Tristan Erney

October 13th, 2021

Intro to Cryptology

Hands On Exercise 6 – DES

Starting Plaintext :: 123456789abcdef

Starting Key :: 123456789abcdef

Generating keys for 16 rounds...

|- keygen round 1

|- C :: e199541

|- D :: 5599e01

|- subK :: e1995415599e01

|- key :: b02679b49a5

|- keygen round 2

|- C :: c332a83

|- D :: ab33c02

|- subK :: c332a83ab33c02

|- key :: 69a659256a26

|- keygen round 3

|- C :: ccaa0f

|- D :: accf00a

|- subK :: ccaa0faccf00a

|- key :: 45d48ab428d2

|- keygen round 4

|- C :: 332a83c

|- D :: b33c02a

|- subK :: 332a83cb33c02a

|- key :: 7289d2a58257

|- keygen round 5

|- C :: ccaa0f0

|- D :: ccf00aa

|- subK :: ccaa0f0ccf00aa

|- key :: 3ce80317a6c2

|- keygen round 6

|- C :: 32a83c3

|- D :: 33c02ab

|- subK :: 32a83c333c02ab

|- key :: 23251e3c8545

|- keygen round 7

|- C :: caa0f0c

|- D :: cf00aac

|- subK :: caa0f0ccf00aac

|- key :: 6c04950ae4c6

|- keygen round 8

|- C :: 2a83c33

|- D :: 3c02ab3

|- subK :: 2a83c333c02ab3

|- key :: 5788386ce581

|- keygen round 9

|- C :: 5507866

|- D :: 7805566

|- subK :: 55078667805566

|- key :: c0c9e926b839

|- keygen round 10

|- C :: 541e199

|- D :: e015599

|- subK :: 541e199e015599

|- key :: 91e307631d72

|- keygen round 11

|- C :: 5078665

|- D :: 8055667

|- subK :: 50786658055667

|- key :: 211f830d893a

|- keygen round 12

|- C :: 41e1995

|- D :: 15599e

|- subK :: 41e1995015599e

|- key :: 7130e5455c54

|- keygen round 13

|- C :: 786655

|- D :: 556678

|- subK :: 7866550556678

|- key :: 91c4d04980fc

|- keygen round 14

|- C :: 1e19954

|- D :: 15599e0

|- subK :: 1e1995415599e0

|- key :: 5443b681dc8d

|- keygen round 15

|- C :: 7866550

|- D :: 5566780

|- subK :: 78665505566780

|- key :: b691050a16b5

|- keygen round 16

|- C :: f0ccaa0

|- D :: aaccf00

|- subK :: f0ccaa0aaccf00

|- key :: ca3d03b87032

Initial L and R values...

L0 :: cc00ccff

R0 :: f0aaf0aa

Round 1

E( f0aaf0aa ) :: 7a15557a1555

7a15557a1555 ^ b02679b49a5 :: 711732e15cf0

F( f0aaf0aa, b02679b49a5 ) :: 921c209c

L1 :: f0aaf0aa

R1 :: 5e1cec63

Round 2

E( 5e1cec63 ) :: afc0f9758306

afc0f9758306 ^ 69a659256a26 :: c666a050e920

F( 5e1cec63, 69a659256a26 ) :: 724bcce3

L2 :: 5e1cec63

R2 :: 82e13c49

Round 3

E( 82e13c49 ) :: c057029f8253

c057029f8253 ^ 45d48ab428d2 :: 8583882baa81

F( 82e13c49, 45d48ab428d2 ) :: 1789ae9a

L3 :: 82e13c49

R3 :: 499542f9

Round 4

E( 499542f9 ) :: a53caaa057f2

a53caaa057f2 ^ 7289d2a58257 :: d7b57805d5a5

F( 499542f9, 7289d2a58257 ) :: 8f3776b2

L4 :: 499542f9

R4 :: dd64afb

Round 5

E( dd64afb ) :: 85beac2557f6

85beac2557f6 ^ 3ce80317a6c2 :: b956af32f134

F( dd64afb, 3ce80317a6c2 ) :: 39a346c2

L5 :: dd64afb

R5 :: 7036043b

Round 6

E( 7036043b ) :: ba01ac0081f6

ba01ac0081f6 ^ 23251e3c8545 :: 9924b23c04b3

F( 7036043b, 23251e3c8545 ) :: fc914139

L6 :: 7036043b

R6 :: f1470bc2

Round 7

E( f1470bc2 ) :: 7a2a0e857e05

7a2a0e857e05 ^ 6c04950ae4c6 :: 162e9b8f9ac3

F( f1470bc2, 6c04950ae4c6 ) :: 497a8b7e

L7 :: f1470bc2

R7 :: 394c8f45

Round 8

E( 394c8f45 ) :: 9f2a5945ea0a

9f2a5945ea0a ^ 5788386ce581 :: c8a261290f8b

F( 394c8f45, 5788386ce581 ) :: c5cacc84

L8 :: 394c8f45

R8 :: 348dc746

Round 9

E( 348dc746 ) :: 1a945be0ea0c

1a945be0ea0c ^ c0c9e926b839 :: da5db2c65235

F( 348dc746, c0c9e926b839 ) :: ca3d8f83

L9 :: 348dc746

R9 :: f37100c6

Round 10

E( f37100c6 ) :: 7a6ba280160d

7a6ba280160d ^ 91e307631d72 :: eb88a5e30b7f

F( f37100c6, 91e307631d72 ) :: 8af6e8d

L10 :: f37100c6

R10 :: 3c22a9cb

Round 11

E( 3c22a9cb ) :: 9f8105553e56

9f8105553e56 ^ 211f830d893a :: be9e8658b76c

F( 3c22a9cb, 211f830d893a ) :: f946c3af

L11 :: 3c22a9cb

R11 :: a37c369

Round 12

E( a37c369 ) :: 8541afe06b52

8541afe06b52 ^ 7130e5455c54 :: f4714aa53706

F( a37c369, 7130e5455c54 ) :: 6050f630

L12 :: a37c369

R12 :: 5c725ffb

Round 13

E( 5c725ffb ) :: af83a42ffff6

af83a42ffff6 ^ 91c4d04980fc :: 3e4774667f0a

F( 5c725ffb, 91c4d04980fc ) :: fe4349bf

L13 :: 5c725ffb

R13 :: f4748ad6

Round 14

E( f4748ad6 ) :: 7a83a94556ad

7a83a94556ad ^ 5443b681dc8d :: 2ec01fc48a20

F( f4748ad6, 5443b681dc8d ) :: 905e7bb5

L14 :: f4748ad6

R14 :: cc2c244e

Round 15

E( cc2c244e ) :: 65815810825d

65815810825d ^ b691050a16b5 :: d3105d1a94e8

F( cc2c244e, b691050a16b5 ) :: 7ec97d42

L15 :: cc2c244e

R15 :: 8abdf794

Round 16

E( 8abdf794 ) :: 4555fbfafca9

4555fbfafca9 ^ ca3d03b87032 :: 8f68f8428c9b

F( 8abdf794, ca3d03b87032 ) :: cbb29535

L16 :: 8abdf794

R16 :: 79eb17b

Ending Ciphertext :: 6dd97ab13f2d09be

DES.cpp

#include <iostream>

#include <iomanip>

#include <bitset>

#include <stdlib.h>

#include <stdint.h>

template <uint64\_t I>

std::string printb(uint64\_t b) {

std::bitset<I> x(b);

return x.to\_string();

}

// S-Boxes 1-8

uint8\_t sboxes[][64] = {

{

14, 4, 13, 1, 2, 15, 11, 8, 3, 10, 6, 12, 5, 9, 0, 7,

0, 15, 7, 4, 14, 2, 13, 1, 10, 6, 12, 11, 9, 5, 3, 8,

4, 1, 14, 8, 13, 6, 2, 11, 15, 12, 9, 7, 3, 10, 5, 0,

15, 12, 8, 2, 4, 9, 1, 7, 5, 11, 3, 14, 10, 0, 6, 13

},

{

15, 1, 8, 14, 6, 11, 3, 4, 9, 7, 2, 13, 12, 0, 5, 10,

3, 13, 4, 7, 15, 2, 8, 14, 12, 0, 1, 10, 6, 9, 11, 5,

0, 14, 7, 11, 10, 4, 13, 1, 5, 8, 12, 6, 9, 3, 2, 15,

13, 8, 10, 1, 3, 15, 4, 2, 11, 6, 7, 12, 0, 5, 14, 9

},

{

10, 0, 9, 14, 6, 3, 15, 5, 1, 13, 12, 7, 11, 4, 2, 8,

13, 7, 0, 9, 3, 4, 6, 10, 2, 8, 5, 14, 12, 11, 15, 1,

13, 6, 4, 9, 8, 15, 3, 0, 11, 1, 2, 12, 5, 10, 14, 7,

1, 10, 13, 0, 6, 9, 8, 7, 4, 15, 14, 3, 11, 5, 2, 12

},

{

7, 13, 14, 3, 0, 6, 9, 10, 1, 2, 8, 5, 11, 12, 4, 15,

13, 8, 11, 5, 6, 15, 0, 3, 4, 7, 2, 12, 1, 10, 14, 7,

10, 6, 9, 0, 12, 11, 7, 13, 15, 1, 3, 14, 5, 2, 8, 4,

3, 15, 0, 6, 10, 1, 13, 8, 9, 4, 5, 11, 12, 7, 2, 14

},

{

2, 12, 4, 1, 7, 10, 11, 6, 8, 5, 3, 15, 13, 0, 14, 9,

14, 11, 2, 12, 4, 7, 13, 1, 5, 0, 15, 10, 3, 9, 8, 6,

4, 2, 1, 11, 10, 13, 7, 8, 15, 9, 12, 5, 6, 3, 0, 14,

11, 8, 12, 7, 1, 14, 2, 13, 6, 15, 0, 9, 10, 4, 5, 3

},

{

12, 1, 10, 15, 9, 2, 6, 8, 0, 13, 3, 4, 14, 7, 5, 11,

10, 15, 4, 2, 7, 12, 9, 5, 6, 1, 13, 14, 0, 11, 3, 8,

9, 14, 15, 5, 2, 8, 12, 3, 7, 0, 4, 10, 1, 13, 11, 6,

4, 3, 2, 12, 9, 5, 15, 10, 11, 14, 1, 7, 6, 0, 8, 13

},

{

4, 11, 2, 14, 15, 0, 8, 13, 3, 12, 9, 7, 5, 10, 6, 1,

13, 0, 11, 7, 4, 9, 1, 10, 14, 3, 5, 12, 2, 15, 8, 6,

1, 4, 11, 13, 12, 3, 7, 14, 10, 15, 6, 8, 0, 5, 9, 2,

6, 11, 13, 8, 1, 4, 10, 7, 9, 5, 0, 15, 14, 2, 3, 12

},

{

13, 2, 8, 4, 6, 15, 11, 1, 10, 9, 3, 14, 5, 0, 12, 7,

1, 15, 13, 8, 10, 3, 7, 4, 12, 5, 6, 11, 0, 14, 9, 2,

7, 11, 4, 1, 9, 12, 14, 2, 0, 6, 10, 13, 15, 3, 5, 8,

2, 1, 14, 7, 4, 10, 8, 13, 15, 12, 9, 0, 3, 5, 6, 11

}

};

// Permutations

uint8\_t ip[] = {

58, 50, 42, 34, 26, 18, 10, 2,

60, 52, 44, 36, 28, 20, 12, 4,

62, 54, 46, 38, 30, 22, 14, 6,

64, 56, 48, 40, 32, 24, 16, 8,

57, 49, 41, 33, 25, 17, 9, 1,

59, 51, 43, 35, 27, 19, 11, 3,

61, 53, 45, 37, 29, 21, 13, 5,

63, 55, 47, 39, 31, 23, 15, 7

};

uint8\_t fp[] = {

40, 8, 48, 16, 56, 24, 64, 32,

39, 7, 47, 15, 55, 23, 63, 31,

38, 6, 46, 14, 54, 22, 62, 30,

37, 5, 45, 13, 53, 21, 61, 29,

36, 4, 44, 12, 52, 20, 60, 28,

35, 3, 43, 11, 51, 19, 59, 27,

34, 2, 42, 10, 50, 18, 58, 26,

33, 1, 41, 9, 49, 17, 57, 25

};

uint8\_t p[] = {

16, 7, 20, 21, 29, 12, 28, 17,

1, 15, 23, 26, 5, 18, 31, 10,

2, 8, 24, 14, 32, 27, 3, 9,

19, 13, 30, 6, 22, 11, 4, 25

};

uint8\_t pc1[] = { // LEFT

57, 49, 41, 33, 25, 17, 9,

1, 58, 50, 42, 34, 26, 18,

10, 2, 59, 51, 43, 35, 27,

19, 11, 3, 60, 52, 44, 36,

// RIGHT

63, 55, 47, 39, 31, 23, 15,

7, 62, 54, 46, 38, 30, 22,

14, 6, 61, 53, 45, 37, 29,

21, 13, 5, 28, 20, 12, 4

};

uint8\_t pc2[] = {

14, 17, 11, 24, 1, 5,

3, 28, 15, 6, 21, 10,

23, 19, 12, 4, 26, 8,

16, 7, 27, 20, 13, 2,

41, 52, 31, 37, 47, 55,

30, 40, 51, 45, 33, 48,

44, 49, 39, 56, 34, 53,

46, 42, 50, 36, 29, 32

};

uint64\_t E(uint32\_t x) {

uint8\_t ep[] = {

31, 0, 1, 2, 3, 4,

3, 4, 5, 6, 7, 8,

7, 8, 9, 10, 11, 12,

11, 12, 13, 14, 15, 16,

15, 16, 17, 18, 19, 20,

19, 20, 21, 22, 23, 24,

23, 24, 25, 26, 27, 28,

27, 28, 29, 30, 31, 0

};

uint64\_t e = 0;

for (int i = 0; i < 48; i += 1) {

e <<= 1;

e |= 0x01 & (x >> (31 - ep[i]));

}

return e;

}

uint64\_t F(uint32\_t R, uint64\_t K) {

uint64\_t eR = E(R);

uint64\_t eR\_xor\_K = eR ^ K;

std::cout << "\t\tE( " << std::hex << R << " ) :: " << eR << "\n";

std::cout << "\t\t" << eR << " ^ " << K << " :: " << eR\_xor\_K << "\n";

const uint8\_t RMASK = 0x21;

const uint8\_t CMASK = 0x1e;

uint32\_t sbox\_res = 0;

for (uint8\_t i = 0; i < 8; i += 1) {

// get bit 0 and 5 from each block of eR\_xor\_K

uint8\_t block = (uint8\_t) ((eR\_xor\_K >> 42 - (i \* 6)));

uint8\_t row = (uint8\_t) (RMASK & block);

row = ((0x20 & row) >> 4) | (0x01 & row);

uint8\_t col = (uint8\_t) (CMASK & block);

col = col >> 1;

sbox\_res <<= 4;

sbox\_res |= 0xf & sboxes[i][row \* 16 + col];

}

// permute with p table

uint32\_t res = 0;

for (int i = 0; i < 32; i += 1) {

res <<= 1;

res |= 0x01 & (sbox\_res >> (32 - p[i]));

}

std::cout << "\t\tF( " << R << ", " << K << " ) :: " << res << std::dec << "\n";

return res;

}

uint64\_t\* key\_gen(uint64\_t K) {

uint64\_t\* keys = (uint64\_t\*)malloc(sizeof(uint64\_t) \* 16);

uint8\_t K\_shift\_schedule[] = {

1, 1, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 1

};

std::cout << "Generating keys for 16 rounds...\n";

uint64\_t sK = 0;

uint32\_t C = 0, D = 0;

for (uint8\_t i = 0; i < 56; i += 1) {

sK <<= 1;

sK |= 0x01 & (K >> 64 - pc1[i]);

}

C = (uint32\_t)(0x000000000fffffff & (sK >> 28));

D = (uint32\_t)(0x000000000fffffff & sK);

// generate keys

for (uint8\_t round = 0; round < 16; round += 1) {

// permute key with PC1

std::cout << "|- keygen round " << std::dec << round + 1 << "\n";

{

uint8\_t shift = K\_shift\_schedule[round];

C = 0x0fffffff & ( (C << shift) | (C >> 28 - shift) );

D = 0x0fffffff & ( (D << shift) | (D >> 28 - shift) );

uint64\_t subK = 0;

subK = ((uint64\_t)C << 28) | (uint64\_t)(D);

// PC2

for (uint64\_t i = 0; i < 48; i += 1) {

keys[round] <<= 1;

keys[round] |= 0x01 & (subK >> (56 - pc2[i]));

}

std::cout << std::hex <<

" |- C :: " << C << "\n" <<

" |- D :: " << D << "\n" <<

" |- subK :: " << subK << "\n";

}

std::cout << " |- key :: " << std::hex << keys[round] << "\n";

}

std::cout << std::dec << "\n";

return keys;

}

uint64\_t des\_encrypt(uint64\_t P, uint64\_t\* Ks) {

uint64\_t cipher = 0;

// initial permutation

uint64\_t permute\_P = 0;

for (uint64\_t i = 0; i < 64; i += 1) {

permute\_P <<= 1;

permute\_P |= 0x01 & (P >> (64 - ip[i]));

}

uint32\_t L = (uint32\_t)(permute\_P >> 32);

uint32\_t R = (uint32\_t)(0x00000000ffffffff & permute\_P);

std::cout << "Initial L and R values...\n";

std::cout << "L0 :: " << std::hex << L << "\n";

std::cout << "R0 :: " << std::hex << R << std::dec << "\n\n";

for (uint32\_t i = 0; i < 16; i += 1) {

std::cout << "Round " << i + 1 << "\n";

uint32\_t L\_ = R;

uint32\_t R\_ = L ^ F(R, Ks[i]);

std::cout << "\tL" << i + 1 << " :: " << std::hex << L\_ << std::dec << "\n";

std::cout << "\tR" << i + 1 << " :: " << std::hex << R\_ << std::dec << "\n";

L = L\_;

R = R\_;

}

std::cout << "\n";

uint32\_t tmp = L;

L = R;

R = tmp;

cipher = ((uint64\_t)L << 32) | (uint64\_t)R;

uint64\_t fp\_cipher = 0;

for (uint8\_t i = 0; i < 64; i += 1) {

fp\_cipher <<= 1;

fp\_cipher |= 0x01 & (cipher >> (64 - fp[i]));

}

return fp\_cipher;

}

int main() {

uint64\_t P = 0x0123456789abcdef;

uint64\_t K = 0x0123456789abcdef;

std::cout << "Starting Plaintext :: " << std::hex << P << "\n" <<

"Starting Key :: " << std::hex << K << "\n";

uint64\_t\* Ks = key\_gen(K);

uint64\_t C = des\_encrypt(P, Ks);

std::cout << "Ending Ciphertext :: " << std::hex << C << "\n";

return 0;

}