# -\*- coding: utf-8 -\*-

"""

Created on Tue Jul 23 15:06:07 2019

@author: xxx

"""

from hashlib import md5

from base64 import b64decode

from base64 import b64encode

from Crypto.Cipher import AES

from Crypto.Random import get\_random\_bytes

from Crypto.Util.Padding import pad, unpad

class AESCipher:

def \_\_init\_\_(self, key):

self.key = md5(key.encode('utf8')).digest()

def encrypt(self, data):

iv = get\_random\_bytes(AES.block\_size)

self.cipher = AES.new(self.key, AES.MODE\_CBC, iv)

return b64encode(iv + self.cipher.encrypt(pad(data.encode('utf-8'),

AES.block\_size)))

def decrypt(self, data):

raw = b64decode(data)

self.cipher = AES.new(self.key, AES.MODE\_CBC, raw[:AES.block\_size])

return unpad(self.cipher.decrypt(raw[AES.block\_size:]), AES.block\_size)

if \_\_name\_\_ == '\_\_main\_\_':

print('TESTING ENCRYPTION')

msg = input('Message...: ')

pwd = input('Password..: ')

print('Ciphertext:', AESCipher(pwd).encrypt(msg).decode('utf-8'))

print('\nTESTING DECRYPTION')

cte = input('Ciphertext: ')

pwd = input('Password..: ')

print('Message...:', AESCipher(pwd).decrypt(cte).decode('utf-8'))

Pycrypte-2.6.1>src

Librería AES

/\*\*

\* rijndael-alg-fst.c

\*

\* @version 3.0 (December 2000)

\*

\* Optimised ANSI C code for the Rijndael cipher (now AES)

\*

\* @author Vincent Rijmen <vincent.rijmen@esat.kuleuven.ac.be>

\* @author Antoon Bosselaers <antoon.bosselaers@esat.kuleuven.ac.be>

\* @author Paulo Barreto <paulo.barreto@terra.com.br>

\*

\* This code is hereby placed in the public domain.

\*

\* THIS SOFTWARE IS PROVIDED BY THE AUTHORS ''AS IS'' AND ANY EXPRESS

\* OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED

\* WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE

\* ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHORS OR CONTRIBUTORS BE

\* LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR

\* CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF

\* SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR

\* BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY,

\* WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE

\* OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE,

\* EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

\*/

#include <assert.h>

#include <stdlib.h>

#include "Python.h"

#define MODULE\_NAME \_AES

#define BLOCK\_SIZE 16

#define KEY\_SIZE 0

#define MAXKC (256/32)

#define MAXKB (256/8)

#define MAXNR 14

typedef unsigned char u8;

typedef unsigned short u16;

typedef unsigned int u32;

typedef struct {

u32 ek[ 4\*(MAXNR+1) ];

u32 dk[ 4\*(MAXNR+1) ];

int rounds;

} block\_state;

static void rijndaelEncrypt(u32 rk[/\*4\*(Nr + 1)\*/], int Nr, const u8 pt[16], u8 ct[16]);

static void rijndaelDecrypt(u32 rk[/\*4\*(Nr + 1)\*/], int Nr, const u8 ct[16], u8 pt[16]);

#ifdef INTERMEDIATE\_VALUE\_KAT

static void rijndaelEncryptRound(const u32 rk[/\*4\*(Nr + 1)\*/], int Nr, u8 block[16], int rounds);

static void rijndaelDecryptRound(const u32 rk[/\*4\*(Nr + 1)\*/], int Nr, u8 block[16], int rounds);

#endif /\* INTERMEDIATE\_VALUE\_KAT \*/

/\*

Te0[x] = S [x].[02, 01, 01, 03];

Te1[x] = S [x].[03, 02, 01, 01];

Te2[x] = S [x].[01, 03, 02, 01];

Te3[x] = S [x].[01, 01, 03, 02];

Te4[x] = S [x].[01, 01, 01, 01];

Td0[x] = Si[x].[0e, 09, 0d, 0b];

Td1[x] = Si[x].[0b, 0e, 09, 0d];

Td2[x] = Si[x].[0d, 0b, 0e, 09];

Td3[x] = Si[x].[09, 0d, 0b, 0e];

Td4[x] = Si[x].[01, 01, 01, 01];

\*/

static const u32 Te0[256] = {

0xc66363a5U, 0xf87c7c84U, 0xee777799U, 0xf67b7b8dU,

0xfff2f20dU, 0xd66b6bbdU, 0xde6f6fb1U, 0x91c5c554U,

0x60303050U, 0x02010103U, 0xce6767a9U, 0x562b2b7dU,

0xe7fefe19U, 0xb5d7d762U, 0x4dababe6U, 0xec76769aU,

0x8fcaca45U, 0x1f82829dU, 0x89c9c940U, 0xfa7d7d87U,

0xeffafa15U, 0xb25959ebU, 0x8e4747c9U, 0xfbf0f00bU,

0x41adadecU, 0xb3d4d467U, 0x5fa2a2fdU, 0x45afafeaU,

0x239c9cbfU, 0x53a4a4f7U, 0xe4727296U, 0x9bc0c05bU,

0x75b7b7c2U, 0xe1fdfd1cU, 0x3d9393aeU, 0x4c26266aU,

0x6c36365aU, 0x7e3f3f41U, 0xf5f7f702U, 0x83cccc4fU,

0x6834345cU, 0x51a5a5f4U, 0xd1e5e534U, 0xf9f1f108U,

0xe2717193U, 0xabd8d873U, 0x62313153U, 0x2a15153fU,

0x0804040cU, 0x95c7c752U, 0x46232365U, 0x9dc3c35eU,

0x30181828U, 0x379696a1U, 0x0a05050fU, 0x2f9a9ab5U,

0x0e070709U, 0x24121236U, 0x1b80809bU, 0xdfe2e23dU,

0xcdebeb26U, 0x4e272769U, 0x7fb2b2cdU, 0xea75759fU,

0x1209091bU, 0x1d83839eU, 0x582c2c74U, 0x341a1a2eU,

0x361b1b2dU, 0xdc6e6eb2U, 0xb45a5aeeU, 0x5ba0a0fbU,

0xa45252f6U, 0x763b3b4dU, 0xb7d6d661U, 0x7db3b3ceU,

0x5229297bU, 0xdde3e33eU, 0x5e2f2f71U, 0x13848497U,

0xa65353f5U, 0xb9d1d168U, 0x00000000U, 0xc1eded2cU,

0x40202060U, 0xe3fcfc1fU, 0x79b1b1c8U, 0xb65b5bedU,

0xd46a6abeU, 0x8dcbcb46U, 0x67bebed9U, 0x7239394bU,

0x944a4adeU, 0x984c4cd4U, 0xb05858e8U, 0x85cfcf4aU,

0xbbd0d06bU, 0xc5efef2aU, 0x4faaaae5U, 0xedfbfb16U,

0x864343c5U, 0x9a4d4dd7U, 0x66333355U, 0x11858594U,

0x8a4545cfU, 0xe9f9f910U, 0x04020206U, 0xfe7f7f81U,

0xa05050f0U, 0x783c3c44U, 0x259f9fbaU, 0x4ba8a8e3U,

0xa25151f3U, 0x5da3a3feU, 0x804040c0U, 0x058f8f8aU,

0x3f9292adU, 0x219d9dbcU, 0x70383848U, 0xf1f5f504U,

0x63bcbcdfU, 0x77b6b6c1U, 0xafdada75U, 0x42212163U,

0x20101030U, 0xe5ffff1aU, 0xfdf3f30eU, 0xbfd2d26dU,

0x81cdcd4cU, 0x180c0c14U, 0x26131335U, 0xc3ecec2fU,

0xbe5f5fe1U, 0x359797a2U, 0x884444ccU, 0x2e171739U,

0x93c4c457U, 0x55a7a7f2U, 0xfc7e7e82U, 0x7a3d3d47U,

0xc86464acU, 0xba5d5de7U, 0x3219192bU, 0xe6737395U,

0xc06060a0U, 0x19818198U, 0x9e4f4fd1U, 0xa3dcdc7fU,

0x44222266U, 0x542a2a7eU, 0x3b9090abU, 0x0b888883U,

0x8c4646caU, 0xc7eeee29U, 0x6bb8b8d3U, 0x2814143cU,

0xa7dede79U, 0xbc5e5ee2U, 0x160b0b1dU, 0xaddbdb76U,

0xdbe0e03bU, 0x64323256U, 0x743a3a4eU, 0x140a0a1eU,

0x924949dbU, 0x0c06060aU, 0x4824246cU, 0xb85c5ce4U,

0x9fc2c25dU, 0xbdd3d36eU, 0x43acacefU, 0xc46262a6U,

0x399191a8U, 0x319595a4U, 0xd3e4e437U, 0xf279798bU,

0xd5e7e732U, 0x8bc8c843U, 0x6e373759U, 0xda6d6db7U,

0x018d8d8cU, 0xb1d5d564U, 0x9c4e4ed2U, 0x49a9a9e0U,

0xd86c6cb4U, 0xac5656faU, 0xf3f4f407U, 0xcfeaea25U,

0xca6565afU, 0xf47a7a8eU, 0x47aeaee9U, 0x10080818U,

0x6fbabad5U, 0xf0787888U, 0x4a25256fU, 0x5c2e2e72U,

0x381c1c24U, 0x57a6a6f1U, 0x73b4b4c7U, 0x97c6c651U,

0xcbe8e823U, 0xa1dddd7cU, 0xe874749cU, 0x3e1f1f21U,

0x964b4bddU, 0x61bdbddcU, 0x0d8b8b86U, 0x0f8a8a85U,

0xe0707090U, 0x7c3e3e42U, 0x71b5b5c4U, 0xcc6666aaU,

0x904848d8U, 0x06030305U, 0xf7f6f601U, 0x1c0e0e12U,

0xc26161a3U, 0x6a35355fU, 0xae5757f9U, 0x69b9b9d0U,

0x17868691U, 0x99c1c158U, 0x3a1d1d27U, 0x279e9eb9U,

0xd9e1e138U, 0xebf8f813U, 0x2b9898b3U, 0x22111133U,

0xd26969bbU, 0xa9d9d970U, 0x078e8e89U, 0x339494a7U,

0x2d9b9bb6U, 0x3c1e1e22U, 0x15878792U, 0xc9e9e920U,

0x87cece49U, 0xaa5555ffU, 0x50282878U, 0xa5dfdf7aU,

0x038c8c8fU, 0x59a1a1f8U, 0x09898980U, 0x1a0d0d17U,

0x65bfbfdaU, 0xd7e6e631U, 0x844242c6U, 0xd06868b8U,

0x824141c3U, 0x299999b0U, 0x5a2d2d77U, 0x1e0f0f11U,

0x7bb0b0cbU, 0xa85454fcU, 0x6dbbbbd6U, 0x2c16163aU,

};

static const u32 Te1[256] = {

0xa5c66363U, 0x84f87c7cU, 0x99ee7777U, 0x8df67b7bU,

0x0dfff2f2U, 0xbdd66b6bU, 0xb1de6f6fU, 0x5491c5c5U,

0x50603030U, 0x03020101U, 0xa9ce6767U, 0x7d562b2bU,

0x19e7fefeU, 0x62b5d7d7U, 0xe64dababU, 0x9aec7676U,

0x458fcacaU, 0x9d1f8282U, 0x4089c9c9U, 0x87fa7d7dU,

0x15effafaU, 0xebb25959U, 0xc98e4747U, 0x0bfbf0f0U,

0xec41adadU, 0x67b3d4d4U, 0xfd5fa2a2U, 0xea45afafU,

0xbf239c9cU, 0xf753a4a4U, 0x96e47272U, 0x5b9bc0c0U,

0xc275b7b7U, 0x1ce1fdfdU, 0xae3d9393U, 0x6a4c2626U,

0x5a6c3636U, 0x417e3f3fU, 0x02f5f7f7U, 0x4f83ccccU,

0x5c683434U, 0xf451a5a5U, 0x34d1e5e5U, 0x08f9f1f1U,

0x93e27171U, 0x73abd8d8U, 0x53623131U, 0x3f2a1515U,

0x0c080404U, 0x5295c7c7U, 0x65462323U, 0x5e9dc3c3U,

0x28301818U, 0xa1379696U, 0x0f0a0505U, 0xb52f9a9aU,

0x090e0707U, 0x36241212U, 0x9b1b8080U, 0x3ddfe2e2U,

0x26cdebebU, 0x694e2727U, 0xcd7fb2b2U, 0x9fea7575U,

0x1b120909U, 0x9e1d8383U, 0x74582c2cU, 0x2e341a1aU,

0x2d361b1bU, 0xb2dc6e6eU, 0xeeb45a5aU, 0xfb5ba0a0U,

0xf6a45252U, 0x4d763b3bU, 0x61b7d6d6U, 0xce7db3b3U,

0x7b522929U, 0x3edde3e3U, 0x715e2f2fU, 0x97138484U,

0xf5a65353U, 0x68b9d1d1U, 0x00000000U, 0x2cc1ededU,

0x60402020U, 0x1fe3fcfcU, 0xc879b1b1U, 0xedb65b5bU,

0xbed46a6aU, 0x468dcbcbU, 0xd967bebeU, 0x4b723939U,

0xde944a4aU, 0xd4984c4cU, 0xe8b05858U, 0x4a85cfcfU,

0x6bbbd0d0U, 0x2ac5efefU, 0xe54faaaaU, 0x16edfbfbU,

0xc5864343U, 0xd79a4d4dU, 0x55663333U, 0x94118585U,

0xcf8a4545U, 0x10e9f9f9U, 0x06040202U, 0x81fe7f7fU,

0xf0a05050U, 0x44783c3cU, 0xba259f9fU, 0xe34ba8a8U,

0xf3a25151U, 0xfe5da3a3U, 0xc0804040U, 0x8a058f8fU,

0xad3f9292U, 0xbc219d9dU, 0x48703838U, 0x04f1f5f5U,

0xdf63bcbcU, 0xc177b6b6U, 0x75afdadaU, 0x63422121U,

0x30201010U, 0x1ae5ffffU, 0x0efdf3f3U, 0x6dbfd2d2U,

0x4c81cdcdU, 0x14180c0cU, 0x35261313U, 0x2fc3ececU,

0xe1be5f5fU, 0xa2359797U, 0xcc884444U, 0x392e1717U,

0x5793c4c4U, 0xf255a7a7U, 0x82fc7e7eU, 0x477a3d3dU,

0xacc86464U, 0xe7ba5d5dU, 0x2b321919U, 0x95e67373U,

0xa0c06060U, 0x98198181U, 0xd19e4f4fU, 0x7fa3dcdcU,

0x66442222U, 0x7e542a2aU, 0xab3b9090U, 0x830b8888U,

0xca8c4646U, 0x29c7eeeeU, 0xd36bb8b8U, 0x3c281414U,

0x79a7dedeU, 0xe2bc5e5eU, 0x1d160b0bU, 0x76addbdbU,

0x3bdbe0e0U, 0x56643232U, 0x4e743a3aU, 0x1e140a0aU,

0xdb924949U, 0x0a0c0606U, 0x6c482424U, 0xe4b85c5cU,

0x5d9fc2c2U, 0x6ebdd3d3U, 0xef43acacU, 0xa6c46262U,

0xa8399191U, 0xa4319595U, 0x37d3e4e4U, 0x8bf27979U,

0x32d5e7e7U, 0x438bc8c8U, 0x596e3737U, 0xb7da6d6dU,

0x8c018d8dU, 0x64b1d5d5U, 0xd29c4e4eU, 0xe049a9a9U,

0xb4d86c6cU, 0xfaac5656U, 0x07f3f4f4U, 0x25cfeaeaU,

0xafca6565U, 0x8ef47a7aU, 0xe947aeaeU, 0x18100808U,

0xd56fbabaU, 0x88f07878U, 0x6f4a2525U, 0x725c2e2eU,

0x24381c1cU, 0xf157a6a6U, 0xc773b4b4U, 0x5197c6c6U,

0x23cbe8e8U, 0x7ca1ddddU, 0x9ce87474U, 0x213e1f1fU,

0xdd964b4bU, 0xdc61bdbdU, 0x860d8b8bU, 0x850f8a8aU,

0x90e07070U, 0x427c3e3eU, 0xc471b5b5U, 0xaacc6666U,

0xd8904848U, 0x05060303U, 0x01f7f6f6U, 0x121c0e0eU,

0xa3c26161U, 0x5f6a3535U, 0xf9ae5757U, 0xd069b9b9U,

0x91178686U, 0x5899c1c1U, 0x273a1d1dU, 0xb9279e9eU,

0x38d9e1e1U, 0x13ebf8f8U, 0xb32b9898U, 0x33221111U,

0xbbd26969U, 0x70a9d9d9U, 0x89078e8eU, 0xa7339494U,

0xb62d9b9bU, 0x223c1e1eU, 0x92158787U, 0x20c9e9e9U,

0x4987ceceU, 0xffaa5555U, 0x78502828U, 0x7aa5dfdfU,

0x8f038c8cU, 0xf859a1a1U, 0x80098989U, 0x171a0d0dU,

0xda65bfbfU, 0x31d7e6e6U, 0xc6844242U, 0xb8d06868U,

0xc3824141U, 0xb0299999U, 0x775a2d2dU, 0x111e0f0fU,

0xcb7bb0b0U, 0xfca85454U, 0xd66dbbbbU, 0x3a2c1616U,

};

static const u32 Te2[256] = {

0x63a5c663U, 0x7c84f87cU, 0x7799ee77U, 0x7b8df67bU,

0xf20dfff2U, 0x6bbdd66bU, 0x6fb1de6fU, 0xc55491c5U,

0x30506030U, 0x01030201U, 0x67a9ce67U, 0x2b7d562bU,

0xfe19e7feU, 0xd762b5d7U, 0xabe64dabU, 0x769aec76U,

0xca458fcaU, 0x829d1f82U, 0xc94089c9U, 0x7d87fa7dU,

0xfa15effaU, 0x59ebb259U, 0x47c98e47U, 0xf00bfbf0U,

0xadec41adU, 0xd467b3d4U, 0xa2fd5fa2U, 0xafea45afU,

0x9cbf239cU, 0xa4f753a4U, 0x7296e472U, 0xc05b9bc0U,

0xb7c275b7U, 0xfd1ce1fdU, 0x93ae3d93U, 0x266a4c26U,

0x365a6c36U, 0x3f417e3fU, 0xf702f5f7U, 0xcc4f83ccU,

0x345c6834U, 0xa5f451a5U, 0xe534d1e5U, 0xf108f9f1U,

0x7193e271U, 0xd873abd8U, 0x31536231U, 0x153f2a15U,

0x040c0804U, 0xc75295c7U, 0x23654623U, 0xc35e9dc3U,

0x18283018U, 0x96a13796U, 0x050f0a05U, 0x9ab52f9aU,

0x07090e07U, 0x12362412U, 0x809b1b80U, 0xe23ddfe2U,

0xeb26cdebU, 0x27694e27U, 0xb2cd7fb2U, 0x759fea75U,

0x091b1209U, 0x839e1d83U, 0x2c74582cU, 0x1a2e341aU,

0x1b2d361bU, 0x6eb2dc6eU, 0x5aeeb45aU, 0xa0fb5ba0U,

0x52f6a452U, 0x3b4d763bU, 0xd661b7d6U, 0xb3ce7db3U,

0x297b5229U, 0xe33edde3U, 0x2f715e2fU, 0x84971384U,

0x53f5a653U, 0xd168b9d1U, 0x00000000U, 0xed2cc1edU,

0x20604020U, 0xfc1fe3fcU, 0xb1c879b1U, 0x5bedb65bU,

0x6abed46aU, 0xcb468dcbU, 0xbed967beU, 0x394b7239U,

0x4ade944aU, 0x4cd4984cU, 0x58e8b058U, 0xcf4a85cfU,

0xd06bbbd0U, 0xef2ac5efU, 0xaae54faaU, 0xfb16edfbU,

0x43c58643U, 0x4dd79a4dU, 0x33556633U, 0x85941185U,

0x45cf8a45U, 0xf910e9f9U, 0x02060402U, 0x7f81fe7fU,

0x50f0a050U, 0x3c44783cU, 0x9fba259fU, 0xa8e34ba8U,

0x51f3a251U, 0xa3fe5da3U, 0x40c08040U, 0x8f8a058fU,

0x92ad3f92U, 0x9dbc219dU, 0x38487038U, 0xf504f1f5U,

0xbcdf63bcU, 0xb6c177b6U, 0xda75afdaU, 0x21634221U,

0x10302010U, 0xff1ae5ffU, 0xf30efdf3U, 0xd26dbfd2U,

0xcd4c81cdU, 0x0c14180cU, 0x13352613U, 0xec2fc3ecU,

0x5fe1be5fU, 0x97a23597U, 0x44cc8844U, 0x17392e17U,

0xc45793c4U, 0xa7f255a7U, 0x7e82fc7eU, 0x3d477a3dU,

0x64acc864U, 0x5de7ba5dU, 0x192b3219U, 0x7395e673U,

0x60a0c060U, 0x81981981U, 0x4fd19e4fU, 0xdc7fa3dcU,

0x22664422U, 0x2a7e542aU, 0x90ab3b90U, 0x88830b88U,

0x46ca8c46U, 0xee29c7eeU, 0xb8d36bb8U, 0x143c2814U,

0xde79a7deU, 0x5ee2bc5eU, 0x0b1d160bU, 0xdb76addbU,

0xe03bdbe0U, 0x32566432U, 0x3a4e743aU, 0x0a1e140aU,

0x49db9249U, 0x060a0c06U, 0x246c4824U, 0x5ce4b85cU,

0xc25d9fc2U, 0xd36ebdd3U, 0xacef43acU, 0x62a6c462U,

0x91a83991U, 0x95a43195U, 0xe437d3e4U, 0x798bf279U,

0xe732d5e7U, 0xc8438bc8U, 0x37596e37U, 0x6db7da6dU,

0x8d8c018dU, 0xd564b1d5U, 0x4ed29c4eU, 0xa9e049a9U,

0x6cb4d86cU, 0x56faac56U, 0xf407f3f4U, 0xea25cfeaU,

0x65afca65U, 0x7a8ef47aU, 0xaee947aeU, 0x08181008U,

0xbad56fbaU, 0x7888f078U, 0x256f4a25U, 0x2e725c2eU,

0x1c24381cU, 0xa6f157a6U, 0xb4c773b4U, 0xc65197c6U,

0xe823cbe8U, 0xdd7ca1ddU, 0x749ce874U, 0x1f213e1fU,

0x4bdd964bU, 0xbddc61bdU, 0x8b860d8bU, 0x8a850f8aU,

0x7090e070U, 0x3e427c3eU, 0xb5c471b5U, 0x66aacc66U,

0x48d89048U, 0x03050603U, 0xf601f7f6U, 0x0e121c0eU,

0x61a3c261U, 0x355f6a35U, 0x57f9ae57U, 0xb9d069b9U,

0x86911786U, 0xc15899c1U, 0x1d273a1dU, 0x9eb9279eU,

0xe138d9e1U, 0xf813ebf8U, 0x98b32b98U, 0x11332211U,

0x69bbd269U, 0xd970a9d9U, 0x8e89078eU, 0x94a73394U,

0x9bb62d9bU, 0x1e223c1eU, 0x87921587U, 0xe920c9e9U,

0xce4987ceU, 0x55ffaa55U, 0x28785028U, 0xdf7aa5dfU,

0x8c8f038cU, 0xa1f859a1U, 0x89800989U, 0x0d171a0dU,

0xbfda65bfU, 0xe631d7e6U, 0x42c68442U, 0x68b8d068U,

0x41c38241U, 0x99b02999U, 0x2d775a2dU, 0x0f111e0fU,

0xb0cb7bb0U, 0x54fca854U, 0xbbd66dbbU, 0x163a2c16U,

};

static const u32 Te3[256] = {

0x6363a5c6U, 0x7c7c84f8U, 0x777799eeU, 0x7b7b8df6U,

0xf2f20dffU, 0x6b6bbdd6U, 0x6f6fb1deU, 0xc5c55491U,

0x30305060U, 0x01010302U, 0x6767a9ceU, 0x2b2b7d56U,

0xfefe19e7U, 0xd7d762b5U, 0xababe64dU, 0x76769aecU,

0xcaca458fU, 0x82829d1fU, 0xc9c94089U, 0x7d7d87faU,

0xfafa15efU, 0x5959ebb2U, 0x4747c98eU, 0xf0f00bfbU,

0xadadec41U, 0xd4d467b3U, 0xa2a2fd5fU, 0xafafea45U,

0x9c9cbf23U, 0xa4a4f753U, 0x727296e4U, 0xc0c05b9bU,

0xb7b7c275U, 0xfdfd1ce1U, 0x9393ae3dU, 0x26266a4cU,

0x36365a6cU, 0x3f3f417eU, 0xf7f702f5U, 0xcccc4f83U,

0x34345c68U, 0xa5a5f451U, 0xe5e534d1U, 0xf1f108f9U,

0x717193e2U, 0xd8d873abU, 0x31315362U, 0x15153f2aU,

0x04040c08U, 0xc7c75295U, 0x23236546U, 0xc3c35e9dU,

0x18182830U, 0x9696a137U, 0x05050f0aU, 0x9a9ab52fU,

0x0707090eU, 0x12123624U, 0x80809b1bU, 0xe2e23ddfU,

0xebeb26cdU, 0x2727694eU, 0xb2b2cd7fU, 0x75759feaU,

0x09091b12U, 0x83839e1dU, 0x2c2c7458U, 0x1a1a2e34U,

0x1b1b2d36U, 0x6e6eb2dcU, 0x5a5aeeb4U, 0xa0a0fb5bU,

0x5252f6a4U, 0x3b3b4d76U, 0xd6d661b7U, 0xb3b3ce7dU,

0x29297b52U, 0xe3e33eddU, 0x2f2f715eU, 0x84849713U,

0x5353f5a6U, 0xd1d168b9U, 0x00000000U, 0xeded2cc1U,

0x20206040U, 0xfcfc1fe3U, 0xb1b1c879U, 0x5b5bedb6U,

0x6a6abed4U, 0xcbcb468dU, 0xbebed967U, 0x39394b72U,

0x4a4ade94U, 0x4c4cd498U, 0x5858e8b0U, 0xcfcf4a85U,

0xd0d06bbbU, 0xefef2ac5U, 0xaaaae54fU, 0xfbfb16edU,

0x4343c586U, 0x4d4dd79aU, 0x33335566U, 0x85859411U,

0x4545cf8aU, 0xf9f910e9U, 0x02020604U, 0x7f7f81feU,

0x5050f0a0U, 0x3c3c4478U, 0x9f9fba25U, 0xa8a8e34bU,

0x5151f3a2U, 0xa3a3fe5dU, 0x4040c080U, 0x8f8f8a05U,

0x9292ad3fU, 0x9d9dbc21U, 0x38384870U, 0xf5f504f1U,

0xbcbcdf63U, 0xb6b6c177U, 0xdada75afU, 0x21216342U,

0x10103020U, 0xffff1ae5U, 0xf3f30efdU, 0xd2d26dbfU,

0xcdcd4c81U, 0x0c0c1418U, 0x13133526U, 0xecec2fc3U,

0x5f5fe1beU, 0x9797a235U, 0x4444cc88U, 0x1717392eU,

0xc4c45793U, 0xa7a7f255U, 0x7e7e82fcU, 0x3d3d477aU,

0x6464acc8U, 0x5d5de7baU, 0x19192b32U, 0x737395e6U,

0x6060a0c0U, 0x81819819U, 0x4f4fd19eU, 0xdcdc7fa3U,

0x22226644U, 0x2a2a7e54U, 0x9090ab3bU, 0x8888830bU,

0x4646ca8cU, 0xeeee29c7U, 0xb8b8d36bU, 0x14143c28U,

0xdede79a7U, 0x5e5ee2bcU, 0x0b0b1d16U, 0xdbdb76adU,

0xe0e03bdbU, 0x32325664U, 0x3a3a4e74U, 0x0a0a1e14U,

0x4949db92U, 0x06060a0cU, 0x24246c48U, 0x5c5ce4b8U,

0xc2c25d9fU, 0xd3d36ebdU, 0xacacef43U, 0x6262a6c4U,

0x9191a839U, 0x9595a431U, 0xe4e437d3U, 0x79798bf2U,

0xe7e732d5U, 0xc8c8438bU, 0x3737596eU, 0x6d6db7daU,

0x8d8d8c01U, 0xd5d564b1U, 0x4e4ed29cU, 0xa9a9e049U,

0x6c6cb4d8U, 0x5656faacU, 0xf4f407f3U, 0xeaea25cfU,

0x6565afcaU, 0x7a7a8ef4U, 0xaeaee947U, 0x08081810U,

0xbabad56fU, 0x787888f0U, 0x25256f4aU, 0x2e2e725cU,

0x1c1c2438U, 0xa6a6f157U, 0xb4b4c773U, 0xc6c65197U,

0xe8e823cbU, 0xdddd7ca1U, 0x74749ce8U, 0x1f1f213eU,

0x4b4bdd96U, 0xbdbddc61U, 0x8b8b860dU, 0x8a8a850fU,

0x707090e0U, 0x3e3e427cU, 0xb5b5c471U, 0x6666aaccU,

0x4848d890U, 0x03030506U, 0xf6f601f7U, 0x0e0e121cU,

0x6161a3c2U, 0x35355f6aU, 0x5757f9aeU, 0xb9b9d069U,

0x86869117U, 0xc1c15899U, 0x1d1d273aU, 0x9e9eb927U,

0xe1e138d9U, 0xf8f813ebU, 0x9898b32bU, 0x11113322U,

0x6969bbd2U, 0xd9d970a9U, 0x8e8e8907U, 0x9494a733U,

0x9b9bb62dU, 0x1e1e223cU, 0x87879215U, 0xe9e920c9U,

0xcece4987U, 0x5555ffaaU, 0x28287850U, 0xdfdf7aa5U,

0x8c8c8f03U, 0xa1a1f859U, 0x89898009U, 0x0d0d171aU,

0xbfbfda65U, 0xe6e631d7U, 0x4242c684U, 0x6868b8d0U,

0x4141c382U, 0x9999b029U, 0x2d2d775aU, 0x0f0f111eU,

0xb0b0cb7bU, 0x5454fca8U, 0xbbbbd66dU, 0x16163a2cU,

};

static const u32 Te4[256] = {

0x63636363U, 0x7c7c7c7cU, 0x77777777U, 0x7b7b7b7bU,

0xf2f2f2f2U, 0x6b6b6b6bU, 0x6f6f6f6fU, 0xc5c5c5c5U,

0x30303030U, 0x01010101U, 0x67676767U, 0x2b2b2b2bU,

0xfefefefeU, 0xd7d7d7d7U, 0xababababU, 0x76767676U,

0xcacacacaU, 0x82828282U, 0xc9c9c9c9U, 0x7d7d7d7dU,

0xfafafafaU, 0x59595959U, 0x47474747U, 0xf0f0f0f0U,

0xadadadadU, 0xd4d4d4d4U, 0xa2a2a2a2U, 0xafafafafU,

0x9c9c9c9cU, 0xa4a4a4a4U, 0x72727272U, 0xc0c0c0c0U,

0xb7b7b7b7U, 0xfdfdfdfdU, 0x93939393U, 0x26262626U,

0x36363636U, 0x3f3f3f3fU, 0xf7f7f7f7U, 0xccccccccU,

0x34343434U, 0xa5a5a5a5U, 0xe5e5e5e5U, 0xf1f1f1f1U,

0x71717171U, 0xd8d8d8d8U, 0x31313131U, 0x15151515U,

0x04040404U, 0xc7c7c7c7U, 0x23232323U, 0xc3c3c3c3U,

0x18181818U, 0x96969696U, 0x05050505U, 0x9a9a9a9aU,

0x07070707U, 0x12121212U, 0x80808080U, 0xe2e2e2e2U,

0xebebebebU, 0x27272727U, 0xb2b2b2b2U, 0x75757575U,

0x09090909U, 0x83838383U, 0x2c2c2c2cU, 0x1a1a1a1aU,

0x1b1b1b1bU, 0x6e6e6e6eU, 0x5a5a5a5aU, 0xa0a0a0a0U,

0x52525252U, 0x3b3b3b3bU, 0xd6d6d6d6U, 0xb3b3b3b3U,

0x29292929U, 0xe3e3e3e3U, 0x2f2f2f2fU, 0x84848484U,

0x53535353U, 0xd1d1d1d1U, 0x00000000U, 0xededededU,

0x20202020U, 0xfcfcfcfcU, 0xb1b1b1b1U, 0x5b5b5b5bU,

0x6a6a6a6aU, 0xcbcbcbcbU, 0xbebebebeU, 0x39393939U,

0x4a4a4a4aU, 0x4c4c4c4cU, 0x58585858U, 0xcfcfcfcfU,

0xd0d0d0d0U, 0xefefefefU, 0xaaaaaaaaU, 0xfbfbfbfbU,

0x43434343U, 0x4d4d4d4dU, 0x33333333U, 0x85858585U,

0x45454545U, 0xf9f9f9f9U, 0x02020202U, 0x7f7f7f7fU,

0x50505050U, 0x3c3c3c3cU, 0x9f9f9f9fU, 0xa8a8a8a8U,

0x51515151U, 0xa3a3a3a3U, 0x40404040U, 0x8f8f8f8fU,

0x92929292U, 0x9d9d9d9dU, 0x38383838U, 0xf5f5f5f5U,

0xbcbcbcbcU, 0xb6b6b6b6U, 0xdadadadaU, 0x21212121U,

0x10101010U, 0xffffffffU, 0xf3f3f3f3U, 0xd2d2d2d2U,

0xcdcdcdcdU, 0x0c0c0c0cU, 0x13131313U, 0xececececU,

0x5f5f5f5fU, 0x97979797U, 0x44444444U, 0x17171717U,

0xc4c4c4c4U, 0xa7a7a7a7U, 0x7e7e7e7eU, 0x3d3d3d3dU,

0x64646464U, 0x5d5d5d5dU, 0x19191919U, 0x73737373U,

0x60606060U, 0x81818181U, 0x4f4f4f4fU, 0xdcdcdcdcU,

0x22222222U, 0x2a2a2a2aU, 0x90909090U, 0x88888888U,

0x46464646U, 0xeeeeeeeeU, 0xb8b8b8b8U, 0x14141414U,

0xdedededeU, 0x5e5e5e5eU, 0x0b0b0b0bU, 0xdbdbdbdbU,

0xe0e0e0e0U, 0x32323232U, 0x3a3a3a3aU, 0x0a0a0a0aU,

0x49494949U, 0x06060606U, 0x24242424U, 0x5c5c5c5cU,

0xc2c2c2c2U, 0xd3d3d3d3U, 0xacacacacU, 0x62626262U,

0x91919191U, 0x95959595U, 0xe4e4e4e4U, 0x79797979U,

0xe7e7e7e7U, 0xc8c8c8c8U, 0x37373737U, 0x6d6d6d6dU,

0x8d8d8d8dU, 0xd5d5d5d5U, 0x4e4e4e4eU, 0xa9a9a9a9U,

0x6c6c6c6cU, 0x56565656U, 0xf4f4f4f4U, 0xeaeaeaeaU,

0x65656565U, 0x7a7a7a7aU, 0xaeaeaeaeU, 0x08080808U,

0xbabababaU, 0x78787878U, 0x25252525U, 0x2e2e2e2eU,

0x1c1c1c1cU, 0xa6a6a6a6U, 0xb4b4b4b4U, 0xc6c6c6c6U,

0xe8e8e8e8U, 0xddddddddU, 0x74747474U, 0x1f1f1f1fU,

0x4b4b4b4bU, 0xbdbdbdbdU, 0x8b8b8b8bU, 0x8a8a8a8aU,

0x70707070U, 0x3e3e3e3eU, 0xb5b5b5b5U, 0x66666666U,

0x48484848U, 0x03030303U, 0xf6f6f6f6U, 0x0e0e0e0eU,

0x61616161U, 0x35353535U, 0x57575757U, 0xb9b9b9b9U,

0x86868686U, 0xc1c1c1c1U, 0x1d1d1d1dU, 0x9e9e9e9eU,

0xe1e1e1e1U, 0xf8f8f8f8U, 0x98989898U, 0x11111111U,

0x69696969U, 0xd9d9d9d9U, 0x8e8e8e8eU, 0x94949494U,

0x9b9b9b9bU, 0x1e1e1e1eU, 0x87878787U, 0xe9e9e9e9U,

0xcecececeU, 0x55555555U, 0x28282828U, 0xdfdfdfdfU,

0x8c8c8c8cU, 0xa1a1a1a1U, 0x89898989U, 0x0d0d0d0dU,

0xbfbfbfbfU, 0xe6e6e6e6U, 0x42424242U, 0x68686868U,

0x41414141U, 0x99999999U, 0x2d2d2d2dU, 0x0f0f0f0fU,

0xb0b0b0b0U, 0x54545454U, 0xbbbbbbbbU, 0x16161616U,

};

static const u32 Td0[256] = {

0x51f4a750U, 0x7e416553U, 0x1a17a4c3U, 0x3a275e96U,

0x3bab6bcbU, 0x1f9d45f1U, 0xacfa58abU, 0x4be30393U,

0x2030fa55U, 0xad766df6U, 0x88cc7691U, 0xf5024c25U,

0x4fe5d7fcU, 0xc52acbd7U, 0x26354480U, 0xb562a38fU,

0xdeb15a49U, 0x25ba1b67U, 0x45ea0e98U, 0x5dfec0e1U,

0xc32f7502U, 0x814cf012U, 0x8d4697a3U, 0x6bd3f9c6U,

0x038f5fe7U, 0x15929c95U, 0xbf6d7aebU, 0x955259daU,

0xd4be832dU, 0x587421d3U, 0x49e06929U, 0x8ec9c844U,

0x75c2896aU, 0xf48e7978U, 0x99583e6bU, 0x27b971ddU,

0xbee14fb6U, 0xf088ad17U, 0xc920ac66U, 0x7dce3ab4U,

0x63df4a18U, 0xe51a3182U, 0x97513360U, 0x62537f45U,

0xb16477e0U, 0xbb6bae84U, 0xfe81a01cU, 0xf9082b94U,

0x70486858U, 0x8f45fd19U, 0x94de6c87U, 0x527bf8b7U,

0xab73d323U, 0x724b02e2U, 0xe31f8f57U, 0x6655ab2aU,

0xb2eb2807U, 0x2fb5c203U, 0x86c57b9aU, 0xd33708a5U,

0x302887f2U, 0x23bfa5b2U, 0x02036abaU, 0xed16825cU,

0x8acf1c2bU, 0xa779b492U, 0xf307f2f0U, 0x4e69e2a1U,

0x65daf4cdU, 0x0605bed5U, 0xd134621fU, 0xc4a6fe8aU,

0x342e539dU, 0xa2f355a0U, 0x058ae132U, 0xa4f6eb75U,

0x0b83ec39U, 0x4060efaaU, 0x5e719f06U, 0xbd6e1051U,

0x3e218af9U, 0x96dd063dU, 0xdd3e05aeU, 0x4de6bd46U,

0x91548db5U, 0x71c45d05U, 0x0406d46fU, 0x605015ffU,

0x1998fb24U, 0xd6bde997U, 0x894043ccU, 0x67d99e77U,

0xb0e842bdU, 0x07898b88U, 0xe7195b38U, 0x79c8eedbU,

0xa17c0a47U, 0x7c420fe9U, 0xf8841ec9U, 0x00000000U,

0x09808683U, 0x322bed48U, 0x1e1170acU, 0x6c5a724eU,

0xfd0efffbU, 0x0f853856U, 0x3daed51eU, 0x362d3927U,

0x0a0fd964U, 0x685ca621U, 0x9b5b54d1U, 0x24362e3aU,

0x0c0a67b1U, 0x9357e70fU, 0xb4ee96d2U, 0x1b9b919eU,

0x80c0c54fU, 0x61dc20a2U, 0x5a774b69U, 0x1c121a16U,

0xe293ba0aU, 0xc0a02ae5U, 0x3c22e043U, 0x121b171dU,

0x0e090d0bU, 0xf28bc7adU, 0x2db6a8b9U, 0x141ea9c8U,

0x57f11985U, 0xaf75074cU, 0xee99ddbbU, 0xa37f60fdU,

0xf701269fU, 0x5c72f5bcU, 0x44663bc5U, 0x5bfb7e34U,

0x8b432976U, 0xcb23c6dcU, 0xb6edfc68U, 0xb8e4f163U,

0xd731dccaU, 0x42638510U, 0x13972240U, 0x84c61120U,

0x854a247dU, 0xd2bb3df8U, 0xaef93211U, 0xc729a16dU,

0x1d9e2f4bU, 0xdcb230f3U, 0x0d8652ecU, 0x77c1e3d0U,

0x2bb3166cU, 0xa970b999U, 0x119448faU, 0x47e96422U,

0xa8fc8cc4U, 0xa0f03f1aU, 0x567d2cd8U, 0x223390efU,

0x87494ec7U, 0xd938d1c1U, 0x8ccaa2feU, 0x98d40b36U,

0xa6f581cfU, 0xa57ade28U, 0xdab78e26U, 0x3fadbfa4U,

0x2c3a9de4U, 0x5078920dU, 0x6a5fcc9bU, 0x547e4662U,

0xf68d13c2U, 0x90d8b8e8U, 0x2e39f75eU, 0x82c3aff5U,

0x9f5d80beU, 0x69d0937cU, 0x6fd52da9U, 0xcf2512b3U,

0xc8ac993bU, 0x10187da7U, 0xe89c636eU, 0xdb3bbb7bU,

0xcd267809U, 0x6e5918f4U, 0xec9ab701U, 0x834f9aa8U,

0xe6956e65U, 0xaaffe67eU, 0x21bccf08U, 0xef15e8e6U,

0xbae79bd9U, 0x4a6f36ceU, 0xea9f09d4U, 0x29b07cd6U,

0x31a4b2afU, 0x2a3f2331U, 0xc6a59430U, 0x35a266c0U,

0x744ebc37U, 0xfc82caa6U, 0xe090d0b0U, 0x33a7d815U,

0xf104984aU, 0x41ecdaf7U, 0x7fcd500eU, 0x1791f62fU,

0x764dd68dU, 0x43efb04dU, 0xccaa4d54U, 0xe49604dfU,

0x9ed1b5e3U, 0x4c6a881bU, 0xc12c1fb8U, 0x4665517fU,

0x9d5eea04U, 0x018c355dU, 0xfa877473U, 0xfb0b412eU,

0xb3671d5aU, 0x92dbd252U, 0xe9105633U, 0x6dd64713U,

0x9ad7618cU, 0x37a10c7aU, 0x59f8148eU, 0xeb133c89U,

0xcea927eeU, 0xb761c935U, 0xe11ce5edU, 0x7a47b13cU,

0x9cd2df59U, 0x55f2733fU, 0x1814ce79U, 0x73c737bfU,

0x53f7cdeaU, 0x5ffdaa5bU, 0xdf3d6f14U, 0x7844db86U,

0xcaaff381U, 0xb968c43eU, 0x3824342cU, 0xc2a3405fU,

0x161dc372U, 0xbce2250cU, 0x283c498bU, 0xff0d9541U,

0x39a80171U, 0x080cb3deU, 0xd8b4e49cU, 0x6456c190U,

0x7bcb8461U, 0xd532b670U, 0x486c5c74U, 0xd0b85742U,

};

static const u32 Td1[256] = {

0x5051f4a7U, 0x537e4165U, 0xc31a17a4U, 0x963a275eU,

0xcb3bab6bU, 0xf11f9d45U, 0xabacfa58U, 0x934be303U,

0x552030faU, 0xf6ad766dU, 0x9188cc76U, 0x25f5024cU,

0xfc4fe5d7U, 0xd7c52acbU, 0x80263544U, 0x8fb562a3U,

0x49deb15aU, 0x6725ba1bU, 0x9845ea0eU, 0xe15dfec0U,

0x02c32f75U, 0x12814cf0U, 0xa38d4697U, 0xc66bd3f9U,

0xe7038f5fU, 0x9515929cU, 0xebbf6d7aU, 0xda955259U,

0x2dd4be83U, 0xd3587421U, 0x2949e069U, 0x448ec9c8U,

0x6a75c289U, 0x78f48e79U, 0x6b99583eU, 0xdd27b971U,

0xb6bee14fU, 0x17f088adU, 0x66c920acU, 0xb47dce3aU,

0x1863df4aU, 0x82e51a31U, 0x60975133U, 0x4562537fU,

0xe0b16477U, 0x84bb6baeU, 0x1cfe81a0U, 0x94f9082bU,

0x58704868U, 0x198f45fdU, 0x8794de6cU, 0xb7527bf8U,

0x23ab73d3U, 0xe2724b02U, 0x57e31f8fU, 0x2a6655abU,

0x07b2eb28U, 0x032fb5c2U, 0x9a86c57bU, 0xa5d33708U,

0xf2302887U, 0xb223bfa5U, 0xba02036aU, 0x5ced1682U,

0x2b8acf1cU, 0x92a779b4U, 0xf0f307f2U, 0xa14e69e2U,

0xcd65daf4U, 0xd50605beU, 0x1fd13462U, 0x8ac4a6feU,

0x9d342e53U, 0xa0a2f355U, 0x32058ae1U, 0x75a4f6ebU,

0x390b83ecU, 0xaa4060efU, 0x065e719fU, 0x51bd6e10U,

0xf93e218aU, 0x3d96dd06U, 0xaedd3e05U, 0x464de6bdU,

0xb591548dU, 0x0571c45dU, 0x6f0406d4U, 0xff605015U,

0x241998fbU, 0x97d6bde9U, 0xcc894043U, 0x7767d99eU,

0xbdb0e842U, 0x8807898bU, 0x38e7195bU, 0xdb79c8eeU,

0x47a17c0aU, 0xe97c420fU, 0xc9f8841eU, 0x00000000U,

0x83098086U, 0x48322bedU, 0xac1e1170U, 0x4e6c5a72U,

0xfbfd0effU, 0x560f8538U, 0x1e3daed5U, 0x27362d39U,

0x640a0fd9U, 0x21685ca6U, 0xd19b5b54U, 0x3a24362eU,

0xb10c0a67U, 0x0f9357e7U, 0xd2b4ee96U, 0x9e1b9b91U,

0x4f80c0c5U, 0xa261dc20U, 0x695a774bU, 0x161c121aU,

0x0ae293baU, 0xe5c0a02aU, 0x433c22e0U, 0x1d121b17U,

0x0b0e090dU, 0xadf28bc7U, 0xb92db6a8U, 0xc8141ea9U,

0x8557f119U, 0x4caf7507U, 0xbbee99ddU, 0xfda37f60U,

0x9ff70126U, 0xbc5c72f5U, 0xc544663bU, 0x345bfb7eU,

0x768b4329U, 0xdccb23c6U, 0x68b6edfcU, 0x63b8e4f1U,

0xcad731dcU, 0x10426385U, 0x40139722U, 0x2084c611U,

0x7d854a24U, 0xf8d2bb3dU, 0x11aef932U, 0x6dc729a1U,

0x4b1d9e2fU, 0xf3dcb230U, 0xec0d8652U, 0xd077c1e3U,

0x6c2bb316U, 0x99a970b9U, 0xfa119448U, 0x2247e964U,

0xc4a8fc8cU, 0x1aa0f03fU, 0xd8567d2cU, 0xef223390U,

0xc787494eU, 0xc1d938d1U, 0xfe8ccaa2U, 0x3698d40bU,

0xcfa6f581U, 0x28a57adeU, 0x26dab78eU, 0xa43fadbfU,

0xe42c3a9dU, 0x0d507892U, 0x9b6a5fccU, 0x62547e46U,

0xc2f68d13U, 0xe890d8b8U, 0x5e2e39f7U, 0xf582c3afU,

0xbe9f5d80U, 0x7c69d093U, 0xa96fd52dU, 0xb3cf2512U,

0x3bc8ac99U, 0xa710187dU, 0x6ee89c63U, 0x7bdb3bbbU,

0x09cd2678U, 0xf46e5918U, 0x01ec9ab7U, 0xa8834f9aU,

0x65e6956eU, 0x7eaaffe6U, 0x0821bccfU, 0xe6ef15e8U,

0xd9bae79bU, 0xce4a6f36U, 0xd4ea9f09U, 0xd629b07cU,

0xaf31a4b2U, 0x312a3f23U, 0x30c6a594U, 0xc035a266U,

0x37744ebcU, 0xa6fc82caU, 0xb0e090d0U, 0x1533a7d8U,

0x4af10498U, 0xf741ecdaU, 0x0e7fcd50U, 0x2f1791f6U,

0x8d764dd6U, 0x4d43efb0U, 0x54ccaa4dU, 0xdfe49604U,

0xe39ed1b5U, 0x1b4c6a88U, 0xb8c12c1fU, 0x7f466551U,

0x049d5eeaU, 0x5d018c35U, 0x73fa8774U, 0x2efb0b41U,

0x5ab3671dU, 0x5292dbd2U, 0x33e91056U, 0x136dd647U,

0x8c9ad761U, 0x7a37a10cU, 0x8e59f814U, 0x89eb133cU,

0xeecea927U, 0x35b761c9U, 0xede11ce5U, 0x3c7a47b1U,

0x599cd2dfU, 0x3f55f273U, 0x791814ceU, 0xbf73c737U,

0xea53f7cdU, 0x5b5ffdaaU, 0x14df3d6fU, 0x867844dbU,

0x81caaff3U, 0x3eb968c4U, 0x2c382434U, 0x5fc2a340U,

0x72161dc3U, 0x0cbce225U, 0x8b283c49U, 0x41ff0d95U,

0x7139a801U, 0xde080cb3U, 0x9cd8b4e4U, 0x906456c1U,

0x617bcb84U, 0x70d532b6U, 0x74486c5cU, 0x42d0b857U,

};

static const u32 Td2[256] = {

0xa75051f4U, 0x65537e41U, 0xa4c31a17U, 0x5e963a27U,

0x6bcb3babU, 0x45f11f9dU, 0x58abacfaU, 0x03934be3U,

0xfa552030U, 0x6df6ad76U, 0x769188ccU, 0x4c25f502U,

0xd7fc4fe5U, 0xcbd7c52aU, 0x44802635U, 0xa38fb562U,

0x5a49deb1U, 0x1b6725baU, 0x0e9845eaU, 0xc0e15dfeU,

0x7502c32fU, 0xf012814cU, 0x97a38d46U, 0xf9c66bd3U,

0x5fe7038fU, 0x9c951592U, 0x7aebbf6dU, 0x59da9552U,

0x832dd4beU, 0x21d35874U, 0x692949e0U, 0xc8448ec9U,

0x896a75c2U, 0x7978f48eU, 0x3e6b9958U, 0x71dd27b9U,

0x4fb6bee1U, 0xad17f088U, 0xac66c920U, 0x3ab47dceU,

0x4a1863dfU, 0x3182e51aU, 0x33609751U, 0x7f456253U,

0x77e0b164U, 0xae84bb6bU, 0xa01cfe81U, 0x2b94f908U,

0x68587048U, 0xfd198f45U, 0x6c8794deU, 0xf8b7527bU,

0xd323ab73U, 0x02e2724bU, 0x8f57e31fU, 0xab2a6655U,

0x2807b2ebU, 0xc2032fb5U, 0x7b9a86c5U, 0x08a5d337U,

0x87f23028U, 0xa5b223bfU, 0x6aba0203U, 0x825ced16U,

0x1c2b8acfU, 0xb492a779U, 0xf2f0f307U, 0xe2a14e69U,

0xf4cd65daU, 0xbed50605U, 0x621fd134U, 0xfe8ac4a6U,

0x539d342eU, 0x55a0a2f3U, 0xe132058aU, 0xeb75a4f6U,

0xec390b83U, 0xefaa4060U, 0x9f065e71U, 0x1051bd6eU,

0x8af93e21U, 0x063d96ddU, 0x05aedd3eU, 0xbd464de6U,

0x8db59154U, 0x5d0571c4U, 0xd46f0406U, 0x15ff6050U,

0xfb241998U, 0xe997d6bdU, 0x43cc8940U, 0x9e7767d9U,

0x42bdb0e8U, 0x8b880789U, 0x5b38e719U, 0xeedb79c8U,

0x0a47a17cU, 0x0fe97c42U, 0x1ec9f884U, 0x00000000U,

0x86830980U, 0xed48322bU, 0x70ac1e11U, 0x724e6c5aU,

0xfffbfd0eU, 0x38560f85U, 0xd51e3daeU, 0x3927362dU,

0xd9640a0fU, 0xa621685cU, 0x54d19b5bU, 0x2e3a2436U,

0x67b10c0aU, 0xe70f9357U, 0x96d2b4eeU, 0x919e1b9bU,

0xc54f80c0U, 0x20a261dcU, 0x4b695a77U, 0x1a161c12U,

0xba0ae293U, 0x2ae5c0a0U, 0xe0433c22U, 0x171d121bU,

0x0d0b0e09U, 0xc7adf28bU, 0xa8b92db6U, 0xa9c8141eU,

0x198557f1U, 0x074caf75U, 0xddbbee99U, 0x60fda37fU,

0x269ff701U, 0xf5bc5c72U, 0x3bc54466U, 0x7e345bfbU,

0x29768b43U, 0xc6dccb23U, 0xfc68b6edU, 0xf163b8e4U,

0xdccad731U, 0x85104263U, 0x22401397U, 0x112084c6U,

0x247d854aU, 0x3df8d2bbU, 0x3211aef9U, 0xa16dc729U,

0x2f4b1d9eU, 0x30f3dcb2U, 0x52ec0d86U, 0xe3d077c1U,

0x166c2bb3U, 0xb999a970U, 0x48fa1194U, 0x642247e9U,

0x8cc4a8fcU, 0x3f1aa0f0U, 0x2cd8567dU, 0x90ef2233U,

0x4ec78749U, 0xd1c1d938U, 0xa2fe8ccaU, 0x0b3698d4U,

0x81cfa6f5U, 0xde28a57aU, 0x8e26dab7U, 0xbfa43fadU,

0x9de42c3aU, 0x920d5078U, 0xcc9b6a5fU, 0x4662547eU,

0x13c2f68dU, 0xb8e890d8U, 0xf75e2e39U, 0xaff582c3U,

0x80be9f5dU, 0x937c69d0U, 0x2da96fd5U, 0x12b3cf25U,

0x993bc8acU, 0x7da71018U, 0x636ee89cU, 0xbb7bdb3bU,

0x7809cd26U, 0x18f46e59U, 0xb701ec9aU, 0x9aa8834fU,

0x6e65e695U, 0xe67eaaffU, 0xcf0821bcU, 0xe8e6ef15U,

0x9bd9bae7U, 0x36ce4a6fU, 0x09d4ea9fU, 0x7cd629b0U,

0xb2af31a4U, 0x23312a3fU, 0x9430c6a5U, 0x66c035a2U,

0xbc37744eU, 0xcaa6fc82U, 0xd0b0e090U, 0xd81533a7U,

0x984af104U, 0xdaf741ecU, 0x500e7fcdU, 0xf62f1791U,

0xd68d764dU, 0xb04d43efU, 0x4d54ccaaU, 0x04dfe496U,

0xb5e39ed1U, 0x881b4c6aU, 0x1fb8c12cU, 0x517f4665U,

0xea049d5eU, 0x355d018cU, 0x7473fa87U, 0x412efb0bU,

0x1d5ab367U, 0xd25292dbU, 0x5633e910U, 0x47136dd6U,

0x618c9ad7U, 0x0c7a37a1U, 0x148e59f8U, 0x3c89eb13U,

0x27eecea9U, 0xc935b761U, 0xe5ede11cU, 0xb13c7a47U,

0xdf599cd2U, 0x733f55f2U, 0xce791814U, 0x37bf73c7U,

0xcdea53f7U, 0xaa5b5ffdU, 0x6f14df3dU, 0xdb867844U,

0xf381caafU, 0xc43eb968U, 0x342c3824U, 0x405fc2a3U,

0xc372161dU, 0x250cbce2U, 0x498b283cU, 0x9541ff0dU,

0x017139a8U, 0xb3de080cU, 0xe49cd8b4U, 0xc1906456U,

0x84617bcbU, 0xb670d532U, 0x5c74486cU, 0x5742d0b8U,

};

static const u32 Td3[256] = {

0xf4a75051U, 0x4165537eU, 0x17a4c31aU, 0x275e963aU,

0xab6bcb3bU, 0x9d45f11fU, 0xfa58abacU, 0xe303934bU,

0x30fa5520U, 0x766df6adU, 0xcc769188U, 0x024c25f5U,

0xe5d7fc4fU, 0x2acbd7c5U, 0x35448026U, 0x62a38fb5U,

0xb15a49deU, 0xba1b6725U, 0xea0e9845U, 0xfec0e15dU,

0x2f7502c3U, 0x4cf01281U, 0x4697a38dU, 0xd3f9c66bU,

0x8f5fe703U, 0x929c9515U, 0x6d7aebbfU, 0x5259da95U,

0xbe832dd4U, 0x7421d358U, 0xe0692949U, 0xc9c8448eU,

0xc2896a75U, 0x8e7978f4U, 0x583e6b99U, 0xb971dd27U,

0xe14fb6beU, 0x88ad17f0U, 0x20ac66c9U, 0xce3ab47dU,

0xdf4a1863U, 0x1a3182e5U, 0x51336097U, 0x537f4562U,

0x6477e0b1U, 0x6bae84bbU, 0x81a01cfeU, 0x082b94f9U,

0x48685870U, 0x45fd198fU, 0xde6c8794U, 0x7bf8b752U,

0x73d323abU, 0x4b02e272U, 0x1f8f57e3U, 0x55ab2a66U,

0xeb2807b2U, 0xb5c2032fU, 0xc57b9a86U, 0x3708a5d3U,

0x2887f230U, 0xbfa5b223U, 0x036aba02U, 0x16825cedU,

0xcf1c2b8aU, 0x79b492a7U, 0x07f2f0f3U, 0x69e2a14eU,

0xdaf4cd65U, 0x05bed506U, 0x34621fd1U, 0xa6fe8ac4U,

0x2e539d34U, 0xf355a0a2U, 0x8ae13205U, 0xf6eb75a4U,

0x83ec390bU, 0x60efaa40U, 0x719f065eU, 0x6e1051bdU,

0x218af93eU, 0xdd063d96U, 0x3e05aeddU, 0xe6bd464dU,

0x548db591U, 0xc45d0571U, 0x06d46f04U, 0x5015ff60U,

0x98fb2419U, 0xbde997d6U, 0x4043cc89U, 0xd99e7767U,

0xe842bdb0U, 0x898b8807U, 0x195b38e7U, 0xc8eedb79U,

0x7c0a47a1U, 0x420fe97cU, 0x841ec9f8U, 0x00000000U,

0x80868309U, 0x2bed4832U, 0x1170ac1eU, 0x5a724e6cU,

0x0efffbfdU, 0x8538560fU, 0xaed51e3dU, 0x2d392736U,

0x0fd9640aU, 0x5ca62168U, 0x5b54d19bU, 0x362e3a24U,

0x0a67b10cU, 0x57e70f93U, 0xee96d2b4U, 0x9b919e1bU,

0xc0c54f80U, 0xdc20a261U, 0x774b695aU, 0x121a161cU,

0x93ba0ae2U, 0xa02ae5c0U, 0x22e0433cU, 0x1b171d12U,

0x090d0b0eU, 0x8bc7adf2U, 0xb6a8b92dU, 0x1ea9c814U,

0xf1198557U, 0x75074cafU, 0x99ddbbeeU, 0x7f60fda3U,

0x01269ff7U, 0x72f5bc5cU, 0x663bc544U, 0xfb7e345bU,

0x4329768bU, 0x23c6dccbU, 0xedfc68b6U, 0xe4f163b8U,

0x31dccad7U, 0x63851042U, 0x97224013U, 0xc6112084U,

0x4a247d85U, 0xbb3df8d2U, 0xf93211aeU, 0x29a16dc7U,

0x9e2f4b1dU, 0xb230f3dcU, 0x8652ec0dU, 0xc1e3d077U,

0xb3166c2bU, 0x70b999a9U, 0x9448fa11U, 0xe9642247U,

0xfc8cc4a8U, 0xf03f1aa0U, 0x7d2cd856U, 0x3390ef22U,

0x494ec787U, 0x38d1c1d9U, 0xcaa2fe8cU, 0xd40b3698U,

0xf581cfa6U, 0x7ade28a5U, 0xb78e26daU, 0xadbfa43fU,

0x3a9de42cU, 0x78920d50U, 0x5fcc9b6aU, 0x7e466254U,

0x8d13c2f6U, 0xd8b8e890U, 0x39f75e2eU, 0xc3aff582U,

0x5d80be9fU, 0xd0937c69U, 0xd52da96fU, 0x2512b3cfU,

0xac993bc8U, 0x187da710U, 0x9c636ee8U, 0x3bbb7bdbU,

0x267809cdU, 0x5918f46eU, 0x9ab701ecU, 0x4f9aa883U,

0x956e65e6U, 0xffe67eaaU, 0xbccf0821U, 0x15e8e6efU,

0xe79bd9baU, 0x6f36ce4aU, 0x9f09d4eaU, 0xb07cd629U,

0xa4b2af31U, 0x3f23312aU, 0xa59430c6U, 0xa266c035U,

0x4ebc3774U, 0x82caa6fcU, 0x90d0b0e0U, 0xa7d81533U,

0x04984af1U, 0xecdaf741U, 0xcd500e7fU, 0x91f62f17U,

0x4dd68d76U, 0xefb04d43U, 0xaa4d54ccU, 0x9604dfe4U,

0xd1b5e39eU, 0x6a881b4cU, 0x2c1fb8c1U, 0x65517f46U,

0x5eea049dU, 0x8c355d01U, 0x877473faU, 0x0b412efbU,

0x671d5ab3U, 0xdbd25292U, 0x105633e9U, 0xd647136dU,

0xd7618c9aU, 0xa10c7a37U, 0xf8148e59U, 0x133c89ebU,

0xa927eeceU, 0x61c935b7U, 0x1ce5ede1U, 0x47b13c7aU,

0xd2df599cU, 0xf2733f55U, 0x14ce7918U, 0xc737bf73U,

0xf7cdea53U, 0xfdaa5b5fU, 0x3d6f14dfU, 0x44db8678U,

0xaff381caU, 0x68c43eb9U, 0x24342c38U, 0xa3405fc2U,

0x1dc37216U, 0xe2250cbcU, 0x3c498b28U, 0x0d9541ffU,

0xa8017139U, 0x0cb3de08U, 0xb4e49cd8U, 0x56c19064U,

0xcb84617bU, 0x32b670d5U, 0x6c5c7448U, 0xb85742d0U,

};

static const u32 Td4[256] = {

0x52525252U, 0x09090909U, 0x6a6a6a6aU, 0xd5d5d5d5U,

0x30303030U, 0x36363636U, 0xa5a5a5a5U, 0x38383838U,

0xbfbfbfbfU, 0x40404040U, 0xa3a3a3a3U, 0x9e9e9e9eU,

0x81818181U, 0xf3f3f3f3U, 0xd7d7d7d7U, 0xfbfbfbfbU,

0x7c7c7c7cU, 0xe3e3e3e3U, 0x39393939U, 0x82828282U,

0x9b9b9b9bU, 0x2f2f2f2fU, 0xffffffffU, 0x87878787U,

0x34343434U, 0x8e8e8e8eU, 0x43434343U, 0x44444444U,

0xc4c4c4c4U, 0xdedededeU, 0xe9e9e9e9U, 0xcbcbcbcbU,

0x54545454U, 0x7b7b7b7bU, 0x94949494U, 0x32323232U,

0xa6a6a6a6U, 0xc2c2c2c2U, 0x23232323U, 0x3d3d3d3dU,

0xeeeeeeeeU, 0x4c4c4c4cU, 0x95959595U, 0x0b0b0b0bU,

0x42424242U, 0xfafafafaU, 0xc3c3c3c3U, 0x4e4e4e4eU,

0x08080808U, 0x2e2e2e2eU, 0xa1a1a1a1U, 0x66666666U,

0x28282828U, 0xd9d9d9d9U, 0x24242424U, 0xb2b2b2b2U,

0x76767676U, 0x5b5b5b5bU, 0xa2a2a2a2U, 0x49494949U,

0x6d6d6d6dU, 0x8b8b8b8bU, 0xd1d1d1d1U, 0x25252525U,

0x72727272U, 0xf8f8f8f8U, 0xf6f6f6f6U, 0x64646464U,

0x86868686U, 0x68686868U, 0x98989898U, 0x16161616U,

0xd4d4d4d4U, 0xa4a4a4a4U, 0x5c5c5c5cU, 0xccccccccU,

0x5d5d5d5dU, 0x65656565U, 0xb6b6b6b6U, 0x92929292U,

0x6c6c6c6cU, 0x70707070U, 0x48484848U, 0x50505050U,

0xfdfdfdfdU, 0xededededU, 0xb9b9b9b9U, 0xdadadadaU,

0x5e5e5e5eU, 0x15151515U, 0x46464646U, 0x57575757U,

0xa7a7a7a7U, 0x8d8d8d8dU, 0x9d9d9d9dU, 0x84848484U,

0x90909090U, 0xd8d8d8d8U, 0xababababU, 0x00000000U,

0x8c8c8c8cU, 0xbcbcbcbcU, 0xd3d3d3d3U, 0x0a0a0a0aU,

0xf7f7f7f7U, 0xe4e4e4e4U, 0x58585858U, 0x05050505U,

0xb8b8b8b8U, 0xb3b3b3b3U, 0x45454545U, 0x06060606U,

0xd0d0d0d0U, 0x2c2c2c2cU, 0x1e1e1e1eU, 0x8f8f8f8fU,

0xcacacacaU, 0x3f3f3f3fU, 0x0f0f0f0fU, 0x02020202U,

0xc1c1c1c1U, 0xafafafafU, 0xbdbdbdbdU, 0x03030303U,

0x01010101U, 0x13131313U, 0x8a8a8a8aU, 0x6b6b6b6bU,

0x3a3a3a3aU, 0x91919191U, 0x11111111U, 0x41414141U,

0x4f4f4f4fU, 0x67676767U, 0xdcdcdcdcU, 0xeaeaeaeaU,

0x97979797U, 0xf2f2f2f2U, 0xcfcfcfcfU, 0xcecececeU,

0xf0f0f0f0U, 0xb4b4b4b4U, 0xe6e6e6e6U, 0x73737373U,

0x96969696U, 0xacacacacU, 0x74747474U, 0x22222222U,

0xe7e7e7e7U, 0xadadadadU, 0x35353535U, 0x85858585U,

0xe2e2e2e2U, 0xf9f9f9f9U, 0x37373737U, 0xe8e8e8e8U,

0x1c1c1c1cU, 0x75757575U, 0xdfdfdfdfU, 0x6e6e6e6eU,

0x47474747U, 0xf1f1f1f1U, 0x1a1a1a1aU, 0x71717171U,

0x1d1d1d1dU, 0x29292929U, 0xc5c5c5c5U, 0x89898989U,

0x6f6f6f6fU, 0xb7b7b7b7U, 0x62626262U, 0x0e0e0e0eU,

0xaaaaaaaaU, 0x18181818U, 0xbebebebeU, 0x1b1b1b1bU,

0xfcfcfcfcU, 0x56565656U, 0x3e3e3e3eU, 0x4b4b4b4bU,

0xc6c6c6c6U, 0xd2d2d2d2U, 0x79797979U, 0x20202020U,

0x9a9a9a9aU, 0xdbdbdbdbU, 0xc0c0c0c0U, 0xfefefefeU,

0x78787878U, 0xcdcdcdcdU, 0x5a5a5a5aU, 0xf4f4f4f4U,

0x1f1f1f1fU, 0xddddddddU, 0xa8a8a8a8U, 0x33333333U,

0x88888888U, 0x07070707U, 0xc7c7c7c7U, 0x31313131U,

0xb1b1b1b1U, 0x12121212U, 0x10101010U, 0x59595959U,

0x27272727U, 0x80808080U, 0xececececU, 0x5f5f5f5fU,

0x60606060U, 0x51515151U, 0x7f7f7f7fU, 0xa9a9a9a9U,

0x19191919U, 0xb5b5b5b5U, 0x4a4a4a4aU, 0x0d0d0d0dU,

0x2d2d2d2dU, 0xe5e5e5e5U, 0x7a7a7a7aU, 0x9f9f9f9fU,

0x93939393U, 0xc9c9c9c9U, 0x9c9c9c9cU, 0xefefefefU,

0xa0a0a0a0U, 0xe0e0e0e0U, 0x3b3b3b3bU, 0x4d4d4d4dU,

0xaeaeaeaeU, 0x2a2a2a2aU, 0xf5f5f5f5U, 0xb0b0b0b0U,

0xc8c8c8c8U, 0xebebebebU, 0xbbbbbbbbU, 0x3c3c3c3cU,

0x83838383U, 0x53535353U, 0x99999999U, 0x61616161U,

0x17171717U, 0x2b2b2b2bU, 0x04040404U, 0x7e7e7e7eU,

0xbabababaU, 0x77777777U, 0xd6d6d6d6U, 0x26262626U,

0xe1e1e1e1U, 0x69696969U, 0x14141414U, 0x63636363U,

0x55555555U, 0x21212121U, 0x0c0c0c0cU, 0x7d7d7d7dU,

};

static const u32 rcon[] = {

0x01000000, 0x02000000, 0x04000000, 0x08000000,

0x10000000, 0x20000000, 0x40000000, 0x80000000,

0x1B000000, 0x36000000, /\* for 128-bit blocks, Rijndael never uses more than 10 rcon values \*/

};

#define SWAP(x) (\_lrotl(x, 8) & 0x00ff00ff | \_lrotr(x, 8) & 0xff00ff00)

#ifdef \_MSC\_VER

#define GETU32(p) SWAP(\*((u32 \*)(p)))

#define PUTU32(ct, st) { \*((u32 \*)(ct)) = SWAP((st)); }

#else

#define GETU32(pt) (((u32)(pt)[0] << 24) ^ ((u32)(pt)[1] << 16) ^ ((u32)(pt)[2] << 8) ^ ((u32)(pt)[3]))

#define PUTU32(ct, st) { (ct)[0] = (u8)((st) >> 24); (ct)[1] = (u8)((st) >> 16); (ct)[2] = (u8)((st) >> 8); (ct)[3] = (u8)(st); }

#endif

/\*\*

\* Expand the cipher key into the encryption key schedule.

\*

\* @return the number of rounds for the given cipher key size.

\*/

static int rijndaelKeySetupEnc(u32 rk[/\*4\*(Nr + 1)\*/], const u8 cipherKey[], int keyBits) {

int i = 0;

u32 temp;

rk[0] = GETU32(cipherKey );

rk[1] = GETU32(cipherKey + 4);

rk[2] = GETU32(cipherKey + 8);

rk[3] = GETU32(cipherKey + 12);

if (keyBits == 128) {

for (;;) {

temp = rk[3];

rk[4] = rk[0] ^

(Te4[(temp >> 16) & 0xff] & 0xff000000) ^

(Te4[(temp >> 8) & 0xff] & 0x00ff0000) ^

(Te4[(temp ) & 0xff] & 0x0000ff00) ^

(Te4[(temp >> 24) ] & 0x000000ff) ^

rcon[i];

rk[5] = rk[1] ^ rk[4];

rk[6] = rk[2] ^ rk[5];

rk[7] = rk[3] ^ rk[6];

if (++i == 10) {

return 10;

}

rk += 4;

}

}

rk[4] = GETU32(cipherKey + 16);

rk[5] = GETU32(cipherKey + 20);

if (keyBits == 192) {

for (;;) {

temp = rk[ 5];

rk[ 6] = rk[ 0] ^

(Te4[(temp >> 16) & 0xff] & 0xff000000) ^

(Te4[(temp >> 8) & 0xff] & 0x00ff0000) ^

(Te4[(temp ) & 0xff] & 0x0000ff00) ^

(Te4[(temp >> 24) ] & 0x000000ff) ^

rcon[i];

rk[ 7] = rk[ 1] ^ rk[ 6];

rk[ 8] = rk[ 2] ^ rk[ 7];

rk[ 9] = rk[ 3] ^ rk[ 8];

if (++i == 8) {

return 12;

}

rk[10] = rk[ 4] ^ rk[ 9];

rk[11] = rk[ 5] ^ rk[10];

rk += 6;

}

}

rk[6] = GETU32(cipherKey + 24);

rk[7] = GETU32(cipherKey + 28);

if (keyBits == 256) {

for (;;) {

temp = rk[ 7];

rk[ 8] = rk[ 0] ^

(Te4[(temp >> 16) & 0xff] & 0xff000000) ^

(Te4[(temp >> 8) & 0xff] & 0x00ff0000) ^

(Te4[(temp ) & 0xff] & 0x0000ff00) ^

(Te4[(temp >> 24) ] & 0x000000ff) ^

rcon[i];

rk[ 9] = rk[ 1] ^ rk[ 8];

rk[10] = rk[ 2] ^ rk[ 9];

rk[11] = rk[ 3] ^ rk[10];

if (++i == 7) {

return 14;

}

temp = rk[11];

rk[12] = rk[ 4] ^

(Te4[(temp >> 24) ] & 0xff000000) ^

(Te4[(temp >> 16) & 0xff] & 0x00ff0000) ^

(Te4[(temp >> 8) & 0xff] & 0x0000ff00) ^

(Te4[(temp ) & 0xff] & 0x000000ff);

rk[13] = rk[ 5] ^ rk[12];

rk[14] = rk[ 6] ^ rk[13];

rk[15] = rk[ 7] ^ rk[14];

rk += 8;

}

}

return 0;

}

/\*\*

\* Expand the cipher key into the decryption key schedule.

\*

\* @return the number of rounds for the given cipher key size.

\*/

static int rijndaelKeySetupDec(u32 rk[/\*4\*(Nr + 1)\*/], const u8 cipherKey[], int keyBits) {

int Nr, i, j;

u32 temp;

/\* expand the cipher key: \*/

Nr = rijndaelKeySetupEnc(rk, cipherKey, keyBits);

/\* invert the order of the round keys: \*/

for (i = 0, j = 4\*Nr; i < j; i += 4, j -= 4) {

temp = rk[i ]; rk[i ] = rk[j ]; rk[j ] = temp;

temp = rk[i + 1]; rk[i + 1] = rk[j + 1]; rk[j + 1] = temp;

temp = rk[i + 2]; rk[i + 2] = rk[j + 2]; rk[j + 2] = temp;

temp = rk[i + 3]; rk[i + 3] = rk[j + 3]; rk[j + 3] = temp;

}

/\* apply the inverse MixColumn transform to all round keys but the first and the last: \*/

for (i = 1; i < Nr; i++) {

rk += 4;

rk[0] =

Td0[Te4[(rk[0] >> 24) ] & 0xff] ^

Td1[Te4[(rk[0] >> 16) & 0xff] & 0xff] ^

Td2[Te4[(rk[0] >> 8) & 0xff] & 0xff] ^

Td3[Te4[(rk[0] ) & 0xff] & 0xff];

rk[1] =

Td0[Te4[(rk[1] >> 24) ] & 0xff] ^

Td1[Te4[(rk[1] >> 16) & 0xff] & 0xff] ^

Td2[Te4[(rk[1] >> 8) & 0xff] & 0xff] ^

Td3[Te4[(rk[1] ) & 0xff] & 0xff];

rk[2] =

Td0[Te4[(rk[2] >> 24) ] & 0xff] ^

Td1[Te4[(rk[2] >> 16) & 0xff] & 0xff] ^

Td2[Te4[(rk[2] >> 8) & 0xff] & 0xff] ^

Td3[Te4[(rk[2] ) & 0xff] & 0xff];

rk[3] =

Td0[Te4[(rk[3] >> 24) ] & 0xff] ^

Td1[Te4[(rk[3] >> 16) & 0xff] & 0xff] ^

Td2[Te4[(rk[3] >> 8) & 0xff] & 0xff] ^

Td3[Te4[(rk[3] ) & 0xff] & 0xff];

}

return Nr;

}

static void rijndaelEncrypt(u32 rk[/\*4\*(Nr + 1)\*/], int Nr, const u8 pt[16], u8 ct[16]) {

u32 s0, s1, s2, s3, t0, t1, t2, t3;

#ifndef FULL\_UNROLL

int r;

#endif /\* ?FULL\_UNROLL \*/

/\*

\* map byte array block to cipher state

\* and add initial round key:

\*/

s0 = GETU32(pt ) ^ rk[0];

s1 = GETU32(pt + 4) ^ rk[1];

s2 = GETU32(pt + 8) ^ rk[2];

s3 = GETU32(pt + 12) ^ rk[3];

#ifdef FULL\_UNROLL

/\* round 1: \*/

t0 = Te0[s0 >> 24] ^ Te1[(s1 >> 16) & 0xff] ^ Te2[(s2 >> 8) & 0xff] ^ Te3[s3 & 0xff] ^ rk[ 4];

t1 = Te0[s1 >> 24] ^ Te1[(s2 >> 16) & 0xff] ^ Te2[(s3 >> 8) & 0xff] ^ Te3[s0 & 0xff] ^ rk[ 5];

t2 = Te0[s2 >> 24] ^ Te1[(s3 >> 16) & 0xff] ^ Te2[(s0 >> 8) & 0xff] ^ Te3[s1 & 0xff] ^ rk[ 6];

t3 = Te0[s3 >> 24] ^ Te1[(s0 >> 16) & 0xff] ^ Te2[(s1 >> 8) & 0xff] ^ Te3[s2 & 0xff] ^ rk[ 7];

/\* round 2: \*/

s0 = Te0[t0 >> 24] ^ Te1[(t1 >> 16) & 0xff] ^ Te2[(t2 >> 8) & 0xff] ^ Te3[t3 & 0xff] ^ rk[ 8];

s1 = Te0[t1 >> 24] ^ Te1[(t2 >> 16) & 0xff] ^ Te2[(t3 >> 8) & 0xff] ^ Te3[t0 & 0xff] ^ rk[ 9];

s2 = Te0[t2 >> 24] ^ Te1[(t3 >> 16) & 0xff] ^ Te2[(t0 >> 8) & 0xff] ^ Te3[t1 & 0xff] ^ rk[10];

s3 = Te0[t3 >> 24] ^ Te1[(t0 >> 16) & 0xff] ^ Te2[(t1 >> 8) & 0xff] ^ Te3[t2 & 0xff] ^ rk[11];

/\* round 3: \*/

t0 = Te0[s0 >> 24] ^ Te1[(s1 >> 16) & 0xff] ^ Te2[(s2 >> 8) & 0xff] ^ Te3[s3 & 0xff] ^ rk[12];

t1 = Te0[s1 >> 24] ^ Te1[(s2 >> 16) & 0xff] ^ Te2[(s3 >> 8) & 0xff] ^ Te3[s0 & 0xff] ^ rk[13];

t2 = Te0[s2 >> 24] ^ Te1[(s3 >> 16) & 0xff] ^ Te2[(s0 >> 8) & 0xff] ^ Te3[s1 & 0xff] ^ rk[14];

t3 = Te0[s3 >> 24] ^ Te1[(s0 >> 16) & 0xff] ^ Te2[(s1 >> 8) & 0xff] ^ Te3[s2 & 0xff] ^ rk[15];

/\* round 4: \*/

s0 = Te0[t0 >> 24] ^ Te1[(t1 >> 16) & 0xff] ^ Te2[(t2 >> 8) & 0xff] ^ Te3[t3 & 0xff] ^ rk[16];

s1 = Te0[t1 >> 24] ^ Te1[(t2 >> 16) & 0xff] ^ Te2[(t3 >> 8) & 0xff] ^ Te3[t0 & 0xff] ^ rk[17];

s2 = Te0[t2 >> 24] ^ Te1[(t3 >> 16) & 0xff] ^ Te2[(t0 >> 8) & 0xff] ^ Te3[t1 & 0xff] ^ rk[18];

s3 = Te0[t3 >> 24] ^ Te1[(t0 >> 16) & 0xff] ^ Te2[(t1 >> 8) & 0xff] ^ Te3[t2 & 0xff] ^ rk[19];

/\* round 5: \*/

t0 = Te0[s0 >> 24] ^ Te1[(s1 >> 16) & 0xff] ^ Te2[(s2 >> 8) & 0xff] ^ Te3[s3 & 0xff] ^ rk[20];

t1 = Te0[s1 >> 24] ^ Te1[(s2 >> 16) & 0xff] ^ Te2[(s3 >> 8) & 0xff] ^ Te3[s0 & 0xff] ^ rk[21];

t2 = Te0[s2 >> 24] ^ Te1[(s3 >> 16) & 0xff] ^ Te2[(s0 >> 8) & 0xff] ^ Te3[s1 & 0xff] ^ rk[22];

t3 = Te0[s3 >> 24] ^ Te1[(s0 >> 16) & 0xff] ^ Te2[(s1 >> 8) & 0xff] ^ Te3[s2 & 0xff] ^ rk[23];

/\* round 6: \*/

s0 = Te0[t0 >> 24] ^ Te1[(t1 >> 16) & 0xff] ^ Te2[(t2 >> 8) & 0xff] ^ Te3[t3 & 0xff] ^ rk[24];

s1 = Te0[t1 >> 24] ^ Te1[(t2 >> 16) & 0xff] ^ Te2[(t3 >> 8) & 0xff] ^ Te3[t0 & 0xff] ^ rk[25];

s2 = Te0[t2 >> 24] ^ Te1[(t3 >> 16) & 0xff] ^ Te2[(t0 >> 8) & 0xff] ^ Te3[t1 & 0xff] ^ rk[26];

s3 = Te0[t3 >> 24] ^ Te1[(t0 >> 16) & 0xff] ^ Te2[(t1 >> 8) & 0xff] ^ Te3[t2 & 0xff] ^ rk[27];

/\* round 7: \*/

t0 = Te0[s0 >> 24] ^ Te1[(s1 >> 16) & 0xff] ^ Te2[(s2 >> 8) & 0xff] ^ Te3[s3 & 0xff] ^ rk[28];

t1 = Te0[s1 >> 24] ^ Te1[(s2 >> 16) & 0xff] ^ Te2[(s3 >> 8) & 0xff] ^ Te3[s0 & 0xff] ^ rk[29];

t2 = Te0[s2 >> 24] ^ Te1[(s3 >> 16) & 0xff] ^ Te2[(s0 >> 8) & 0xff] ^ Te3[s1 & 0xff] ^ rk[30];

t3 = Te0[s3 >> 24] ^ Te1[(s0 >> 16) & 0xff] ^ Te2[(s1 >> 8) & 0xff] ^ Te3[s2 & 0xff] ^ rk[31];

/\* round 8: \*/

s0 = Te0[t0 >> 24] ^ Te1[(t1 >> 16) & 0xff] ^ Te2[(t2 >> 8) & 0xff] ^ Te3[t3 & 0xff] ^ rk[32];

s1 = Te0[t1 >> 24] ^ Te1[(t2 >> 16) & 0xff] ^ Te2[(t3 >> 8) & 0xff] ^ Te3[t0 & 0xff] ^ rk[33];

s2 = Te0[t2 >> 24] ^ Te1[(t3 >> 16) & 0xff] ^ Te2[(t0 >> 8) & 0xff] ^ Te3[t1 & 0xff] ^ rk[34];

s3 = Te0[t3 >> 24] ^ Te1[(t0 >> 16) & 0xff] ^ Te2[(t1 >> 8) & 0xff] ^ Te3[t2 & 0xff] ^ rk[35];

/\* round 9: \*/

t0 = Te0[s0 >> 24] ^ Te1[(s1 >> 16) & 0xff] ^ Te2[(s2 >> 8) & 0xff] ^ Te3[s3 & 0xff] ^ rk[36];

t1 = Te0[s1 >> 24] ^ Te1[(s2 >> 16) & 0xff] ^ Te2[(s3 >> 8) & 0xff] ^ Te3[s0 & 0xff] ^ rk[37];

t2 = Te0[s2 >> 24] ^ Te1[(s3 >> 16) & 0xff] ^ Te2[(s0 >> 8) & 0xff] ^ Te3[s1 & 0xff] ^ rk[38];

t3 = Te0[s3 >> 24] ^ Te1[(s0 >> 16) & 0xff] ^ Te2[(s1 >> 8) & 0xff] ^ Te3[s2 & 0xff] ^ rk[39];

if (Nr > 10) {

/\* round 10: \*/

s0 = Te0[t0 >> 24] ^ Te1[(t1 >> 16) & 0xff] ^ Te2[(t2 >> 8) & 0xff] ^ Te3[t3 & 0xff] ^ rk[40];

s1 = Te0[t1 >> 24] ^ Te1[(t2 >> 16) & 0xff] ^ Te2[(t3 >> 8) & 0xff] ^ Te3[t0 & 0xff] ^ rk[41];

s2 = Te0[t2 >> 24] ^ Te1[(t3 >> 16) & 0xff] ^ Te2[(t0 >> 8) & 0xff] ^ Te3[t1 & 0xff] ^ rk[42];

s3 = Te0[t3 >> 24] ^ Te1[(t0 >> 16) & 0xff] ^ Te2[(t1 >> 8) & 0xff] ^ Te3[t2 & 0xff] ^ rk[43];

/\* round 11: \*/

t0 = Te0[s0 >> 24] ^ Te1[(s1 >> 16) & 0xff] ^ Te2[(s2 >> 8) & 0xff] ^ Te3[s3 & 0xff] ^ rk[44];

t1 = Te0[s1 >> 24] ^ Te1[(s2 >> 16) & 0xff] ^ Te2[(s3 >> 8) & 0xff] ^ Te3[s0 & 0xff] ^ rk[45];

t2 = Te0[s2 >> 24] ^ Te1[(s3 >> 16) & 0xff] ^ Te2[(s0 >> 8) & 0xff] ^ Te3[s1 & 0xff] ^ rk[46];

t3 = Te0[s3 >> 24] ^ Te1[(s0 >> 16) & 0xff] ^ Te2[(s1 >> 8) & 0xff] ^ Te3[s2 & 0xff] ^ rk[47];

if (Nr > 12) {

/\* round 12: \*/

s0 = Te0[t0 >> 24] ^ Te1[(t1 >> 16) & 0xff] ^ Te2[(t2 >> 8) & 0xff] ^ Te3[t3 & 0xff] ^ rk[48];

s1 = Te0[t1 >> 24] ^ Te1[(t2 >> 16) & 0xff] ^ Te2[(t3 >> 8) & 0xff] ^ Te3[t0 & 0xff] ^ rk[49];

s2 = Te0[t2 >> 24] ^ Te1[(t3 >> 16) & 0xff] ^ Te2[(t0 >> 8) & 0xff] ^ Te3[t1 & 0xff] ^ rk[50];

s3 = Te0[t3 >> 24] ^ Te1[(t0 >> 16) & 0xff] ^ Te2[(t1 >> 8) & 0xff] ^ Te3[t2 & 0xff] ^ rk[51];

/\* round 13: \*/

t0 = Te0[s0 >> 24] ^ Te1[(s1 >> 16) & 0xff] ^ Te2[(s2 >> 8) & 0xff] ^ Te3[s3 & 0xff] ^ rk[52];

t1 = Te0[s1 >> 24] ^ Te1[(s2 >> 16) & 0xff] ^ Te2[(s3 >> 8) & 0xff] ^ Te3[s0 & 0xff] ^ rk[53];

t2 = Te0[s2 >> 24] ^ Te1[(s3 >> 16) & 0xff] ^ Te2[(s0 >> 8) & 0xff] ^ Te3[s1 & 0xff] ^ rk[54];

t3 = Te0[s3 >> 24] ^ Te1[(s0 >> 16) & 0xff] ^ Te2[(s1 >> 8) & 0xff] ^ Te3[s2 & 0xff] ^ rk[55];

}

}

rk += Nr << 2;

#else /\* !FULL\_UNROLL \*/

/\*

\* Nr - 1 full rounds:

\*/

r = Nr >> 1;

for (;;) {

t0 =

Te0[(s0 >> 24) ] ^

Te1[(s1 >> 16) & 0xff] ^

Te2[(s2 >> 8) & 0xff] ^

Te3[(s3 ) & 0xff] ^

rk[4];

t1 =

Te0[(s1 >> 24) ] ^

Te1[(s2 >> 16) & 0xff] ^

Te2[(s3 >> 8) & 0xff] ^

Te3[(s0 ) & 0xff] ^

rk[5];

t2 =

Te0[(s2 >> 24) ] ^

Te1[(s3 >> 16) & 0xff] ^

Te2[(s0 >> 8) & 0xff] ^

Te3[(s1 ) & 0xff] ^

rk[6];

t3 =

Te0[(s3 >> 24) ] ^

Te1[(s0 >> 16) & 0xff] ^

Te2[(s1 >> 8) & 0xff] ^

Te3[(s2 ) & 0xff] ^

rk[7];

rk += 8;

if (--r == 0) {

break;

}

s0 =

Te0[(t0 >> 24) ] ^

Te1[(t1 >> 16) & 0xff] ^

Te2[(t2 >> 8) & 0xff] ^

Te3[(t3 ) & 0xff] ^

rk[0];

s1 =

Te0[(t1 >> 24) ] ^

Te1[(t2 >> 16) & 0xff] ^

Te2[(t3 >> 8) & 0xff] ^

Te3[(t0 ) & 0xff] ^

rk[1];

s2 =

Te0[(t2 >> 24) ] ^

Te1[(t3 >> 16) & 0xff] ^

Te2[(t0 >> 8) & 0xff] ^

Te3[(t1 ) & 0xff] ^

rk[2];

s3 =

Te0[(t3 >> 24) ] ^

Te1[(t0 >> 16) & 0xff] ^

Te2[(t1 >> 8) & 0xff] ^

Te3[(t2 ) & 0xff] ^

rk[3];

}

#endif /\* ?FULL\_UNROLL \*/

/\*

\* apply last round and

\* map cipher state to byte array block:

\*/

s0 =

(Te4[(t0 >> 24) ] & 0xff000000) ^

(Te4[(t1 >> 16) & 0xff] & 0x00ff0000) ^

(Te4[(t2 >> 8) & 0xff] & 0x0000ff00) ^

(Te4[(t3 ) & 0xff] & 0x000000ff) ^

rk[0];

PUTU32(ct , s0);

s1 =

(Te4[(t1 >> 24) ] & 0xff000000) ^

(Te4[(t2 >> 16) & 0xff] & 0x00ff0000) ^

(Te4[(t3 >> 8) & 0xff] & 0x0000ff00) ^

(Te4[(t0 ) & 0xff] & 0x000000ff) ^

rk[1];

PUTU32(ct + 4, s1);

s2 =

(Te4[(t2 >> 24) ] & 0xff000000) ^

(Te4[(t3 >> 16) & 0xff] & 0x00ff0000) ^

(Te4[(t0 >> 8) & 0xff] & 0x0000ff00) ^

(Te4[(t1 ) & 0xff] & 0x000000ff) ^

rk[2];

PUTU32(ct + 8, s2);

s3 =

(Te4[(t3 >> 24) ] & 0xff000000) ^

(Te4[(t0 >> 16) & 0xff] & 0x00ff0000) ^

(Te4[(t1 >> 8) & 0xff] & 0x0000ff00) ^

(Te4[(t2 ) & 0xff] & 0x000000ff) ^

rk[3];

PUTU32(ct + 12, s3);

}

static void rijndaelDecrypt(u32 rk[/\*4\*(Nr + 1)\*/], int Nr, const u8 ct[16], u8 pt[16]) {

u32 s0, s1, s2, s3, t0, t1, t2, t3;

#ifndef FULL\_UNROLL

int r;

#endif /\* ?FULL\_UNROLL \*/

/\*

\* map byte array block to cipher state

\* and add initial round key:

\*/

s0 = GETU32(ct ) ^ rk[0];

s1 = GETU32(ct + 4) ^ rk[1];

s2 = GETU32(ct + 8) ^ rk[2];

s3 = GETU32(ct + 12) ^ rk[3];

#ifdef FULL\_UNROLL

/\* round 1: \*/

t0 = Td0[s0 >> 24] ^ Td1[(s3 >> 16) & 0xff] ^ Td2[(s2 >> 8) & 0xff] ^ Td3[s1 & 0xff] ^ rk[ 4];

t1 = Td0[s1 >> 24] ^ Td1[(s0 >> 16) & 0xff] ^ Td2[(s3 >> 8) & 0xff] ^ Td3[s2 & 0xff] ^ rk[ 5];

t2 = Td0[s2 >> 24] ^ Td1[(s1 >> 16) & 0xff] ^ Td2[(s0 >> 8) & 0xff] ^ Td3[s3 & 0xff] ^ rk[ 6];

t3 = Td0[s3 >> 24] ^ Td1[(s2 >> 16) & 0xff] ^ Td2[(s1 >> 8) & 0xff] ^ Td3[s0 & 0xff] ^ rk[ 7];

/\* round 2: \*/

s0 = Td0[t0 >> 24] ^ Td1[(t3 >> 16) & 0xff] ^ Td2[(t2 >> 8) & 0xff] ^ Td3[t1 & 0xff] ^ rk[ 8];

s1 = Td0[t1 >> 24] ^ Td1[(t0 >> 16) & 0xff] ^ Td2[(t3 >> 8) & 0xff] ^ Td3[t2 & 0xff] ^ rk[ 9];

s2 = Td0[t2 >> 24] ^ Td1[(t1 >> 16) & 0xff] ^ Td2[(t0 >> 8) & 0xff] ^ Td3[t3 & 0xff] ^ rk[10];

s3 = Td0[t3 >> 24] ^ Td1[(t2 >> 16) & 0xff] ^ Td2[(t1 >> 8) & 0xff] ^ Td3[t0 & 0xff] ^ rk[11];

/\* round 3: \*/

t0 = Td0[s0 >> 24] ^ Td1[(s3 >> 16) & 0xff] ^ Td2[(s2 >> 8) & 0xff] ^ Td3[s1 & 0xff] ^ rk[12];

t1 = Td0[s1 >> 24] ^ Td1[(s0 >> 16) & 0xff] ^ Td2[(s3 >> 8) & 0xff] ^ Td3[s2 & 0xff] ^ rk[13];

t2 = Td0[s2 >> 24] ^ Td1[(s1 >> 16) & 0xff] ^ Td2[(s0 >> 8) & 0xff] ^ Td3[s3 & 0xff] ^ rk[14];

t3 = Td0[s3 >> 24] ^ Td1[(s2 >> 16) & 0xff] ^ Td2[(s1 >> 8) & 0xff] ^ Td3[s0 & 0xff] ^ rk[15];

/\* round 4: \*/

s0 = Td0[t0 >> 24] ^ Td1[(t3 >> 16) & 0xff] ^ Td2[(t2 >> 8) & 0xff] ^ Td3[t1 & 0xff] ^ rk[16];

s1 = Td0[t1 >> 24] ^ Td1[(t0 >> 16) & 0xff] ^ Td2[(t3 >> 8) & 0xff] ^ Td3[t2 & 0xff] ^ rk[17];

s2 = Td0[t2 >> 24] ^ Td1[(t1 >> 16) & 0xff] ^ Td2[(t0 >> 8) & 0xff] ^ Td3[t3 & 0xff] ^ rk[18];

s3 = Td0[t3 >> 24] ^ Td1[(t2 >> 16) & 0xff] ^ Td2[(t1 >> 8) & 0xff] ^ Td3[t0 & 0xff] ^ rk[19];

/\* round 5: \*/

t0 = Td0[s0 >> 24] ^ Td1[(s3 >> 16) & 0xff] ^ Td2[(s2 >> 8) & 0xff] ^ Td3[s1 & 0xff] ^ rk[20];

t1 = Td0[s1 >> 24] ^ Td1[(s0 >> 16) & 0xff] ^ Td2[(s3 >> 8) & 0xff] ^ Td3[s2 & 0xff] ^ rk[21];

t2 = Td0[s2 >> 24] ^ Td1[(s1 >> 16) & 0xff] ^ Td2[(s0 >> 8) & 0xff] ^ Td3[s3 & 0xff] ^ rk[22];

t3 = Td0[s3 >> 24] ^ Td1[(s2 >> 16) & 0xff] ^ Td2[(s1 >> 8) & 0xff] ^ Td3[s0 & 0xff] ^ rk[23];

/\* round 6: \*/

s0 = Td0[t0 >> 24] ^ Td1[(t3 >> 16) & 0xff] ^ Td2[(t2 >> 8) & 0xff] ^ Td3[t1 & 0xff] ^ rk[24];

s1 = Td0[t1 >> 24] ^ Td1[(t0 >> 16) & 0xff] ^ Td2[(t3 >> 8) & 0xff] ^ Td3[t2 & 0xff] ^ rk[25];

s2 = Td0[t2 >> 24] ^ Td1[(t1 >> 16) & 0xff] ^ Td2[(t0 >> 8) & 0xff] ^ Td3[t3 & 0xff] ^ rk[26];

s3 = Td0[t3 >> 24] ^ Td1[(t2 >> 16) & 0xff] ^ Td2[(t1 >> 8) & 0xff] ^ Td3[t0 & 0xff] ^ rk[27];

/\* round 7: \*/

t0 = Td0[s0 >> 24] ^ Td1[(s3 >> 16) & 0xff] ^ Td2[(s2 >> 8) & 0xff] ^ Td3[s1 & 0xff] ^ rk[28];

t1 = Td0[s1 >> 24] ^ Td1[(s0 >> 16) & 0xff] ^ Td2[(s3 >> 8) & 0xff] ^ Td3[s2 & 0xff] ^ rk[29];

t2 = Td0[s2 >> 24] ^ Td1[(s1 >> 16) & 0xff] ^ Td2[(s0 >> 8) & 0xff] ^ Td3[s3 & 0xff] ^ rk[30];

t3 = Td0[s3 >> 24] ^ Td1[(s2 >> 16) & 0xff] ^ Td2[(s1 >> 8) & 0xff] ^ Td3[s0 & 0xff] ^ rk[31];

/\* round 8: \*/

s0 = Td0[t0 >> 24] ^ Td1[(t3 >> 16) & 0xff] ^ Td2[(t2 >> 8) & 0xff] ^ Td3[t1 & 0xff] ^ rk[32];

s1 = Td0[t1 >> 24] ^ Td1[(t0 >> 16) & 0xff] ^ Td2[(t3 >> 8) & 0xff] ^ Td3[t2 & 0xff] ^ rk[33];

s2 = Td0[t2 >> 24] ^ Td1[(t1 >> 16) & 0xff] ^ Td2[(t0 >> 8) & 0xff] ^ Td3[t3 & 0xff] ^ rk[34];

s3 = Td0[t3 >> 24] ^ Td1[(t2 >> 16) & 0xff] ^ Td2[(t1 >> 8) & 0xff] ^ Td3[t0 & 0xff] ^ rk[35];

/\* round 9: \*/

t0 = Td0[s0 >> 24] ^ Td1[(s3 >> 16) & 0xff] ^ Td2[(s2 >> 8) & 0xff] ^ Td3[s1 & 0xff] ^ rk[36];

t1 = Td0[s1 >> 24] ^ Td1[(s0 >> 16) & 0xff] ^ Td2[(s3 >> 8) & 0xff] ^ Td3[s2 & 0xff] ^ rk[37];

t2 = Td0[s2 >> 24] ^ Td1[(s1 >> 16) & 0xff] ^ Td2[(s0 >> 8) & 0xff] ^ Td3[s3 & 0xff] ^ rk[38];

t3 = Td0[s3 >> 24] ^ Td1[(s2 >> 16) & 0xff] ^ Td2[(s1 >> 8) & 0xff] ^ Td3[s0 & 0xff] ^ rk[39];

if (Nr > 10) {

/\* round 10: \*/

s0 = Td0[t0 >> 24] ^ Td1[(t3 >> 16) & 0xff] ^ Td2[(t2 >> 8) & 0xff] ^ Td3[t1 & 0xff] ^ rk[40];

s1 = Td0[t1 >> 24] ^ Td1[(t0 >> 16) & 0xff] ^ Td2[(t3 >> 8) & 0xff] ^ Td3[t2 & 0xff] ^ rk[41];

s2 = Td0[t2 >> 24] ^ Td1[(t1 >> 16) & 0xff] ^ Td2[(t0 >> 8) & 0xff] ^ Td3[t3 & 0xff] ^ rk[42];

s3 = Td0[t3 >> 24] ^ Td1[(t2 >> 16) & 0xff] ^ Td2[(t1 >> 8) & 0xff] ^ Td3[t0 & 0xff] ^ rk[43];

/\* round 11: \*/

t0 = Td0[s0 >> 24] ^ Td1[(s3 >> 16) & 0xff] ^ Td2[(s2 >> 8) & 0xff] ^ Td3[s1 & 0xff] ^ rk[44];

t1 = Td0[s1 >> 24] ^ Td1[(s0 >> 16) & 0xff] ^ Td2[(s3 >> 8) & 0xff] ^ Td3[s2 & 0xff] ^ rk[45];

t2 = Td0[s2 >> 24] ^ Td1[(s1 >> 16) & 0xff] ^ Td2[(s0 >> 8) & 0xff] ^ Td3[s3 & 0xff] ^ rk[46];

t3 = Td0[s3 >> 24] ^ Td1[(s2 >> 16) & 0xff] ^ Td2[(s1 >> 8) & 0xff] ^ Td3[s0 & 0xff] ^ rk[47];

if (Nr > 12) {

/\* round 12: \*/

s0 = Td0[t0 >> 24] ^ Td1[(t3 >> 16) & 0xff] ^ Td2[(t2 >> 8) & 0xff] ^ Td3[t1 & 0xff] ^ rk[48];

s1 = Td0[t1 >> 24] ^ Td1[(t0 >> 16) & 0xff] ^ Td2[(t3 >> 8) & 0xff] ^ Td3[t2 & 0xff] ^ rk[49];

s2 = Td0[t2 >> 24] ^ Td1[(t1 >> 16) & 0xff] ^ Td2[(t0 >> 8) & 0xff] ^ Td3[t3 & 0xff] ^ rk[50];

s3 = Td0[t3 >> 24] ^ Td1[(t2 >> 16) & 0xff] ^ Td2[(t1 >> 8) & 0xff] ^ Td3[t0 & 0xff] ^ rk[51];

/\* round 13: \*/

t0 = Td0[s0 >> 24] ^ Td1[(s3 >> 16) & 0xff] ^ Td2[(s2 >> 8) & 0xff] ^ Td3[s1 & 0xff] ^ rk[52];

t1 = Td0[s1 >> 24] ^ Td1[(s0 >> 16) & 0xff] ^ Td2[(s3 >> 8) & 0xff] ^ Td3[s2 & 0xff] ^ rk[53];

t2 = Td0[s2 >> 24] ^ Td1[(s1 >> 16) & 0xff] ^ Td2[(s0 >> 8) & 0xff] ^ Td3[s3 & 0xff] ^ rk[54];

t3 = Td0[s3 >> 24] ^ Td1[(s2 >> 16) & 0xff] ^ Td2[(s1 >> 8) & 0xff] ^ Td3[s0 & 0xff] ^ rk[55];

}

}

rk += Nr << 2;

#else /\* !FULL\_UNROLL \*/

/\*

\* Nr - 1 full rounds:

\*/

r = Nr >> 1;

for (;;) {

t0 =

Td0[(s0 >> 24) ] ^

Td1[(s3 >> 16) & 0xff] ^

Td2[(s2 >> 8) & 0xff] ^

Td3[(s1 ) & 0xff] ^

rk[4];

t1 =

Td0[(s1 >> 24) ] ^

Td1[(s0 >> 16) & 0xff] ^

Td2[(s3 >> 8) & 0xff] ^

Td3[(s2 ) & 0xff] ^

rk[5];

t2 =

Td0[(s2 >> 24) ] ^

Td1[(s1 >> 16) & 0xff] ^

Td2[(s0 >> 8) & 0xff] ^

Td3[(s3 ) & 0xff] ^

rk[6];

t3 =

Td0[(s3 >> 24) ] ^

Td1[(s2 >> 16) & 0xff] ^

Td2[(s1 >> 8) & 0xff] ^

Td3[(s0 ) & 0xff] ^

rk[7];

rk += 8;

if (--r == 0) {

break;

}

s0 =

Td0[(t0 >> 24) ] ^

Td1[(t3 >> 16) & 0xff] ^

Td2[(t2 >> 8) & 0xff] ^

Td3[(t1 ) & 0xff] ^

rk[0];

s1 =

Td0[(t1 >> 24) ] ^

Td1[(t0 >> 16) & 0xff] ^

Td2[(t3 >> 8) & 0xff] ^

Td3[(t2 ) & 0xff] ^

rk[1];

s2 =

Td0[(t2 >> 24) ] ^

Td1[(t1 >> 16) & 0xff] ^

Td2[(t0 >> 8) & 0xff] ^

Td3[(t3 ) & 0xff] ^

rk[2];

s3 =

Td0[(t3 >> 24) ] ^

Td1[(t2 >> 16) & 0xff] ^

Td2[(t1 >> 8) & 0xff] ^

Td3[(t0 ) & 0xff] ^

rk[3];

}

#endif /\* ?FULL\_UNROLL \*/

/\*

\* apply last round and

\* map cipher state to byte array block:

\*/

s0 =

(Td4[(t0 >> 24) ] & 0xff000000) ^

(Td4[(t3 >> 16) & 0xff] & 0x00ff0000) ^

(Td4[(t2 >> 8) & 0xff] & 0x0000ff00) ^

(Td4[(t1 ) & 0xff] & 0x000000ff) ^

rk[0];

PUTU32(pt , s0);

s1 =

(Td4[(t1 >> 24) ] & 0xff000000) ^

(Td4[(t0 >> 16) & 0xff] & 0x00ff0000) ^

(Td4[(t3 >> 8) & 0xff] & 0x0000ff00) ^

(Td4[(t2 ) & 0xff] & 0x000000ff) ^

rk[1];

PUTU32(pt + 4, s1);

s2 =

(Td4[(t2 >> 24) ] & 0xff000000) ^

(Td4[(t1 >> 16) & 0xff] & 0x00ff0000) ^

(Td4[(t0 >> 8) & 0xff] & 0x0000ff00) ^

(Td4[(t3 ) & 0xff] & 0x000000ff) ^

rk[2];

PUTU32(pt + 8, s2);

s3 =

(Td4[(t3 >> 24) ] & 0xff000000) ^

(Td4[(t2 >> 16) & 0xff] & 0x00ff0000) ^

(Td4[(t1 >> 8) & 0xff] & 0x0000ff00) ^

(Td4[(t0 ) & 0xff] & 0x000000ff) ^

rk[3];

PUTU32(pt + 12, s3);

}

#ifdef INTERMEDIATE\_VALUE\_KAT

static void rijndaelEncryptRound(const u32 rk[/\*4\*(Nr + 1)\*/], int Nr, u8 block[16], int rounds) {

int r;

u32 s0, s1, s2, s3, t0, t1, t2, t3;

/\*

\* map byte array block to cipher state

\* and add initial round key:

\*/

s0 = GETU32(block ) ^ rk[0];

s1 = GETU32(block + 4) ^ rk[1];

s2 = GETU32(block + 8) ^ rk[2];

s3 = GETU32(block + 12) ^ rk[3];

rk += 4;

/\*

\* Nr - 1 full rounds:

\*/

for (r = (rounds < Nr ? rounds : Nr - 1); r > 0; r--) {

t0 =

Te0[(s0 >> 24) ] ^

Te1[(s1 >> 16) & 0xff] ^

Te2[(s2 >> 8) & 0xff] ^

Te3[(s3 ) & 0xff] ^

rk[0];

t1 =

Te0[(s1 >> 24) ] ^

Te1[(s2 >> 16) & 0xff] ^

Te2[(s3 >> 8) & 0xff] ^

Te3[(s0 ) & 0xff] ^

rk[1];

t2 =

Te0[(s2 >> 24) ] ^

Te1[(s3 >> 16) & 0xff] ^

Te2[(s0 >> 8) & 0xff] ^

Te3[(s1 ) & 0xff] ^

rk[2];

t3 =

Te0[(s3 >> 24) ] ^

Te1[(s0 >> 16) & 0xff] ^

Te2[(s1 >> 8) & 0xff] ^

Te3[(s2 ) & 0xff] ^

rk[3];

s0 = t0;

s1 = t1;

s2 = t2;

s3 = t3;

rk += 4;

}

/\*

\* apply last round and

\* map cipher state to byte array block:

\*/

if (rounds == Nr) {

t0 =

(Te4[(s0 >> 24) ] & 0xff000000) ^

(Te4[(s1 >> 16) & 0xff] & 0x00ff0000) ^

(Te4[(s2 >> 8) & 0xff] & 0x0000ff00) ^

(Te4[(s3 ) & 0xff] & 0x000000ff) ^

rk[0];

t1 =

(Te4[(s1 >> 24) ] & 0xff000000) ^

(Te4[(s2 >> 16) & 0xff] & 0x00ff0000) ^

(Te4[(s3 >> 8) & 0xff] & 0x0000ff00) ^

(Te4[(s0 ) & 0xff] & 0x000000ff) ^

rk[1];

t2 =

(Te4[(s2 >> 24) ] & 0xff000000) ^

(Te4[(s3 >> 16) & 0xff] & 0x00ff0000) ^

(Te4[(s0 >> 8) & 0xff] & 0x0000ff00) ^

(Te4[(s1 ) & 0xff] & 0x000000ff) ^

rk[2];

t3 =

(Te4[(s3 >> 24) ] & 0xff000000) ^

(Te4[(s0 >> 16) & 0xff] & 0x00ff0000) ^

(Te4[(s1 >> 8) & 0xff] & 0x0000ff00) ^

(Te4[(s2 ) & 0xff] & 0x000000ff) ^

rk[3];

s0 = t0;

s1 = t1;

s2 = t2;

s3 = t3;

}

PUTU32(block , s0);

PUTU32(block + 4, s1);

PUTU32(block + 8, s2);

PUTU32(block + 12, s3);

}

static void rijndaelDecryptRound(const u32 rk[/\*4\*(Nr + 1)\*/], int Nr, u8 block[16], int rounds) {

int r;

u32 s0, s1, s2, s3, t0, t1, t2, t3;

/\*

\* map byte array block to cipher state

\* and add initial round key:

\*/

s0 = GETU32(block ) ^ rk[0];

s1 = GETU32(block + 4) ^ rk[1];

s2 = GETU32(block + 8) ^ rk[2];

s3 = GETU32(block + 12) ^ rk[3];

rk += 4;

/\*

\* Nr - 1 full rounds:

\*/

for (r = (rounds < Nr ? rounds : Nr) - 1; r > 0; r--) {

t0 =

Td0[(s0 >> 24) ] ^

Td1[(s3 >> 16) & 0xff] ^

Td2[(s2 >> 8) & 0xff] ^

Td3[(s1 ) & 0xff] ^

rk[0];

t1 =

Td0[(s1 >> 24) ] ^

Td1[(s0 >> 16) & 0xff] ^

Td2[(s3 >> 8) & 0xff] ^

Td3[(s2 ) & 0xff] ^

rk[1];

t2 =

Td0[(s2 >> 24) ] ^

Td1[(s1 >> 16) & 0xff] ^

Td2[(s0 >> 8) & 0xff] ^

Td3[(s3 ) & 0xff] ^

rk[2];

t3 =

Td0[(s3 >> 24) ] ^

Td1[(s2 >> 16) & 0xff] ^

Td2[(s1 >> 8) & 0xff] ^

Td3[(s0 ) & 0xff] ^

rk[3];

s0 = t0;

s1 = t1;

s2 = t2;

s3 = t3;

rk += 4;

}

/\*

\* complete the last round and

\* map cipher state to byte array block:

\*/

t0 =

(Td4[(s0 >> 24) ] & 0xff000000) ^

(Td4[(s3 >> 16) & 0xff] & 0x00ff0000) ^

(Td4[(s2 >> 8) & 0xff] & 0x0000ff00) ^

(Td4[(s1 ) & 0xff] & 0x000000ff);

t1 =

(Td4[(s1 >> 24) ] & 0xff000000) ^

(Td4[(s0 >> 16) & 0xff] & 0x00ff0000) ^

(Td4[(s3 >> 8) & 0xff] & 0x0000ff00) ^

(Td4[(s2 ) & 0xff] & 0x000000ff);

t2 =

(Td4[(s2 >> 24) ] & 0xff000000) ^

(Td4[(s1 >> 16) & 0xff] & 0x00ff0000) ^

(Td4[(s0 >> 8) & 0xff] & 0x0000ff00) ^

(Td4[(s3 ) & 0xff] & 0x000000ff);

t3 =

(Td4[(s3 >> 24) ] & 0xff000000) ^

(Td4[(s2 >> 16) & 0xff] & 0x00ff0000) ^

(Td4[(s1 >> 8) & 0xff] & 0x0000ff00) ^

(Td4[(s0 ) & 0xff] & 0x000000ff);

if (rounds == Nr) {

t0 ^= rk[0];

t1 ^= rk[1];

t2 ^= rk[2];

t3 ^= rk[3];

}

PUTU32(block , t0);

PUTU32(block + 4, t1);

PUTU32(block + 8, t2);

PUTU32(block + 12, t3);

}

#endif /\* INTERMEDIATE\_VALUE\_KAT \*/

static void block\_init(block\_state \*state, unsigned char \*key,

int keylen)

{

int Nr = 0;

if (keylen != 16 && keylen != 24 && keylen != 32) {

PyErr\_SetString(PyExc\_ValueError,

"AES key must be either 16, 24, or 32 bytes long");

return;

}

switch (keylen) {

case(16): Nr = 10; break;

case(24): Nr = 12; break;

case(32): Nr = 14; break;

}

state->rounds = Nr;

rijndaelKeySetupEnc(state->ek, key, keylen\*8);

rijndaelKeySetupDec(state->dk, key, keylen\*8);

}

static void block\_encrypt(block\_state \*self, u8 \*in, u8 \*out)

{

rijndaelEncrypt(self->ek, self->rounds, in, out);

}

static void block\_decrypt(block\_state \*self, u8 \*in, u8 \*out)

{

rijndaelDecrypt(self->dk, self->rounds, in, out);

}

#include "block\_template.c"