```c


/* Log table using 0xe5 (229) as the generator */
unsigned char ltable[256] = {
    0x00, 0xff, 0xc8, 0x08, 0x91, 0x10, 0xd0, 0x36,
    0x5a, 0x3e, 0xd8, 0x43, 0x99, 0x77, 0xfe, 0x18,
    0x23, 0x20, 0x07, 0x70, 0xa1, 0x6c, 0x0c, 0x7f,
    0x62, 0x8b, 0x40, 0x46, 0xc7, 0x4b, 0xe0, 0x0e,
    0xeb, 0x16, 0xe8, 0xad, 0xcf, 0xcd, 0x39, 0x53,
    0x6a, 0x27, 0x35, 0x93, 0xd4, 0x4e, 0x48, 0xc3,
    0x2b, 0x79, 0x54, 0x28, 0x09, 0x78, 0x0f, 0x21,
    0x90, 0x87, 0x14, 0x2a, 0xa9, 0x9c, 0xd6, 0x74,
    0xb4, 0x7c, 0xde, 0xed, 0xb1, 0x86, 0x76, 0xa4,
    0x98, 0xe2, 0x96, 0x8f, 0x02, 0x32, 0x1c, 0xc1,
    0x33, 0xee, 0xef, 0x81, 0xfd, 0x30, 0x5c, 0x13,
    0x9d, 0x29, 0x17, 0xc4, 0x11, 0x44, 0x8c, 0x80,
    0xf3, 0x73, 0x42, 0x1e, 0x1d, 0xb5, 0xf0, 0x12,
    0xd1, 0x5b, 0x41, 0xa2, 0xd7, 0x2c, 0xe9, 0xd5,
    0x59, 0xcb, 0x50, 0xa8, 0xdc, 0xfc, 0xf2, 0x56,
    0x72, 0xa6, 0x65, 0x2f, 0x9f, 0x9b, 0x3d, 0xba,
    0x7d, 0xc2, 0x45, 0x82, 0xa7, 0x57, 0xb6, 0xa3,
    0x7a, 0x75, 0x4f, 0xae, 0x3f, 0x37, 0x6d, 0x47,
    0x61, 0xbe, 0xab, 0xd3, 0x5f, 0xb0, 0x58, 0xaf,
    0xca, 0x5e, 0xfa, 0x85, 0xe4, 0x4d, 0x8a, 0x05,
    0xfb, 0x60, 0xb7, 0x7b, 0xb8, 0x26, 0x4a, 0x67,
    0xc6, 0x1a, 0xf8, 0x69, 0x25, 0xb3, 0xdb, 0xbd,
    0x66, 0xdd, 0xf1, 0xd2, 0xdf, 0x03, 0x8d, 0x34,
    0xd9, 0x92, 0x0d, 0x63, 0x55, 0xaa, 0x49, 0xec,
    0xbc, 0x95, 0x3c, 0x84, 0x0b, 0xf5, 0xe6, 0xe7,
    0xe5, 0xac, 0x7e, 0x6e, 0xb9, 0xf9, 0xda, 0x8e,
    0x9a, 0xc9, 0x24, 0xe1, 0x0a, 0x15, 0x6b, 0x3a,
    0xa0, 0x51, 0xf4, 0xea, 0xb2, 0x97, 0x9e, 0x5d,
    0x22, 0x88, 0x94, 0xce, 0x19, 0x01, 0x71, 0x4c,
    0xa5, 0xe3, 0xc5, 0x31, 0xbb, 0xcc, 0x1f, 0x2d,
    0x3b, 0x52, 0x6f, 0xf6, 0x2e, 0x89, 0xf7, 0xc0,
    0x68, 0x1b, 0x64, 0x04, 0x06, 0xbf, 0x83, 0x38 };


/* Anti-log table: */
unsigned char atable[256] = {
    0x01, 0xe5, 0x4c, 0xb5, 0xfb, 0x9f, 0xfc, 0x12,
    0x03, 0x34, 0xd4, 0xc4, 0x16, 0xba, 0x1f, 0x36,
    0x05, 0x5c, 0x67, 0x57, 0x3a, 0xd5, 0x21, 0x5a,
    0x0f, 0xe4, 0xa9, 0xf9, 0x4e, 0x64, 0x63, 0xee,
    0x11, 0x37, 0xe0, 0x10, 0xd2, 0xac, 0xa5, 0x29,
    0x33, 0x59, 0x3b, 0x30, 0x6d, 0xef, 0xf4, 0x7b,
    0x55, 0xeb, 0x4d, 0x50, 0xb7, 0x2a, 0x07, 0x8d,
    0xff, 0x26, 0xd7, 0xf0, 0xc2, 0x7e, 0x09, 0x8c,
    0x1a, 0x6a, 0x62, 0x0b, 0x5d, 0x82, 0x1b, 0x8f,
    0x2e, 0xbe, 0xa6, 0x1d, 0xe7, 0x9d, 0x2d, 0x8a,
    0x72, 0xd9, 0xf1, 0x27, 0x32, 0xbc, 0x77, 0x85,
    0x96, 0x70, 0x08, 0x69, 0x56, 0xdf, 0x99, 0x94,
    0xa1, 0x90, 0x18, 0xbb, 0xfa, 0x7a, 0xb0, 0xa7,
    0xf8, 0xab, 0x28, 0xd6, 0x15, 0x8e, 0xcb, 0xf2,
    0x13, 0xe6, 0x78, 0x61, 0x3f, 0x89, 0x46, 0x0d,
    0x35, 0x31, 0x88, 0xa3, 0x41, 0x80, 0xca, 0x17,
    0x5f, 0x53, 0x83, 0xfe, 0xc3, 0x9b, 0x45, 0x39,
    0xe1, 0xf5, 0x9e, 0x19, 0x5e, 0xb6, 0xcf, 0x4b,
    0x38, 0x04, 0xb9, 0x2b, 0xe2, 0xc1, 0x4a, 0xdd,
    0x48, 0x0c, 0xd0, 0x7d, 0x3d, 0x58, 0xde, 0x7c,
    0xd8, 0x14, 0x6b, 0x87, 0x47, 0xe8, 0x79, 0x84,
    0x73, 0x3c, 0xbd, 0x92, 0xc9, 0x23, 0x8b, 0x97,
    0x95, 0x44, 0xdc, 0xad, 0x40, 0x65, 0x86, 0xa2,
    0xa4, 0xcc, 0x7f, 0xec, 0xc0, 0xaf, 0x91, 0xfd,
    0xf7, 0x4f, 0x81, 0x2f, 0x5b, 0xea, 0xa8, 0x1c,
    0x02, 0xd1, 0x98, 0x71, 0xed, 0x25, 0xe3, 0x24,
    0x06, 0x68, 0xb3, 0x93, 0x2c, 0x6f, 0x3e, 0x6c,
    0x0a, 0xb8, 0xce, 0xae, 0x74, 0xb1, 0x42, 0xb4,
    0x1e, 0xd3, 0x49, 0xe9, 0x9c, 0xc8, 0xc6, 0xc7,
    0x22, 0x6e, 0xdb, 0x20, 0xbf, 0x43, 0x51, 0x52,
    0x66, 0xb2, 0x76, 0x60, 0xda, 0xc5, 0xf3, 0xf6,
    0xaa, 0xcd, 0x9a, 0xa0, 0x75, 0x54, 0x0e, 0x01 };


unsigned char gmultiply(unsigned char a, unsigned char b)
{
    unsigned char product = 0;
    unsigned char counter;
    unsigned char hi_bit_set;

    for (counter = 0; counter < 8; counter++)
    {
        if ((b & 1) == 1)
            product ^= a;

        hi_bit_set = (a & 0x80);
        a <<= 1;

        if (hi_bit_set == 0x80)
            a ^= 0x1b;

        b >>= 1;
    }

    return product;
}

unsigned char gmul_inverse(unsigned char in) {
    /* 0 is self inverting */
    if (in == 0)
        return 0;
    else
        return atable[(255 - ltable[in])];
}
```

```c



void rotate(unsigned char * in)
{
    unsigned char a = in[0];
    unsigned char c;

    for (c = 0; c < 3; c++)
        in[c] = in[c + 1];

    in[3] = a;
    return;
}


unsigned char rcon(unsigned char in)
{
    unsigned char c = 1;
    if (in == 0) return 0;

    while (in != 1)
    {
        c = gmultiply(c, 2);
        in--;
    }

    return c;
}

/* Calculate the s-box for a given number */
unsigned char csbox(unsigned char in)
{
    unsigned char c, s, x;
    s = x = gmul_inverse(in);

    for (c = 0; c < 4; c++)
    {
        /* One bit circular rotate to the left */
        s = (s << 1) | (s >> 7);
        /* xor with x */
        x ^= s;
    }

    x ^= 99; /* 0x63 */
    return x;
}

/* This is the core key expansion, which, given a 4-byte value, does some scrambling. */
void schedule_core(unsigned char * in, unsigned char i)
{
    char a;

    /* Rotate the input 8 bits to the left */
    rotate(in);

    /* Apply Rijndael's s-box on all 4 bytes. */
    for (a = 0; a < 4; a++)
        in[a] = csbox(in[a]);

    /* On just the first byte, add 2^i to the byte */
    in[0] ^= rcon(i);
}


void print_expand_key(unsigned char * in)
{
    unsigned char t[4];

    /* let c = 16, since first sub-key is user-supplied key */
    unsigned char c = 16;
    unsigned char i = 1;
    unsigned char a;
    int round = 1;

    /* Since we have 11 rounds (Round 0 up to round 10 for 16-byte key), we need 11 sets of
    16 bytes each, for 128-bit mode (AES). */
    while (c < 176)
    {
        /* Copy the temp variable over from the last 4-byte block */
        for (a = 0; a < 4; a++)
            t[a] = in[a + c - 4];

        if (c > 16)
            for (a = 0; a < 4; a++)
                printf("%02x ", t[a]);

        /* Every four blocks (4-bytes) do a complex calculation */
        if (c % 16 == 0)
        {
            schedule_core(t, i);
            i++;
            printf("\nRound: %d:\n     Key: ", round++);
        }

        for (a = 0; a < 4; a++)
        {
            in[c] = in[c - 16] ^ t[a];
            c++;
            if (c > 172) printf("%02x ", in[c - 1]);
        }
    }
```

```
}

```