REPORT

Information Security Project #1

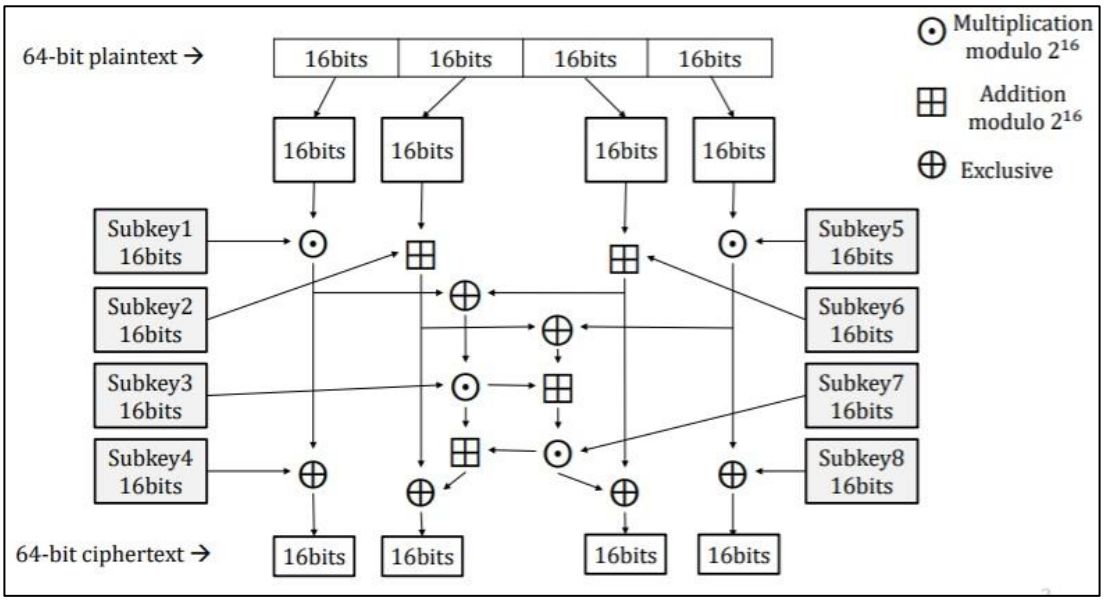
Block Cipher

학과	컴퓨터학과
학번	2015410056
이름	김지윤
제출 일자	2018/10/14
담당 교수	허준범 교수님

Project #1: Block Cipher

Find the key of given one-round symmetric-key block cipher.

■ Given Block Cipher - Encryption Workflow



- ✓ **Block size**는 64-bit이다
- ✓ **Key**는 8개의 16-bit subkey로 이루어져 있다.
- ✓ Encryption과정에서 세 가지 연산이 사용된다
  - a) Multiplication modulo 2^{16}
  - b) Addition modulo 2^{16}
  - c) XOR
- ✓ 64-bit의 plaintext block은 네 부분(1^{st}, 2^{nd}, 3^{rd}, 4^{th} subtext) 으로 나뉘어 encrypt되고 각 subtext는 permutation없이 순서대로 ciphertext가 된다.
- ✓ 네 개의 16-bit subtext 중 1^{st}, 4^{th} subtext는 독립적인 연산이 가능하다. (input subtext와 두 개의 subkey 만으로 output subtext를 계산할 수 있다.)

■ Approach to Find the Key

8개의 subkey 중 subkey3과 subkey7을 제외한 나머지 6개는 모두 랜덤하게 생성되기 때문에 각 subkey에 대해 모든 범위(0 ~ 2^{16}-1)를 테스트한다면 시간이 오래 걸릴 수 있다. (key 조합 65536^6 X 4개의 plain/ciphertext pair 테스트 ≒ 3.16 X 10^{11})

시간 효율성을 위해 아래의 두 가지 방법을 통해 연산의 횟수를 줄였다.

1. Even/Odd Test

Encryption 과정에는 세 가지 연산이 사용된다. (Multiplication modulo $2^{16,}$ Addition modulo $2^{16,}$ XOR) 세 연산 모두 결과의 even/odd 여부가 연산이 받는 두 operand의 even/odd 조합에 따라 결정되는데 이를 정리해보면 아래 표와 같다.

Multiplication mod 2^{16}			**Addition mod 2^{16}**			**XOR**		
Input 1	Input 2	Result	Input 1	Input 2	Result	Input 1	Input 2	Result
Even	Even	Even	Even	Even	Even	Even	Even	Even
Even	Odd	Even	Even	Odd	Odd	Even	Odd	Odd
Odd	Even	Even	Odd	Even	Odd	Odd	Even	Odd
Odd	Odd	Odd	Odd	Odd	Even	Odd	Odd	Even

4개의 plaintext, ciphertext의 even/odd 조합과 위의 표를 맞춰보면 가능한 subkeys의 even/odd 조합을 알 수 있다.

1) 1st subtext, 4th subtext (related subkey: subkey1, subkey4, subkey5, subkey8)

먼저 encryption 연산이 비교적 간단한 1st, 4th subtext부터 살펴보면

- ✓ 1st 16-bit plain/cipher pair: {(601**8**, 3AC**5**), (0A8**1**, 192**C**), (2E7**0**, 0B2**D**), (F77**8**, 7BA**5**)}
- ✓ 4th 16-bit plain/cipher pair: {(84A**9**, 724**E**), (DA5**A**, 69E**D**), (45A**0**, D8F**7**), (4AB**1**, 2DF**6**)}

Plaintext와 cipher text가 서로 반대 관계에 있는 것을 알 수 있다. ((plain, cipher)가 (even, odd) 또는 (odd, even))

이를 토대로 1st, 4th의 두 연산 결과를 정리해보면 아래 표와 같고,

Mul-mod		**XOR**			**Mul-mod**		**XOR**		
Input	Subkey	Mid	Subkey	Result	Input	Subkey	Mid	Subkey	result
Even	**Even**	Even	Even	Even	Odd	**Even**	Even	**Even**	Even
			Odd	Odd				Odd	Odd
	Odd	Even	Even	Even		**Odd**	Odd	Even	Odd
			Odd	Odd				**Odd**	Even

Input plaintext가 even일 경우와 odd일 경우를 모두 만족하려면 multiplication mod 연산과 XOR 연산의 operand인 subkey 둘 다 odd여야 한다.

따라서 subkey1, subkey4, subkey5, subkey8이 될 수 있는 값은 0이상 65535이하의 홀수가 된다.

2) 2nd subtext, 3rd subtext (related subkey: subkey1, subkey2, subkey3, subkey4, subkey5, subkey6, subkey7, subkey8)

2nd, 3rd subtext에 대해서도 위와 같이 각 input, output subtext와 encryption 연산에 관여하는 subkey가 even일 경우, odd일 경우를 나누어서 subkey의 가능한 even / odd 조합을 알아낸다. 두 subtext의 연산에는 모든 subkey가 관여한다고 볼 수 있지만 위에서 서술했듯이 subkey1, subkey4, subkey5, subkey8의 값은 홀수임이 확실하므로 나머지 네 개의 subkey에 대해 가능한 조합을 찾는다.

Subkey2, subkey3, subkey6, subkey7의 even/odd 조합은 총 여섯 가지가 가능하다.

	Subkey2	Subkey3	Subkey6	Subkey7
Case 1	Even	Even	Odd	Odd
Case 2	Even	Odd	Even	Even
Case 3	Even	Odd	Even	odd
Case 4	Even	Odd	Odd	Odd
Case 5	Odd	Even	Even	Even
Case 6	Odd	Even	Even	Odd

2. 1st subtext (subkey1, subkey4), 4th subtext (subkey5, subkey8) 먼저 계산

1st subtext와 4th subtext의 encrypt과정에는 input인 subtext, 그리고 각각에 해당되는 두 개의 subkeys 외에 연산에 관여하는 것이 없다.

8개의 subkey조합을 먼저 만들어내고 이후에 모든 연산을 진행한다면 불필요한 반복구간이 생기게 된다. 이러한 상황을 막기 위해 독립적인 연산이 가능한 1st, 4th subtext의 encrypt 부분을 먼저 수행하여 valid한 subkey1, 4, 5, 8을 구한 후 이를 나머지 subkey 2, 3, 6, 7을 찾는데 사용한다.

■ Code Description

주어진 block cipher의 8 subkeys를 찾기 위해 가장 먼저 위에 서술했던 것처럼 가능한 subkey의 even odd 조합을 찾는 함수를 만들었고 그 결과를 토대로 8개의 subkey를 찾는 부분을 구현하였다. Subkey를 찾는 함수는 네 개의 subkey (subkey1, 4, 5, 8)를 먼저 찾고 나머지는 미리 계산된 subkey값을 이용하여 찾도록 했다.

- ✓ 개발언어: C++
- ✓ 개발환경: Visual Studio 2017

1. Operation

encryption과정의 세가지 연산은 아래와 같이 구현한다. (operand: a, b)

- ✓ Multiplication modulo 2^{16}: (a*b) & (2^{16}-1)
- ✓ Addition modulo 2^{16}: (a*b) & (2^{16}-1)
- ✓ XOR: (a^b)

Modulo 연산은 % 대신에 bitwise operation (AND)을 사용하는 것이 연산 속도가 더 빠르다. (…■Reference)

2. EvenOdd_test 함수

Subkey2, subkey3, subkey6, subkey7의 가능한 even odd 조합을 찾는 함수이다.

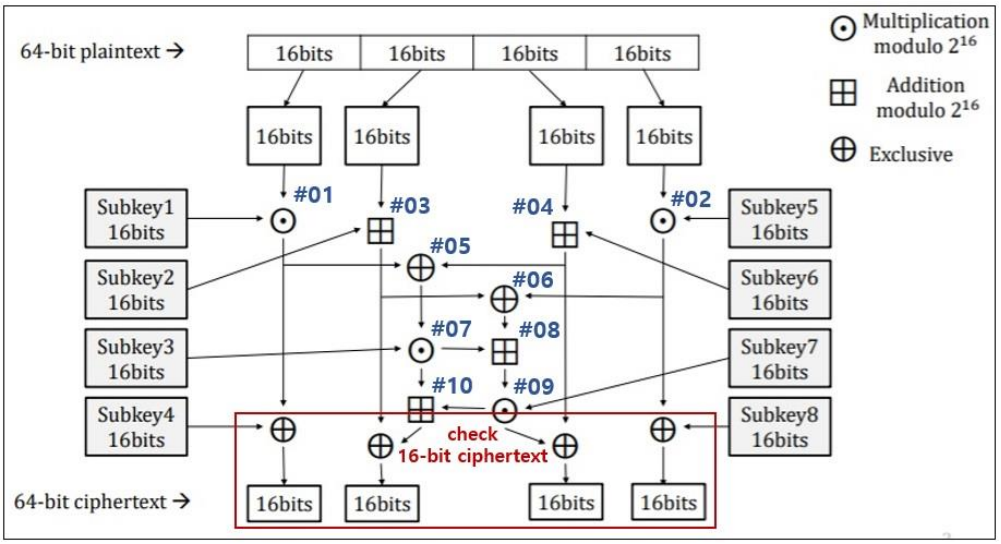
a) Variables

- ✓ input (vector <vector <int>>): plaintext의 even/odd조합을 나타낸다. 0은 짝수를, 1은 홀수를 나타낸다.
- ✓ output (vector <vector <int>>): ciphertext의 even/odd조합을 나타낸다.
- ✓ val (vector <int>): 각 연산의 결과(even/odd)를 임시로 저장하는 vector이다.
- ✓ Subkeys (vector <int>): test할 subkey의 값을 담는 vector이다.

b) 동작 과정

1) Subkey 1, 4, 5, 8이 홀수이기 때문에 (∵ ■Approach to Find the Key-1-1)) subkeys[1], subkeys[4], subkeys[5], subkeys[8]에는 모두 1을 대입한다
2) 나머지 subkey에 대한 모든 even/odd 조합을 만들어내는 4중 for문(범위는 0이상

1이하 – 짝수 또는 홀수)을 만들고 그 내부에서 연산을 차례로 수행한다. 연산 순서는 아래 그림에 붙은 숫자 순서대로 진행하고 각 연산의 결과를 vector 'val'에 저장한다. Even/odd만을 계산하는 것이기 때문에 모든 연산은 modulo 2^{16}대신에 modulo 2로 계산한다.



3) 연산의 결과가 각 ciphertext의 even/odd와 일치한다면 subkey조합을 출력한다.

3. Check1st16bits 함수

첫번째 16-bit plaintext의 encryption에 관여하는 valid한 subkey1, subkey4의 pair를 찾아서 리턴하는 함수이다. Subkey1과 subkey4의 범위를 0이상 2^{16}미만의 홀수로 잡고 for문을 돌려서 목표 값을 찾는다.

4. Check4th16bits 함수

Check1st16bits와 동일하게 동작하며 subkey5와 subkey8을 찾는다.

5. Find8subkeys 함수

8개의 subkey를 찾아 출력하는 함수이다. 함수의 전체적인 동작 과정과 주요 변수들은 'EvenOdd_test' 함수와 비슷하지만 크게 두 가지 차이점이 있다.

1) Subkey1, 4, 5, 8을 1(odd)로 하드코딩하는 대신에 'Check1st16bits', 'Check4th16bits'를 호출하여 valid한 subkey 4개를 먼저 계산하고 함수의 리턴값을 대입한다.

2) 이중 for문으로 subkey2, subkey6을 랜덤으로 생성하고 이것이 subkey2, 3, 6, 7의 even/odd 조합 여섯 가지 case 중 하나에 해당되는지 체크한다.

```
for (int k2 = 0; k2 < bitsize; ++k2) {
  if ((k2 & 5) == 5)
    continue;
  for (int k6 = 0; k6 < bitsize; ++k6) {
    if ((k2 & 5) == 0) { //case1
      if ((k6 & 5) != 5)
        continue;
    }
    else if ((k2 & 5) == 4) { //case2, case3, case4
      if ((k6 & 5) == 1)
        continue;
    }
    else { //if ((k2 & 5) == 1)
      if ((k6 & 5) == 1) //case5
        continue;
      if ((k6 & 5) == 5) //case6
        continue;
    }

    subkeys[2] = k2;
    subkeys[3] = (k2 >> 2);
    subkeys[6] = k6;
    subkeys[7] = (k6 >> 2);

    /*--------- subkeys 8개 모두 생성 ---------*/
```

	K2	K3	K6	K7
case1	0	0	1	1
case2	0	1	0	0
case3	0	1	0	1
case4	0	1	1	1
case5	1	0	0	0
case6	1	0	0	1

위의 테이블은 ■Approach to Find the Key-1-2)에서 계산한 subkey2, 3, 6, 7의 even/odd 조합을 나타낸다.

그리고 subkey3과 subkey7은 각각 subkey2와 subkey6를 2-bit right shift 한 결과이다.

따라서 네 가지 subkey가 위의 조합에 해당되는지는 key2와 key6의 가장 오른쪽 bit와 오른쪽에서 세번째 bit를 체크함으로써 확인할 수 있고 두 bit는 'AND 3(101)' 연산을 통해 확인할 수 있다.

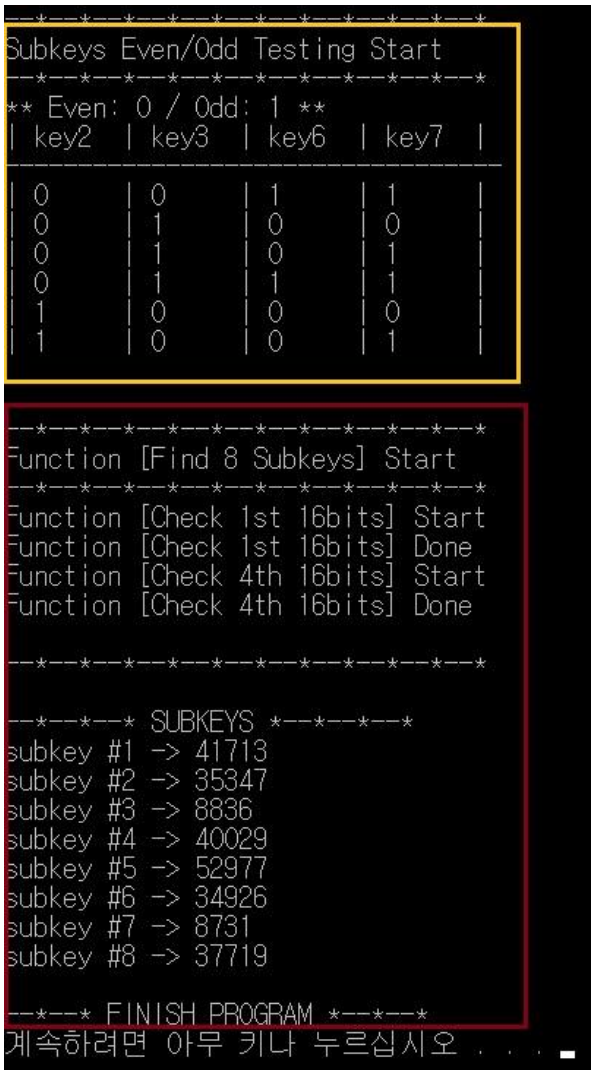
그 뒤의 연산은 'EvenOdd_test'함수의 동작 순서와 동일하다. Encryption workflow의 연산을 순서대로 진행하고 그 결과를 vector 'val'에 저장해 다음 연산에 사용한다. 연산이 끝난 뒤엔 ciphertext와 비교해보고 일치한다면 subkey를 출력한다.

6. Main 함수

'EvenOdd_test' 함수와 'Find8subkeys'함수를 순서대로 호출하여 subkey들의 even/odd조합, 그리고 최종 결과인 8개의 subkey를 순서대로 출력하도록 했다.

■ Result

1. Screen capture of the running program



노란색 네모 친 부분이 'EvenOdd_test'의 결과를 출력한 것이고

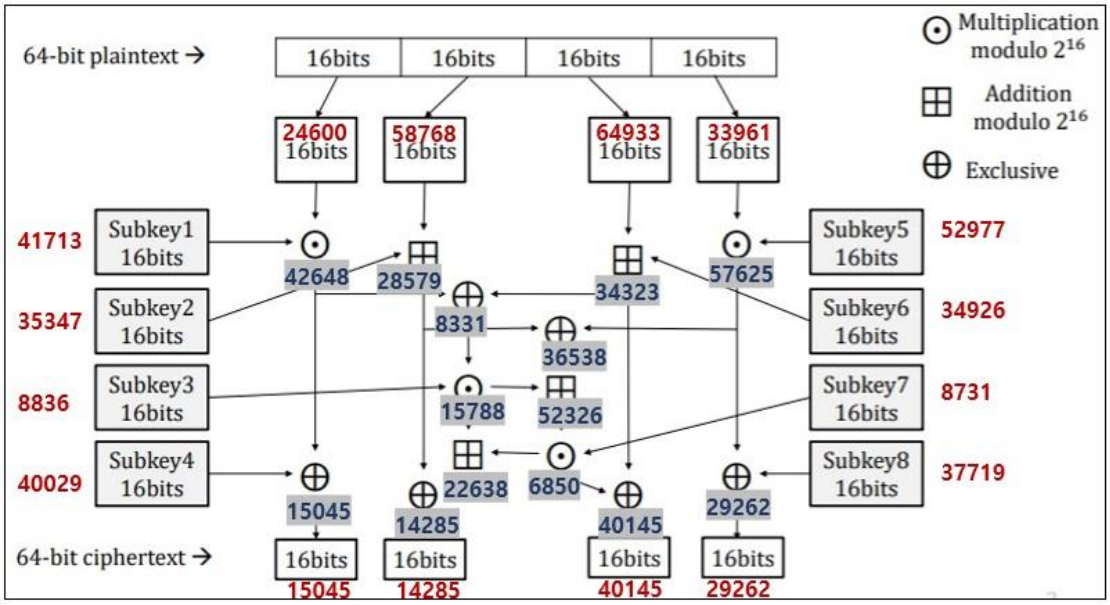
빨간색 네모 친 부분이 'Find8subkeys'의 결과이다.

2. Answer

	in decimal	in hex
Subkey #1	41713	A2F1
Subkey #2	35347	8A13
Subkey #3	8836	2284
Subkey #4	40029	9C5D
Subkey #5	52977	CEF1
Subkey #6	34926	886E
Subkey #7	8731	221B
Subkey #8	37719	9357

3. Verifying answer

결과로 나온 subkey와 첫번째 plaintext, ciphertext pair로 encryption workflow를 그대로 따라 계산해보면 아래와 같다.



마지막 연산의 결과값과 ciphertext가 서로 일치하는 것을 알 수 있다.

■ References

- ✓ http://mziccard.me/2015/05/08/modulo-and-division-vs-bitwise-operations/
- ✓ https://kulms.korea.ac.kr/bbcswebdav/pid-2235843-dt-content-rid-5945425_1/courses/20182R0136COSE35402/project1%281%29.pdf