**정보보호 - Project 2**

고려대학교 컴퓨터학과

2015410113 신채호

**Analyzing hash function**

**0. 실행환경**

- Python3 이용

**1 Round.**

Ti를 i번째 라운드의 T 값이라고 하면, final output Y는

Y = (A0 ⊕ H0 ⊕ T0, B0 ⊕ A0, C0 ⊕ B0, D0 ⊕ C0, E0 ⊕ D0, F0 ⊕ E0, G0 ⊕ F0, H0 ⊕ G0)이다.

Y의 각 32bit 값을 Y[0] ~ Y[7] 이라고 하면 32bit A0의 값이 주어질 때,

B0 = Y[1] ⊕ A0, C0 = Y[2] ⊕ B0, D0 = Y[3] ⊕ C0, E0 = Y[4] ⊕ D0,

F0 = Y[5] ⊕ E0, G0 = Y[6] ⊕ F0, , H0 = Y[7] ⊕ G0 이므로, A0부터 H0까지 모두 구할 수 있다.

A0 ~ H0가 주어지면 inverse MDS와 inverse Sbox를 이용해 W0도 구할 수 있으므로,

그렇다면 모든 232개의 A0에 대하여 각각 B0~H0까지 그리고 마지막으로 W0를 구한다음, 구한 A0~W0와 임의의 고정된 W1~W15(1 round hash function의 경우 input output에 영향을 주지 않음)를 hash function을 돌렸을 때, 우리의 output Y와 똑같이 나오는 A0가 존재한다면, 이 A0를 통해 구한 X = (A0, B0, C0, D0, E0, F0, G0, H0, W0, ~W15) 가 input이 된다.

총 232번의 hash operation이 필요하다.

**2 Round.**

마찬가지로 final output Y를 나타내면

Y = (A0 ⊕ G0 ⊕ T1, B0 ⊕ H0 ⊕ T0, C0 ⊕ A0, D0 ⊕ B0, E0 ⊕ C0, F0 ⊕ D0, G0 ⊕ E0, H0 ⊕ F0)이다.

Round 1과 마찬가지로, 32bit A0의 값이 주어질 때, C0, E0, G0, T0 값 모두 알 수 있다. Y[0]부터 Y[7]까지 모두 더하면 T0 ⊕ T1 인데, T1의 값을 알고 있으므로 T0도 구할 수 있다. B0, D0, F0, H0는 나머지 값들을 이용해 구할 수 없으므로 B0도 고정된 값으로 주면, A0~H0, T0, T1 모두 구할 수 있고, T0와 T1을 구하면 inverse MDS와 inverseSbox를 이용해 W0와 W1도 구할 수 있다. 그럼, 임의의 W2~W15에 대하여 X = (A0, B0, C0, D0, E0, F0, G0, H0, W0, W1, ~W15)가 input이 된다.

총 232 \* 232 = 264 번의 hash operation이 필요하다.

**3~7 Rounds.**

3 round: Y = (A0 ⊕ F0 ⊕ T2, B0 ⊕ G0 ⊕ T1, C0 ⊕ H0 ⊕ T0, D0 ⊕ A0, E0 ⊕ B0, F0 ⊕ C0, G0 ⊕ D0, H0 ⊕ E0)이다. 1, 2 round와 똑 같은 방법으로 A0, B0, C0가 주어지면 임의의 W3~W15에 대하여 input X = (A0, B0, C0, D0, E0, F0, G0, H0, W0, W1, W2, ~W15)를 구할 수 있다.

총 232 \* 232 \* 232 = 296 번의 hash operation이 필요하다.

4 round: Y = (A0 ⊕ E0 ⊕ T3, B0 ⊕ F0 ⊕ T2, C0 ⊕ G0 ⊕ T1, D0 ⊕ H0 ⊕ T0, E0 ⊕ A0, F0 ⊕ B0, G0 ⊕ C0, H0 ⊕ D0)이다. 1, 2 round와 똑 같은 방법으로 A0, B0, C0, D0가 주어지면 임의의 W4~W15에 대하여 input X = (A0, B0, C0, D0, E0, F0, G0, H0, W0, W1, W2, W3 ~W15)를 구할 수 있다.

총 232 \* 232 \* 232 \* 232 = 2128 번의 hash operation이 필요하다.

5 round: Y = (A0 ⊕ D0 ⊕ T4, B0 ⊕ E0 ⊕ T3, C0 ⊕ F0 ⊕ T2, D0 ⊕ G0 ⊕ T1, E0 ⊕ H0 ⊕ T0, F0 ⊕ A0, G0 ⊕ B0, H0 ⊕ C0)이다. 1, 2 round와 똑 같은 방법으로 A0, B0, C0, D0, E0가 주어지면 임의의 W5~W15에 대하여 input X = (A0, B0, C0, D0, E0, F0, G0, H0, W0, W1, W2, W3, W4 ~W15)를 구할 수 있다.

총 232 \* 232 \* 232 \* 232 \* 232 = 2160 번의 hash operation이 필요하다.

6 round: Y = (A0 ⊕ C0 ⊕ T5, B0 ⊕ D0 ⊕ T4, C0 ⊕ E0 ⊕ T3, D0 ⊕ F0 ⊕ T2, E0 ⊕ G0 ⊕ T1, F0 ⊕ H0 ⊕ T0, G0 ⊕ A0, H0 ⊕ B0)이다. 1, 2 round와 똑 같은 방법으로 A0, B0, C0, D0, E0, F0가 주어지면 임의의 W6~W15에 대하여 input X = (A0, B0, C0, D0, E0, F0, G0, H0, W0, W1, W2, W3, W4, W5 ~W15)를 구할 수 있다.

총 232 \* 232 \* 232 \* 232 \* 232 \* 232 = 2192 번의 hash operation이 필요하다.

6 round: Y = (A0 ⊕ C0 ⊕ T5, B0 ⊕ D0 ⊕ T4, C0 ⊕ E0 ⊕ T3, D0 ⊕ F0 ⊕ T2, E0 ⊕ G0 ⊕ T1, F0 ⊕ H0 ⊕ T0, G0 ⊕ A0, H0 ⊕ B0)이다. 1, 2 round와 똑 같은 방법으로 A0, B0, C0, D0, E0, F0가 주어지면 임의의 W6~W15에 대하여 input X = (A0, B0, C0, D0, E0, F0, G0, H0, W0, W1, W2, W3, W4, W5 ~W15)를 구할 수 있다.

총 232 \* 232 \* 232 \* 232 \* 232 \* 232 = 2192 번의 hash operation이 필요하다.

7 round: Y = (A0 ⊕ B0 ⊕ T6, B0 ⊕ C0 ⊕ T5, C0 ⊕ D0 ⊕ T4, D0 ⊕ E0 ⊕ T3, E0 ⊕ F0 ⊕ T2, F0 ⊕ G0 ⊕ T1, G0 ⊕ H0 ⊕ T0, H0 ⊕ A0)이다. 1, 2 round와 똑 같은 방법으로 A0, B0, C0, D0, E0, F0, G0가 주어지면 임의의 W7~W15에 대하여 input X = (A0, B0, C0, D0, E0, F0, G0, H0, W0, W1, W2, W3, W4, W5, W6, ~W15)를 구할 수 있다.

총 232 \* 232 \* 232 \* 232 \* 232 \* 232 \* 232 = 2224 번의 hash operation이 필요하다.

**8 Round.**

8 round 부터는 A0~H0 모두를 알아야 하므로 2256 번의 계산이 필요하므로 최대 round는 7이다.

Source Code:

import random  
import numpy as np  
import itertools  
from copy import copy  
  
Sbox = np.array(  
 [[0x63, 0x7C, 0x77, 0x7B, 0xF2, 0x6B, 0x6F, 0xC5, 0x30, 0x01, 0x67, 0x2B, 0xFE, 0xD7, 0xAB, 0x76],  
 [0xCA, 0x82, 0xC9, 0x7D, 0xFA, 0x59, 0x47, 0xF0, 0xAD, 0xD4, 0xA2, 0xAF, 0x9C, 0xA4, 0x72, 0xC0],  
 [0xB7, 0xFD, 0x93, 0x26, 0x36, 0x3F, 0xF7, 0xCC, 0x34, 0xA5, 0xE5, 0xF1, 0x71, 0xD8, 0x31, 0x15],  
 [0x04, 0xC7, 0x23, 0xC3, 0x18, 0x96, 0x05, 0x9A, 0x07, 0x12, 0x80, 0xE2, 0xEB, 0x27, 0xB2, 0x75],  
 [0x09, 0x83, 0x2C, 0x1A, 0x1B, 0x6E, 0x5A, 0xA0, 0x52, 0x3B, 0xD6, 0xB3, 0x29, 0xE3, 0x2F, 0x84],  
 [0x53, 0xD1, 0x00, 0xED, 0x20, 0xFC, 0xB1, 0x5B, 0x6A, 0xCB, 0xBE, 0x39, 0x4A, 0x4C, 0x58, 0xCF],  
 [0xD0, 0xEF, 0xAA, 0xFB, 0x43, 0x4D, 0x33, 0x85, 0x45, 0xF9, 0x02, 0x7F, 0x50, 0x3C, 0x9F, 0xA8],  
 [0x51, 0xA3, 0x40, 0x8F, 0x92, 0x9D, 0x38, 0xF5, 0xBC, 0xB6, 0xDA, 0x21, 0x10, 0xFF, 0xF3, 0xD2],  
 [0xCD, 0x0C, 0x13, 0xEC, 0x5F, 0x97, 0x44, 0x17, 0xC4, 0xA7, 0x7E, 0x3D, 0x64, 0x5D, 0x19, 0x73],  
 [0x60, 0x81, 0x4F, 0xDC, 0x22, 0x2A, 0x90, 0x88, 0x46, 0xEE, 0xB8, 0x14, 0xDE, 0x5E, 0x0B, 0xDB],  
 [0xE0, 0x32, 0x3A, 0x0A, 0x49, 0x06, 0x24, 0x5C, 0xC2, 0xD3, 0xAC, 0x62, 0x91, 0x95, 0xE4, 0x79],  
 [0xE7, 0xC8, 0x37, 0x6D, 0x8D, 0xD5, 0x4E, 0xA9, 0x6C, 0x56, 0xF4, 0xEA, 0x65, 0x7A, 0xAE, 0x08],  
 [0xBA, 0x78, 0x25, 0x2E, 0x1C, 0xA6, 0xB4, 0xC6, 0xE8, 0xDD, 0x74, 0x1F, 0x4B, 0xBD, 0x8B, 0x8A],  
 [0x70, 0x3E, 0xB5, 0x66, 0x48, 0x03, 0xF6, 0x0E, 0x61, 0x35, 0x57, 0xB9, 0x86, 0xC1, 0x1D, 0x9E],  
 [0xE1, 0xF8, 0x98, 0x11, 0x69, 0xD9, 0x8E, 0x94, 0x9B, 0x1E, 0x87, 0xE9, 0xCE, 0x55, 0x28, 0xDF],  
 [0x8C, 0xA1, 0x89, 0x0D, 0xBF, 0xE6, 0x42, 0x68, 0x41, 0x99, 0x2D, 0x0F, 0xB0, 0x54, 0xBB, 0x16]]  
)  
  
SboxInv = np.array([  
 [0x52, 0x09, 0x6a, 0xd5, 0x30, 0x36, 0xa5, 0x38, 0xbf, 0x40, 0xa3, 0x9e, 0x81, 0xf3, 0xd7, 0xfb],  
 [0x7c, 0xe3, 0x39, 0x82, 0x9b, 0x2f, 0xff, 0x87, 0x34, 0x8e, 0x43, 0x44, 0xc4, 0xde, 0xe9, 0xcb],  
 [0x54, 0x7b, 0x94, 0x32, 0xa6, 0xc2, 0x23, 0x3d, 0xee, 0x4c, 0x95, 0x0b, 0x42, 0xfa, 0xc3, 0x4e],  
 [0x08, 0x2e, 0xa1, 0x66, 0x28, 0xd9, 0x24, 0xb2, 0x76, 0x5b, 0xa2, 0x49, 0x6d, 0x8b, 0xd1, 0x25],  
 [0x72, 0xf8, 0xf6, 0x64, 0x86, 0x68, 0x98, 0x16, 0xd4, 0xa4, 0x5c, 0xcc, 0x5d, 0x65, 0xb6, 0x92],  
 [0x6c, 0x70, 0x48, 0x50, 0xfd, 0xed, 0xb9, 0xda, 0x5e, 0x15, 0x46, 0x57, 0xa7, 0x8d, 0x9d, 0x84],  
 [0x90, 0xd8, 0xab, 0x00, 0x8c, 0xbc, 0xd3, 0x0a, 0xf7, 0xe4, 0x58, 0x05, 0xb8, 0xb3, 0x45, 0x06],  
 [0xd0, 0x2c, 0x1e, 0x8f, 0xca, 0x3f, 0x0f, 0x02, 0xc1, 0xaf, 0xbd, 0x03, 0x01, 0x13, 0x8a, 0x6b],  
 [0x3a, 0x91, 0x11, 0x41, 0x4f, 0x67, 0xdc, 0xea, 0x97, 0xf2, 0xcf, 0xce, 0xf0, 0xb4, 0xe6, 0x73],  
 [0x96, 0xac, 0x74, 0x22, 0xe7, 0xad, 0x35, 0x85, 0xe2, 0xf9, 0x37, 0xe8, 0x1c, 0x75, 0xdf, 0x6e],  
 [0x47, 0xf1, 0x1a, 0x71, 0x1d, 0x29, 0xc5, 0x89, 0x6f, 0xb7, 0x62, 0x0e, 0xaa, 0x18, 0xbe, 0x1b],  
 [0xfc, 0x56, 0x3e, 0x4b, 0xc6, 0xd2, 0x79, 0x20, 0x9a, 0xdb, 0xc0, 0xfe, 0x78, 0xcd, 0x5a, 0xf4],  
 [0x1f, 0xdd, 0xa8, 0x33, 0x88, 0x07, 0xc7, 0x31, 0xb1, 0x12, 0x10, 0x59, 0x27, 0x80, 0xec, 0x5f],  
 [0x60, 0x51, 0x7f, 0xa9, 0x19, 0xb5, 0x4a, 0x0d, 0x2d, 0xe5, 0x7a, 0x9f, 0x93, 0xc9, 0x9c, 0xef],  
 [0xa0, 0xe0, 0x3b, 0x4d, 0xae, 0x2a, 0xf5, 0xb0, 0xc8, 0xeb, 0xbb, 0x3c, 0x83, 0x53, 0x99, 0x61],  
 [0x17, 0x2b, 0x04, 0x7e, 0xba, 0x77, 0xd6, 0x26, 0xe1, 0x69, 0x14, 0x63, 0x55, 0x21, 0x0c, 0x7d]  
 ])  
  
#bit 로테이션을 위한 함수  
def rotl(val, r\_bits):  
 bin\_val = bin(val)  
 max\_bits = len(bin\_val)-2  
 real\_rotation = r\_bits%max\_bits  
 right = bin\_val[2:real\_rotation+2]  
 left = bin\_val[-(max\_bits-r\_bits):]  
 return int(left+right,2)  
  
#output binary를 num개의 블록으로 나눠주는 함수  
def slice\_output(output, max\_len, num):  
 output\_bin = bin(output)  
 output\_bin = output\_bin[2:]  
 while len(output\_bin) != max\_len:  
 output\_bin = '0'+output\_bin  
 sliced\_list = []  
 #print(output\_bin)  
 dif = int(max\_len/num)  
 for k in range(0,len(output\_bin),dif):  
 sliced\_list.append(int('0b'+output\_bin[k:k+dif],2))  
 #print(sliced\_list)  
 return sliced\_list  
  
#slice된 수를 다시 1개의 integer로 합쳐주는 함수  
def concat\_output(output, current\_len):  
 num = len(output)  
 result = ''  
 for n in range(num):  
 out\_bin = bin(output[n])  
 out\_bin = out\_bin[2:]  
 while len(out\_bin) != current\_len:  
 out\_bin = '0'+out\_bin  
 result += out\_bin  
 result = int('0b'+result,2)  
 return result  
  
  
def sub\_byte(input):  
 input\_sliced = slice\_output(input, 8, 2)  
 return Sbox[input\_sliced[0]][input\_sliced[1]]  
  
  
def sub\_byte\_inverse(input):  
 input\_sliced = slice\_output(input, 8, 2)  
 return SboxInv[input\_sliced[0]][input\_sliced[1]]  
  
#GF(2^8)을 통한 modulo reduction  
def mod\_reduction(input):  
 if len(bin(input))>8+2:  
 input ^= 0b00011011  
 input = int('0b'+bin(input)[-8:],2)  
 return input  
  
def MDS(input\_sliced):  
 output = [0,0,0,0]  
 output[0] = mod\_reduction(input\_sliced[0]<<1) ^ mod\_reduction(input\_sliced[1] ^ mod\_reduction(input\_sliced[1]<<1)) \  
 ^ input\_sliced[2] ^ input\_sliced[3]  
 output[1] = mod\_reduction(input\_sliced[1]<<1) ^ mod\_reduction(input\_sliced[2] ^ mod\_reduction(input\_sliced[2]<<1)) \  
 ^ input\_sliced[0] ^ input\_sliced[3]  
 output[2] = mod\_reduction(input\_sliced[2]<<1) ^ mod\_reduction(input\_sliced[3] ^ mod\_reduction(input\_sliced[3]<<1)) \  
 ^ input\_sliced[0] ^ input\_sliced[1]  
 output[3] = mod\_reduction(input\_sliced[3]<<1) ^ mod\_reduction(input\_sliced[0] ^ mod\_reduction(input\_sliced[0]<<1)) \  
 ^ input\_sliced[1] ^ input\_sliced[2]  
 return output  
  
#GF2^8) multiplication  
def GMul (a, b):  
 p = 0  
 for counter in range(8):  
 if b&1 != 0:  
 p^=a  
  
 hi\_bit\_set = (a & 0x80) != 0  
 a<<=1  
 if hi\_bit\_set:  
 a ^= 0x1B  
 b>>=1  
 return p % 256  
  
def MDS\_inverse(input\_sliced):  
 output = [0,0,0,0]  
 output[0] = GMul(int("0x0E",16),input\_sliced[0]) ^ GMul(int("0x0B",16), input\_sliced[1]) ^ \  
 GMul(int("0x0D",16),input\_sliced[2]) ^ GMul(int("0x09",16),input\_sliced[3])  
 output[1] = GMul(int("0x0E",16),input\_sliced[1]) ^ GMul(int("0x0B",16), input\_sliced[2]) ^ \  
 GMul(int("0x0D",16),input\_sliced[3]) ^ GMul(int("0x09",16),input\_sliced[0])  
 output[2] = GMul(int("0x0E",16),input\_sliced[2]) ^ GMul(int("0x0B",16), input\_sliced[3]) ^ \  
 GMul(int("0x0D",16),input\_sliced[0]) ^ GMul(int("0x09",16),input\_sliced[1])  
 output[3] = GMul(int("0x0E",16),input\_sliced[3]) ^ GMul(int("0x0B",16), input\_sliced[0]) ^ \  
 GMul(int("0x0D",16),input\_sliced[1]) ^ GMul(int("0x09",16),input\_sliced[2])  
 return output  
  
#최종 hash 함수  
def hash(sliced\_in, round):  
 global W\_array  
 result = copy(sliced\_in)  
 out = np.zeros(8)  
 for r in range(round):  
 sliced\_sliced\_in = [slice\_output(num, 32, 4) for num in result]  
 tmp\_for = copy(result)  
 T = slice\_output(W\_array[r], 32, 4)  
  
 T = MDS([sub\_byte(T[0]^sliced\_sliced\_in[0][0]), sub\_byte(T[1]^sliced\_sliced\_in[0][1]), sub\_byte(T[2]^sliced\_sliced\_in[0][2]), sub\_byte(T[3]^sliced\_sliced\_in[0][3])])  
 T = MDS([sub\_byte(T[0]^sliced\_sliced\_in[1][0]), sub\_byte(T[1]^sliced\_sliced\_in[1][1]), sub\_byte(T[2]^sliced\_sliced\_in[1][2]), sub\_byte(T[3]^sliced\_sliced\_in[1][3])])  
 T = MDS([sub\_byte(T[0]^sliced\_sliced\_in[2][0]), sub\_byte(T[1]^sliced\_sliced\_in[2][1]), sub\_byte(T[2]^sliced\_sliced\_in[2][2]), sub\_byte(T[3]^sliced\_sliced\_in[2][3])])  
 T = MDS([sub\_byte(T[0]^sliced\_sliced\_in[3][0]), sub\_byte(T[1]^sliced\_sliced\_in[3][1]), sub\_byte(T[2]^sliced\_sliced\_in[3][2]), sub\_byte(T[3]^sliced\_sliced\_in[3][3])])  
 T = MDS([sub\_byte(T[0]^sliced\_sliced\_in[4][0]), sub\_byte(T[1]^sliced\_sliced\_in[4][1]), sub\_byte(T[2]^sliced\_sliced\_in[4][2]), sub\_byte(T[3]^sliced\_sliced\_in[4][3])])  
 T = MDS([sub\_byte(T[0]^sliced\_sliced\_in[5][0]), sub\_byte(T[1]^sliced\_sliced\_in[5][1]), sub\_byte(T[2]^sliced\_sliced\_in[5][2]), sub\_byte(T[3]^sliced\_sliced\_in[5][3])])  
 T = MDS([sub\_byte(T[0]^sliced\_sliced\_in[6][0]), sub\_byte(T[1]^sliced\_sliced\_in[6][1]), sub\_byte(T[2]^sliced\_sliced\_in[6][2]), sub\_byte(T[3]^sliced\_sliced\_in[6][3])])  
 T\_concat = concat\_output(T,8)  
 result[0] = tmp\_for[7] ^ T\_concat  
 for n in range(1,8):  
 result[n] = tmp\_for[n-1]  
 sliced\_in = np.array(sliced\_in)  
 result = np.array(result)  
 result = sliced\_in ^ result  
 return result  
  
#slice된 수를 binary 혹은 hex 형태의 string으로 치환해주는 함수  
def sliced\_to\_bin(sliced\_in, hexOrbin):  
 res = ""  
 if hexOrbin == 0:  
 for slice in sliced\_in:  
 tmp = hex(slice)  
 tmp = tmp[2:]  
 while len(tmp) != 8:  
 tmp = "0"+tmp  
 res += tmp  
 else:  
 for slice in sliced\_in:  
 tmp = bin(slice)  
 tmp = tmp[2:]  
 res += tmp  
 return res  
  
def append\_W\_array():  
 global W\_array  
 length = len(W\_array)  
 if len(W\_array) >16:  
 W\_array.append(rotl(W\_array[i - 3], 1) ^ rotl(W\_array[i - 8], 6) ^ rotl(W\_array[i - 14], 11) ^ W\_array[i - 16])  
Y = random.randint(0,2\*\*256)  
Y\_sliced = slice\_output(Y,256,8)  
maxR = 32  
  
W = random.randint(0,2\*\*32)  
W\_array = []  
for k in range(16):  
 W\_array.append(random.randint(0,2\*\*32))  
  
for i in range(16,maxR):  
 W\_array.append(rotl(W\_array[i-3],1) ^ rotl(W\_array[i-8],6) ^ rotl(W\_array[i-14],11) ^ W\_array[i-16])  
np.array(W\_array)  
bin\_vectorized = np.vectorize(lambda i: bin(i))  
  
W\_bin\_array = bin\_vectorized(W\_array)  
  
  
print("--------------1 Round--------------")  
  
X = np.zeros(24).astype(np.int64)  
for k in range(0,2\*\*16):  
 A\_array = [random.randint(0,2\*\*32) for i in range(2\*\*16\*k, 2\*\*16\*(k+1))]  
 for A in A\_array:  
 #B0 = B1 XOR A0  
 X[0] = A  
 #B0 = Y[1] XOR A0, --- , H0 = Y[7] XOR G0  
 for n in range(1,8):  
 X[n] = Y\_sliced[n] ^ X[n-1]  
 #W0 찾기  
 T0 = X[0] ^ Y\_sliced[0] ^ X[7]  
 W = slice\_output(T0, 32, 4)  
 for i in range(6,-1, -1):  
 W = MDS\_inverse(W) ^ X[i]  
 W\_array[0] = concat\_output(W,8)  
 res = hash(X[:8], 1)  
 #A0를 통해 얻은 hash 값과 Y 비교  
 if list(res) == Y\_sliced:  
 print("Answer X-> ")  
 print(X)  
 print("Answer X\_hex")  
 print(sliced\_to\_bin(X,0))  
 print("Answer X\_bin")  
 print(sliced\_to\_bin(X,1))  
 break  
 print("finished")