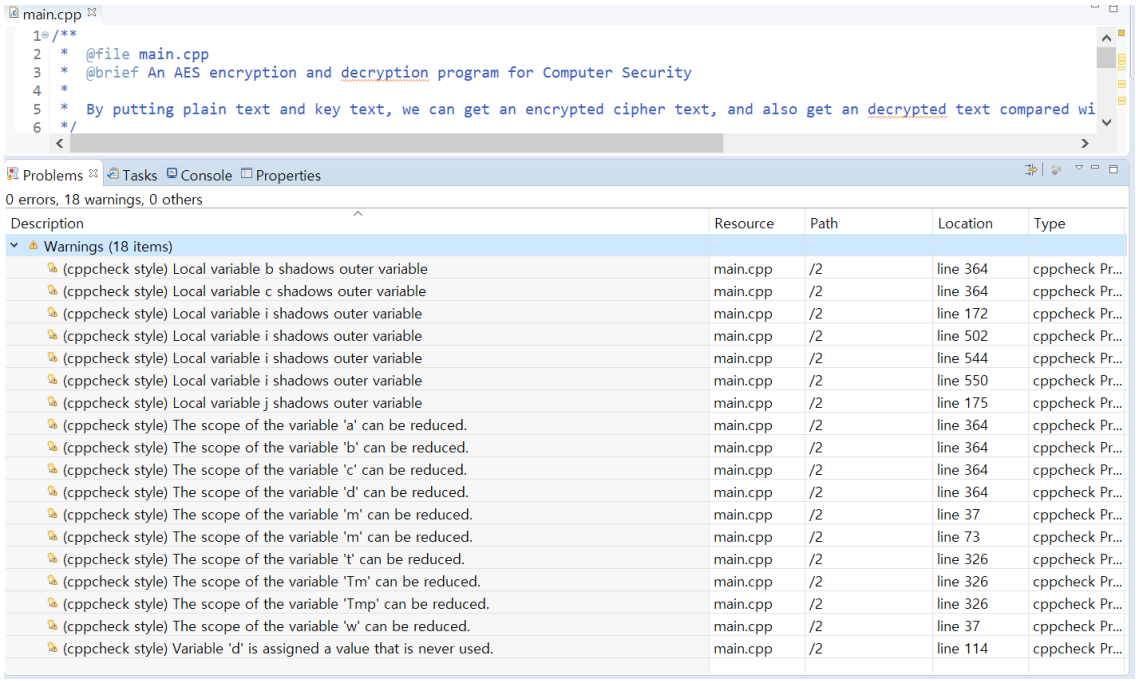
```
/// Reading binary file
fp = fopen("key.bin", "rb");
fp1 = fopen("plain.bin", "rb");
if (!fp) {
    perror("key.txt");
    exit(1);
}
if (!fp1) {
    perror("plain.txt");
    exit(1);
}
/// Printing plain text array
while (fread((void*)buf, 1, sizeof(buf), fp1)) {
    if(!fread((void*)buf, 1, sizeof(buf), fp1)){
        break;      /// Indicating error
    }
```

According to the C Standard, the fopen() function returns a NULL value to indicate that an error occurred. This compliant solution tests for this condition before reading from a file to eliminate the chance of operating on the wrong portion of the file if fopen() fails.

```
/// Reading binary file
fp = fopen("key.bin", "rb");
fp1 = fopen("plain.bin", "rb");
if (fp==NULL) {     /// fopen returns NULL when error occurs
    perror("key.txt");
    exit(1);
}
if (fp1==NULL) {    /// fopen returns NULL when error occurs
    perror("plain.txt");
    exit(1);
}
/// Printing plain text array
while (fread((void*)buf, 1, sizeof(buf), fp1)) {
```

The code is modified because fopen could return NULL, not FALSE. By changing the code, we can handle the error exactly.

Application of corrections from errors by cppcheck



main.cpp

```
/**
 *  @file main.cpp
 *  @brief An AES encryption and decryption program for Computer Security
 *
 *  By putting plain text and key text, we can get an encrypted cipher text, and also get an decrypted text compared wi
 */
```

Problems | Tasks | Console | Properties

0 errors, 18 warnings, 0 others

Description	Resource	Path	Location	Type
Warnings (18 items)				
(cppcheck style) Local variable b shadows outer variable	main.cpp	/2	line 364	cppcheck Pr...
(cppcheck style) Local variable c shadows outer variable	main.cpp	/2	line 364	cppcheck Pr...
(cppcheck style) Local variable i shadows outer variable	main.cpp	/2	line 172	cppcheck Pr...
(cppcheck style) Local variable i shadows outer variable	main.cpp	/2	line 502	cppcheck Pr...
(cppcheck style) Local variable i shadows outer variable	main.cpp	/2	line 544	cppcheck Pr...
(cppcheck style) Local variable i shadows outer variable	main.cpp	/2	line 550	cppcheck Pr...
(cppcheck style) Local variable j shadows outer variable	main.cpp	/2	line 175	cppcheck Pr...
(cppcheck style) The scope of the variable 'a' can be reduced.	main.cpp	/2	line 364	cppcheck Pr...
(cppcheck style) The scope of the variable 'b' can be reduced.	main.cpp	/2	line 364	cppcheck Pr...
(cppcheck style) The scope of the variable 'c' can be reduced.	main.cpp	/2	line 364	cppcheck Pr...
(cppcheck style) The scope of the variable 'd' can be reduced.	main.cpp	/2	line 364	cppcheck Pr...
(cppcheck style) The scope of the variable 'm' can be reduced.	main.cpp	/2	line 37	cppcheck Pr...
(cppcheck style) The scope of the variable 'm' can be reduced.	main.cpp	/2	line 73	cppcheck Pr...
(cppcheck style) The scope of the variable 't' can be reduced.	main.cpp	/2	line 326	cppcheck Pr...
(cppcheck style) The scope of the variable 'Tm' can be reduced.	main.cpp	/2	line 326	cppcheck Pr...
(cppcheck style) The scope of the variable 'Tmp' can be reduced.	main.cpp	/2	line 326	cppcheck Pr...
(cppcheck style) The scope of the variable 'w' can be reduced.	main.cpp	/2	line 37	cppcheck Pr...
(cppcheck style) Variable 'd' is assigned a value that is never used.	main.cpp	/2	line 114	cppcheck Pr...

We have 18 warnings detected by cppcheck.

1. Local variable b, c shadows outer variable

```
361 void InvMixColumns()
362 {
363     unsigned char a, b, c, d;
364     for (int i = 0; i < 4; i++)
365     {
366         a = state[0][i];
367         b = state[1][i];
368         c = state[2][i];
369         d = state[3][i];
370
371         /// Operating multiply with fixed polynomial expression by Multiply and XOR operation
372         state[0][i] = Multiply(a, 0x0e) ^ Multiply(b, 0x0b) ^ Multiply(c, 0x0d) ^ Multiply(d, 0x09);
373         state[1][i] = Multiply(a, 0x09) ^ Multiply(b, 0x0e) ^ Multiply(c, 0x0b) ^ Multiply(d, 0x0d);
374         state[2][i] = Multiply(a, 0x0d) ^ Multiply(b, 0x09) ^ Multiply(c, 0x0e) ^ Multiply(d, 0x0b);
```

Unsigned char b, c shadows outer variable. So we changed the code by putting definitions inside 'for' loop. And we changed b and c by b_ and c_

due to arrays defined ahead by b and c. When the variables aren't used outside the loop, it's better to define it inside the loop.

```
361 void InvMixColumns()
362 {
363     for (int i = 0; i < 4; i++)
364     {
365         unsigned char a, b_, c_, d;     /// By defining a,b,c,d here, we can stop them from shadowing outer variables
366         a = state[0][i];
367         b_ = state[1][i];
368         c_ = state[2][i];
369         d = state[3][i];
370 
371         /// Operating multiply with fixed polynomial expression by Multiply and XOR operation
372         state[0][i] = Multiply(a, 0x0e) ^ Multiply(b_, 0x0b) ^ Multiply(c_, 0x0d) ^ Multiply(d, 0x09);
373         state[1][i] = Multiply(a, 0x09) ^ Multiply(b_, 0x0e) ^ Multiply(c_, 0x0b) ^ Multiply(d, 0x0d);
374         state[2][i] = Multiply(a, 0x0d) ^ Multiply(b_, 0x09) ^ Multiply(c_, 0x0e) ^ Multiply(d, 0x0b);
375         state[3][i] = Multiply(a, 0x0b) ^ Multiply(b_, 0x0d) ^ Multiply(c_, 0x09) ^ Multiply(d, 0x0e);
376     }
```

2. Local variable i, j shadows outer variable

```
172     for (int i = 1; i <= Nr; i++)
173     {
174         cout << dec << "ROUND " << i << ": ";
175         for (int j = 0; j < 16; j++)
176         {
177             cout << hex << RoundKey[i * 16 + j] << " ";
178         }
179         cout << endl;
180     }
181 }
182 
183 /**
184  *
```

Problems | Tasks | Console | Properties

0 errors, 18 warnings, 0 others

Description	Resource	Path	Location	Type
Warnings (18 items)				
(cppcheck style) Local variable b shadows outer variable	main.cpp	/2	line 364	cppcheck P
(cppcheck style) Local variable c shadows outer variable	main.cpp	/2	line 364	cppcheck P
(cppcheck style) Local variable i shadows outer variable	main.cpp	/2	line 172	cppcheck P
(cppcheck style) Local variable i shadows outer variable	main.cpp	/2	line 502	cppcheck P

Local variable i and j are already defined before, so we don't have to define them as integer again.

```
    for (i = 1; i <= Nr; i++)          /// Local variable i shadowed outer variable, so we changed it by erasing int
    {
        cout << dec << "ROUND " << i << ": ";
        for (j = 0; j < 16; j++)
        {
            cout << hex << RoundKey[i * 16 + j] << " ";
        }
        cout << endl;
    }
```

We solved the warning by removing int ahead of i and j.

3. The scope of the variable Tmp, Tm, t can be reduced

```
325 void MixColumns()
326 {
327     unsigned char Tmp, Tm, t;
328
329     for (int i = 0; i < 4; i++)
330     {
331         t = state[0][i];
332
333         Tmp = state[0][i] ^ state[1][i] ^ state[2][i] ^ state[3][i];
334         /// execute matrix multiply operation and store in Tm
335         Tm = state[0][i] ^ state[1][i];
336         Tm = xtime(Tm);
337         state[0][i] ^= Tm ^ Tmp;
338         Tm = state[1][i] ^ state[2][i];
339         Tm = xtime(Tm);
340         state[1][i] ^= Tm ^ Tmp;
```

Problems | Tasks | Console | Properties

0 errors, 5 warnings, 0 others

Description	Resource	Path	L
Warnings (5 items)			
(cppcheck style) Local variable a shadows outer variable	main.cpp	/2	li
(cppcheck style) Local variable d shadows outer variable	main.cpp	/2	li
(cppcheck style) The scope of the variable 't' can be reduced.	main.cpp	/2	li
(cppcheck style) The scope of the variable 'Tm' can be reduced.	main.cpp	/2	li

We can solve it by putting the definitions of variable Tmp, Tm and t inside the 'for' loop.

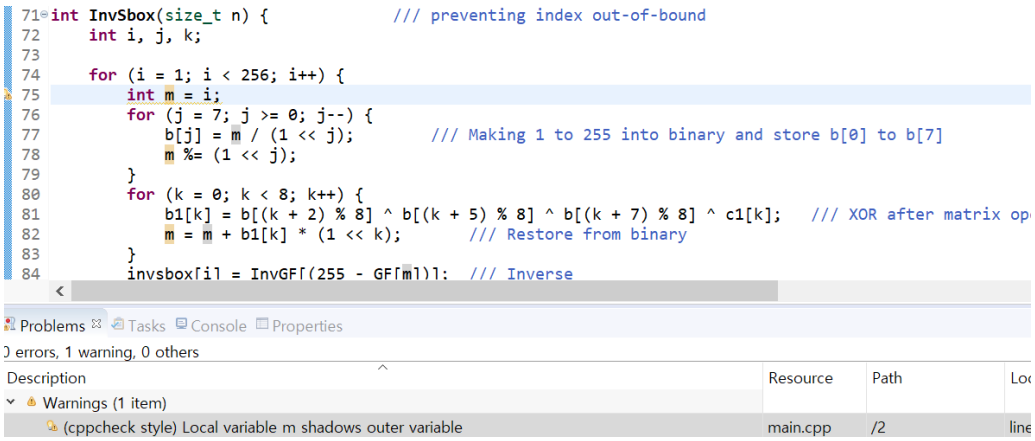
```
325 void MixColumns()
326 {
327
328     for (int i = 0; i < 4; i++)
329     {
330         unsigned char Tmp, Tm, t;           /// putting inside the loop solves warnings
331         t = state[0][i];
332
333         Tmp = state[0][i] ^ state[1][i] ^ state[2][i] ^ state[3][i];
334         /// execute matrix multiply operation and store in Tm
335         Tm = state[0][i] ^ state[1][i];
336         Tm = xtime(Tm);
337         state[0][i] ^= Tm ^ Tmp;
338         Tm = state[1][i] ^ state[2][i];
339         Tm = xtime(Tm);
340         state[1][i] ^= Tm ^ Tmp;
```

Problems | Tasks | Console | Properties

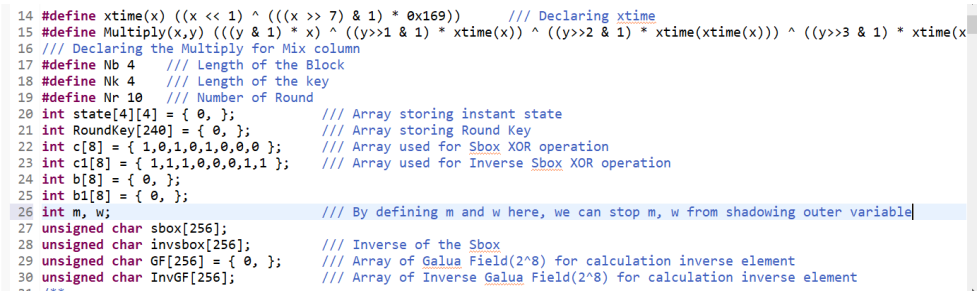
) errors, 2 warnings, 0 others

Description	Resource	Path	Locatio
Warnings (2 items)			
(cppcheck style) Local variable a shadows outer variable	main.cpp	/2	line 39
(cppcheck style) Local variable d shadows outer variable	main.cpp	/2	line 11

4. Local variable m, w shadows outer variable



Local variable m and w shadows outer variable, so we can define it outside of function to make it global variable. And we can remove int ahead of m and w.



By defining m and w here, we can stop m, w from shadowing outer variable.

5. Variable d is assigned a value that is never used.

```
112 void KeyExpansion(int *key)
113 {
114     int i, j, d = 0;
115     unsigned char temp[4], k;
116     cout << "ROUND 0: ";
117     for (i = 0; i < Nk; i++)    /// Storing key value in Round key
118     {
119         RoundKey[4 * i] = key[4 * i];
120         RoundKey[4 * i + 1] = key[4 * i + 1];
121         RoundKey[4 * i + 2] = key[4 * i + 2];
122         RoundKey[4 * i + 3] = key[4 * i + 3];
123     }
124     for (k = 0; k < 16; k++) {
125         cout << RoundKey[k] << " ";
126     }
127     cout << endl;
128
129     while (i < (Nb * (Nr + 1)))
```

Problems | Tasks | Console | Properties

errors, 1 warning, 8 others

Description	Resource	Path
invalid operands of types 'int' and 'int [8]' to binary 'operator*'	main.cpp	/2
make: *** [subdir.mk:20: main.o] Error 1	2	
Warnings (1 item)		
(cppcheck style) Variable 'd' is assigned a value that is never used.	main.cpp	/2

Variable d is assigned with 0, and it is told that the value is never used. So we solved the problem by erasing '=0'.

```
112 void KeyExpansion(int *key)
113 {
114     int i, j, d;        /// solved by cppcheck erasing definition by 0
115     unsigned char temp[4], k;
116     cout << "ROUND 0: ";
117     for (i = 0; i < Nk; i++)    /// Storing key value in Round key
118     {
119         RoundKey[4 * i] = key[4 * i];
120         RoundKey[4 * i + 1] = key[4 * i + 1];
121         RoundKey[4 * i + 2] = key[4 * i + 2];
122         RoundKey[4 * i + 3] = key[4 * i + 3];
123     }
124     for (k = 0; k < 16; k++) {
125         cout << RoundKey[k] << " ";
126     }
127     cout << endl;
```

6. Invalid arguments

```
532     int cipher[128] = { 0, };   /// Array storing cipher text
533     cout << endl << endl << "<------ENCRYPTION-------->" << endl << endl << "KEY EXPANSION" << endl;
534     KeyExpansion(*key);
535     Cipher(plain, cipher);      /// Encryption
536     cout << endl << "CIPHER : ";
537     for (i = 0; i < 16; i++)
538         cout << cipher[i] << " ";
539     cout << endl << endl << "<------DECRYPTION-------->" << endl << endl;
540
541     InvCipher(cipher, decrypted);   /// Decryption
542     cout << "DECRYPTED : ";
543     for (i = 0; i < 16; i++)
544         cout << decrypted[i] << " ";
```

Problems | Tasks | Console | Properties

error, 0 warnings, 0 others

Description	Resource	Path	Location
Errors (1 item)			
Invalid arguments '	main.cpp	/2	line 534

This issue is not existing in our project, because it is already solved. But we can see that parameter key needs to remove * ahead.

```
531
532     int cipher[128] = { 0, };   /// Array storing cipher text
533     cout << endl << endl << "<------ENCRYPTION-------->" << endl << endl << "KEY EXPANSION" << endl;
534     KeyExpansion(key);
535     Cipher(plain, cipher);      /// Encryption
536     cout << endl << "CIPHER : ";
537     for (i = 0; i < 16; i++)
538         cout << cipher[i] << " ";
539     cout << endl << endl << "<------DECRYPTION-------->" << endl << endl;
540
541     InvCipher(cipher, decrypted);   /// Decryption
542     cout << "DECRYPTED : ";
543     for (i = 0; i < 16; i++)
544         cout << decrypted[i] << " ";
```

7. Array accessed out of bounds

```
47      }
48      InvGF[256] = 0;
49
50      for (int i = 1; i < 256; i++) {
51          m = InvGF[(255 - GF[i])];       /// Inverse element
52          for (int j = 7; j >= 0; j--) {
53              b[j] = m / (1 << j);        /// Generating inverse element into binary b[0] to b[7]
54              m %= (1 << j);
55          }
56          for (int k = 0; k < 8; k++) {
57              b1[k] = b[k] ^ b[(k + 4) % 8] ^ b[(k + 5) % 8] ^ b[(k + 6) % 8] ^ b[(k + 7) % 8] ^ c[k];
58              m += b1[k] * (1 << k);      /// Restore from binary
59          }
```

Problems Tasks Console Properties

error, 0 warnings, 0 others

Description	Resource	Path	Loc
Errors (1 item)			
(cppcheck error) Array 'InvGF[256]' accessed at index 256, which is out of bounds.	main.cpp	/2	line

Cppcheck finds error when index is out of bounds. Index of array 'InvGF[256]' should be lower than 256.

```
47      }
48      InvGF[255] = 0;
49
50      for (int i = 1; i < 256; i++) {
51          m = InvGF[(255 - GF[i])];       /// Inverse element
52          for (int j = 7; j >= 0; j--) {
53              b[j] = m / (1 << j);        /// Generating inverse element into binary b[0] to b[7]
54              m %= (1 << j);
55          }
56          for (int k = 0; k < 8; k++) {
57              b1[k] = b[k] ^ b[(k + 4) % 8] ^ b[(k + 5) % 8] ^ b[(k + 6) % 8] ^ b[(k + 7) % 8] ^ c[k];    // ??? ????
58              m += b1[k] * (1 << k);      /// Restore from binary
59          }
```

Problems Tasks Console Properties

0 items

Description	Resource	Path	Location	Type

8. Symbol ‘i’ could not be resolved

```
422
423     /// Storing final encrypted state array in cipher array
424     for (i = 0; i < 4; i++)
425     {
426         for (int j = 0; j < Nb; j++)
427         {
428             cipher[4 * i + j] = state[j][i];
429         }
430     }
431 }
432 /**
433  *
434  * @param [in] cipher The array which stores ciphered text
```

Problems | Tasks | Console | Properties

5 errors, 0 warnings, 0 others

Description	Resource	Path
Errors (5 items)		
Symbol 'i' could not be resolved	main.cpp	/2
Symbol 'i' could not be resolved	main.cpp	/2
Symbol 'i' could not be resolved	main.cpp	/2
Symbol 'i' could not be resolved	main.cpp	/2
Symbol 'i' could not be resolved	main.cpp	/2

Cppcheck checks if the symbol is not resolved. It has to be defined whether if it is a variable. So we already have int ahead of variable i.

```
416         cout << endl;
417     }
418     cout << "Round 10" << endl;
419     SubBytes();
420     ShiftRows();
421     AddRoundKey(Nr);    /// XORing last key and current block
422
423     /// Storing final encrypted state array in cipher array
424     for (int i = 0; i < 4; i++)
425     {
426         for (int j = 0; j < Nb; j++)
427         {
428             cipher[4 * i + j] = state[j][i];
429         }
430     }
431 }
432 /**
433  *
434  * @param [in] cipher The array which stores ciphered text
```

9. Consecutive break are unnecessary

```
500     fp = fopen("key.bin", "rb");
501     fp1 = fopen("plain.bin", "rb");
502     if (fp == NULL) {       /// fopen returns NULL when error occurs
503         perror("key.txt");
504         exit(1);
505         break;
506     }
507     if (fp1 == NULL) {  /// fopen returns NULL when error occurs
508         perror("plain.txt");
509         exit(1);
510     }
511     /// Printing plain text array
512     while (fread((void*)buf, 1, sizeof(buf), fp1)) {
513         cout << "PLAIN : ";
514         for (i = 0; i < 16; i++) {
515             plain[i] = buf[i];
516             cout << plain[i] << " ";
517         }
518         cout << endl;
```

Problems | Tasks | Console | Properties

) errors, 1 warning, 0 others

Description	Resource	Path	Location	Type
Warnings (1 item)				
(cppcheck style) Consecutive return, break, continue, goto or throw statements are unnecessary.	main.cpp	/2	line 505	cppcl

Cppcheck checks whether if consecutive statements are unnecessary to be used. We already don't have any unnecessary statements.

```
498
499     /// Reading binary file
500     fp = fopen("key.bin", "rb");
501     fp1 = fopen("plain.bin", "rb");
502     if (fp == NULL) {       /// fopen returns NULL when error occurs
503         perror("key.txt");
504         exit(1);
505     }
506     if (fp1 == NULL) {  /// fopen returns NULL when error occurs
507         perror("plain.txt");
508         exit(1);
509     }
510     /// Printing plain text array
511     while (fread((void*)buf, 1, sizeof(buf), fp1)) {
512         cout << "PLAIN : ";
513         for (i = 0; i < 16; i++) {
514             plain[i] = buf[i];
```

10. Unmatched ')'

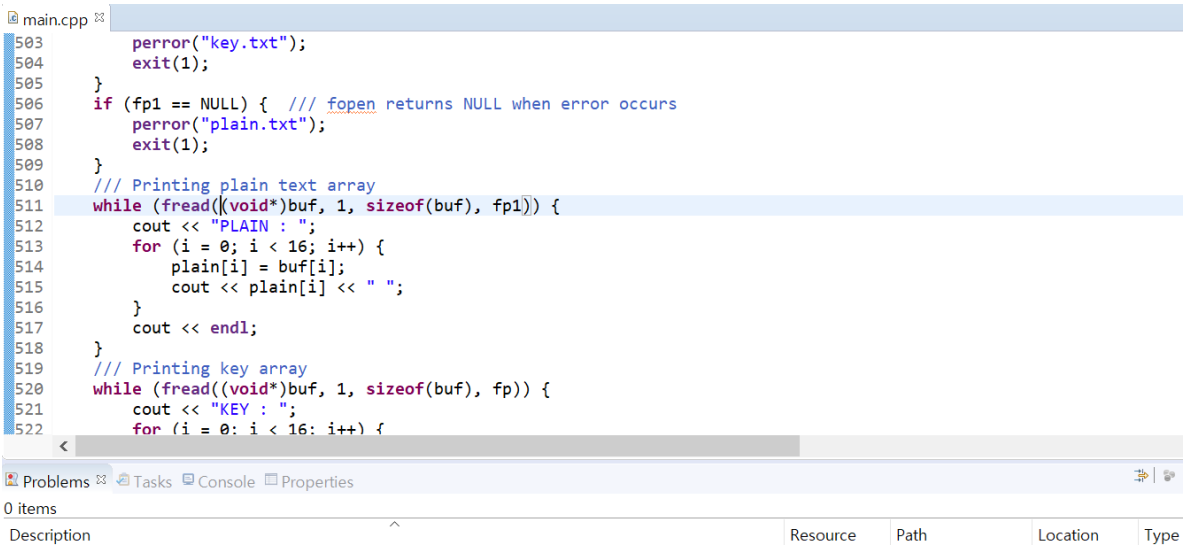
```
510     /// Printing plain text array
511     while (fread(void*)buf, 1, sizeof(buf), fp1)) {
512         cout << "PLAIN : ";
513         for (i = 0; i < 16; i++) {
514             plain[i] = buf[i];
515             cout << plain[i] << " ";
516         }
517         cout << endl;
518     }
519     /// Printing key array
520     while (fread((void*)buf, 1, sizeof(buf), fp)) {
521         cout << "KEY : ";
522         for (i = 0; i < 16; i++) {
```

Problems | Tasks | Console | Properties

1 error, 0 warnings, 0 others

Description	Resource	Path
Errors (1 item)		
(cppcheck error) Unmatched ')'. Configuration: ''.	main.cpp	/2

Cppcheck checks if apostrophe is closed. We already closed well to define fread().



The reduction of the cyclomatic complexity by metriculator

We have 14 functions in our project. And when we sorted it by McCabe, we can see that two functions : KeyExpansion() and main() V(G) are over 10. So we modified the functions to make them same or below than 10.

main.cpp

```
24 int b1[8] = { 0, };
25 int m, w;                          /// By defining m and w here, we can stop m, w from shadowing ou
26 unsigned char a, b_, c_, d;     /// By defining a,b,c,d here, we can stop them from shadowing outer
27 unsigned char sbox[256];
28 unsigned char invsbox[256];        /// Inverse of the Sbox
29 unsigned char GF[256] = { 0, };    /// Array of Galua Field(2^8) for calculation inverse element
30 unsigned char InvGF[256];          /// Array of Inverse Galua Field(2^8) for calculation inverse e
31 /**
32  *
33  * @param [in] num The index of Sbox array
```

Problems | Tasks | Console | Properties | metriculator

Scope (14 items)	LSLOC	EfferentC...	McCabe	NbParams	NbMem...
KeyExpansion(i...	50	0	19	1	0
main(void)	70	0	14	1	0
Sbox(size_t nu...	22	0	6	1	0
Cipher(int * pla...	22	0	6	2	0
InvCipher(int * ...	22	0	6	2	0
InvSbox(size_t n)	13	0	4	1	0
MixColumns()	22	0	4	0	0
InvMixColumns()	15	0	4	0	0
RCon(size_t nu...	7	0	3	1	0
AddRoundKey(...	7	0	3	1	0
SubBytes()	7	0	3	0	0
InvSubBytes()	7	0	3	0	0
ShiftRows()	23	0	3	0	0
InvShiftRows()	23	0	3	0	0

Main function : V(G)=14

```
/**
 * @brief Opening cipher and decrypted binary file and writing the output
 */

void Decrypt() {
    int i;
    unsigned char buf1[16];
    int decrypted[20];  /// Array storing decrypted text
    int cipher[128] = { 0, };   /// Array storing cipher text

    FILE *fp2, *fp3;
    /// Writing cipher text and decrypted text in binary file
    fp2 = fopen("cipher.bin", "wx");
    fp3 = fopen("decrypted.bin", "wx");
    if (!fp2) {
        perror("cipher.bin");
        exit(1);
    }
    if (!fp3) {
        perror("decrypted.bin");
        exit(1);
    }
    /// Writing cipher file
    for (i = 0; i < 16; i++) {
        buf1[i] = cipher[i];
    }
    fwrite(&buf1, 1, sizeof(buf1), fp2);

    /// Writing decrypted file
    for (i = 0; i < 16; i++) {
```

We worked on main function first. We made a new function called Decrypt(), which opens cipher and decrypted binary file and writes the output. By this modification, the V(G) of main() became 10, and it meets our expectation.

Problems | Tasks | Console | Properties | metriculator

Scope (15 items)	LSLOC	EfferentC...	McCabe	NbParams	NbMem...
KeyExpansion(i...	50	0	19	1	0
main(void)	49	0	10	1	0
Sbox(size_t nu...	22	0	6	1	0
Cipher(int * pla...	22	0	6	2	0
InvCipher(int * ...	22	0	6	2	0
Decrypt()	23	0	5	0	0
InvSbox(size_t n)	13	0	4	1	0
MixColumns()	22	0	4	0	0
InvMixColumns()	15	0	4	0	0
RCon(size_t nu...	7	0	3	1	0
AddRoundKey(...	7	0	3	1	0
SubBytes()	7	0	3	0	0
InvSubBytes()	7	0	3	0	0
ShiftRows()	23	0	3	0	0
InvShiftRows()	23	0	3	0	0

KeyExpansion function : V(G)=19

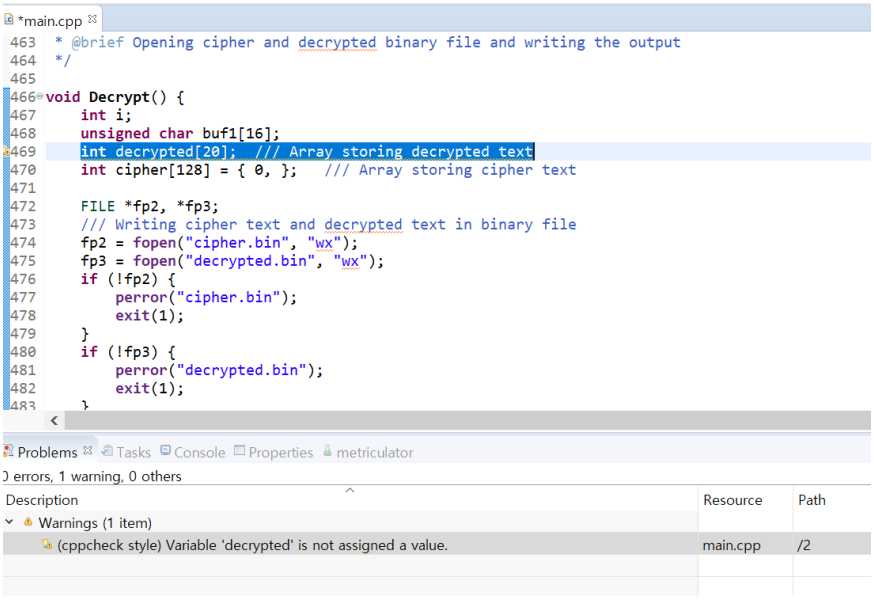
```
149                 temp[3] = Sbox(temp[3]);
150             }
151             temp[0] = temp[0] ^ RCon(i / Nk - 4);       // We changed this line by adding -4 to prevent it from hardcoding
152         }
153         RoundKey[4 * i + 0] = RoundKey[(i - Nk) * 4 + 0] ^ temp[0];
154         RoundKey[4 * i + 1] = RoundKey[(i - Nk) * 4 + 1] ^ temp[1];
155         RoundKey[4 * i + 2] = RoundKey[(i - Nk) * 4 + 2] ^ temp[2];
156         RoundKey[4 * i + 3] = RoundKey[(i - Nk) * 4 + 3] ^ temp[3];
157         i++;
158     }
159     for (i = 1; i <= Nr; i++)
160     {
161         cout << dec << "ROUND " << i << ": ";
162         for (j = 0; j < 16; j++)
163         {
164             cout << hex << RoundKey[i * 16 + j] << " ";
```

Next function was KeyExpansion, which had higher complexity compared to the main function. The code had the part of hardcoding which made the cyclomatic complexity high. So we erased the part, and found the solution by putting minus 4 in the code "`temp[0] = temp[0] ^ RCon(i / Nk);`". After checking the code, it was successfully built and the complexity became 8.

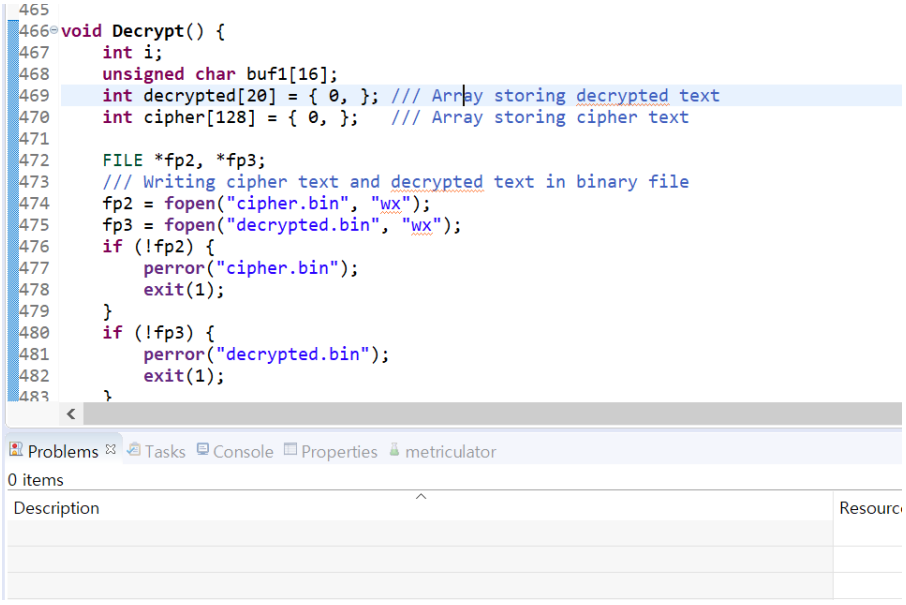
Problems | Tasks | Console | Properties | metriculator

Scope (15 items)	McCabe	EfferentC...	LSLOC	NbParams	NbMem...
main(void)	10	0	49	1	0
KeyExpansion(i...	8	0	38	1	0
Sbox(size_t nu...	6	0	22	1	0
Cipher(int * pla...	6	0	22	2	0
InvCipher(int * ...	6	0	22	2	0
Decrypt()	5	0	23	0	0
InvSbox(size_t n)	4	0	13	1	0
MixColumns()	4	0	22	0	0
InvMixColumns()	4	0	15	0	0
RCon(size_t nu...	3	0	7	1	0
AddRoundKey(...	3	0	7	1	0
SubBytes()	3	0	7	0	0
InvSubBytes()	3	0	7	0	0
ShiftRows()	3	0	23	0	0
InvShiftRows()	3	0	23	0	0

Application of cppcheck to the reviewed modules

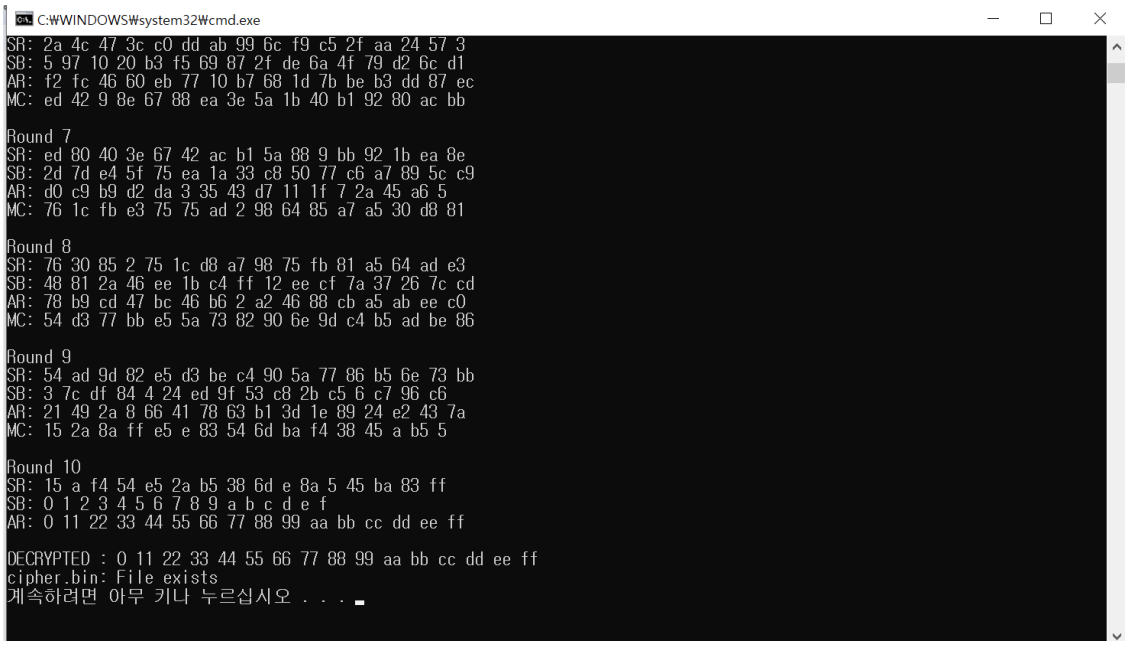


The cppcheck detected warning from new function Decrypt, which is telling that Variable 'decrypted' is not assigned a value. So we defined it like Variable 'cipher', and the warning is removed.

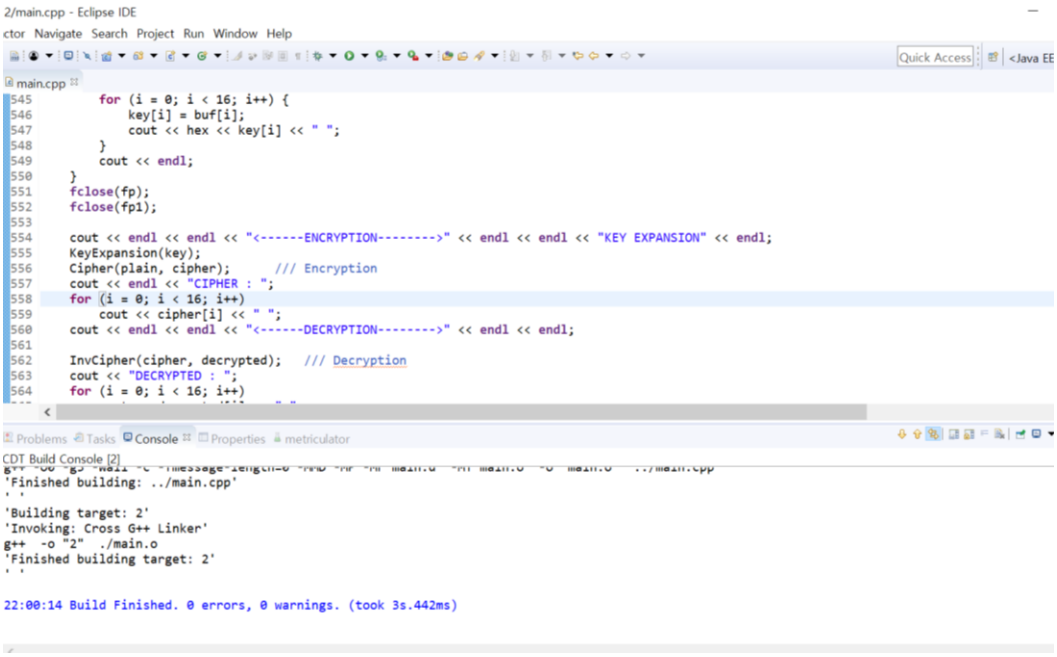


Conclusion

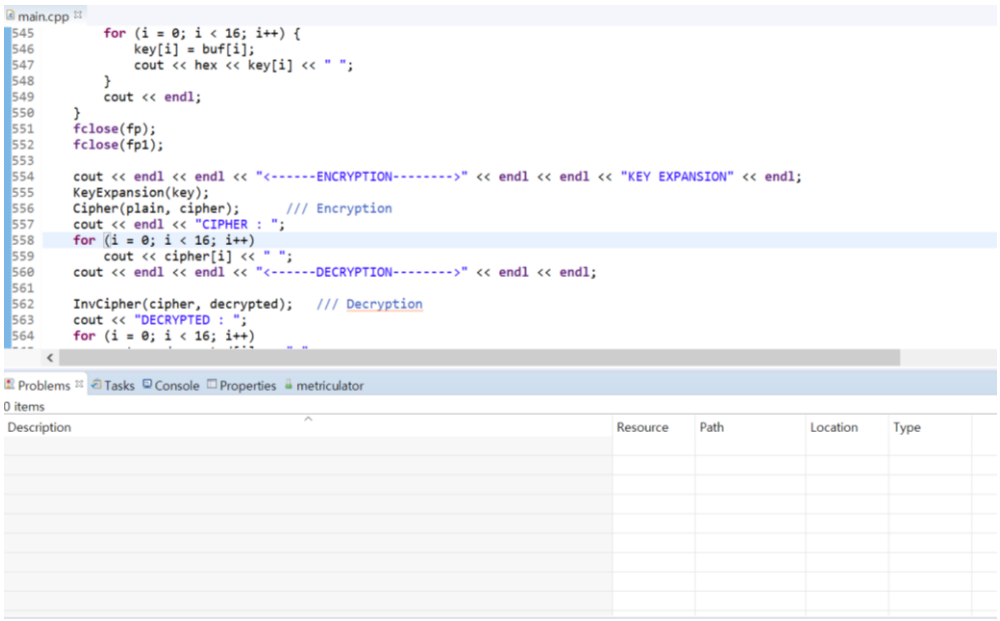
The final modified code of our program works well.



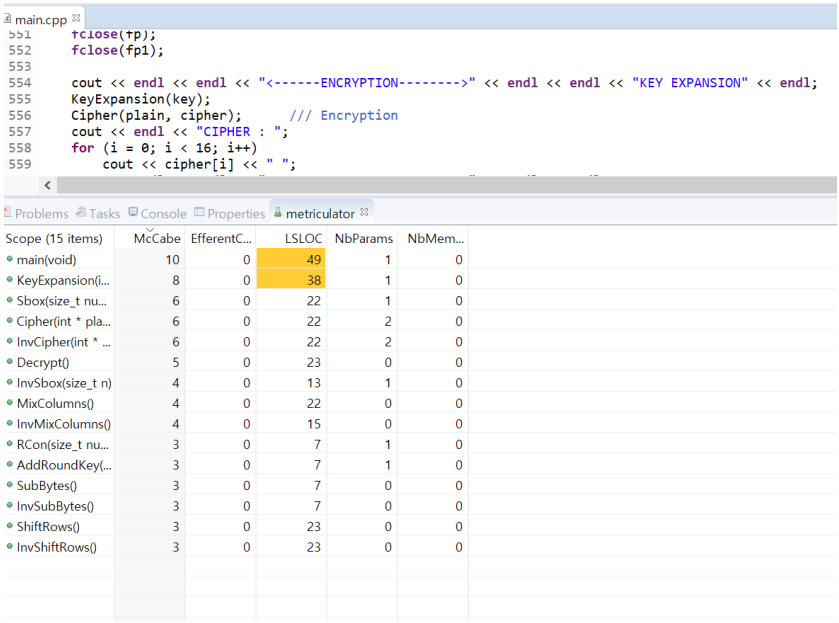
We've met 10 standards of the rules, SEI CERT Coding Standards including C and C++.



We can also see that our code works well with eclipse.



By cppcheck, we don't have any problems to solve.



We've met the regulation of our program which was V(G)≤10. We've reduced cyclomatic complexity of our module, which was main and KeyExpansion

function.

We've met 10 standards of SEI CERT Coding Standards by modifying our code, solved 10 issues from cppcheck by selecting some of the problems that cppcheck is able to detect, and finally reduced cyclomatic complexity of our modules each by same or lower than 10. Our program was finally improved by SEI CERT Coding Standards, Cppcheck, and Metriculator. The potential of error occurring got lowered by handling some exceptions and we checked errors and warnings by cppcheck to ensure the confidence of our program. We have finally checked our complexity by metriculator to make our program work efficiently. We made a change to our code by removing hard coded part and finding a new solution. We can say that our final program's software quality has improved to a better quality.