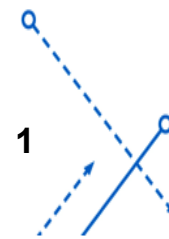


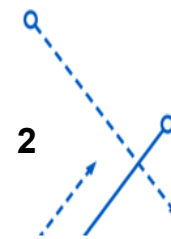
## AES密钥的生成过程

- 以128位种子密钥为例, 假定当前机器的存储模式为Big-Endian(高字节在前, 低字节在后), 设long k[4]是种子密钥, 则后面还需要生成k[4]至k[43]共40个long, 步骤如下:
- $k[4] = k[3]$ ;
- 把k[4]包含的4个字节循环左移1字节;
- ByteSub(&k[4], 4); 把k[4]包含的4个字节全部替换成sbox中的值
- $r = 2^{(i-4)/4} \bmod 0x11B$ ; 计算轮常数r, 其中i是k的下标即4
- $k[4] \text{ 首字节} \wedge = r$ ; k[4]首字节与r异或
- $k[4] \wedge = k[0]$ ; k[4]与k[0]异或
- $k[5] = k[4] \wedge k[1]$ ;
- $k[6] = k[5] \wedge k[2]$ ;
- $k[7] = k[6] \wedge k[3]$ ;



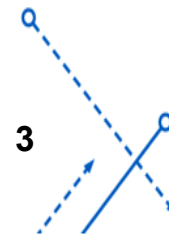
## AES密钥的生成过程

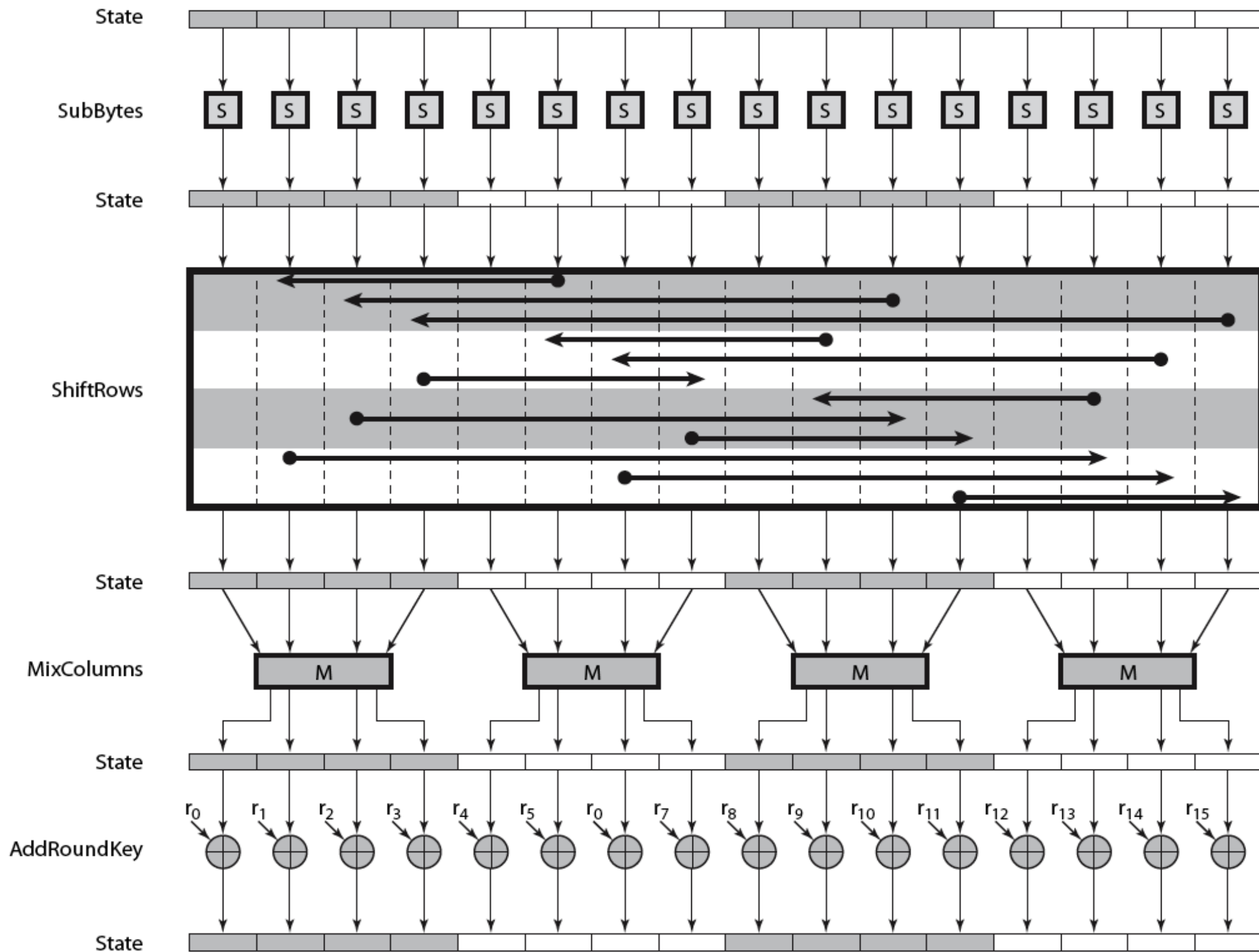
- 以上过程生成了4个long，是一组16字节的key。接下去生成k[8]至k[11]的过程与k[4]至k[7]类似，其中k[8]像k[4]那样需要作特殊处理：
  - $k[8] = k[7]$ ;
  - k[8]包含的4字节循环左移1字节;
  - $\text{ByteSub}(\&k[8], 4)$ ;
  - 轮常数  $r = 21 = 2 \bmod 0x11B$ ;
  - k[8]首字节  $\wedge = r$ ;
  - $k[8] \wedge = k[4]$ ;
  - $k[9] = k[8] \wedge k[5]$ ;
  - $k[10] = k[9] \wedge k[6]$ ;
  - $k[11] = k[10] \wedge k[7]$ ;



## AES密钥的生成过程

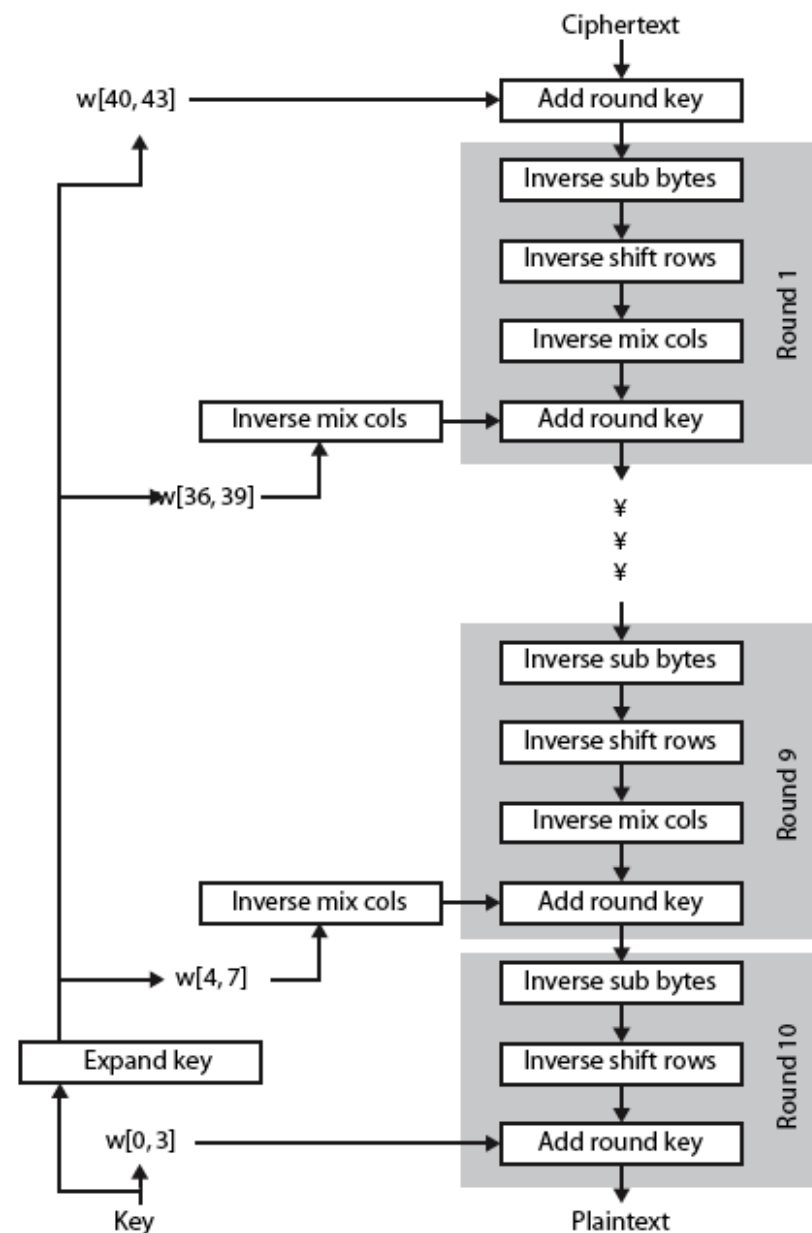
- 按上述步骤可以生成10组密钥, 每组为4个long。请特别注意最后两组的轮常数因为mod 0x11B的缘故与 $2^i$ 的值并不相等。对应于 $i \in [0, 9]$ , 生成10组密钥的轮常数 $r = 2^i \bmod 0x11B$ 的值分别为:
- 0x01 0x02 0x04 0x08 0x10 0x20 0x40 0x80 0x1B 0x36
- 192位及256位种子密钥生成密钥的过程比128位复杂一些, 具体请参考MyAes.c中的函数aes\_set\_key()。





## AES - 解密过程

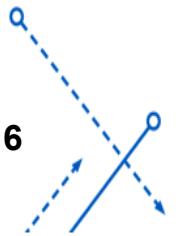
- AES解密与加密不同，步骤与加密相反
- 可以定义一个等价的逆密码，其步骤与加密相同
  - 用每一步的逆步骤
  - 使用不同的key schedule
- 当交换byte substitution和shift rows步骤结果不会改变
  - 因为只处理字节



## ► 调用Openssl库中的AES加密解密函数

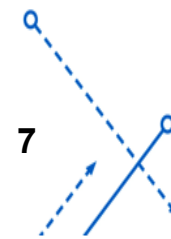
```
#include <openssl/aes.h>
#include <stdio.h>
#pragma comment(lib, "libeay32.lib")
#pragma comment(lib, "ssleay32.lib")
main() {
    AES_KEY k;
    int i;
    unsigned char bufin[100]="A Quick BrownFox";
    unsigned char bufout[100]={0};
    unsigned char hex[100];
    AES_set_encrypt_key("0123456789ABCDEF",
128, &k);
```

```
AES_encrypt(bufin, bufout, &k);
for(i=0; i<16; i++){
    sprintf(hex+i*2, "%02X", bufout[i]);
}
puts(hex);
memset(bufin, 0, sizeof(bufin));
memset(bufout, 0, sizeof(bufout));
for(i=0; i<16; i++) {
    sscanf(hex+i*2, "%02X", &bufin[i]);
}
AES_decrypt(bufin, bufout, &k);
puts(bufout);
}
```



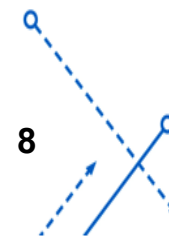
## AES – 实现层面

- 可以在8位CPU上高效实现
- AddRoundKey-简单的字节XOR操作
- SubBytes-使用256个条目的表处理字节
- ShiftRows-简单的字节移位操作
- MixColumns-需要GF ( $2^8$ ) 中的矩阵乘法，可以简化为使用表查找和字节异或



## AES – 实现层面

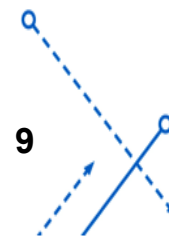
- 可以在32位CPU上高效实现
  - 重新定义使用32位word的步骤
  - 可以预计算4个256word的表格
  - 可以使用4个表查找+4个异或操作来计算每一轮中的每一列
  - 以4Kb的成本存储表格
- 设计人员认为，这种非常高效的实现是选择AES密码的关键因素





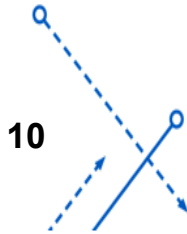
## T-box vs S-box

- 与软件和硬件中的S-Box实现相比，用于解密和组合加密解密单元的AES T-Box实现显示出更好的吞吐量。
- S-box体系结构使用 $8 \times 8$ 查找表和其他的轮操作操作进行加密/解密操作。
- T-box体系结构使用 $8 \times 32$ 查找表和其他的XOR操作进行加密/解密操作。  
(只有查表操作和AddRoundKey操作，其他三个轮操作综合到查表操作中)
- T-box架构使用的内存是S-box的4倍。



## T-box架构概述

- 这种体系结构只允许使用查找表和XOR操作计算整轮。
- 预先计算的查找表表示子字节和混合列转换的组合操作。
- T-box表的大小为 $8 \times 32$ 位。

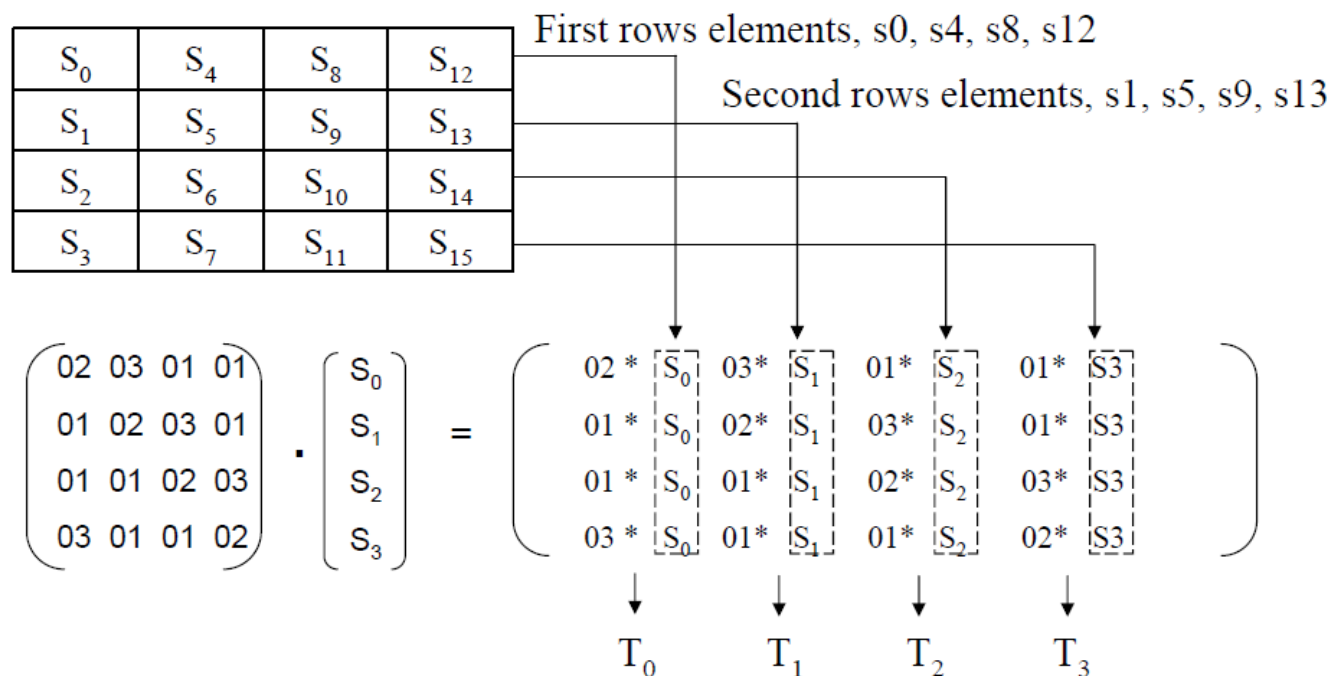


## T-box Tables

- 将每一列作为一个T，其中T0，T1，T2，T3之间需要进行XOR来得到最终的结果。

Mix Column Operation In AES

State (128 bit)



## T-box Tables


- $S[a]$  代表  $a$  经过 Subbyte 变换之后的结果，上面的乘法为  $GF(2^8)$  下的多项式乘法

$$\begin{aligned}
 T_0[a] &= \begin{pmatrix} 02.S[a] \\ S[a] \\ S[a] \\ 03.S[a] \end{pmatrix} & T_1[a] &= \begin{pmatrix} 03.S[a] \\ 02.S[a] \\ S[a] \\ S[a] \end{pmatrix} & T_2[a] &= \begin{pmatrix} S[a] \\ 03.S[a] \\ 02.S[a] \\ S[a] \end{pmatrix} & T_3[a] &= \begin{pmatrix} S[a] \\ S[a] \\ 03.S[a] \\ 02.S[a] \end{pmatrix} \\
 T_0^{-1}[a] &= \begin{pmatrix} 0E.S[a] \\ 09.S[a] \\ 0D.S[a] \\ 0B.S[a] \end{pmatrix} & T_1^{-1}[a] &= \begin{pmatrix} 0B.S[a] \\ 0E.S[a] \\ 09.S[a] \\ 0D.S[a] \end{pmatrix} & T_2^{-1}[a] &= \begin{pmatrix} 0D.S[a] \\ 0B.S[a] \\ 0E.S[a] \\ 09.S[a] \end{pmatrix} & T_3^{-1}[a] &= \begin{pmatrix} 09.S[a] \\ 0D.S[a] \\ 0B.S[a] \\ 0E.S[a] \end{pmatrix}
 \end{aligned}$$



## T-box - 每轮计算过程

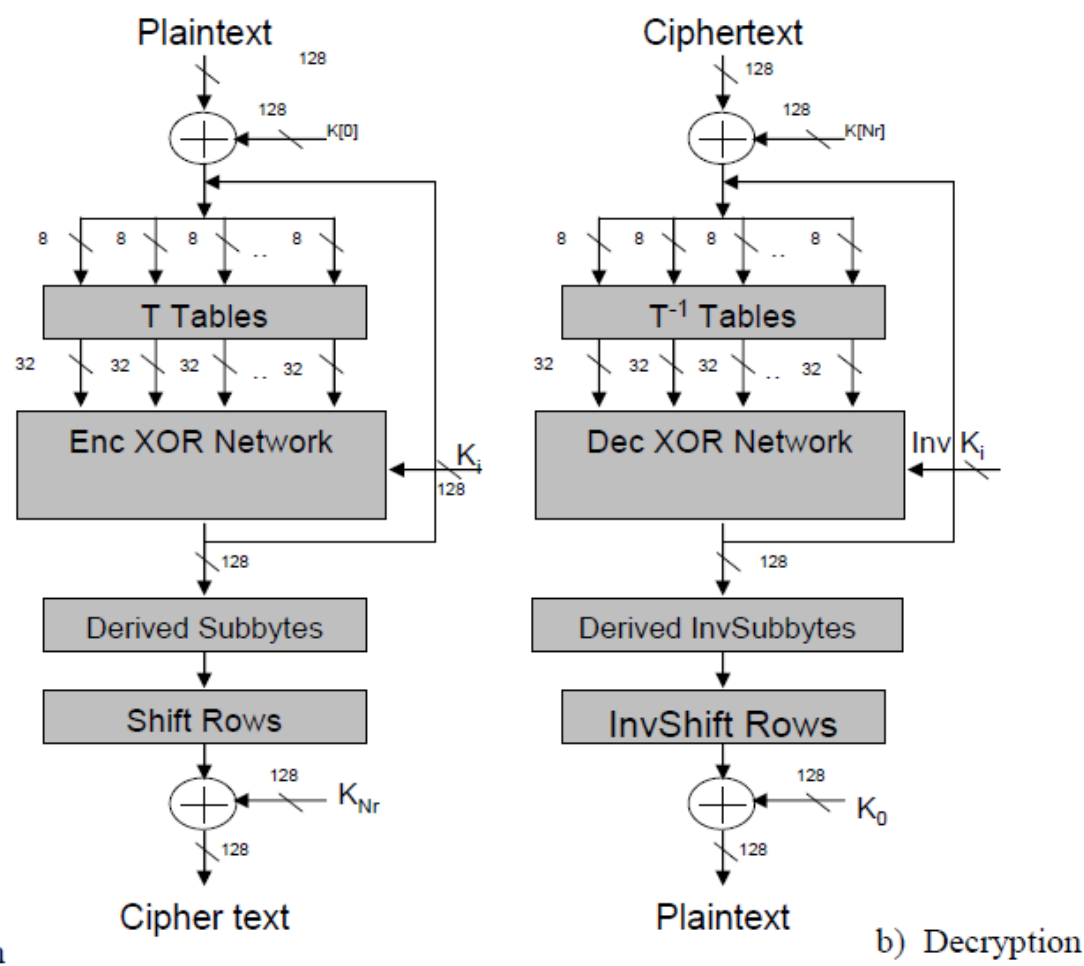
$$\begin{pmatrix} e_{0,j} \\ e_{1,j} \\ e_{2,j} \\ e_{3,j} \end{pmatrix} = T_0[a_{0,j}] \oplus T_1[a_{1,j+c_1}] \oplus T_2[a_{2,j+c_2}] \oplus T_3[a_{3,j+c_3}] \oplus \begin{pmatrix} K_{0,j} \\ K_{1,j} \\ K_{2,j} \\ K_{3,j} \end{pmatrix}$$

 Mod 4

$$\begin{pmatrix} e_{0,j} \\ e_{1,j} \\ e_{2,j} \\ e_{3,j} \end{pmatrix} = T_0[a_{0,j}] \oplus \text{Rotbyte}(T_0[a_{1,j+c_1}]) \oplus \text{Rotbyte}(T_0[a_{2,j+c_2}]) \oplus \text{Rotbyte}(T_0[a_{3,j+c_3}]) \oplus K_j$$

j- indicates key word

## T-box - 架构



## AES-NI

- 英特尔高级加密标准（AES）新指令（AESNI）：专门为实现AES做优化的指令集。
  - AES-NI是英特尔和AMD于2008年3月提出的针对微处理器的x86指令集架构的扩展。
- 性能提升：
  - 使用AES-NI预期的性能改进将取决于应用程序以及应用程序在加密和解密上花费的时间。在算法层面，使用AES-NI可以显著提高AES的速度。对于非并行模式的AES操作，如CBC encrypt AES-NI，与完全软件化的方法相比，性能提高了2-3倍。
  - 对于CBC解密和CTR等可并行化模式，AES-NI可以比软件解决方案提高10倍。
- 安全性提升：
  - 除了提高性能之外，新指令还有助于解决最近发现的AES侧通道攻击。AES-NI指令完全在硬件中执行解密和加密，无需软件查找表。因此，使用AES-NI可以降低旁道攻击的风险，并大大提高AES性能。

## AES-NI -其他体系架构中的硬件加速

### ➤ ARM 架构

- Allwinner
  - A10 and A20 using "safety system"
- Broadcom
  - Bcm5801 / bcm5805 / bcm5820 using "security processor"
- Qualcomm
  - Snapdragon 805 and later
- Samsung
  - Exynos 3 and later

### ➤ 其他架构

- Atmel AVR



## AES-NI的使用

### ➤ 使用标准库

- OpenSSL: SSL和TLS的开源库实现。支持多种加密功能，包括AES
- Intel® Integrated Performance Primitives (IPP) crypto: 包括AES在内的多种密码算法的多核就绪、高度优化实现的广泛库。
- Microsoft Cryptography API: Next Generation: Microsoft\*操作系统附带的API为应用程序提供加密服务。

### ➤ 使用C/C++或汇编语言

- Gcc/g++ : C/C开源GNU编译器++
- Intel® C/C++ compiler : 英特尔C/C编译器工具+
- Microsoft Visual C++ : 适用于Windows操作系统的C/C++编译器工具



## AES-NI - 指令集

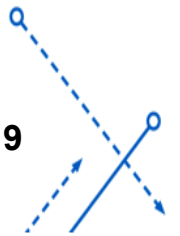
- AESENC 执行一轮AES加密流
- AESENCLAST 执行最后一轮AES加密流
- AESDEC 执行一轮AES解密流
- AESDECLAST 执行最后一轮AES解密流
- AESKEYGENASSIST 协助生成AES轮密钥
- AESIMC 协助AES逆列混合,执行反向列混淆变换
- PCLMULQDQ 进行无进位乘法 (CLMUL)



## AES-NI

Figure 15. AES-128 Encryption Outlined Code Sequence

```
; AES-128 encryption sequence.  
; The data block is in xmm15.  
; Registers xmm0-xmm10 hold the round keys(from 0 to 10 in this order).  
; In the end, xmm15 holds the encryption result.  
    pxor xmm15, xmm0                ; Whitening step (Round 0)  
    aesenc xmm15, xmm1              ; Round 1  
    aesenc xmm15, xmm2              ; Round 2  
    aesenc xmm15, xmm3              ; Round 3  
    aesenc xmm15, xmm4              ; Round 4  
    aesenc xmm15, xmm5              ; Round 5  
    aesenc xmm15, xmm6              ; Round 6  
    aesenc xmm15, xmm7              ; Round 7  
    aesenc xmm15, xmm8              ; Round 8  
    aesenc xmm15, xmm9              ; Round 9  
    aesenclast xmm15, xmm10         ; Round 10
```



## AES – 作业

### ➤ 重写核心函数

下载<http://10.71.45.100/bhh/myaes.zip> , 解压缩得到MyAes.c

(20分)

MyAes.c中部分函数的源代码已删除, 这些函数经编译后生成的机器码已转移到aes.lib内。

这些已删除的函数包括:

```
void build_sbox_inverse(void);
void aes_polynomial_mul(unsigned char x[4], unsigned char y[4], unsigned char z[4]);
void ByteSubInverse(unsigned char *p, int n);
void ShiftRowInverse(unsigned char *p);
void MixColumnInverse(unsigned char *p, unsigned char a[4], int do_mul);
void aes_decrypt(unsigned char *bufin, unsigned char *bufout, unsigned char *key);
```

请对本程序(MyAes.c)做以下修改:

- 重写上述函数
- 删除#pragma comment(lib, "aes.lib")

要求:

- 除sbox及sbox\_inverse外, 函数中不可以查其它表来代替计算
- 源代码中已公开的函数可以随意调用
- 修改后的程序必须能正常编译运行
- 运行产生的输出结果与本程序一致



Thank you!

