VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

B.Tech. Winter Semester 2024-25
School Of Computer Science and Engineering (SCOPE)

Digital Assignment - IV

Cryptography and Network Security Lab

Apurva Mishra: 22BCE2791
Date: 16 February, 2025

Contents

1 Expand Key

1.1 Code

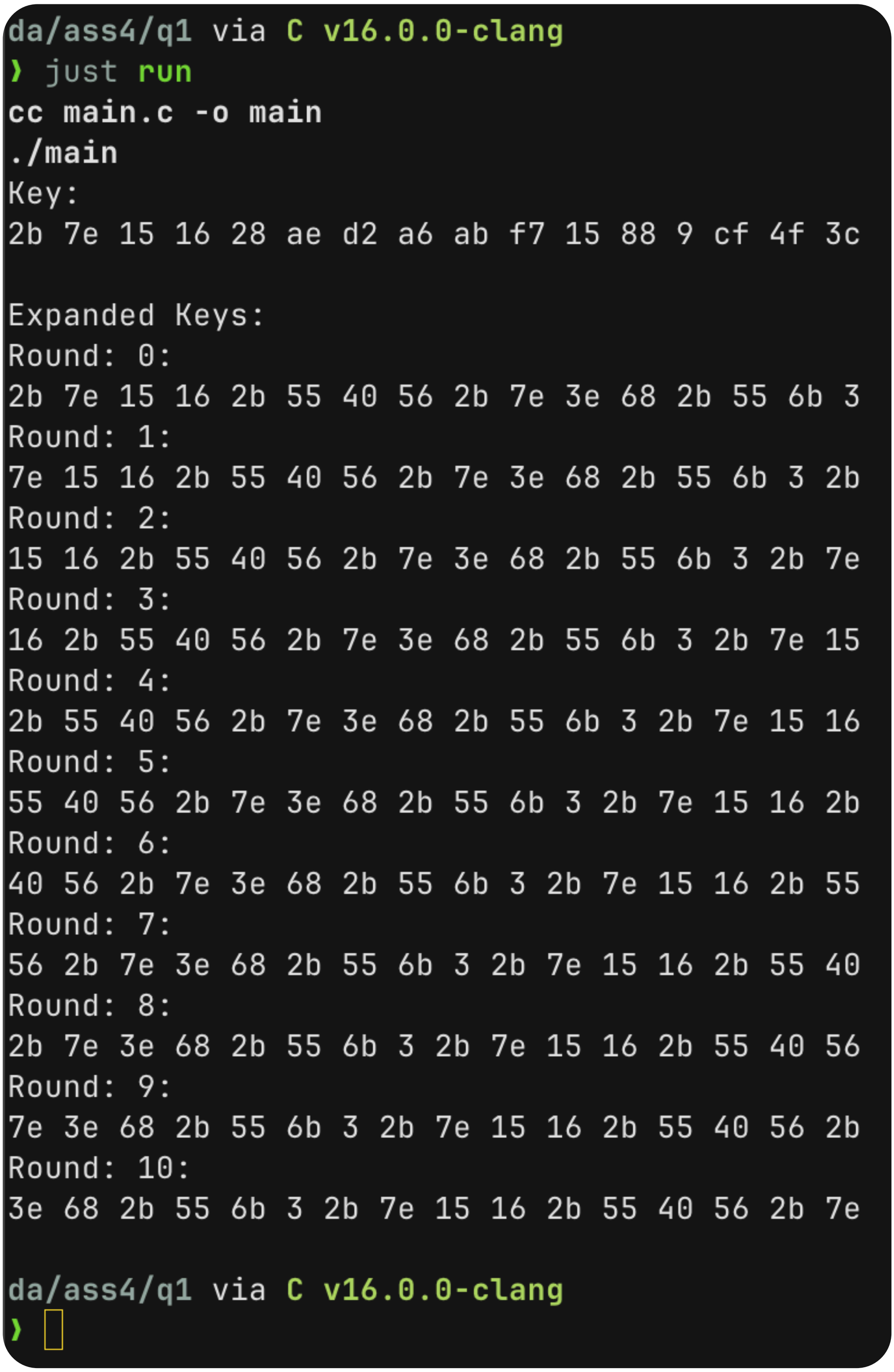
Code 0: main.c

```c
//AES
// Key Expansion
#include <stdint.h>
#include <stdio.h>

// sbox for byte substitution g:2
static const uint8_t sbox[16][16] = {
    {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}};

// Rcon values (round constants) g:3
static const uint8_t rcon[10] = {0x01, 0x02, 0x04, 0x08, 0x10,
                                 0x20, 0x40, 0x80, 0x1b, 0x36};

// g:1
uint32_t left_rotate_word(uint32_t word) { return (word << 8) || (word >> 24); }
```

```c

// g:2, based on sbox
uint32_t byte_substitute(uint32_t word) {
  uint32_t result = 0;

  // for each byte
  for (int i = 0; i < 4; i++) {
    uint8_t byte = (word & (15 << (i * 8))) >> (i * 8);
    int row = byte >> 4;
    int col = byte << 4 >> 4;

    result = (result << 4) + sbox[row][col];
  }

  return result;
}

// g:3
// rcon of form (con, 0, 0, 0)
uint32_t add_row_const(uint32_t word, int round_number) {
  uint32_t rconst = 0;
  rconst = (rconst | rcon[round_number]) << 24;

  return word ^ rconst;
}

// helper to get g our of key
uint32_t assign_g(uint8_t *key, uint8_t *expanded_key, int base, int first) {
  uint32_t res = 0;
  for (int i = 0; i < 4; i++) {
    if (first) {
      res = (res | key[base]) << ((3 - i) * 8);
    } else {
      res = (res | expanded_key[base]) << ((3 - i) * 8);
    }
  }

  return res;
}

uint32_t update_g(uint8_t g, int round_number) {
  g = left_rotate_word(g);
  g = byte_substitute(g);
  g = add_row_const(g, round_number);

  return g;
}

void expand_key(uint8_t *key, uint8_t *expanded_key) {

  for (int i = 0; i < 44; i++) {
    int base = i * 4;
    uint32_t g = 0;

    if (i == 0) {
      expanded_key[0] = key[0];
      expanded_key[1] = key[1];
      expanded_key[2] = key[2];
      expanded_key[3] = key[3];

      g = assign_g(key, expanded_key, i, 1);
      g = update_g(g, i / 4);
    } else {
      // iterate over each byte in word
      for (int j = 0; j < 4; j++) {
        // use g from prev
        if (j == 0) {
          int base = (i * 4) + j;
          int prev_start = ((i - 1) * 4) + j;

          expanded_key[base] = g ^ expanded_key[prev_start];

          // base case
        } else {
          int base = (i * 4) + j;
          int prev_start = ((i - 1) * 4) + j;

          expanded_key[base] =
              expanded_key[base - 1] ^ expanded_key[prev_start];
        }
      }

      // prep g for next iter
      g = assign_g(key, expanded_key, i, 0);
      g = update_g(g, i / 4);
    }
  }
}

int main() {
  uint8_t key[16] = {0x2b, 0x7e, 0x15, 0x16, 0x28, 0xae, 0xd2, 0xa6,
                     0xab, 0xf7, 0x15, 0x88, 0x09, 0xcf, 0x4f, 0x3c};
  uint8_t expanded_key[176];

  printf("Key:\n");
  for (int i = 0; i < 16; i++) {
    printf("%x ", key[i]);
  }
  printf("\n\n");

  expand_key(key, expanded_key);

  printf("Expanded Keys: \n");
  for (int i = 0; i < 11; i++) {
    printf("Round: %d:\n", i);
    for (int j = 0; j < 16; j++) {
```

```c
      printf("%x ", expanded_key[i + j]);
    }

    printf("\n");
  }
  return 0;
}

```

1.2 Output



```
da/ass4/q1 via C v16.0.0-clang
❯ just run
cc main.c -o main
./main
Key:
2b 7e 15 16 28 ae d2 a6 ab f7 15 88 9 cf 4f 3c

Expanded Keys:
Round: 0:
2b 7e 15 16 2b 55 40 56 2b 7e 3e 68 2b 55 6b 3
Round: 1:
7e 15 16 2b 55 40 56 2b 7e 3e 68 2b 55 6b 3 2b
Round: 2:
15 16 2b 55 40 56 2b 7e 3e 68 2b 55 6b 3 2b 7e
Round: 3:
16 2b 55 40 56 2b 7e 3e 68 2b 55 6b 3 2b 7e 15
Round: 4:
2b 55 40 56 2b 7e 3e 68 2b 55 6b 3 2b 7e 15 16
Round: 5:
55 40 56 2b 7e 3e 68 2b 55 6b 3 2b 7e 15 16 2b
Round: 6:
40 56 2b 7e 3e 68 2b 55 6b 3 2b 7e 15 16 2b 55
Round: 7:
56 2b 7e 3e 68 2b 55 6b 3 2b 7e 15 16 2b 55 40
Round: 8:
2b 7e 3e 68 2b 55 6b 3 2b 7e 15 16 2b 55 40 56
Round: 9:
7e 3e 68 2b 55 6b 3 2b 7e 15 16 2b 55 40 56 2b
Round: 10:
3e 68 2b 55 6b 3 2b 7e 15 16 2b 55 40 56 2b 7e

da/ass4/q1 via C v16.0.0-clang
❯
```

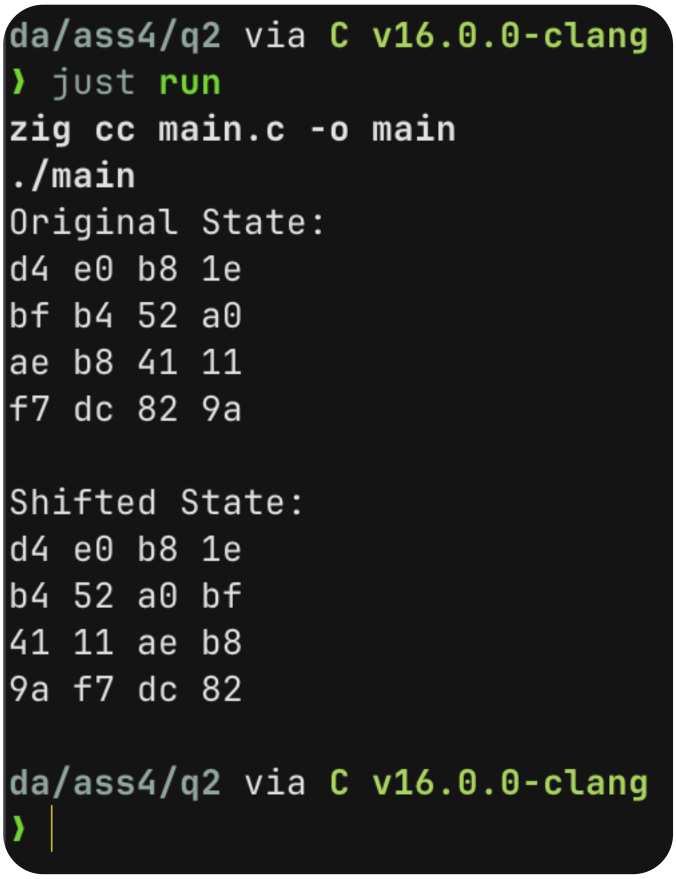
2 Shift Rows

2.1 Code

Code 0: main.c

```c
#include <stdio.h>
#include <stdint.h>

// left rotate row once
void left_rotate(uint8_t *row) {
  uint8_t first = row[0];

  for (int i = 0; i < 3; i++) {
    row[i] = row[i + 1];
  }

  row[3] = first;
}

void shift_rows(uint8_t state[4][4]) {
  for (int i = 0; i < 4; i++) {
    for (int j = 0; j < i; j++) {
      left_rotate(state[i]);
    }
  }
}

int main() {
    uint8_t state[4][4] = {
        {0xd4, 0xe0, 0xb8, 0x1e},
        {0xbf, 0xb4, 0x52, 0xa0},
        {0xae, 0xb8, 0x41, 0x11},
        {0xf7, 0xdc, 0x82, 0x9a}
    };

    printf("Original State:\n");
    for (int i = 0; i < 4; i++) {
        for (int j = 0; j < 4; j++) {
            printf("%02x ", state[i][j]);
        }
        printf("\n");
    }

    shift_rows(state);

    printf("\nShifted State:\n");
    for (int i = 0; i < 4; i++) {
        for (int j = 0; j < 4; j++) {
            printf("%02x ", state[i][j]);
        }
        printf("\n");
    }

    return 0;
}

```

2.2 Output



```
da/ass4/q2 via C v16.0.0-clang
❯ just run
zig cc main.c -o main
./main
Original State:
d4 e0 b8 1e
bf b4 52 a0
ae b8 41 11
f7 dc 82 9a

Shifted State:
d4 e0 b8 1e
b4 52 a0 bf
41 11 ae b8
9a f7 dc 82

da/ass4/q2 via C v16.0.0-clang
❯
```

3 Mix Columns

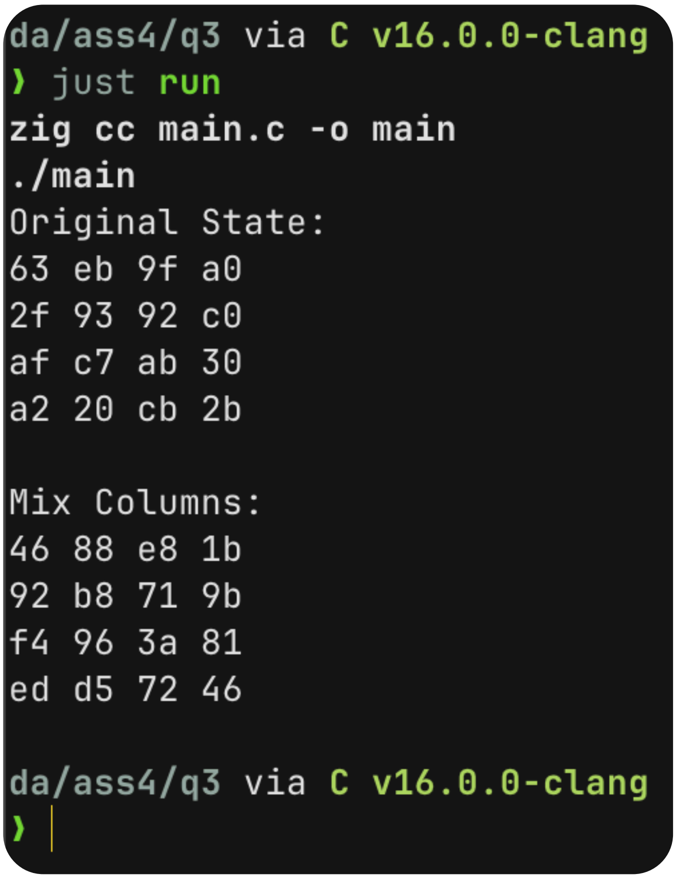
3.1 Code

Code 0: main.c

```c
#include <stdio.h>
#include <stdint.h>


const uint8_t fixed_mat[4][4] = {
        {0x02, 0x03, 0x01, 0x01},
        {0x01, 0x02, 0x03, 0x01},
        {0x01, 0x01, 0x02, 0x03},
        {0x03, 0x01, 0x01, 0x02}
    };

// matrix multiplication at gives pose (i, j) for f_mat and s_mat
// instead of adding values, we xor them
uint8_t mul(const uint8_t fixed_mat[4][4], uint8_t state_mat[4][4], int row,
int col) {
  uint8_t res = 0;
  for (int i = 0; i < 4; i++) {
    // for first iter, xor might give wrong value
    // so just save it at first using or op
    if (i == 0) {
      res |= fixed_mat[row][i] * state_mat[i][col];
    } else {
      res ^= fixed_mat[row][i] * state_mat[i][col];
    }
  }

  return res;
}

// fixed_mat * mix_column
void mix_columns(uint8_t state_mat[4][4]) {
    uint8_t temp[4][4] = {4};

    // iter over rows
    // and mat mul
    for (int i = 0; i < 4; i++) {
      for (int j = 0; j < 4; j++) {
        temp[i][j] = mul(fixed_mat, state_mat, i, j);
      }
    }

    for (int i = 0; i < 4; i++) {
      for (int j = 0; j < 4; j++) {
        state_mat[i][j] = temp[i][j];
      }
    }
}

int main() {
    uint8_t state[4][4] = {
        {0x63, 0xEB, 0x9F, 0xA0},
        {0x2F, 0x93, 0x92, 0xC0},
        {0xAF, 0xC7, 0xAB, 0x30},
        {0xA2, 0x20, 0xCB, 0x2B}
    };

    printf("Original State:\n");
    for (int i = 0; i < 4; i++) {
        for (int j = 0; j < 4; j++) {
            printf("%02x ", state[i][j]);
        }
        printf("\n");
    }

    mix_columns(state);

    printf("\nMix Columns:\n");
```

```c
    for (int i = 0; i < 4; i++) {
        for (int j = 0; j < 4; j++) {
            printf("%02x ", state[i][j]);
        }
        printf("\n");
    }

    return 0;
}
```

3.2 Output



```
da/ass4/q3 via C v16.0.0-clang
❯ just run
zig cc main.c -o main
./main
Original State:
63 eb 9f a0
2f 93 92 c0
af c7 ab 30
a2 20 cb 2b

Mix Columns:
46 88 e8 1b
92 b8 71 9b
f4 96 3a 81
ed d5 72 46

da/ass4/q3 via C v16.0.0-clang
❯
```