基于汉明纠错编码的 AES 硬件容错设计与实现

唐 明[1], 张国平[2], 张焕国[1]
(1. 武汉大学计算机学院, 湖北武汉 430079; 2. 武汉数字工程研究所, 湖北武汉 430074)

摘 要: 提出一种 AES 硬件容错设计可避免攻击者利用在 AES 设计环节中插入故障位实现攻击. 在原有 AES 硬件设计中加入汉明码纠错电路, 能自动纠正同一字节内的所有单比特故障, 硬件仿真实验证明, 故障发现率接近 100%. 针对不同 AES 设计结构和测试点配置对纠错电路的资源及速度进行了分析, 实验结果表明我们提出的硬件容错设计有很强的可行性.

关键词: 高级加密标准; 汉明码; 容错; 纠错码; 故障发现
中图分类号: TP309 文献标识码: A 文章编号: 0372-2112 (2005) 11-2013-04

Automatic Error Correcting Hardware Implementation of AES Algorithm

TANG Ming[1], ZHANG Guo-ping[2], ZHANG Huan-guo[1]
(1. *School of Computer, Wuhan University, Wuhan, Hubei 430072, China;* 2. *709th Research Institution, Wuhan, Hubei 430074, China*)

Abstract: We propose a fault tolerance AES hardware implementation to prevent attackers from injecting faults in the process of AES design. The design adds Hamming Code error correction circuits into the original AES implementation and it can correct all single fault in the same byte automatically. Hardware simulation shows that the ratio of fault detection is close to 100%. The results of analyzing of different architecture and configuration of check points certificate that our proposal is very practical.

Key words: advanced encryption standard(AES); hamming code; fault tolerance; error correcting codes; fault detection

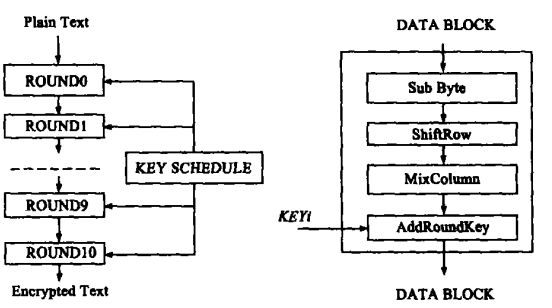
1 引言

AES 算法一经提出就随之出现了许多针对其软硬件设计的攻击方法, 而目前的热点攻击方法又大多基于在 AES 设计环节中插入故障位[5] 实现. 就 AES 硬件设计中如何避免插入故障位的问题, 我们提出了一种基于汉明纠错码的 AES 硬件容错设计方案, 该设计可及时发现是否存在对 AES 运算环节中间结果的篡改, 从而防止基于插入故障位的攻击.

目前针对 AES 算法中插入故障位的硬件检测方法主要包括两种: 一种是基于冗余设计的检测方法(如文[6]中使用基于资源冗余的故障检测机制发现硬件加密单元中的故障, 和文[7]提出的利用时间冗余方法检测故障等), 这些方法需要同时将加密和解密算法配置到一块硬件电路(或 FPGA)中, 检测原理是在加密处理完成后立即进行解密从而发现故障, 很显然这些方法需要占用大量资源和测试时间, 并且因为 AES 硬件设计多采用分时下载加密和解密配置的设计方法, 所以基于冗余设计测试方法的可行性较差; 另一种是由 Guido Bertoni 等提出的基于奇偶校验码的测试方法, 这种方法对插入奇数比特的故障覆盖率为 100%, 它利用每个字节中生成的偶校验位对是否出现插入故障进行判断. 虽然奇偶校验是非常简单的一种校验码, 但测试判断时需占用较多硬件设计资源(大约占整个设计的 10-20%[4]), 而若要进行故障定位(纠错)则需要使用更复杂的逻辑电路. 以上两种方法都可直接用于 AES 硬件设计但需要占用较多硬件资源, 相比较而言, 我们设计的纠错电路利用了汉明码的纠错能力和较强的故障位发现能力, 无须增加额外电路即可实现发现故障的目的. 硬件仿真实验证明, 我们的设计方案对故障的发现率接近 100% 且具有很强的可行性.

2 AES 算法硬件实现结构

AES 算法硬件实现主要采用两种结构: 流水线结构和循环结构. 这两种结构针对不同的应用需求, 由于流水线结构比全并行结构有更高的速度/资源比, 所以成为高速 AES 算法硬件设计的主流结构; 循环结构则采取与原算法基本相同的流程, 相同轮次共享逻辑电路使这种结构所占资源较少而成为低成本设计的典型结构. 两种设计结构如图 1 所示.



(a)流水线结构实现 AES 算法 (b)循环结构实现 AES 算法

图 1 AES 算法硬件设计结构图[3]

收稿日期: 2005-05-10; 修回日期: 2005-07-29
基金项目: 国家 863 项目(No. 2002AA141051); 国家自然基金(No. 90104005, 60373087, 66973034, 60473023); 国家教育部博士点基金(No. 20020486046)

论文的第五部分将针对这两种设计结构分析纠错电路对原设计速度和资源的影响程度.

3 基于插入故障位的 AES 攻击方法简介

目前针对 AES 算法的攻击多以插入故障位作为猜测密钥或状态位的条件. 下面我们介绍一种针对 Xtime 操作硬件设计的攻击[5].

Xtime 是 MixColumn 设计时常用的操作单元, 若直接用分支语句设计 Xtime 操作, 则很容易受到 Timing Analysis 攻击. 所以在硬件设计中将条件判断转化为逻辑门阵列. 但即便如此, 攻击者仍能利用 AES 算法各运算的中间结果存放在寄存器的特点, 用 optical/ eddy current 攻击对 ShiftRow 与 MixColumn 中间状态的最高位进行复位. 攻击者判断的方法是, 若原状态中的最高位为 0, 则 Xtime 操作的结果正确, 进而加密结果正确(攻击者事先获得加密的正确结果); 若原状态的最高位为 1, 则加密处理有误. 于是根据加密结果攻击者可顺利判断原状态的最高位.

只要对上面的攻击进行简单分析就会发现, 防止这类攻击可从两方面入手: (1) 是不把处理的中间结果存入寄存器, 但由于 AES 加密不可能在一个时钟周期处理完, 并且在 FPGA 设计中要选择寄存器以外的方式存储中间结果是很困难的, 所以这种解决方法难以实现; (2) 是及时发现攻击者对中间结果的篡改, 若能定位故障则可进一步纠正故障使攻击者的篡改无法起作用, 从而破坏攻击的判断依据. 正是考虑到第二种方法的可行性, 我们提出一种基于汉明纠错码的 AES 容错设计方案.

4 基于校验码的 AES 容错设计

首先将 AES 加密算法中两个相邻运算间的传输通道看成是一个受外部干扰的信道, 攻击者的行为看成外部干扰信号. 这时, 发现并纠正 AES 设计中错误比特的任务就转化成如何发现并去掉有扰信道上干扰信号的问题. 除了基于冗余设计的故障检测方法外, 最简单的方法就是利用纠错码对每段可能遭受攻击的信道进行编码和测试. 参考文献[3]中介绍了一种基于奇偶校验码的检测方法, 该方法只适合发现故障, 要纠正故障则需付出较大成本. 而我们提出的故障检测方法基于汉明纠错编码, 无论在故障发现还是纠正故障方面都更具优势并且有很强的可行性.

4.1 基于奇偶校验码的 AES 容错设计[3,4]

测试基本原理是在明文的每个字节后插入一比特偶校验位, 把各校验位和对应字节奇偶性的异或结果作为该字节的错误图样, 由于 AES 硬件设计多采用密钥长度为 128 比特, 所以最终错误集合可组成一个 4* 4 的二维矩阵 $\boldsymbol{E}$, 分析各运算对故障矩阵影响后利用故障矩阵 $\boldsymbol{E}$ 发现故障.

只要是基于校验码的容错设计都需要在原设计中增加校验码编码器和校验码译码器, 分别完成校验码生成与测试, 基于奇偶校验码的检测方法采用在整个加密的最后一轮结束时设置一个测试点的方法. 这种方法在每个运算中都加入了偶校验码发生器, 为配合测试还设置了校验码推测电路. 经过硬件仿真实验发现, 该方法能发现所有在各轮入口处插入的单比特故障和奇数比特故障.

这种测试方法的主要缺点是发现故障的电路成本较高, 且需要使用更复杂的逻辑电路才能定位故障.

4.2 基于汉明码的 AES 容错设计

4.2.1 汉明纠错码 我们选择[12,8]扩展汉明码作为 AES 硬件设计中的纠错码, 下面详细说明[12,8]汉明码的纠错原理. 设 $\boldsymbol{C}$ 是传输的汉明码, $\boldsymbol{S}$ 是伴随阵.

$\boldsymbol{C}=[DB_7, DB_6, DB_5, DB_4, DB_3, DB_2, DB_1, DB_0, CB_3, CB_2, CB_1, CB_0]$, 其中 $DB_i(i\in[0,7])$ 是数据位, $CB_i(i\in[0,3])$ 是校验位. 伴随阵 $\boldsymbol{S}=[S_3, S_2, S_1, S_0]$.

$$CB_3= DB_7\oplus DB_6\oplus DB_5\oplus DB_4;$$
$$S_3= DB_7\oplus DB_6\oplus DB_5\oplus DB_4\oplus CB_3;$$
$$CB_2= DB_7\oplus DB_3\oplus DB_2\oplus DB_1;$$
$$S_2= DB_7\oplus DB_3\oplus DB_2\oplus DB_1\oplus CB_2;$$
$$CB_1= DB_6\oplus DB_5\oplus DB_3\oplus DB_2\oplus DB_0;$$
$$S_1= DB_6\oplus DB_5\oplus DB_3\oplus DB_2\oplus DB_0\oplus CB_1;$$
$$CB_0= DB_6\oplus DB_4\oplus DB_3\oplus DB_1\oplus DB_0;$$
$$S_0= DB_6\oplus DB_4\oplus DB_3\oplus DB_1\oplus DB_0\oplus CB_0.$$

表 1 显示了利用伴随阵中的元素组合可以定位 12 位汉明码中的单个故障.

表 1 故障对照表

伴随矩阵	出错位												无错
	DB_7	DB_6	DB_5	DB_4	DB_3	DB_2	DB_1	DB_0	CB_3	CB_2	CB_1	CB_0	
S_3	1	1	1	1	0	0	0	0	1	0	0	0	0
S_2	1	0	0	0	1	1	1	0	0	1	0	0	0
S_1	0	1	1	0	1	1	0	1	0	0	1	0	0
S_0	0	1	0	1	1	0	1	1	0	0	0	1	0

4.2.2 检测方法 接下来将详细介绍测试点的设置方法和加入纠错功能后运算部件的硬件实现方式.

4.2.2.1 测试点的设置 测试点将直接影响测试的效果和成本, 下面说明三种测试点设置方法.

(1)在每个运算的输入端设置汉明码译码器, 输出端设置汉明码发生器: 这样将原来的 8 比特数据扩展成 12 比特传输到下一个运算的输入端, 而各运算(除初始轮外)的输入端加入的汉明码译码器对出错状态进行纠正. 除初始轮次外, 每轮加密处理需要分别使用 3 个汉明码发生器和译码器, 一轮加密处理的结构如图 2 所示. 这种设计结构可以快速准确发现并纠正错误, 但为此付出的成本也是最大的且对原系统加密处理速度的影响也最大.

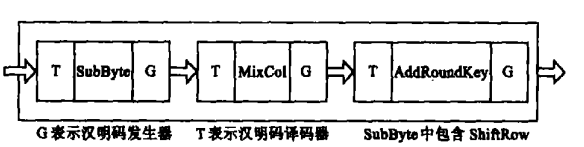


图 2 第一种测试点配置结构图

(2)在每轮的结尾处设置测试点: 在每轮最后一个运算的结尾处设置校验码发生器, 由于后一级汉明码可由前一级生成(具体推导过程见附录), 所以由校验码独立推导电路生成各运算的校验码, 通过比较每轮结尾处产生的校验码与独立推导电路产生的校验码可发现故障, 结构如图 3 所示.

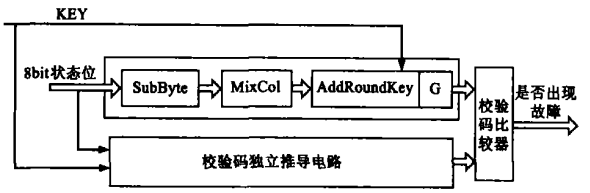


图 3　第二种测试点配置结构图

由图 3 可发现, 这种设置在一轮加密处理中只能减少两组纠错电路所用的时间, 而校验码的独立推导电路需另外占用资源, 所以这种设置不是最合理的.

(3) 在加密流程的最后一轮输出端设置译码器: 与第二种设置类似, 只在加密流程的第一个和最后一个运算的输出端加入汉明码发生器, 通过与校验码独立推导电路的结果比较后可发现故障. 若 AES 算法采用流水线结构设计, 则这种测试点配置方法可大幅度减少运算输入端和输出端的纠错电路, 同时降低对每组数据(128 比特) 原有处理时间的影响, 但另一方面, 这种配置与第二种配置一样要增加校验码独立推导电路, 并且流水线结构之所以能提高系统处理速度主要依赖大批量的待处理数据, 而不是靠仅提高一组数据的处理速度, 所以这种配置对循环结构的处理速度影响更明显.

以上三种测试点配置方法均可用于流水线结构和循环结构, 具体性能参数将在论文第五部分详细说明.

4.2.2.2　加入纠错功能后的运算部件　以 *SubByte* 运算为例, 选择第一种测试点配置方法说明 *AES* 的各运算部件加入纠错功能后电路结构的变化.

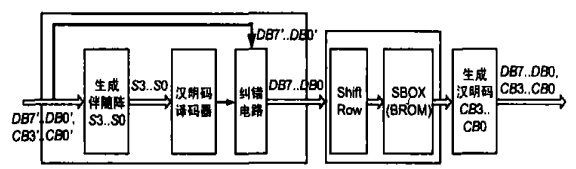


图 4　加入纠错功能后的 SubByte 结构图

SubByte(含 ShiftRow): 硬件设计中常将 *ShiftRow* 与 *SubByte* 合并. *SubByte* 用查表方式实现, 采用 *XILINX* 公司的 *Virtex*2 芯片, *SBOX* 占用 *BlockRAM* 资源(将 *BRAM* 设计成 *ROM*). 为把 *ROM* 的输出直接作为该运算的输出(否则利用 *BRAM* 资源很难同时在一拍内实现 *SubByte* 和 *ShiftRow* 两种运算), 将 *ShiftRow* 放在 *SubByte* 之前实现(两种运算的顺序不影响结果).

5　容错设计的硬件仿真结果

硬件仿真实验中采用的芯片分别是 XILINX 公司的 XC2V3K 芯片(流水线设计结构) 和 XC2S50 芯片(循环设计结构)、设计工具 ISE WebPack5.1、仿真工具采用 ModelSim5.6.

先对插入单比特故障进行硬件仿真, 再结合仿真数据分析容错设计对多比特故障的检测能力.

5.1　插入单比特故障的硬件仿真结果

容错设计能发现并纠正所有单比特故障, 而原设计的结构和测试点的配置都会影响容错设计的资源和处理速度, 表 2 显示了三种测试点配置在不同结构中所占的资源与时钟周期.

表 2　不同测试点配置在不同设计结构中对资源和速度影响情况

流水线结构				循环结构			
测试点配置	增加的资源		增加的时钟周期数	测试点配置	增加的资源		增加的时钟周期数
	Slices	BRAM(bit)			Slices	BRAM(bit)	
配置 1	435	0	59	配置 1	45	0	59
配置 2	113	10K	20	配置 2	12	1K	20
配置 3	105	10K	3	配置 3	17	1K	3

说明: 原设计所用的时钟频率= 25MHz, 表中的资源和时钟周期数是由纠错电路产生的, 故障覆盖率= 100%, 测试点配置分类同论文 4.2.2.1, 三种测试点配置方法在流水线结构中增加的时钟数只是针对第一组数据的处理时间, 这种影响会随着数据分组数的增加而逐渐降低.

从表 2 中可以看到, 第三种测试点配置方法在流水线结构和循环结构中都是最优的, 但第一种配置可以及时纠正所有单比特故障, 而第二和第三种配置则只能保证发现所有单比特故障, 所以应根据实际需要选择以上三种配置.

5.2　插入多比特故障的硬件仿真结果

若多比特故障分别插入在多个不同字节中, 则汉明码纠错电路可纠正所有故障, 故障覆盖率达 100%. 但当两位或两位以上故障同时插入到一个中间状态字节时, 需进行如下分析(设所有状态位被篡改的概率相同) :

(1) 当插入同一中间状态的故障数为奇数时, 根据 4.2.1 中伴随阵的公式, 故障发现率为 100%.

(2) 当插入同一中间状态的故障数是偶数且有故障位发生在校验位时, 通过在运算的输入端加入校验码生成电路配合比较电路可发现故障, 这类故障的发现率等于 100%.

(3) 若偶数个故障出现在伴随阵四个公式中的同一个公式的数据位时, 会出现故障屏蔽的情况, 但这种情况出现的概率很低, 实验结果如表 3 所示.

最后将我们的设计与其他测试方法进行测试性能与占用资源方面的比较(如表 3 所示) . 可以发现虽然 Guido Bertoni 提出的测试方法在多比特故障发现率方面略好于我们的设计, 但因测试增加的资源却远高于我们的设计, 所以在综合比较测试性能与成本后我们的设计显示出更高的可行度.

表 3　同类方法测试效果与资源比较

测试方法	故障发现率(%)						增加的资源	
	单比特	奇数比特	2 比特	4 比特	6 比特	8 比特	Slices	BRAM (bit)
Guido Bertoni[6]	NO	100	99.13	99.95	99.99	99.99	178	256* 9
Our Design	100	100	97.56	99.83	99.98	99.99	45	NO

说明: Guido Bertoni 提到的单比特故障是指一比特故障(归入奇数比特), 而表 3 中的单比特是指一个数据字节中插入一比特故障. 为了便于比较增加的芯片资源和时钟周期, 两种设计都采用 XC2S50FPGA 芯片(循环设计结构), 芯片设计工具 ISE WebPack5.1, 仿真工具 ModelSim5.6.

6　总结

为了防止攻击者利用插入故障位对 AES 设计进行攻击, 我们在原 AES 硬件设计中引入一种基于汉明纠错码的故障

检测电路. 通过硬件仿真实验对 AES 不同设计结构中的多种测试配置进行资源和速度的分析比较, 并与其他测试方法在故障发现率及占用资源方面进行比较, 为实际应用提供了参考依据. 仿真实验证明这种新的 AES 纠错电路具有非常高的故障覆盖率和故障发现率, 可发现所有插入同一状态(12 比特) 的单比特和所有奇数位故障, 而偶数位故障屏蔽的概率也非常低. 实验结果表明, 无论从故障检测性能还是对原设计的速度及资源影响上分析我们提出的这种具有自动纠错功能的 AES 硬件设计方案都具有很高的可行性.

附录: 校验码独立推导电路逻辑

为了能清楚说明校验码独立电路是如何生成的, 引入校验码阵列 $\boldsymbol{C}$(容量是 256* 4bit), $\boldsymbol{C}$ 中的元素 E_{ij} 表示对应位置数据(8bit)的汉明校验码, 每个元素又由四比特校验位组成, 即 $E_{ij}=(CB_3, CB_2, CB_1, CB_0)_{ij}$.

(1) SubByte: 由于 SBOX 中的值是固定的, 所以可事先产生 SBOX 对应的校验码阵列 $\boldsymbol{C}$, 且各元素互不干扰.

(2) ShiftRow: 只是改变校验码阵列 $\boldsymbol{C}$ 中各元素的位置.

(3) MixColumn: 考虑到论文的篇幅, 只说明 $\boldsymbol{C}$ 中一个元素的推导过程.

$$\begin{bmatrix} S_{00} & S_{01} & S_{02} & S_{03} \\ S_{10} & S_{11} & S_{12} & S_{13} \\ S_{20} & S_{21} & S_{22} & S_{23} \\ S_{30} & S_{31} & S_{32} & S_{33} \end{bmatrix} \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} = \begin{bmatrix} S'_{00} & S'_{01} & S'_{02} & S'_{03} \\ S'_{10} & S'_{11} & S'_{12} & S'_{13} \\ S'_{20} & S'_{21} & S'_{22} & S'_{23} \\ S'_{30} & S'_{31} & S'_{32} & S'_{33} \end{bmatrix} \quad (1)$$

公式(1)表示 MixColumn 的逻辑, 其中 Xtime 的功能就是实现 02* S_{ij}.

$$\begin{aligned} S'_{00} &= 02\times S_{00} \oplus 03\times S_{01} \oplus S_{02} \oplus S_{03} \\ &= Xtime(S_{00}) \oplus XtimeS_{01} \oplus S_{02} \oplus S_{03} \end{aligned} \quad (2)$$

S_{01}, S_{02}, S_{03} 分别有对应的校验码 E_{01}, E_{02} 和 E_{03}, 而 $S_{01}\oplus S_{02}\oplus S_{03}$ 的结果所对应的校验码 $E'=(CB'_3, CB'_2, CB'_1, CB'_0)$ 可以由 E_{01}, E_{02} 和 E_{03} 推导出:

$CB'_3=(DB_{7\_03}\oplus DB_{7\_02}\oplus DB_{7\_01})\oplus(DB_{6\_03}\oplus DB_{6\_02}\oplus DB_{6\_01})\oplus(DB_{5\_03}\oplus DB_{5\_02}, DB_{i\_j}$, 表示 S_j 对应 $\oplus DB_{5\_01})\oplus(DB_{4\_03}\oplus DB_{4\_02}\oplus DB_{4\_01})=CB_{3\_03}\oplus CB_{3\_02}\oplus CB_{3\_01}$

的第 i 个数据位, CB_{i_j} 表示 S_j 对应的第 i 个校验位. 同理, 可得到 CB'_2, CB'_2 和 CB'_0, 并推导出: $E'=E_{01}\oplus E_{02}\oplus E_{03}$.

设 $Xtime(E_{00})=(CB_3, CB_2, CB_1, CB_0)$ 是 $Xtime(S_{00})$ 对应的校验位, 这些校验位可以通过前面的数据位得到:

$$CB_3=DB_{6\_00}\oplus DB_{5\_00}\oplus DB_{4\_00}\oplus DB_{3\_00}\oplus DB_{7\_00},$$

$$CB_2=DB_{6\_00}\oplus DB_{2\_00}\oplus DB_{1\_00}\oplus DB_{0\_00}$$

$$CB_1=DB_{5\_00}\oplus DB_{4\_00}\oplus DB_{2\_00}\oplus DB_{1\_00},$$

$$CB_0=DB_{5\_00}\oplus DB_{3\_00}\oplus DB_{2\_00}\oplus DB_{0\_00}$$

同理由 $Xtime(S_{01})$ 可得到 $Xtime(E_{01})$, 由此可知 S'_{00} 的校验码可通过 MixColumn 的输入端数据位直接生成. 由于 SubByte(含 ShiftRow) 运算的校验码可事先生成, 所以 MixColumn 的校验码也可以不通过加密流程生成, 而是由一套独立电路根据 SubByte 的输入产生.

(4) AddRoundKey: 由于 AddRoundKey 运算只是将对应轮次的密钥与数据进行异或, 所以只要在校验码独立推导电路中加入密钥输入端即可.

参考文献:

[1] Daemen J, Rijmen V. AES Proposal: Rijndael[EB/OL]. http://csrc.nist.gov/encyption/aes/rijndael/Rijndael.pdf, 2002-05-01.

[2] Mcloone M, McCanny J V. High performance single-chip FPGA rijndael algorithm implementations[A]. Cryptographic Hardware and Embedded Systems (CHES 2001)[C]. Heidelberg: Springer-Verlag LNCS2162, 2001.65-76.

[3] Guido Bertoni, Luca Breveglieri, Israel Koren. Error analysis and detection procedures for a hardware implementation of the advanced encryption standard[J]. IEEE Transactions on Computers, 2003, 52(4): 492-505.

[4] Guido Bertoni, Luca Breveglieri, Israel Koren. An efficient hardware-based fault diagnosis scheme for AES: performances and cost[A]. Proceedings of 19th IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems[C]. IEEE Computer Society, Cannes, France, 2004. 130-138.

[5] Johannes Blomer, Jean Pierre Seifert. Fault based cryptanalysis of the advanced encryption standard (AES)[J]. Heidelberg: Springer-Verlag LNCS 2742, 2003, 2742: 162-181.

[6] Karri R, et al. Fault-based side-channel cryptanalysis tolerant rijndael symmetric block cipher architecture[A]. Proceedings of the 2001 IEEE Defect and Fault Tolerance in VLSI Systems[C]. IEEE Computer Society, San Francisco, CA, USA, 2001. 418-426.

[7] Fernández-Gómez S, et al. Concurrent error detection in block ciphers[A]. Proceedings of the 2000 Intern. Test Conference[C]. IEEE Cat. No.00CH37159, Atlantic City, NJ, USA, 2000. 979-984.

[8] 周盛雨, 陈晓敏. 一种纠错编码器的实现[J]. 电子技术, 2003, 30(3): 10-12.

[9] 王新梅, 肖国镇. 纠错码-原理与方法[M]. 西安: 西安电子科技大学出版社, 2001.

作者简介:

唐 明 女, 1976 年生于武汉市, 1999 年获武汉测绘科技大学计算机学院工学学士学位, 2003 年获武汉大学计算机学院工学硕士学位, 现为武汉大学博士研究生, 主要从事信息安全, 容错计算, 芯片设计等.

E-mail: m.tang@126.com; m.tang@sohu.com.

张国平 男, 1976 年生于宁夏固原县, 1999 年获武汉测绘科技大学计算机学院工学学士学位, 现为武汉数字工程研究所研究生, 主要研究领域是计算机网络, 高可用结构及技术.