第34卷 第13期 计 算 机 工 程 2008年7月
Vol.34 No.13 Computer Engineering July 2008
·安全技术· 文章编号：1000—3428(2008)13—0113—02 文献标识码：A 中图分类号：TN918

基于AES算法的Cache Hit旁路攻击

邓高明，赵 强，张 鹏，陈开颜
(军械工程学院计算机工程系，石家庄 050003)

摘 要：AES加密快速实现中利用了查表操作，查表的索引值会影响Cache命中率和加密时间，而查表的索引值和密钥存在密切关系。通过分析AES最后一轮加密过程中查表索引值与密文和最后一轮子密钥的关系，以及它们对Cache命中与否和加密时间长短的影响，提出一种利用Cache hit信息作为旁路信息对AES进行旁路攻击的技术，在Intel Celeron 1.99 GHz和Pentium4 3.6 GHz CPU的环境中，分别在2^{21}和2^{25}个随机明文样本的条件下，在5 min内恢复了OpenSSL v.0.9.8(a)库中AES的128 bit密钥，并介绍防御这种攻击途径的手段。

关键词：旁路攻击；Cache 命中；AES 算法

Cache Hit Side Channel Attack Based on AES

DENG Gao-ming, ZHAO Qiang, ZHANG Peng, CHEN Kai-yan
(Dept. of Computer Engineering, Ordnance Engineering College, Shijiazhuang 050003)

【Abstract】The AES software implementes in the way of looking up tables, while the indices affect the Cache hit and miss, and then the time of the AES encryption, however, the indices have a close connection with the secret key. After analyzing the relationship between the indices, and the ciphertext, and final round sub key in the AES final round encryption, it proposes a novel attack against AES by using the Cache hit information, and validates its feasibility with experiments on Intel Celeron 1.99 GHz and Pentium4 3.6 GHz CPU, recovers the 128 bit AES key in 2^{21} and 2^{25} random plaintexts in 5 min separately, and introduces several countermeasures for protecting the AES.

【Key words】Side Channel Attacks(SCA); Cache hit; AES

1 概述

近来，人们提出了一种以密码设备运行时泄漏的物理信息(如时间、功率、电磁辐射、运行错误等)作为分析渠道的新密码分析技术——旁路密码分析技术(Side Channel Analysis, SCA)。在传统概念下，加密过程可以看作一个由明文P和密钥K作为输入参数，密文C作为输出参数的数学函数$C=F(P,K)$，而在旁路分析条件下，这个函数的输入还应该包括攻击者的输入A和攻击者能够获得的泄漏信息L，也就是$(C,L)=F(P,K,A)$。已有的旁路密码分析技术大多对智能卡或者芯片上实现的密码系统进行分析，主要利用密码芯片工作时泄漏的功率消耗和电磁泄漏信息，计时攻击[1]也是旁路密码攻击中的一种，它通过精确测量加密解密程序在物理设备上的运行时间来进行密码分析，由于计时在个人计算机上能够更加精确地被实现，因此计时攻击也能在个人计算机上得以实现。特别的是，通过精确测量加密程序的运行时间，结合Cache在PC机中的特殊作用，产生了一种利用Cache对PC机进行计时攻击的新方法。本文介绍一种基于Cache行为的针对AES密码实现的攻击。

2 AES加密算法

AES(Advanced Encryption Standard)密码算法在2001年被美国国家技术标准局(NIST)选用。其前身是Rijmen和Daemen提出的Rijndael算法[2]。AES密码算法支持128位、192位和256位密钥的不同版本，本文只关注128位密钥的AES，192位和256位密钥版本只是采用了不同密钥扩展算法和更多的加密轮数。

AES是一个基于有限域运算的迭代密码：第i轮迭代根据16 B的输入状态S_i和一个16 B的子密钥K_i，产生一个16 B的输出状态S_{i+1}。除最后一轮之外，每一轮迭代运算都包括对S_i的以下4个的代数运算：字节代换(SubBytes)，行移位(ShiftRows)，列混淆(MixColumns)，与轮子密钥K_i作异或运算(AddRoundKey)。

为了提高性能，现在常用的AES软件实现将前3个操作合并并且预先进行计算，结果存储在几个大的查找表(T_0, T_1, T_2, T_3)中，每一个查找表将1 B的输入映射到4 B的输出。每一轮的运算都是首先将S_i分割成16 B的小块$S_i^0 S\cdots S_i^{15}$，K_i分割成16 B的小块$\kappa_i^0 K_i^1\cdots K_i^{15}$，之后进行如下的轮加密过程：

$$\begin{cases}(S_{i+1}^0,S_{i+1}^1,S_{i+1}^2,S_{i+1}^3)=T_0[S_i^0]\oplus T_1[S_i^5]\oplus T_2[S_i^{10}]\oplus T_3[S_i^{15}]\oplus(K_i^0,K_i^1,K_i^2,K_i^3)\\(S_{i+1}^4,S_{i+1}^5,S_{i+1}^6,S_{i+1}^7)=T_0[S_i^4]\oplus T_1[S_i^9]\oplus T_2[S_i^{14}]\oplus T_3[S_i^3]\oplus(K_i^4,K_i^5,K_i^6,K_i^7)\\(S_{i+1}^8,S_{i+1}^9,S_{i+1}^{10},S_{i+1}^{11})=T_0[S_i^8]\oplus T_1[S_i^{13}]\oplus T_2[S_i^2]\oplus T_3[S_i^7]\oplus(K_i^8,K_i^9,K_i^{10},K_i^{11})\\(S_{i+1}^{12},S_{i+1}^{13},S_{i+1}^{14},S_{i+1}^{15})=T_0[S_i^{12}]\oplus T_1[S_i^1]\oplus T_2[S_i^6]\oplus T_3[S_i^{11}]\oplus(K_i^{12},K_i^{13},K_i^{14},K_i^{15})\end{cases}\quad(1)$$

轮计算的软件实现采用这种方法效率比较高，因为只进行了16次查表操作和16次4 B的异或操作。在初始密钥被扩展为10个轮密钥之后，一个完整的加密过程首先进行一次16 B明文P和16 B初始密钥K的异或操作(AddRoundKey)，然后进行9次普通的轮加密操作，再加上一次更为简单的最

基金项目：国家自然科学基金资助项目“集成电路芯片电磁泄漏旁路攻击机理及解密研究”(60571037)

作者简介：邓高明(1983－)，男，硕士研究生，主研方向：信息安全；赵 强，博士生导师；张 鹏，博士研究生；陈开颜，副教授、在职博士研究生

收稿日期：2007-11-12 **E-mail：**denggaoming26@163.com

后一轮，最后一轮没有列混淆(MixColumns)操作，这一步是关键的，因为它使软件实现在最后一轮采用一个新的 $T4$ 表。

128 位AES共采用 10 轮加密，但是需要 11 个 16 B的子密钥，因为在轮加密之前有一次 16 B明文P和 16 B初始密钥K的异或操作。这 176 B的密钥都是从 16 B的原始密钥产生出来的，方法是反复地从前一个 16 B块通过一个非线性变换得到后一 16 B的块，直到得到所有 176 B密钥。这个密钥扩展结构在给出扩展密钥的连续 16 B的情况下是可逆的[2]。这一点对攻击者来说是很有用的，因为只要恢复了最后 16 B扩展密钥(或者其他 16 B)就相当于恢复了原始密钥。

以上对AES的描述都是原始的Rijndael报告[2]的一部分。而大多的AES实现对原始的最优Rijndael代码没有进行较大的改变，所以，本文的攻击能够广泛适用。Crypto++5.2.1, LibTomCrypt1.09 和本实验中AES的实现都采用了原始的Rijndael C实现。

3 基于 Cache 攻击的原理

3.1 Cache 原理

Cache 是一个在 CPU 和内存之间的小型高速存储器。当主存中的数据被访问时，这些数据就会将 Cache 中的旧数据替换出去。紧接其后的相同位置的主存访问就只需要从 Cache 中获取数据，速度比从主存中获取数据要快，这被称作 Cache 命中(Cache Hit)。CPU 与 Cache 之间的数据交换是以字为单位，而 Cache 与主存之间的数据交换是以行为单位。行的大小从 Pentium III 的 32 B 到更先进的 Pentium4 64 B 或者 AMD Athlon 处理器的 128 B 不等。

通常AES的查找表的元素是 4 B，一个行大小为 32 B的Cache每一行就包含 8 个连续的表元素，因此令定义这个值为 δ[3]。对AES表中任意一个表元素的索引值(即数组下标)l，如果不考虑其低lbδ位，同一个Cache行中任意两个元素的表索引值l'和l就是相等的，记为$<l>=<l'>$[3]，因为访问l对应的元素时，就会把l'对应的元素也调入到Cache中。

笔者把AES加密看作是一个以$l_1,l_2,\cdots,l_{160}$为索引值的 160 个查找表的操作。如果两次不同的索引l_i,l_j满足$<l_i>=<l_j>$，就会发生一次Cache命中(考虑现代高端CPU Cache大小都比AES的查找表T的大小要大，已经访问过而调入Cache的表元素不会被替换出Cache)。如果$<l_i>\neq<l_j>$，并且l_j先前并没有被调入Cache的话，那么就会发生一次Cache不命中。平均来说，前面的操作所花费的时间要比后面的操作少。

3.2 针对 AES 最后一轮的攻击

为了利用访问 Cache 时的索引不同而导致的时间差异对 AES 进行攻击，本文考虑加密的最后一轮。正如前面所说，AES 加密的最后一轮没有列混淆操作而引入了另一个表 $T4$，将式(1)转化为

$$\begin{cases}(C^0,C^1,C^2,C^3)=(T_4[S_{10}^0]\oplus K_{10}^0,T_4[S_{10}^5]\oplus K_{10}^1,T_4[S_{10}^{10}]\oplus K_{10}^2,T_4[S_{10}^{15}]\oplus K_{10}^3)\\(C^4,C^5,C^6,C^7)=(T_4[S_{10}^4]\oplus K_{10}^4,T_4[S_{10}^9]\oplus K_{10}^5,T_4[S_{10}^{14}]\oplus K_{10}^6,T_4[S_{10}^3]\oplus K_{10}^7)\\(C^8,C^9,C^{10},C^{11})=(T_4[S_{10}^8]\oplus K_{10}^8,T_4[S_{10}^{13}]\oplus K_{10}^9,T_4[S_{10}^2]\oplus K_{10}^{10},T_4[S_{10}^7]\oplus K_{10}^{11})\\(C^{12},C^{13},C^{14},C^{15})=(T_4[S_{10}^{12}]\oplus K_{10}^{12},T_4[S_{10}^1]\oplus K_{10}^{13},T_4[S_{10}^6]\oplus K_{10}^{14},T_4[S_{10}^{11}]\oplus K_{10}^{15})\end{cases}\tag{2}$$

在式(2)中，C是 16 B的密文输出，$T4$ 是AES最后一轮的查找表。对密文的任意两个字节C^i, C^j，设存在一个x使得 $C^i=T_4[S_{10}^x]\oplus K_{10}^i$，存在一个$y$使得 $C^j=T_4[S_{10}^y]\oplus K_{10}^j$。不考虑$x$和$y$的值，只要当 $S_{10}^x=S_{10}^y$，在查找$T4$ 时就会发生一次Cache命中。假设 $S_{10}^x=S_{10}^y$，则 $T_4[S_{10}^x]=T_4[S_{10}^x]=\alpha$，那么 $C^i=\alpha\oplus K_{10}^i$，$C^j=\alpha\oplus K_{10}^j$，也就是 $C^i\oplus C^j=K_{10}^i\oplus K_{10}^j$。反之，若是 $C^i\oplus C^j\neq K_{10}^i\oplus K_{10}^j$，则在查找$T4$ 表的时候就会得到两个不同的值α，β。那么就会有 $\gamma=\alpha\oplus\beta=C^i\oplus C^j\oplus K_{10}^i\oplus K_{10}^j$，因为 $K_{10}^i\oplus K_{10}^j$ 是由密钥决定的，所以对于固定的明文字节差值 $C^i\oplus C^j$ 来说，γ的值就是一个常数。

攻击的方法就是记录每次随机明文加密得到的密文C和对应的加密时间t。在获得足够的样本之后考查每个密文中所有的 $\gamma=C^i\oplus C^j$，并将其对应的时间值累加到一个三维的时间表$t[i,j,\gamma]$中。然后为每个i,j找到一个值γ'_{ij}使得$t[i,j,\gamma'_{ij}]$ 足够小，最后γ'_{ij}的值就是真正的 $\gamma_{ij}=K_{10}^i\oplus K_{10}^j$ 值的精确猜测，因为这个值会导致加密时间明显短。

当考虑 Cache 的行调度策略时，查找 $T4$ 时发生 Cache 命中的条件就不是严格的 $S_{10}^x=S_{10}^y$，而是 $<S_{10}^x>=<S_{10}^y>$，密文 C 和第 10 轮子密钥之间的关系就变成

$$<T4^{-1}[C^i\oplus K_{10}^i]>=<T4^{-1}[C^j\oplus K_{10}^j]>$$

其中，$T4^{-1}[\Delta]$ 表示由表$T4$ 的内容反推表索引值。在这种情况下，每次猜测 K_{10}^i 和 K_{10}^j 的对应的 $256\times256=2^{16}$种情况，并计算所有满足 $<T4^{-1}[C^i\oplus K_{10}^i]>=<T4^{-1}[C^j\oplus K_{10}^j]>$ 的平均时间，正确猜测会受Cache命中的影响，而导致对应的时间比较短。

在猜测获得 16 B 的第 10 轮扩展密钥之后，根据第 2 节介绍的原理，对每一个猜测的第 10 轮子密钥的 16 B 组，攻击程序反推原始密钥，并且用一组已知的明文/密文来检验获得的密钥。

4 试验结果及分析

根据前面攻击原理，编写针对AES的Cache Hit攻击算法，其中利用了根据CPU时间戳进行精确计时的指令RDTSC[4]对AES的加密过程进行精确计时，为了保证在每一次AES加密之前所有的查找表都不在Cache中，需要在每次AES加密之前进行一次Cache“清空”操作，可以首先用CPUID指令[4]查找得到Cache大小为S，然后在每次AES加密之前访问内存中大小为S的一块连续区域，将AES表元素都替换出Cache，同时CPUID指令还能获得Cache的行大小，用于计算$<l>$的值。算法的流程如图 1 所示。

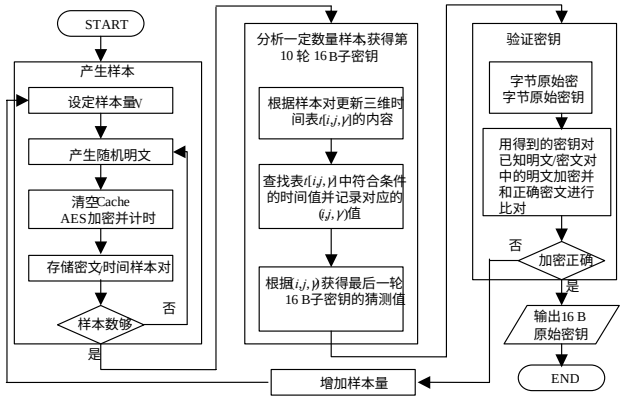

图 1 针对 AES 的 Cache Hit 攻击流程

本文用攻击程序对 OpenSSL v.0.9.8(a)库中的 AES 加密程序进行攻击，使用 RedHat v2.4.20-8 和 gcc v3.2.3，在 Intel Celeron 1.99 GHz，和 Pentium4 3.6 GHz 的 CPU 上分别进行了试验，结果如表 1 所示。 （下转第 129 页）