文章编号:1001 - 9081(2011)06 - 1539 - 04

doi:10.3724/SP.J.1087.2011.01539

AES 算法优化及其在 ARM 上的实现

张月华,张新贺,刘鸿雁

(辽宁科技大学 电子与信息工程学院,辽宁 鞍山 114051)

(zh0865_cn@sina.com)

摘　要:为了提高高级加密标准(AES)算法在 ARM 上的执行效率,针对明文长度和密钥长度均为 128 位的 AES 算法,提出了一种在 ARM 上高效运行并且占用较少 ROM 空间的实现方案。S 盒采用即时计算的方法生成,将列混合和逆列混合修改为针对 32 位字的操作,密钥扩展采用即时密钥扩展。在 S3C2440 处理器上实现的实验结果表明,AES 算法的优化方案可以在 ARM 处理器上高效运行并占用了较少的 ROM 空间。该方案可以应用于存储空间较小的嵌入式系统中。

关键词:高级加密标准;加密;密钥扩展;ARM;算法优化

中图分类号: TP309.2　　**文献标志码**:A

Advanced encryption standard and its software implementation on ARM processor

ZHANG Yue-hua, ZHANG Xin-he, LIU Hong-yan

(*School of Electronic and Information Engineering, University of Science and Technology Liaoning, Anshan Liaoning* 114051, *China*)

Abstract: To improve the efficiency of Advanced Encryption Standard (AES) algorithm on ARM processor, aiming at AES algorithm with 128-bit block length and key length, an optimization method was proposed. The method can speed up execution efficiently on ARM processor while consuming less ROM memory. A theoretical analysis of the Rijndael algorithm and of the proposed optimization was discussed. S box was generated by real-time calculation. The MixColumns and InvMixColumns transformations were amended to execute efficiently on 32-bit processor. On-the-fly key expansion was adapted. Simulation results of the optimized algorithm on S3C2440 processor were presented. The experimental results show that the optimization of AES algorithm can execute efficiently on S3C2440 and consume less ROM memory. The method can be applied to embedded systems with memory constraints.

Key words: Advanced Encryption Standard (AES); encryption; key expansion; ARM; algorithm optimization

2000 年 10 月,Rijndael 算法被选中作为新的高级加密标准(Advanced Encryption Standard,AES),新的标准将代替密钥长度太短的旧 DES 算法[1-2]。该算法汇聚了设计简单、密钥安装快、需要的内存空间少、在所有的平台上运行良好、支持并行处理并且可以抵抗所有已知攻击等优点。AES 算法明文和密钥长度均可独立指定为 128、192 或 256 位。自 AES 算法发布以来,国内外对此进行了各种相关研究,主要集中在 AES 算法在 FPGA、ASIC 上的硬件实现,以及算法的改进方面的研究,并发表了大量的研究成果,参见参考文献[3 - 6]。本文针对明文长度和密钥长度均为 128 位的 AES 算法(记作 AES-128),提出了一种新的算法实现方案并在 ARM 上加以实现。

1　AES 算法简介

AES 加密算法包括字节替换 SubBytes()、行移位 ShiftRows()、列混合 MixColumns()和密钥加 AddRoundKey()等函数[7]。AES 算法结构属于替代/置换(SP)结构,组成每一轮变换的 4 个函数分别属于 S 层、P 层和密钥加层。S 层由 SubBytes()组成,即非线性层,它的作用是确保多轮迭代后结果的高度混乱,其目标是通过一个较小的非线性元素得到一个大的非线性构件。P 层由 ShiftRows()和 MixColumns()组成,即线性层,它的作用是确保多轮迭代后的高度扩散,其目标是使非线性元素取得尽可能简单。密钥加层由 AddRoundKey()组成,主要实现轮密钥与明文或密文的异或运算。

SubBytes():字节替换是 AES 中的唯一的一个非线性变换,该变换可通过一个 S 盒对状态矩阵中的每个字节进行替换运算。S 盒是 256 个字节的查找表。S 盒可通过以下两步变换得到:首先将各个字节表示成有限域 $\mathbf{GF}(2^8)$ 上的形式,并求出各字节的乘法逆元素;然后对乘法逆元素进行仿射变换。

ShiftRows():行移位变换是一个字节移位操作,它对状态矩阵中不同行的数据按照不同的偏移量进行循环左移。对于 AES-128,偏移量分别为 0、1、2、3。

MixColumns():列混合变换是将状态矩阵中每一列的字看做有限域 $\mathbf{GF}(2^8)$ 上的一个多项式,与一个给定的多项式 $c(x)$ 相乘后对 $m(x) = x^8 + x^4 + x^3 + x + 1$ 进行取模运算,运算结果构成新的列。其中 $c(x) = \{03\} * x^3 + \{01\} * x^2 + \{01\} * x + \{02\}$。与一个固定多项式模乘可以表示为一个矩阵乘法,令 $\boldsymbol{b}(x) = \boldsymbol{c}(x) * \boldsymbol{a}(x)$,则:

收稿日期:2010 - 11 - 25;**修回日期**:2011 - 01 - 21。　**基金项目**:鞍山市科委资助项目(2006SH16)。

作者简介:张月华(1979 -),女,河北承德人,讲师,硕士研究生,主要研究方向:信息安全、嵌入式系统; 张新贺(1980 -),男,河北承德人,讲师,硕士研究生,主要研究方向:信息安全、嵌入式系统; 刘鸿雁(1958 -),女,辽宁鞍山人,教授,主要研究方向:信息安全、嵌入式系统、数据库。

$$\begin{bmatrix} \boldsymbol{b}_{0,c} \\ \boldsymbol{b}_{1,c} \\ \boldsymbol{b}_{2,c} \\ \boldsymbol{b}_{3,c} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} \boldsymbol{a}_{0,c} \\ \boldsymbol{a}_{1,c} \\ \boldsymbol{a}_{2,c} \\ \boldsymbol{a}_{3,c} \end{bmatrix}; 0 \leqslant c \leqslant 3$$

该矩阵第一列中的四个字节分别可以用下面的式子表示：

$$\boldsymbol{b}_{0,c} = \{02\} * \boldsymbol{a}_{0,c} \oplus \{03\} * \boldsymbol{a}_{1,c} \oplus \boldsymbol{a}_{2,c} \oplus \boldsymbol{a}_{3,c}$$
$$\boldsymbol{b}_{1,c} = \boldsymbol{a}_{0,c} \oplus \{02\} * \boldsymbol{a}_{1,c} \oplus \{03\} * \boldsymbol{a}_{2,c} \oplus \boldsymbol{a}_{3,c}$$
$$\boldsymbol{b}_{2,c} = \boldsymbol{a}_{0,c} \oplus \boldsymbol{a}_{1,c} \oplus \{02\} * \boldsymbol{a}_{2,c} \oplus \{03\} * \boldsymbol{a}_{3,c}$$
$$\boldsymbol{b}_{3,c} = \{03\} * \boldsymbol{b}_{0,c} \oplus \boldsymbol{b}_{1,c} \oplus \boldsymbol{b}_{2,c} \oplus \{02\} * \boldsymbol{b}_{3,c}$$

其中“*”代表在 $\mathbf{GF}(2^8)$ 上与 $m(x) = x^8 + x^4 + x^3 + x + 1$ 相乘。

AddRoundKey()：密钥加是一个简单的异或运算，即 128 位的状态矩阵与 128 位的轮密钥进行异或。每一轮的轮密钥由初始密钥经密钥扩展函数计算生成。

2 算法优化及实现

AES 加密算法包含字节替换、行移位、列混合、密钥加和密钥扩展等步骤，解密算法包含逆字节替换、逆行移位、逆列混合、密钥加和密钥扩展等步骤。本部分主要介绍字节替换、列混合和密钥扩展的优化及实现方法。

2.1 S 盒的优化及实现

S 盒是一 256 字节的查找表，它可以作为常量存储在 ROM 中，但需要占用一定的存储空间。在本设计中为了尽量少占用 ROM 单元，采用即时计算的方法生成。S 盒可通过以下两步变换得到[8]：

1）将各个字节表示成有限域 $\mathbf{GF}(2^8)$ 上的形式，并求出各字节的乘法逆元素；

假设在伽罗华域 $\mathbf{GF}(2^8)$ 的元素为 x，因为在伽罗华域上有 $x^{28-1} = x^{255} = 1$，所以 x 的逆 $x^{-1} = x^{-1} \cdot x^{255} = x^{254}$。

a）求乘法：假设两个 8 位二进制数为 $A(a_7, a_6, a_5, a_4, a_3, a_2, a_1, a_0)$，$B(b_7, b_6, b_5, b_4, b_3, b_2, b_1, b_0)$。字节 A、B 均可以看做一个系数在 $\mathbf{GF}(2)$ 上的多项式，即：

$$A(x) = a_7x^7 + a_6x^6 + a_5x^5 + a_4x^4 + a_3x^3 + a_2x^2 + a_1x + a_0$$
$$B(x) = b_7x^7 + b_6x^6 + b_5x^5 + b_4x^4 + b_3x^3 + b_2x^2 + b_1x + b_0$$

采用既约多项式 $m(x) = x^8 + x^4 + x^3 + x + 1$ 作为约化多项式来定义乘法 $C(x) = A(x) \cdot B(x)$，则：

$$C(x) = A(x) \cdot B(x) = c_{14}x^{14} + c_{13}x^{13} + c_{12}x^{12} + c_{11}x^{11} + c_{10}x^{10} + c_9x^9 + c_8x^8 + c_7x^7 + c_6x^6 + c_5x^5 + c_4x^4 + c_3x^3 + c_2x^2 + c_1x + c_0$$

通过相乘可以计算出 $C(x)$ 的系数 $c_0 \sim c_{14}$，具体数值在此省略。

利用约化多项式 $m(x) = x^8 + x^4 + x^3 + x + 1$ 进行化简，令 $m(x) = x^8 + x^4 + x^3 + x + 1 = 0$，则：

$$x^8 = x^4 + x^3 + x + 1$$
$$x^9 = x^5 + x^4 + x^2 + x$$
$$x^{10} = x^6 + x^5 + x^3 + x^2$$
$$x^{11} = x^7 + x^6 + x^4 + x^3$$
$$x^{12} = x^8 + x^7 + x^5 + x^4 = (x^4 + x^3 + x + 1) + x^7 + x^5 + x^4 = x^7 + x^5 + x^3 + x + 1$$
$$x^{13} = x^8 + x^6 + x^4 + x^2 + x = (x^4 + x^3 + x + 1) + x^6 + x^4 + x^2 + x = x^6 + x^3 + x^2 + 1$$
$$x^{14} = x^7 + x^4 + x^3 + x$$

将上式带入 $C(x)$ 表达式中，$C(x)$ 简化为 7 次多项式，并令该多项式为 $D(x)$。

$$D(x) = d_7x^7 + d_6x^6 + d_5x^5 + d_4x^4 + d_3x^3 + d_2x^2 + d_1x + d_0$$

其系数如下：

$$d_7 = c_7 \oplus c_{11} \oplus c_{12} \oplus c_{14}$$
$$d_6 = c_6 \oplus c_{10} \oplus c_{11} \oplus c_{13}$$
$$d_5 = c_5 \oplus c_9 \oplus c_{10} \oplus c_{12}$$
$$d_4 = c_4 \oplus c_8 \oplus c_9 \oplus c_{11} \oplus c_{14}$$
$$d_3 = c_3 \oplus c_8 \oplus c_{10} \oplus c_{11} \oplus c_{12} \oplus c_{13} \oplus c_{14}$$
$$d_2 = c_2 \oplus c_9 \oplus c_{10} \oplus c_{13}$$
$$d_1 = c_1 \oplus c_8 \oplus c_9 \oplus c_{12} \oplus c_{14}$$
$$d_0 = c_0 \oplus c_8 \oplus c_{12} \oplus c_{13}$$

b）求乘法逆。

乘法逆实现框图如图 1 所示[9]。图中的“×”表示两个数的乘法。

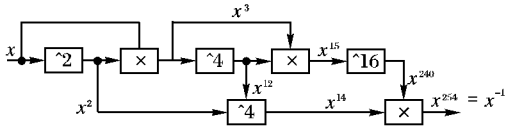

图 1 乘法逆实现

2）对乘法逆元素进行仿射变换。

$$\boldsymbol{b'}_i = \boldsymbol{b}_i \oplus \boldsymbol{b}_{(i+4) \bmod 8} \oplus \boldsymbol{b}_{(i+5) \bmod 8} \oplus \boldsymbol{b}_{(i+6) \bmod 8} \oplus \boldsymbol{b}_{(i+7) \bmod 8} \oplus \boldsymbol{c}_i; 0 \leqslant i \leqslant 7$$

$$\begin{bmatrix} \boldsymbol{b'}_0 \\ \boldsymbol{b'}_1 \\ \boldsymbol{b'}_2 \\ \boldsymbol{b'}_3 \\ \boldsymbol{b'}_4 \\ \boldsymbol{b'}_5 \\ \boldsymbol{b'}_6 \\ \boldsymbol{b'}_7 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} \boldsymbol{b}_0 \\ \boldsymbol{b}_1 \\ \boldsymbol{b}_2 \\ \boldsymbol{b}_3 \\ \boldsymbol{b}_4 \\ \boldsymbol{b}_5 \\ \boldsymbol{b}_6 \\ \boldsymbol{b}_7 \end{bmatrix} \oplus \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$

逆 S 盒可通过如下两部实现：

1）先进行仿射变换的逆变换，即：

$$\begin{bmatrix} \boldsymbol{b'}_0 \\ \boldsymbol{b'}_1 \\ \boldsymbol{b'}_2 \\ \boldsymbol{b'}_3 \\ \boldsymbol{b'}_4 \\ \boldsymbol{b'}_5 \\ \boldsymbol{b'}_6 \\ \boldsymbol{b'}_7 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} \boldsymbol{b}_0 \\ \boldsymbol{b}_1 \\ \boldsymbol{b}_2 \\ \boldsymbol{b}_3 \\ \boldsymbol{b}_4 \\ \boldsymbol{b}_5 \\ \boldsymbol{b}_6 \\ \boldsymbol{b}_7 \end{bmatrix} \oplus \begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

2）计算 $\mathbf{GF}(2^8)$ 上的乘法逆。

2.2 列混合的优化及实现

为了使列混合运算在 ARM 处理器上性能达到最优，将列混合运算进行如下优化：

ARM 处理器一个时钟周期可以处理 32 位数据[10-11]，而列混合是基于字节的变换，为了使其能够在 ARM 上高效运行，将其修改为针对 32 位字的变换[12-13]。令 a_i 为列混合前列构成的字（$0 \leqslant i \leqslant 3$），$b_i$ 为列混合后的列构成的字（$0 \leqslant i \leqslant 3$），列混合运算可表示为：

$$\boldsymbol{b}_0 = \{02\} * \boldsymbol{a}_0 \oplus \{03\} * \boldsymbol{a}_1 \oplus \boldsymbol{a}_2 \oplus \boldsymbol{a}_3$$
$$\boldsymbol{b}_1 = \boldsymbol{a}_0 \oplus \{02\} * \boldsymbol{a}_1 \oplus \{03\} * \boldsymbol{a}_2 \oplus \boldsymbol{a}_3$$
$$\boldsymbol{b}_2 = \boldsymbol{a}_0 \oplus \boldsymbol{a}_1 \oplus \{02\} * \boldsymbol{a}_2 \oplus \{03\} * \boldsymbol{a}_3$$
$$\boldsymbol{b}_3 = \{03\} * \boldsymbol{a}_0 \oplus \boldsymbol{a}_1 \oplus \boldsymbol{a}_2 \oplus \{02\} * \boldsymbol{a}_3$$

其中：$\boldsymbol{a}_i$，$\boldsymbol{b}_i$ 均由 4 个字节组成。在上式中，$\{02\} * \boldsymbol{a}_i$（记做 xtime()）表示 4 个有限域 $\mathbf{GF}(2^8)$ 上的乘法运算。ARM 处理器在一个时钟周期内能够并行处理 4 个有限域 $\mathbf{GF}(2^8)$ 上的乘法或异或运算。针对列混合中所涉及到的运算，可用如下方法来实

现[8-9]:第一步,求和;第二步,计算{02} $*a_i$;第三步,累加。

$$\text{第一步:}\begin{cases} b_0 = a_1 \oplus a_2 \oplus a_3 \\ b_1 = a_2 \oplus a_3 \oplus a_0 \\ b_2 = a_3 \oplus a_0 \oplus a_1 \\ b_2 = a_0 \oplus a_1 \oplus a_2 \end{cases}$$

$$\text{第二步:}\begin{cases} a_0 = \{02\} * a_0 \\ a_1 = \{02\} * a_1 \\ a_2 = \{02\} * a_2 \\ a_3 = \{02\} * a_3 \end{cases}$$

$$\text{第三步:}\begin{cases} b_0 \oplus = a_0 \oplus a_1 \\ b_1 \oplus = a_1 \oplus a_2 \\ b_2 \oplus = a_2 \oplus a_3 \\ b_3 \oplus = a_3 \oplus a_0 \end{cases}$$

完成列混合共需要 4 次 xtime( ) 和 16 次异或运算。

与列混合的实现方法类似,逆列混合可用如下方法实现:假设 a_i 为列混合变换之前状态矩阵的列构成的 32 位字,b_i 为列混合变换之后状态矩阵的列构成的 32 位字,则:

$$b_0 = \{0e\} * a_0 \oplus \{0b\} * a_1 \oplus \{0d\} * a_2 \oplus \{09\} * a_3$$
$$b_1 = \{09\} * a_0 \oplus \{0e\} * a_1 \oplus \{0b\} * a_2 \oplus \{0d\} * a_3$$
$$b_2 = \{0d\} * \alpha_0 \oplus \{09\} * a_1 \oplus \{0e\} * a_2 \oplus \{0b\} * a_3$$
$$b_3 = \{0b\} * a_0 \oplus \{0d\} * a_1 \oplus \{09\} * a_2 \oplus \{0e\} * a_3$$

可分六步实现:

$$\text{第一步:}\begin{cases} c_0 = a_1 \oplus a_2 \oplus a_3 \\ c_1 = a_2 \oplus a_3 \oplus a_0 \\ c_2 = a_3 \oplus a_0 \oplus a_1 \\ c_3 = a_0 \oplus a_1 \oplus a_2 \end{cases}$$

$$\text{第二步:}\begin{cases} d_0 = \{02\} * a_0 \\ d_1 = \{02\} * a_1 \\ d_2 = \{02\} * a_2 \\ d_3 = \{02\} * a_3 \end{cases}$$

$$\text{第三步:}\begin{cases} e_0 = c_0 \oplus d_0 \oplus d_1 \\ e_1 = c_1 \oplus d_1 \oplus d_2 \\ e_2 = c_2 \oplus d_2 \oplus d_3 \\ e_3 = c_3 \oplus d_3 \oplus d_0 \end{cases}$$

$$\text{第四步:}\begin{cases} f_0 = \{02\} * d_0 \\ f_1 = \{02\} * d_1 \\ f_2 = \{02\} * d_2 \\ f_3 = \{02\} * d_3 \end{cases}$$

$$\text{第五步:}\begin{cases} g_0 = \{02\} * f_0 \\ g_1 = \{02\} * f_1 \\ g_2 = \{02\} * f_2 \\ g_3 = \{02\} * f_3 \end{cases}$$

$$\text{第六步:}\begin{cases} b_0 = g_0 \oplus g_1 \oplus g_2 \oplus g_3 \oplus f_0 \oplus f_2 \oplus e_0 \\ b_1 = g_0 \oplus g_1 \oplus g_2 \oplus g_3 \oplus f_1 \oplus f_3 \oplus e_1 \\ b_2 = g_0 \oplus g_1 \oplus g_2 \oplus g_3 \oplus f_0 \oplus f_2 \oplus e_2 \\ b_3 = g_0 \oplus g_1 \oplus g_2 \oplus g_3 \oplus f_1 \oplus f_3 \oplus e_3 \end{cases}$$

完成逆列混合共需要 12 次 xtime( ) 和 40 次异或运算。

2.3　密钥扩展的优化及实现

密钥扩展分为一次性密钥扩展和即时密钥扩展。一次性密钥扩展,是在加密解密开始之前生成所有轮密钥,并将其存储在内存中。即时密钥扩展只保留一轮的密钥 —— 当前轮密钥,下一轮密钥由当前轮密钥计算生成。与一次性密钥扩展相比,即时密钥扩展占用更少的存储单元。在本设计中采用即时密钥扩展。

假设 128 位初始密钥为 $k_0k_1k_2k_3\cdots k_{12}k_{13}k_{14}k_{15}$,将其排列为 4 × 4 的状态矩阵 $\begin{bmatrix} k_0 & k_4 & k_8 & k_{12} \\ k_1 & k_5 & k_9 & k_{13} \\ k_2 & k_6 & k_{10} & k_{14} \\ k_3 & k_7 & k_{11} & k_{15} \end{bmatrix}$,并令 $\boldsymbol{rk}_0 = \begin{bmatrix} k_0 \\ k_1 \\ k_2 \\ k_3 \end{bmatrix}$,$\boldsymbol{rk}_1 = \begin{bmatrix} k_4 \\ k_5 \\ k_6 \\ k_7 \end{bmatrix}$,$\boldsymbol{rk}_2 = \begin{bmatrix} k_8 \\ k_9 \\ k_{10} \\ k_{11} \end{bmatrix}$,$\boldsymbol{rk}_3 = \begin{bmatrix} k_{12} \\ k_{13} \\ k_{14} \\ k_{15} \end{bmatrix}$;其中 $\boldsymbol{rk}_0$、$\boldsymbol{rk}_1$、$\boldsymbol{rk}_2$、$\boldsymbol{rk}_3$ 均为由 4 个字节构成的字。

改进后的加密密钥扩展可由下面的转换过程实现:

$$\begin{aligned} \boldsymbol{rk}'_0 = \boldsymbol{rk}_0 \oplus & ((Sbox(k_{13}) \ \&\ \text{0xffffffff}) << 24) \oplus \\ & ((Sbox(k_{14}) \ \&\ \text{0xffffffff}) << 16) \oplus \\ & ((Sbox(k_{15}) \ \&\ \text{0xffffffff}) << 8) \oplus \\ & ((Sbox(k_{12}) \ \&\ \text{0xffffffff})) \oplus rcon \end{aligned}$$

$$\boldsymbol{rk}'_1 = \boldsymbol{rk}'_0 \oplus \boldsymbol{rk}_1$$
$$\boldsymbol{rk}'_2 = \boldsymbol{rk}'_1 \oplus \boldsymbol{rk}_2$$
$$\boldsymbol{rk}'_3 = \boldsymbol{rk}'_2 \oplus \boldsymbol{rk}_3$$

加密密钥扩展算法的实现如图 2 所示。图 2 中根据初始密钥 $\boldsymbol{rk}_0$,$\boldsymbol{rk}_1$,$\boldsymbol{rk}_2$,$\boldsymbol{rk}_3$ 推算出第一轮加密密钥 $\boldsymbol{rk}'_0$,$\boldsymbol{rk}'_1$,$\boldsymbol{rk}'_2$,$\boldsymbol{rk}'_3$;第二轮密钥可由第一轮密钥按照同样方法生成。

解密密钥扩展是加密密钥扩展的逆过程,不同之处在于在进行解密密钥扩展之前首先要进行加密密钥扩展得到加密结尾轮轮密钥,然后以加密结尾轮密钥作为解密初始轮密钥进行解密密钥扩展。

3　实验数据及算法性能分析

在 S3C2440 开发板上,采用 C 语言编程,在 ARM Developer suite v1.2 开发平台上进行编译,调试。表 1 给出加密轮所涉及到的四个变换以及一轮密钥扩展所消耗的时钟周期数。从表中可看出进行字节替换(S 盒)所消耗的时钟周期数最多,占用的系统资源最多。其次是一轮密钥扩展所消耗的时钟周期数,因为密钥扩展也需要进行 S 盒数据的计算。行移位、列混合和密钥加占用的系统资源非常少,与字节替换和密钥扩展相比可以忽略不计。由于 S 盒中的 256 个数据全都采用即时计算的方法生成,因此 S 盒所消耗的时钟周期数多,这样可以无须在 ROM 中存储数据,减少了 ROM 空间的占用。

表 1　加密轮四个变换及一轮密钥扩展所消耗的时钟周期数对比

操作	Core_Cycles	S_Cycles	A_Cycles	Total
字节替换	98 404	82 697	247 916	330 613
行移位	32	35	105	140
列混合	392	392	1 004	1 396
密钥加	44	56	68	124
一轮密钥扩展	24 592	20 669	62 023	82 692

表 2 给出采用轮不打开方式进行编译得到的部分实验数据。Core_cycles 为系统程序运行时占用的内核时钟周期,S_cycles、A_cycles 分别表示程序运行时地址连续操作和空地址操作占用的时钟周期数。RO 是程序中的指令和常量,RW 是

程序中的已初始化变量,ZI 是程序中的未初始化的变量。从表 2 中可以看出,RO、RW、ZI 空间的占用在 3 种不同的情况下是完全相同的。RO 段存储轮常数(29 × 4 = 116 字节)、明文(16 字节)和初始密钥(16 字节)。RW 段存储针对不同长度的明文和密钥时需要加密的轮数(25 × 4 = 100 字节)。解密算法比加密算法占用的系统资源多,因为首先要进行所有轮的加密密钥扩展,然后即时生成解密轮密钥。

从表 1 和表 2 的数据可以看出,AES 算法可以在 ARM 上高效运行,所占用的系统资源较多。主要是因为 S 盒采用即时计算生成的方法生成,并不是存储在 ROM 中,这样减少了 ROM 的占用,但增加了系统的时钟周期数。这种方法可以应用在低速度要求、小存储容量的嵌入式加密系统中。

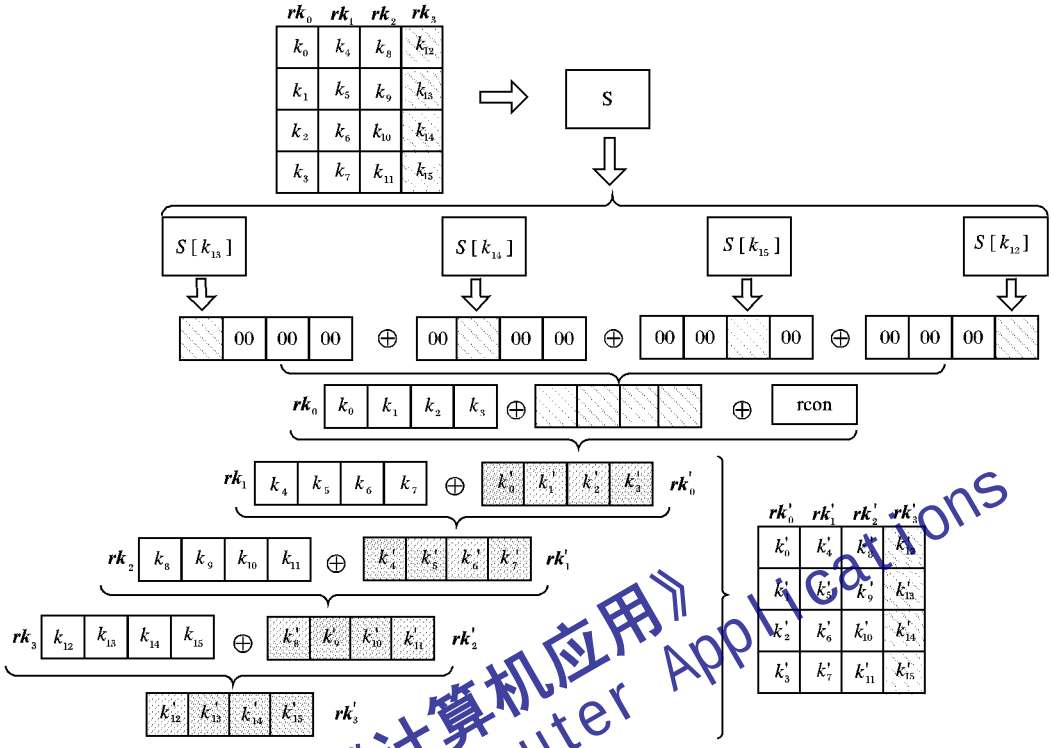

图 2 加密密钥扩展算法

表 2 采用轮不打开方式的时钟周期数和存储空间分配对比

操作	时钟周期数			存储空间			
	Core_Cycles	S_Cycles	A_Cycles	Code	RO Data	RW Data	ZI Data
1 次加密	1 254 665	1 038 545	3 112 539	3672	148	100	4
1 次解密	1 482 761	1 247 266	3 739 030	4 404	148	100	4
1 次加密 + 1 次解密	2 717 440	2 285 824	6 851 585	6 236	148	100	4

4 结语

本文提出 AES 算法的优化方案可以在 ARM 处理器上高效运行。该优化方案只对算法的实现方法进行了优化,并没有改变算法的本身结构,因此没有降低算法的安全性。本文以明文长度和密钥长度均为 128 位的 AES 算法为例,提出了算法的优化方案,该优化方案可以推广应用到其他数据长度的 AES 算法中。

参考文献:

[1] Announcing the Advanced Encryption Atandard (AES): Federal Information Processing Standards FIPS 197[EB/OL]. [2010 - 09 - 01]. http://csrc. nist. gov/publications/fips/fips197/fips - 197. pdf.

[2] DAEMEN J, RIJMEN V. 高级加密标准(AES)算法——Rijndael 的设计[M]. 谷大武, 徐胜波, 译. 北京: 清华大学出版社, 2003.

[3] 王海洋, 陈弘毅. 一种小面积的高吞吐率 AES 协处理器设计[J]. 微电子学与计算机, 2009, 26(6): 12 - 16.

[4] 韩少男, 李晓江. 一种小面积低功率串行 AES 硬件加解密电路[J]. 微电子学, 2010, 40(3): 347 - 353.

[5] 胡亮, 袁巍, 于孟涛, 等. 单向性策略与 AES 密钥生成算法的改进[J]. 吉林大学学报: 工学版, 2009, 1: 137 - 142.

[6] 杨宏志, 韩文报, 沈勇. AES 扩散层的分析及改进方案设计[J]. 计算机工程与应用, 2009, 45(26): 12 - 14.

[7] DAEMEN J, RIJMEN V. AES proposal: Rijndael[EB/OL]. [2010 - 04 - 20]. http://csrc. nist. gov/archive/aes/rijndael/ Rijndael-ammended. pdf.

[8] SATOH A, MORIOKA S, TAKANO K, *et al*. A compact Rijndael hardware architecture with S-box optimization [C]// ASIACRYPT'01: Proceedings of the 7th International Conference on the Theory and Application of Cryptology and Information Security: Advances in Cryptology, LNCS 2248. London: Springer-Verlag, 2001: 239 - 254.

[9] SubBytes transform circuit for AES cipher (Version 1.0) [EB/OL]. [2010 - 04 - 20]. http://www. ie. u-ryukyu. ac. jp/ ~ wada/design04/spec_e. html.

[10] S3C2440A Data sheet [EB/OL]. [2010 - 09 - 01]. http://cn. alldatasheet. com/datasheet-pdf/pdf/83787/SAMSUNG/S3C2440. html. 9 - 50.

[11] ARM Ltd. Website[EB/OL]. [2010 - 09 - 01]. http://www. arm. com.

[12] BERTONI G, BREVEGLIENT L, FRAGNETO P, *et al*. Efficient software implementation of AES on 32-bit platforms[C]// Cryptographic Hardware and Embedded Systems - CHES 2002, LNCS 2523. Berlin: Springer, 2002: 159 - 171.

[13] DARNALL M, KUHLMAN D. AES software implementations on ARM7TDMI[C]// Cryptology-INDOCRYPT 2006, LNCS 4329. Berlin: Springer, 2006: 424 - 435.