Cache攻击——AES密钥恢复

乔珂欣

2017.8.28

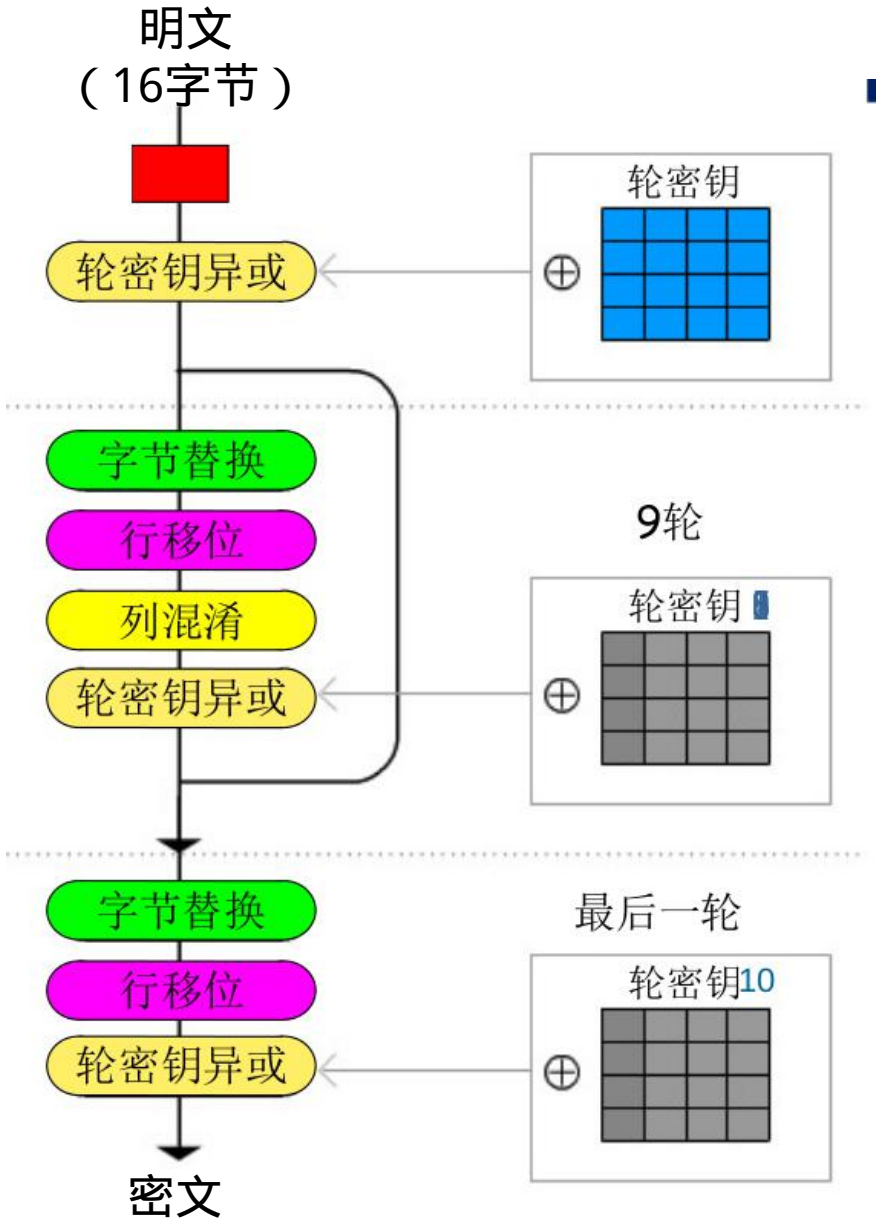
内容

AES算法介绍
- 算法描述
- 查找表实现

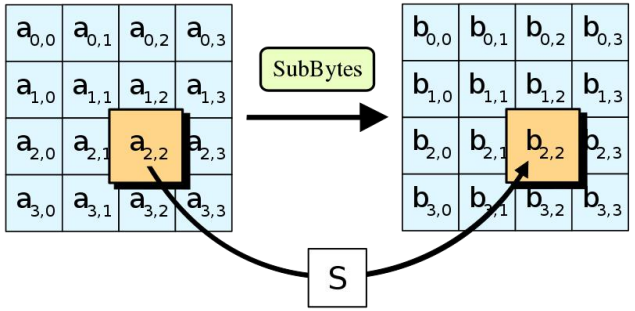
缓存攻击
- 基本原理
- AES 查找表的缓存机制

AES密钥恢复
- OpenSSL1.0.2l 中的 AES
- x86 和 arm 进行
- 演示

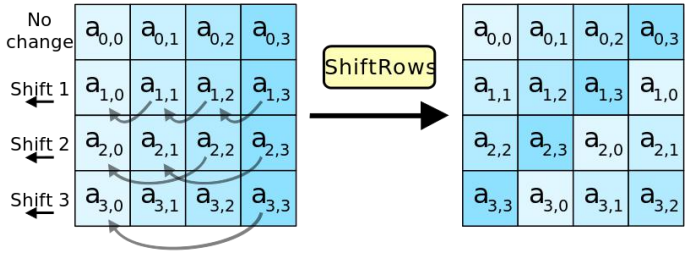
AES128



字节替换：8-bit S盒

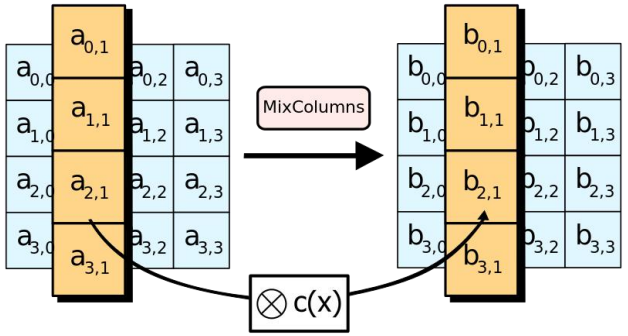


行移位：

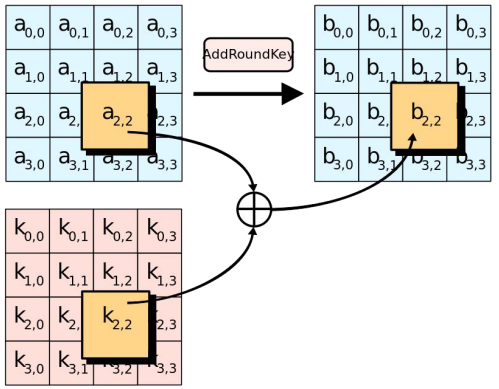


列混淆：每列左乘矩阵

$$\begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix}$$



轮密钥异或：



轮函数的查找表实现

利用代数形式定义一轮的 4 个变换

字节替换

$$b_{i,j} = S[a_{i,j}]$$

行移位

$$\begin{bmatrix} c_{0,j} \\ c_{1,j} \\ c_{2,j} \\ c_{3,j} \end{bmatrix} = \begin{bmatrix} b_{0,j} \\ b_{1,j+1} \\ b_{2,j+2} \\ b_{3,j+3} \end{bmatrix}$$

列混淆

$$\begin{bmatrix} d_{0,j} \\ d_{1,j} \\ d_{2,j} \\ d_{3,j} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} c_{0,j} \\ c_{1,j} \\ c_{2,j} \\ c_{3,j} \end{bmatrix}$$

轮密钥异或

$$\begin{bmatrix} e_{0,j} \\ e_{1,j} \\ e_{2,j} \\ e_{3,j} \end{bmatrix} = \begin{bmatrix} d_{0,j} \\ d_{1,j} \\ d_{2,j} \\ d_{3,j} \end{bmatrix} \oplus \begin{bmatrix} k_{0,j} \\ k_{1,j} \\ k_{2,j} \\ k_{3,j} \end{bmatrix}$$

把 4 个表达式表示成一个等式

$$\begin{bmatrix} e_{0,j} \\ e_{1,j} \\ e_{2,j} \\ e_{3,j} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} S[a_{0,j}] \\ S[a_{1,j+1}] \\ S[a_{2,j+2}] \\ S[a_{3,j+3}] \end{bmatrix} \oplus \begin{bmatrix} k_{0,j} \\ k_{1,j} \\ k_{2,j} \\ k_{3,j} \end{bmatrix}$$

$$= T_0\left(\begin{bmatrix} 02 \\ 01 \\ 01 \\ 03 \end{bmatrix} \cdot S[a_{0,j}]\right) \oplus T_1\left(\begin{bmatrix} 03 \\ 02 \\ 01 \\ 01 \end{bmatrix} \cdot S[a_{1,j+1}]\right)$$

$$\oplus T_2\left(\begin{bmatrix} 01 \\ 03 \\ 02 \\ 01 \end{bmatrix} \cdot S[a_{2,j+2}]\right) \oplus T_3\left(\begin{bmatrix} 01 \\ 01 \\ 03 \\ 02 \end{bmatrix} \cdot S[a_{3,j+3}]\right)$$

$$\oplus \begin{bmatrix} k_{0,j} \\ k_{1,j} \\ k_{2,j} \\ k_{3,j} \end{bmatrix}$$

T_i 输入：8-bit，输出：32-bit

存储为向量: 含 256 个元素，每个元素为 32-bit 字

AES 查找表实现

明文 $\mathbf{p} = (p_0, \cdots, p_{15})$

密钥 $\mathbf{k} = (k_0, \cdots, k_{16})$

轮密钥 $\mathbf{K}^{(r)}, r = 0, \cdots, 10$ $\mathbf{K}^{(r)} = (K_0^{(r)}, K_1^{(r)}, K_2^{(r)}, K_3^{(r)})$

明文与主密钥异或 $x_i^{(0)} = p_i \oplus k_i, i = 0, \cdots, 15$

以字（4字节，一列）为单位进行前 9 轮更新：r=0,...,8

$$
\begin{aligned}
(x_0^{(r+1)}, x_1^{(r+1)}, x_2^{(r+1)}, x_3^{(r+1)}) &\leftarrow T_0[x_0^{(r)}] \oplus T_1[x_5^{(r)}] \oplus T_2[x_{10}^{(r)}] \oplus T_3[x_{15}^{(r)}] \oplus K_0^{(r+1)} \\
(x_4^{(r+1)}, x_5^{(r+1)}, x_6^{(r+1)}, x_7^{(r+1)}) &\leftarrow T_0[x_4^{(r)}] \oplus T_1[x_9^{(r)}] \oplus T_2[x_{14}^{(r)}] \oplus T_3[x_3^{(r)}] \oplus K_1^{(r+1)} \\
(x_8^{(r+1)}, x_9^{(r+1)}, x_{10}^{(r+1)}, x_{11}^{(r+1)}) &\leftarrow T_0[x_8^{(r)}] \oplus T_1[x_{13}^{(r)}] \oplus T_2[x_2^{(r)}] \oplus T_3[x_7^{(r)}] \oplus K_2^{(r+1)} \\
(x_{12}^{(r+1)}, x_{13}^{(r+1)}, x_{14}^{(r+1)}, x_{15}^{(r+1)}) &\leftarrow T_0[x_{12}^{(r)}] \oplus T_1[x_1^{(r)}] \oplus T_2[x_6^{(r)}] \oplus T_3[x_{11}^{(r)}] \oplus K_3^{(r+1)}
\end{aligned} \tag{1}
$$

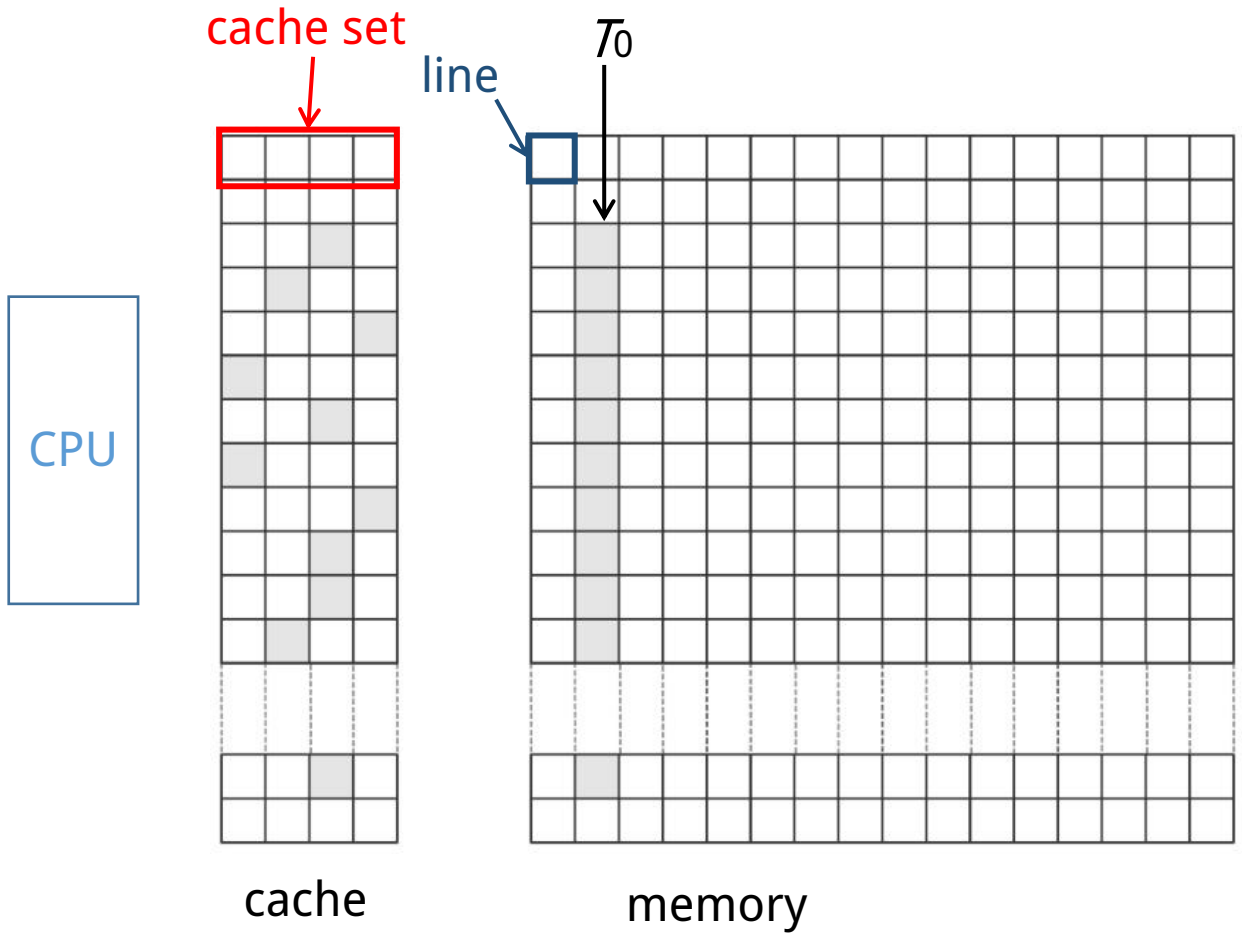
最后一轮：查相同的四个表，通过截断除去列混淆作用（后面讨论）

在一次 AES 加密中，每个查找表被查找了 40 次

缓存

缓存是位于 CPU 与内存之间的临时存储器

- 交换速度快
- 容量小



缓存结构

- 缓存空间分成若干集合 (set)，每个集合含固定数目的段
- 内存空间中的某些特定段竞争使用某个缓存集

对缓存执行的操作

- flush

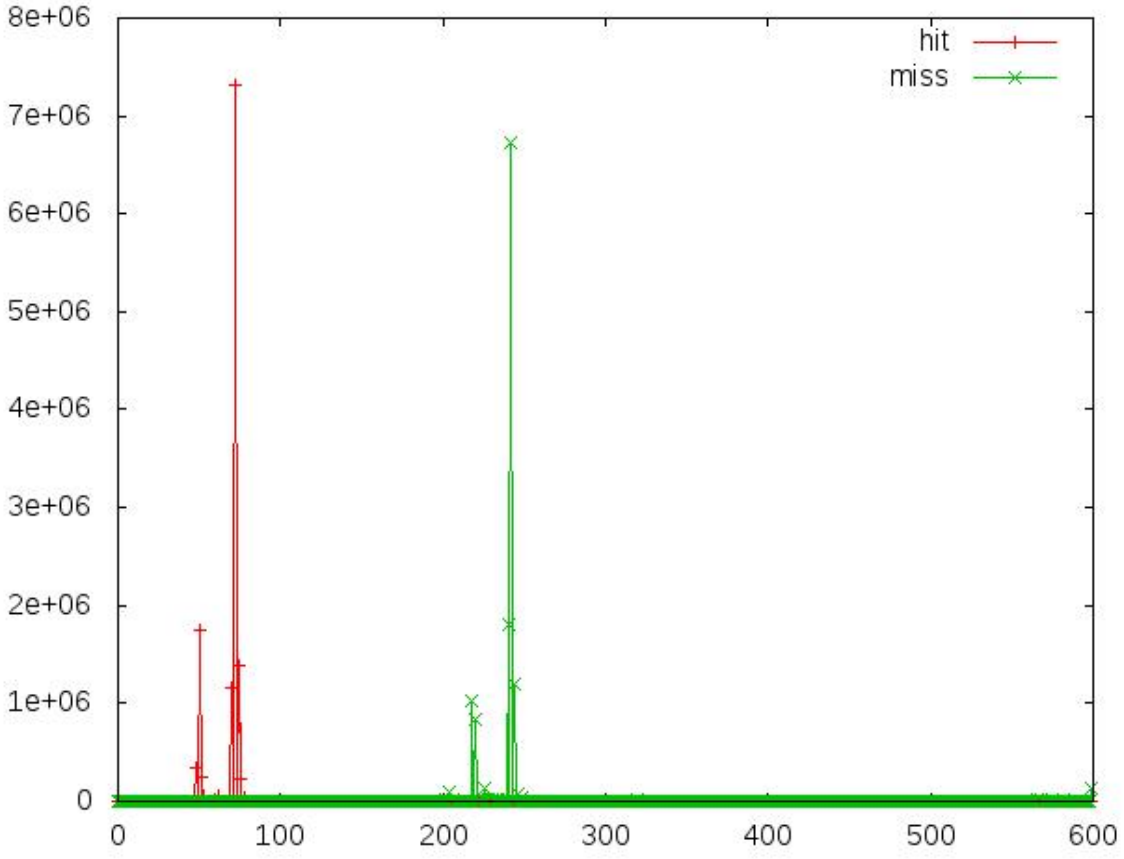
  将某个内存段的内容清除出缓存
- reload

  将某内存段重载入缓存

重载时间：

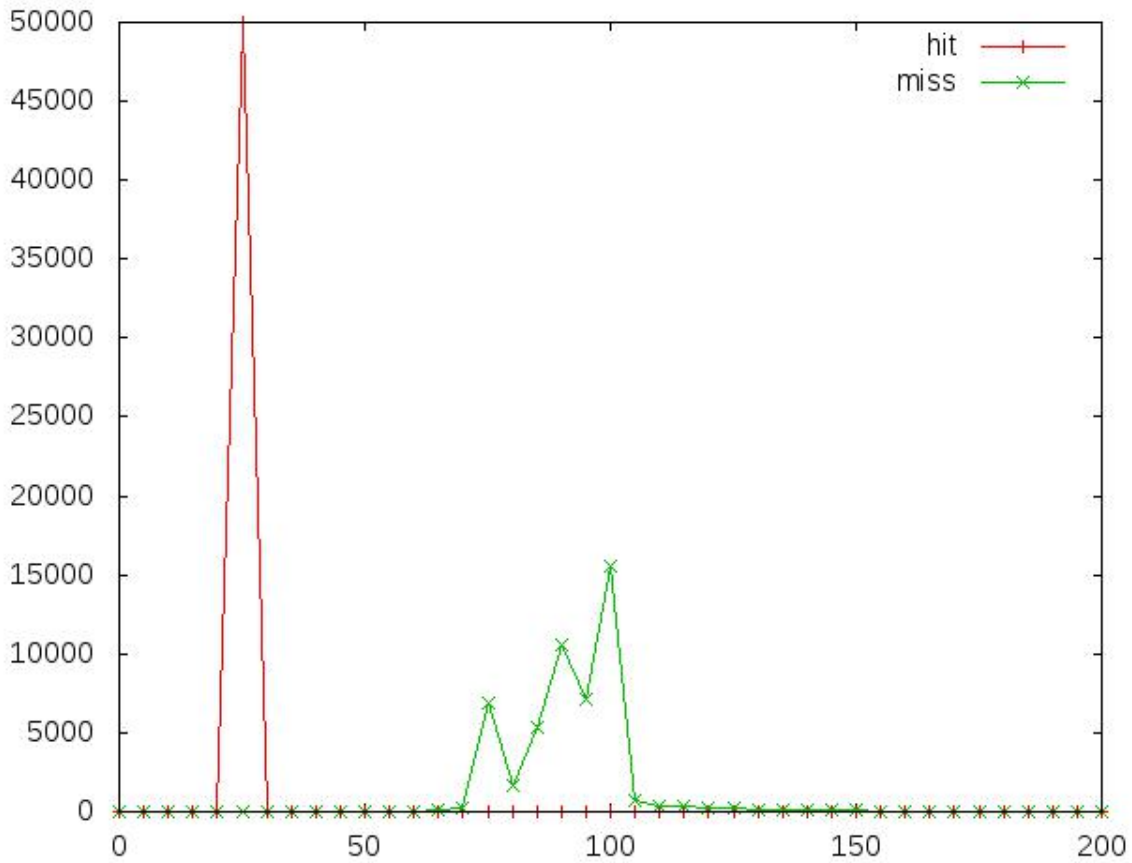
- 命中 (hit)：该内存段中的数据已经在缓存中，用时短；
- 未命中 (miss): 该内存段中的数据没在缓存中，则需从内存中载入，用时长

缓存命中（hit）与未命中（miss）的区分



CPU周期cycle

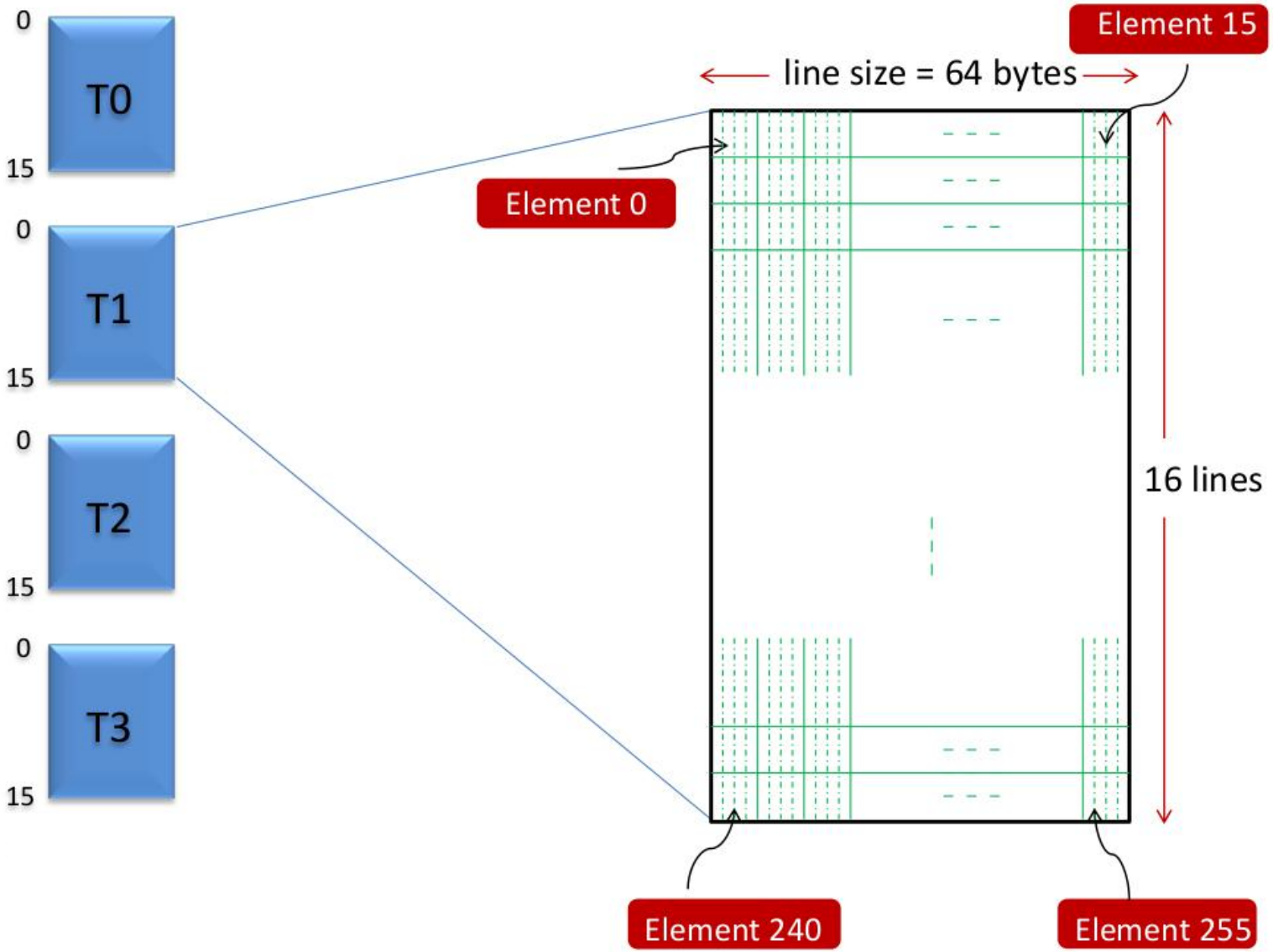
x86，THRESHOLD = 133



线程计数器counter

arm，THRESHOLD = 63

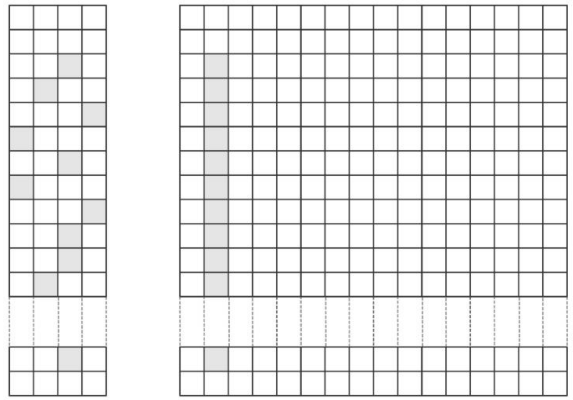
查找表的存储



每个查找表含 256 个元素，每个元素为一个 32-bit 字

- 存储空间以段（line）为单位
- 每段大小为 64 字节，可存储查找表的16个元素
- 每个查找表占用 16 个段，对应16个不同的缓存集合

- 当查找某个元素时，该元素所在的段被缓存



攻击准备

获取 hit 与 miss 的区分点 THRESHOLD

获取查找表在动态库中的偏移位置

- 在OpenSSL1.0.2l 编译产生的 libcrypto.so 动态库中，四个查找表的偏移位置分别为
  x86: 0x16BEC0 0x16BAC0 0x16B6C0 0x16B2C0
  arm: 0x15E800 0x15EC00 0x15F000 0x15F400
- 每个表的表段占据不同的缓存集，无重合
- 无需知道要存入缓存的位置

缓存攻击模式

监听方式采用 flush + reload，即

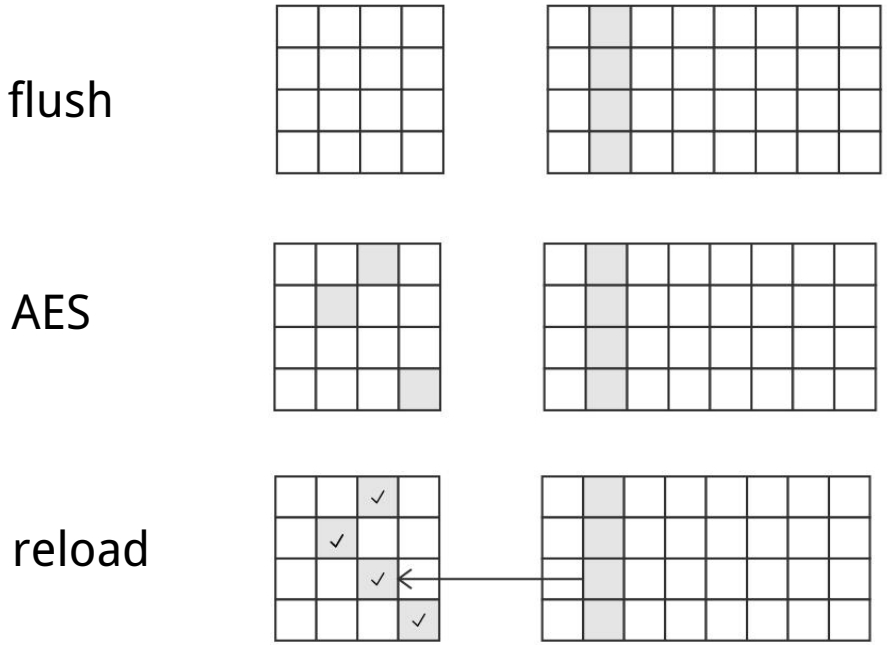
Step 1. flush: 查找表表段清除出缓存

Step 2. AES 加密。用到的表段被载入缓存

Step 3. reload：加载查找表表段，并获得加载时间

若 hit，该表段被使用过

若 miss，该表段未被使用过



x86

```
flush(probe);
AES_encrypt(plaintext, ciphertext, &key_struct);
time = rdtsc();
maccess(probe);
delta = rdtsc() - time;
```

arm

```
libflush_flush(libflush_session, address);
AES_encrypt(plaintext, ciphertext, &key_struct);
delta = libflush_reload_address(libflush_session, address);
```

攻击方法比较

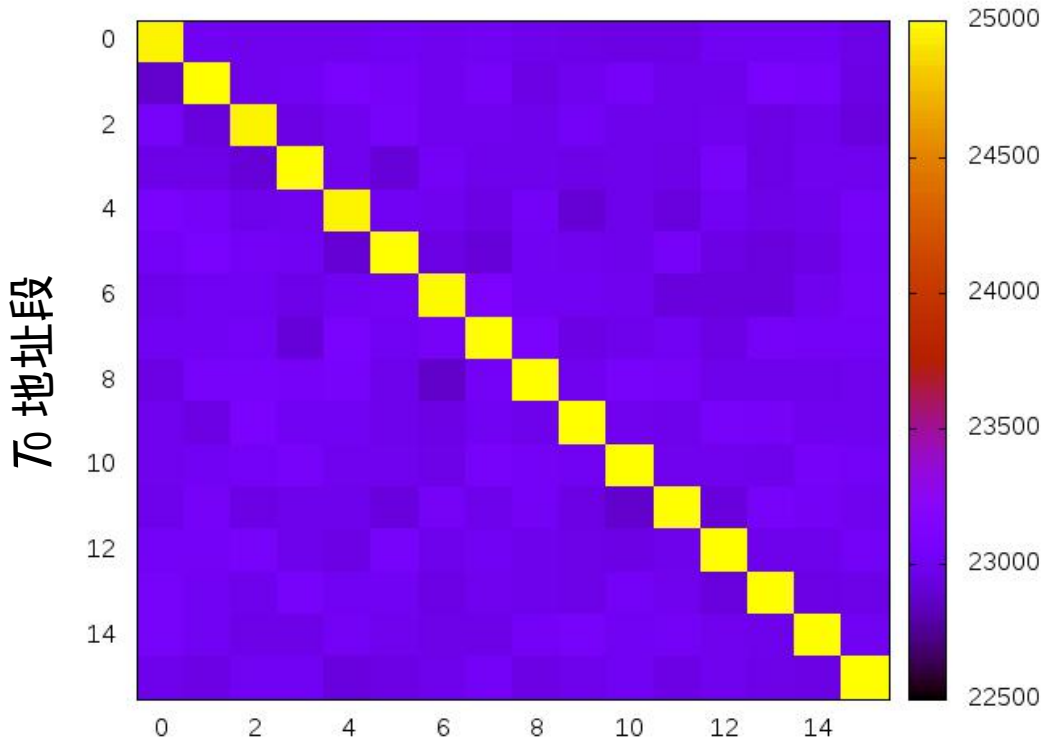
		方法一	方法二	方法三	方法四
利用轮数		第一、二轮	末轮	末轮	末轮
监听 T_i 地址段		任意一段	任意一段	所有 16 段	所有表的所有段
时间复杂度		2^{32} AES $= 2^{18} \cdot N$ AES	2^{26} AES $= 2^{12} \cdot N$ AES	2^{26} 运算 $= 2^{12} \cdot N$ 运算 $+2^4 \cdot N$ AES	2^{26} 运算 $= 2^{12} \cdot N$ 运算 $+N$ AES
		(样本量 $N = 2^{14} = 16384$)			
实际运行时间	x86	1 h	2-3 min	1.6 s	0.5 s
	arm	10 h	9-10 min	30 s	5 s

要点：

- 选取一个查表位置——某轮的某字节
- 密钥比特与该位置的查表元素相关——线性相关、非线性相关
- 利用缓存攻击监听查找表表段的使用情况，从对大量样本的观察中获取可区分特征

统计区分

查找表 T_0 各段 hit 频数统计图



每一个“方格”内进行了 25000 次加密，每一列明文样本满足首轮第一字节查表元素 $(p_0 \oplus k_0)$ 的高 4 比特值固定（在已知密钥下选取 p_0 ，其他明文字节随机）。

对 T0 表的每个地址段进行侦听，统计命中率：

- 地址段 ($(p_0 \oplus k_0)$ / 16)：首轮第一字节处已使用，故命中率为 100%
- 其他地址段：首轮第一字节处未使用，其他 39 处查找也未使用的概率为

$$(1 - \frac{16}{256})^{39} = 0.0807$$

故其他地址段的命中次数低 0.0807.

结论：利用缓存攻击监听查找表表段的使用情况，可以获得特定字节的信息

方法一：前两轮 + 任一表段

Step1: 利用首轮恢复每个密钥字节的高 4 比特

攻击过程

对第 b 个字节:

任选表 $T_{b \bmod 4}$ 的一个表段 L ;

猜测半密钥字节 $k_b[7\sim4]$ (=0,...,15):

构造 N 个首轮第 b 字节查表元素落在表段 L 的明文样本:

flush + AES + reload;

统计表段 L 在该猜测下的命中次数

命中次数最高的猜测为正确猜测

分析

- 密钥猜测正确：

实际的首轮第 b 字节查表元素的确落在 L 表段，L 表段命中率为1；

- 密钥猜测错误：

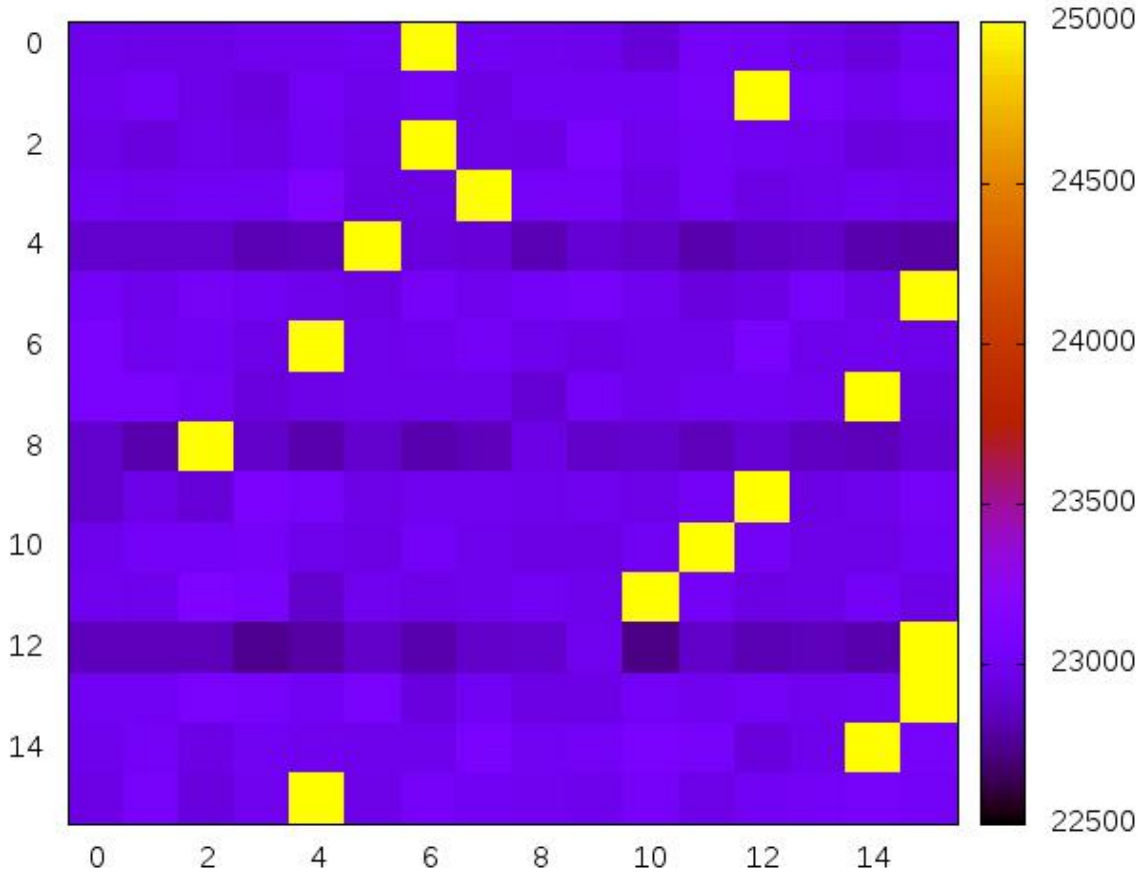
实际的首轮第 b 字节查表元素并未落在表段 L，L 表段命中率为

(1- 0.0807) ≈ 0.92

由于首轮查表元素与密钥是线性关系，只有高 4 比特对查找哪个表段有影响，故只能恢复高 4 比特

示例

字节 b



密钥高 4 比特

加密密钥：
67 c6 69 73 51 ff 4a ec 29 cd ba ab f2 fb e3 46

高 4 比特恢复：
6 c 6 7 5 f 4 e 2 c b a f f e 4

Step2: 利用第二轮恢复剩余比特

考虑第二轮如下四个查表元素

T_2 $$x_2^{(1)} = s(p_0 \oplus k_0) \oplus s(p_5 \oplus k_5) \oplus 2 \bullet s(p_{10} \oplus k_{10}) \oplus 3 \bullet s(p_{15} \oplus k_{15}) \oplus s(k_{15}) \oplus k_2 \quad (2)$$

T_1 $$x_5^{(1)} = s(p_4 \oplus k_4) \oplus 2 \bullet s(p_9 \oplus k_9) \oplus 3 \bullet s(p_{14} \oplus k_{14}) \oplus s(p_3 \oplus k_3) \oplus s(k_{14}) \oplus k_1 \oplus k_5$$

T_0 $$x_8^{(1)} = 2 \bullet (p_8 \oplus k_8) \oplus 3 \bullet s(p_{13} \oplus k_{13}) \oplus s(p_2 \oplus k_2) \oplus s(p_7 \oplus k_7) \oplus s(k_{13}) \oplus k_0 \oplus k_4 \oplus k_8 \oplus 1$$

T_3 $$x_{15}^{(1)} = 3 \bullet s(p_{12} \oplus k_{12}) \oplus s(p_1 \oplus k_1) \oplus s(p_6 \oplus k_6) \oplus 2 \bullet s(p_{11} \oplus k_{11}) \oplus s(k_{12}) \oplus k_{15} \oplus k_3 \oplus k_7 \oplus k_{11}$$

16 个密钥字节分为四组，以第一组为例，查表元素为 $x_2^{(1)}$ ，影响所查表段的密钥比特为 k_0, k_5, k_{10}, k_{15} 的未知比特，

攻击过程　　时间复杂度： $4 \times 2^{16} \times N$ AES

对一个密钥四元组:

任选所查查找表的一个表段 L;

猜测四个半密钥字节的组合 (=0,...,2^16-1):

构造 N 个第二轮查表元素落在表段 L 的明文样本:

flush + AES + reload;

统计表段 L 在该猜测下的命中次数

命中次数最高的猜测为正确猜测

分析

- 密钥猜测正确：

实际的第二轮查表元素的确落在 L 表段，L 表段命中率为1；

- 密钥猜测错误：

实际的第二轮查表元素并未落在表段 L，L 表段命中率为

(1-0.0807) ≈ 0.92

```
Key used for encryption
d7 92 52 7f 92 cc 9b 70 58 b2 99 74 ab c0 6a 34
```

```
Wait for Group 0 ...
0x7f927c0000 0x7f927c0040 key byte 0 = d7
key byte 5 = cc
key byte 10 = 99
key byte 15 = 34
Wait for Group 1 ...
0x7f927bfc00 0x7f927bfc40 key byte 3 = 7f
key byte 4 = 92
key byte 9 = b2
key byte 14 = 6a
Wait for Group 2 ...
0x7f927bf800 0x7f927bf840 key byte 2 = 52
key byte 7 = 70
key byte 8 = 58
key byte 13 = c0
Wait for Group 3 ...
0x7f927c0400 0x7f927c0440 key byte 1 = 92
key byte 6 = 9b
key byte 11 = 74
key byte 12 = ab
Full key recoverd
d7 92 52 7f 92 cc 9b 70 58 b2 99 74 ab c0 6a 34
finish

real    591m45.459s
user    847m12.140s
sys     336m15.980s
```

- 实际试验中选取了 2 个地址段进行监听，两次结果互相验证
- 每个地址段选取 10000 样本

缺点：用时太长！
x86: 1小时
arm: 10小时

方法二：最后一轮 + 任一表段

最后一轮查表元素

$$\begin{bmatrix} \mathcal{S}^{-1}(c_0 \oplus k_0^{10}) & \mathcal{S}^{-1}(c_4 \oplus k_4^{10}) & \mathcal{S}^{-1}(c_8 \oplus k_8^{10}) & \mathcal{S}^{-1}(c_{12} \oplus k_{12}^{10}) \\ \mathcal{S}^{-1}(c_{13} \oplus k_{13}^{10}) & \mathcal{S}^{-1}(c_1 \oplus k_1^{10}) & \mathcal{S}^{-1}(c_5 \oplus k_5^{10}) & \mathcal{S}^{-1}(c_9 \oplus k_9^{10}) \\ \mathcal{S}^{-1}(c_{10} \oplus k_{10}^{10}) & \mathcal{S}^{-1}(c_{14} \oplus k_{14}^{10}) & \mathcal{S}^{-1}(c_2 \oplus k_2^{10}) & \mathcal{S}^{-1}(c_6 \oplus k_6^{10}) \\ \mathcal{S}^{-1}(c_7 \oplus k_7^{10}) & \mathcal{S}^{-1}(c_{11} \oplus k_{11}^{10}) & \mathcal{S}^{-1}(c_{15} \oplus k_{15}^{10}) & \mathcal{S}^{-1}(c_3 \oplus k_3^{10}) \end{bmatrix}$$

使用的查找表

T_2	T_2	T_2	T_2
T_3	T_3	T_3	T_3
T_0	T_0	T_0	T_0
T_1	T_1	T_1	T_1

最后一轮轮密钥与查表元素是非线性关系，可恢复全部密钥比特

最后一轮加密实现代码

```
/*
 * apply last round and
 * map cipher state to byte array block:
 */
s0 =
    (Te2[(t0 >> 24)       ] & 0xff000000) ^
    (Te3[(t1 >> 16) & 0xff] & 0x00ff0000) ^
    (Te0[(t2 >>  8) & 0xff] & 0x0000ff00) ^
    (Te1[(t3      ) & 0xff] & 0x000000ff) ^
    rk[0];
PUTU32(out     , s0);
s1 =
    (Te2[(t1 >> 24)       ] & 0xff000000) ^
    (Te3[(t2 >> 16) & 0xff] & 0x00ff0000) ^
    (Te0[(t3 >>  8) & 0xff] & 0x0000ff00) ^
    (Te1[(t0      ) & 0xff] & 0x000000ff) ^
    rk[1];
PUTU32(out +  4, s1);
s2 =
    (Te2[(t2 >> 24)       ] & 0xff000000) ^
    (Te3[(t3 >> 16) & 0xff] & 0x00ff0000) ^
    (Te0[(t0 >>  8) & 0xff] & 0x0000ff00) ^
    (Te1[(t1      ) & 0xff] & 0x000000ff) ^
    rk[2];
PUTU32(out +  8, s2);
s3 =
    (Te2[(t3 >> 24)       ] & 0xff000000) ^
    (Te3[(t0 >> 16) & 0xff] & 0x00ff0000) ^
    (Te0[(t1 >>  8) & 0xff] & 0x0000ff00) ^
    (Te1[(t2      ) & 0xff] & 0x000000ff) ^
    rk[3];
PUTU32(out + 12, s3);
```

查表元素与明文“相距”太远，无法构造明文，故明文样本随机，根据密文和最后一轮猜测密钥计算查表元素值，将样本分为两类

H0：查表元素值落在监听表段
H1：查表元素值未落在监听表段

计算两类未命中率之差： $Diff\_ratio = \frac{H1_{miss}}{H1_{total}} - \frac{H0_{miss}}{H0_{total}}$

攻击过程

对第 b 个字节:
任选表 T(b mod 4 +2) mod 4 的一个表段 L ;
猜测最后一轮轮密钥字节 kb (=0,...,255):
对 N 个随机明文样本:
flush + AES + reload;
按查表元素值分为两类

计算统计量 Diff_ratio

Diff_ratio 最高的猜测为正确猜测

分析

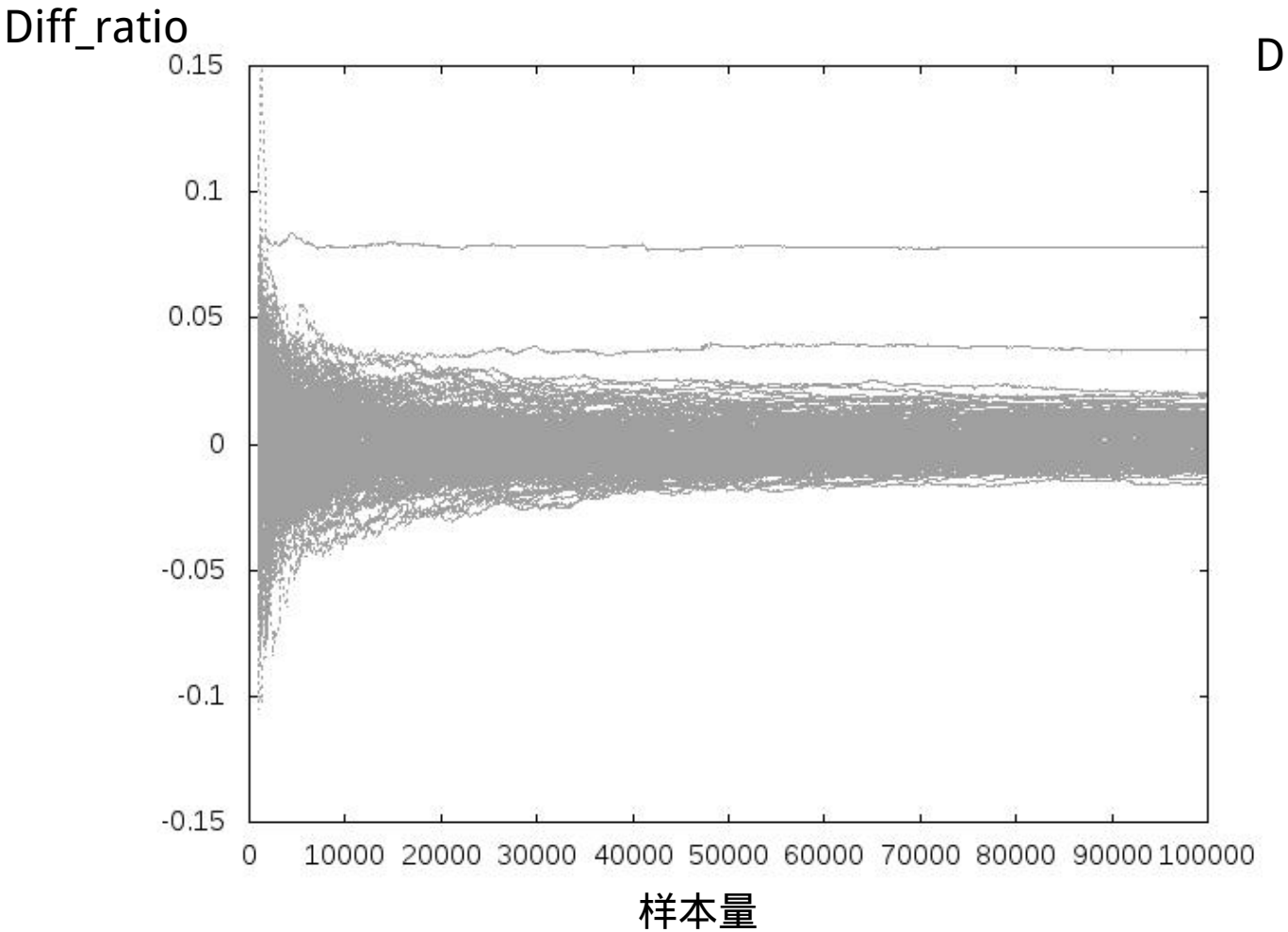
- 密钥猜测正确：

最后一轮查表元素计算值与真实值相等，对样本的分类符合真实情况，Diff_ratio 值为
0.0807；

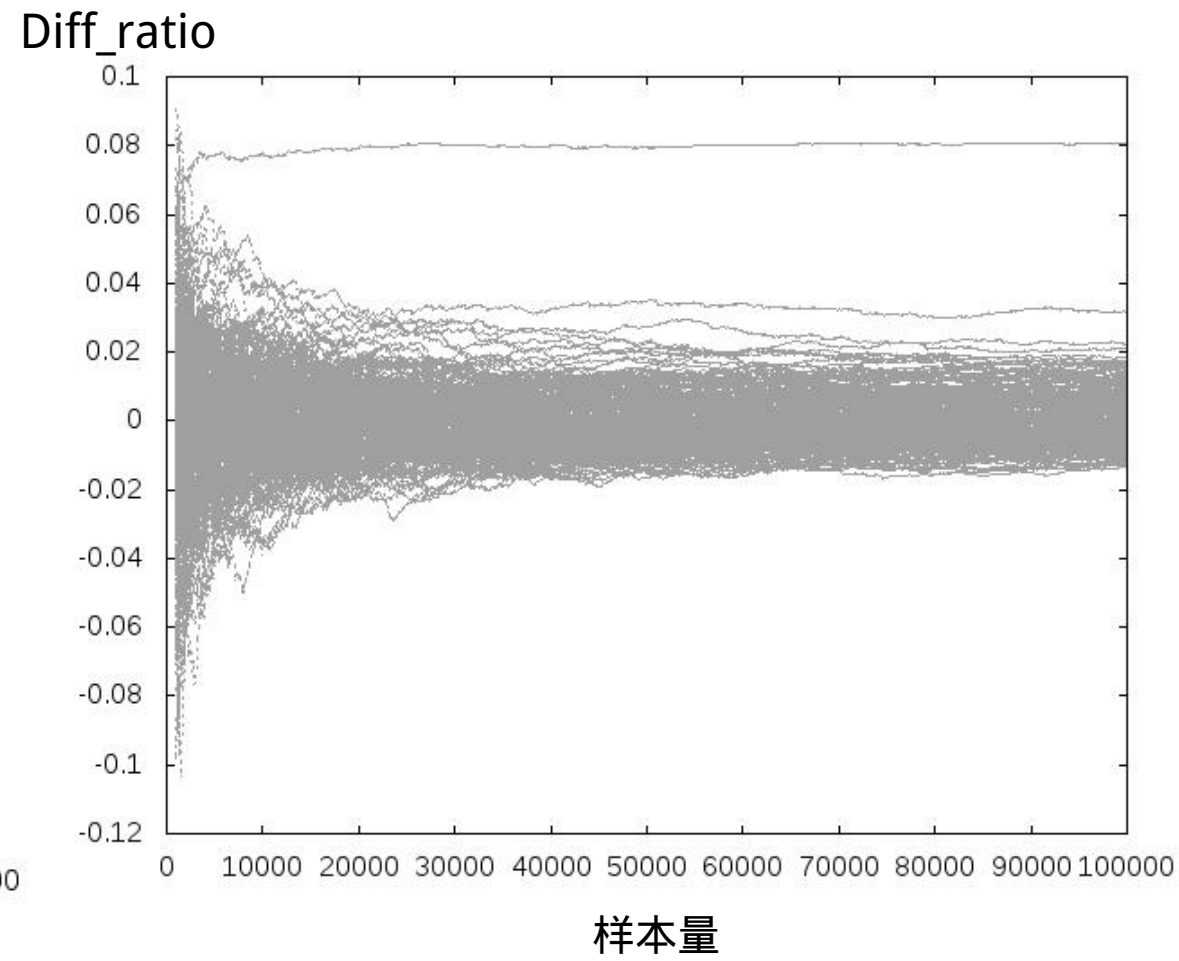
- 密钥猜测错误：

最后一轮查表元素计算值与真实值不等，对样本的分类随机，Diff_ratio 值为
0；

时间复杂度： $2^4 \times 2^8 \times N$ AES



arm 架构下不同密钥下的 miss 率之差



x86 架构下不同密钥下的 miss 率之差

方法三：最后一轮 + 所有表段

每次加密只监听一个表段浪费了其他表段的缓存信息，监听所有表段以利用：

每个被查找的表段推荐出 16 个密钥候选值

攻击过程

对第 b 个字节:
　对 N 个随机明文样本:
　　flush：清空表 $T_{(b \bmod 4 + 2) \bmod 4}$ 的所有 16 段;
　　AES;
　　reload 表 $T_{(b \bmod 4 + 2) \bmod 4}$ 的每个段：
　　若 hit，则对该段每个可能的查表元素 m：
　　密钥猜测 S(m)⊕ciphertext[b] 计数++；

推荐率最高的密钥字节为正确密钥

时间复杂度：

$$2^4 \times N \times (\mathrm{AES} + 2^4 \times 2^4)$$

分析

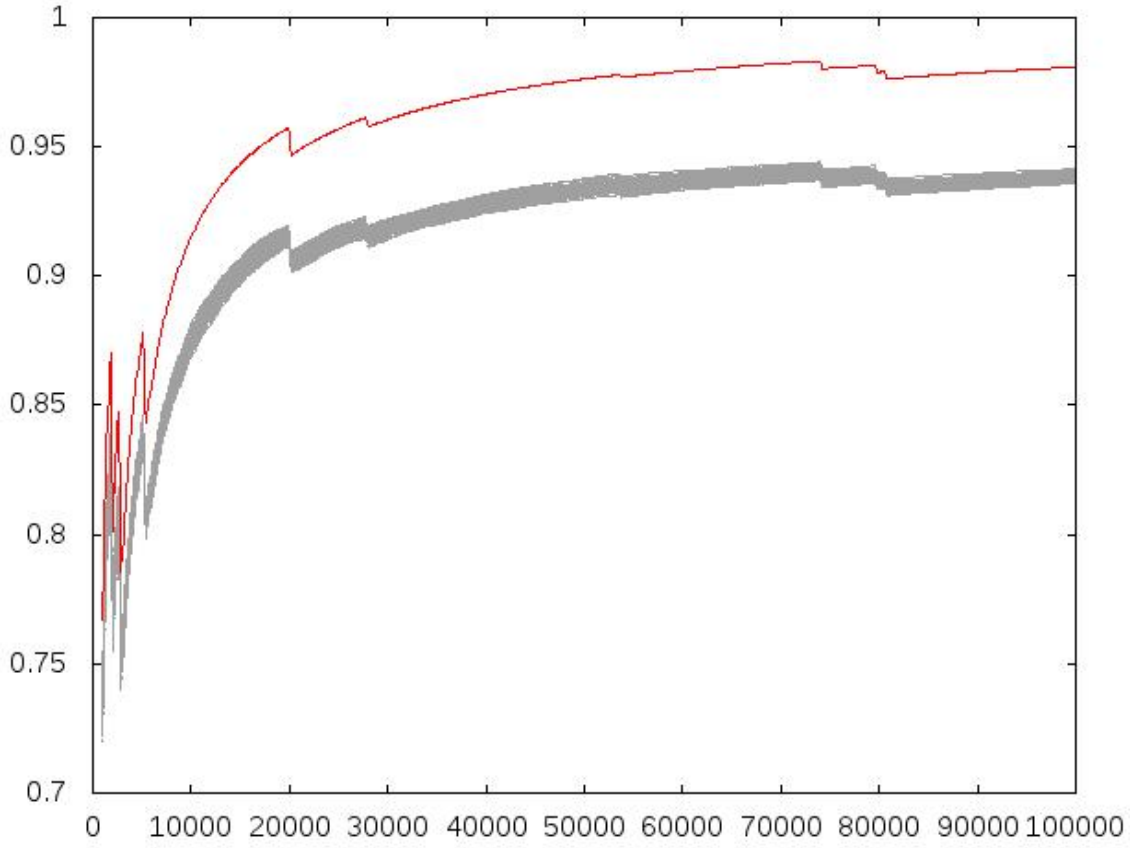
- 正确的密钥将被所有样本推荐，推荐率为 1；
- *一次加密平均用到几个表段？*

$$\begin{aligned}\bar{L} &= 16 \times (1 - (\frac{15}{16})^{40}) \\ &\approx 14.7895\end{aligned}$$

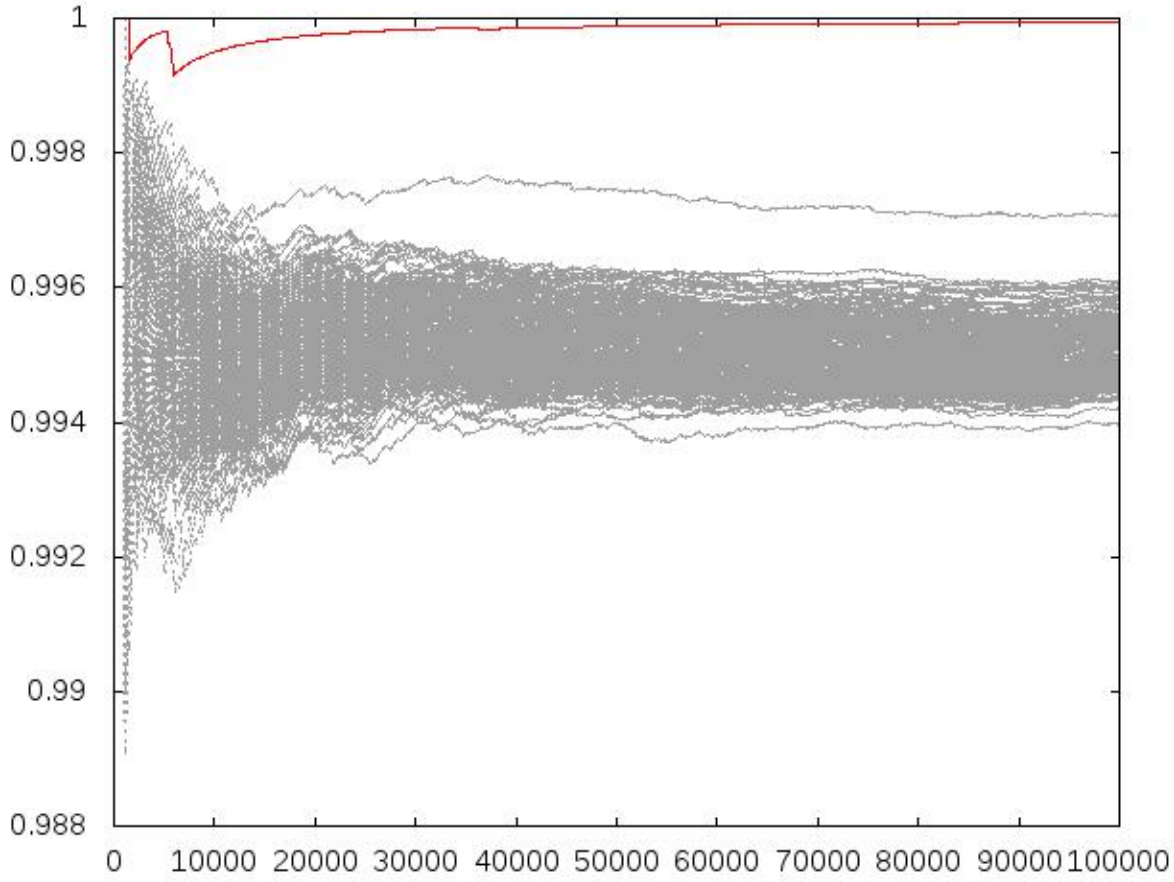
- 错误密钥的推荐率

$$\begin{aligned}&1 - Pr[\text{错误密钥未被推荐}] \\ =&1 - Pr[\text{错误密钥下的查表段未被使用}] \\ &\times (1 - Pr[\text{错误密钥下的查表段未被使用但仍被推荐}]) \\ =&1 - (\frac{15}{16})^{40} \times (1 - \frac{16}{256} \times \bar{L}) \\ =&1 - (\frac{15}{16})^{80} \\ \approx&0.9943\end{aligned}$$

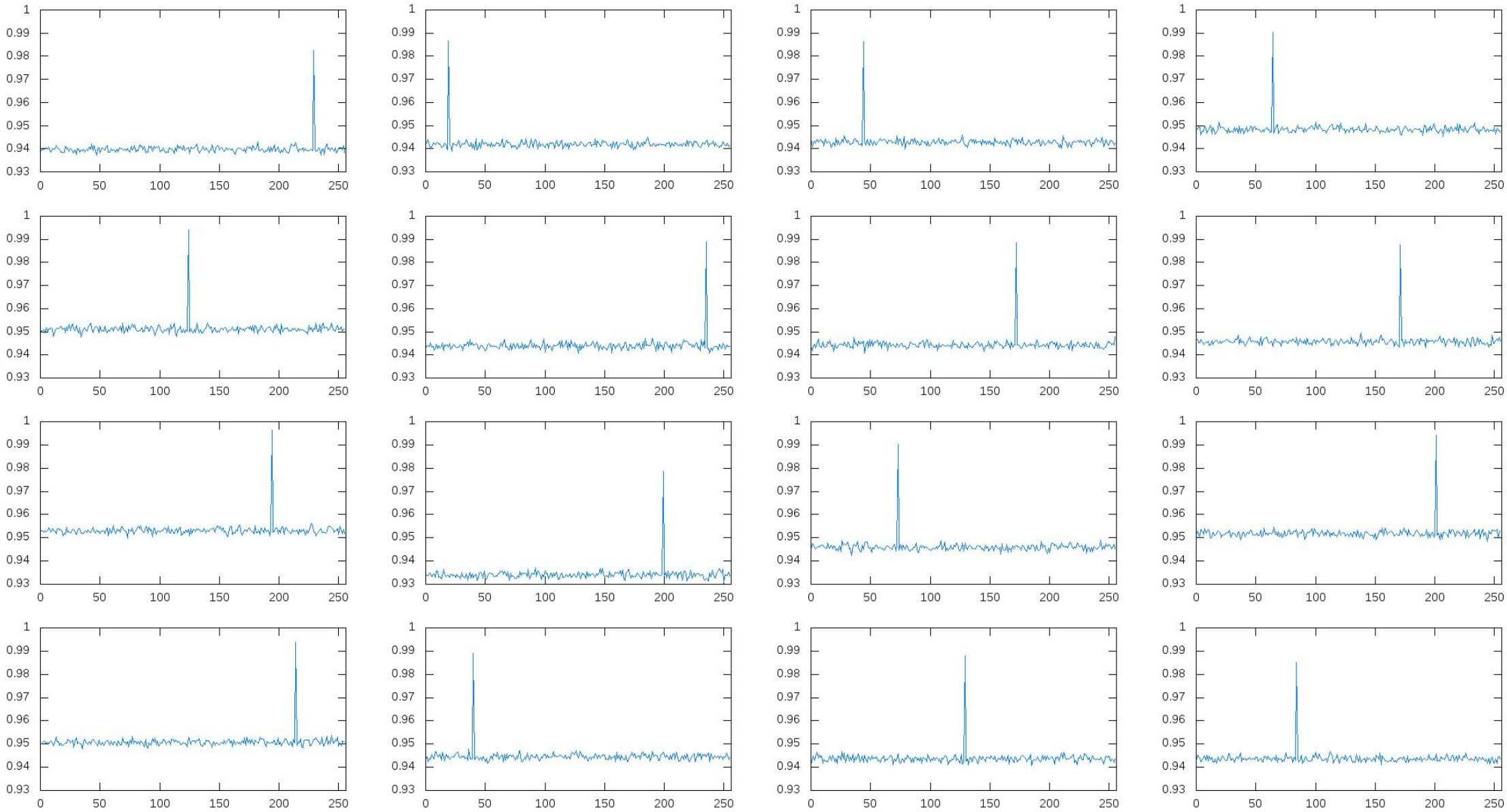
不同架构下的样本量与密钥推荐率关系



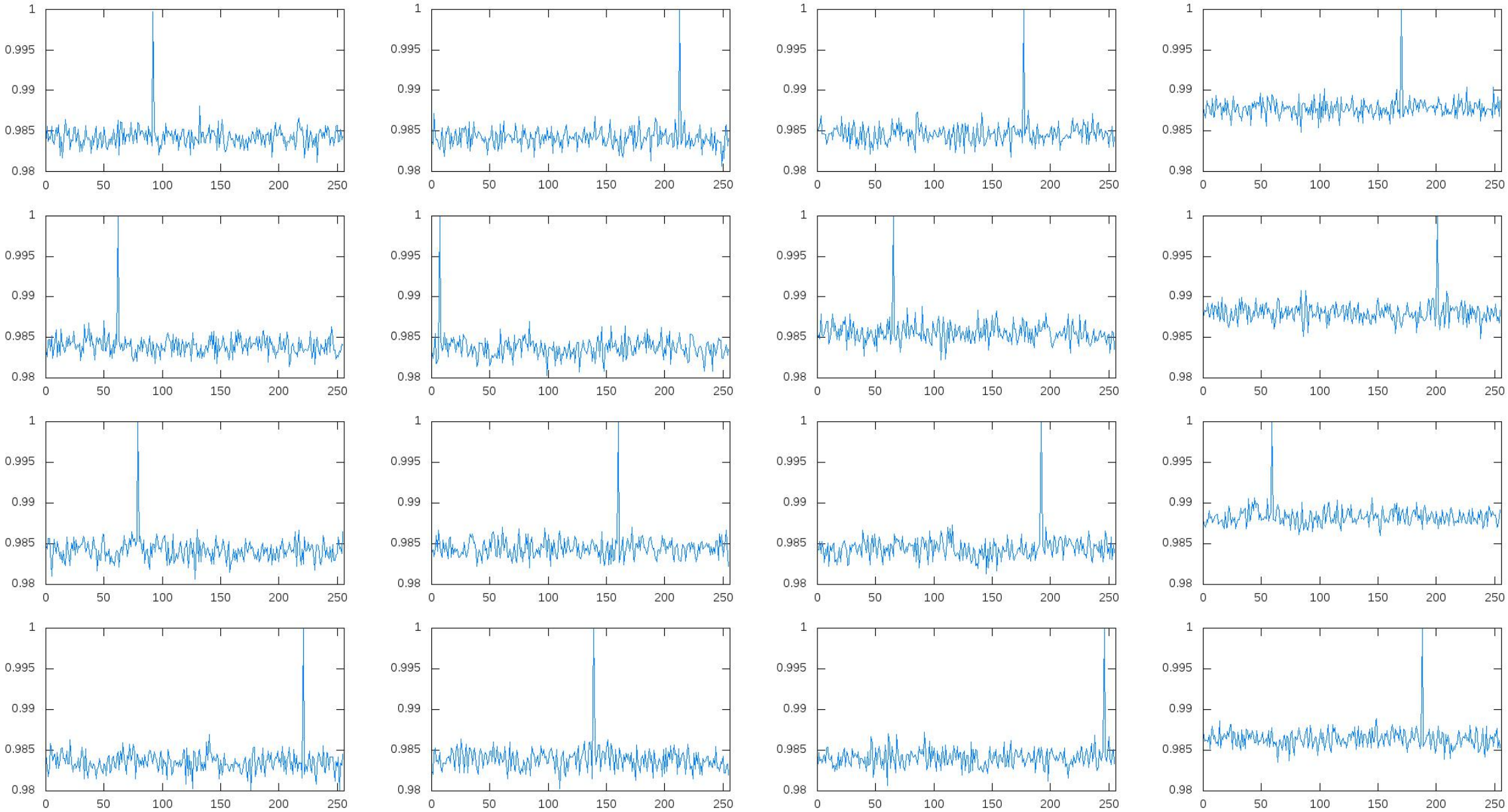
arm



x86



arm 架构下样本量取 N=50000 的推荐率



x86 架构下样本量取 N=16000 的推荐率

方法四：最后一轮 + 所有表、所有表段

注意到前述方法对每个字节单独处理，而实际上一个查找表在一轮中被四个字节查找，对一个查找表的监听可以同时给出四个密钥字节的信息，进一步降低复杂度

攻击过程

对 N 个随机明文样本:

flush：清空表 *T0, T1, T2, T3* 的所有段;

AES;

reload 表 *T0, T1, T2, T3* 的每个段：

若 hit，则对该段每个可能的查表元素 m：

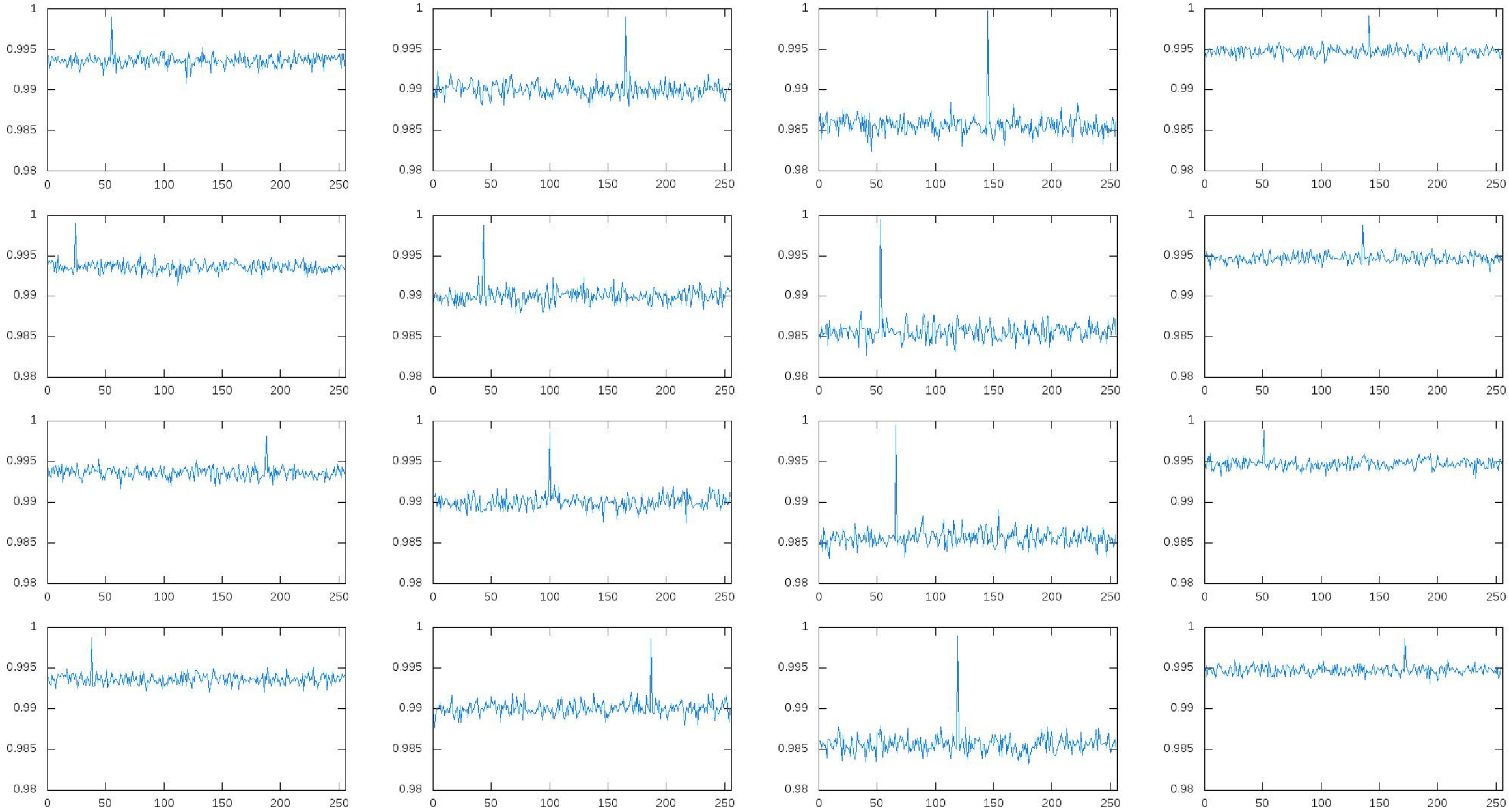
四个字节上的密钥猜测 S(m)⊕ciphertext[b] 计数++；

每个字节上推荐率最高的密钥猜测为正确密钥

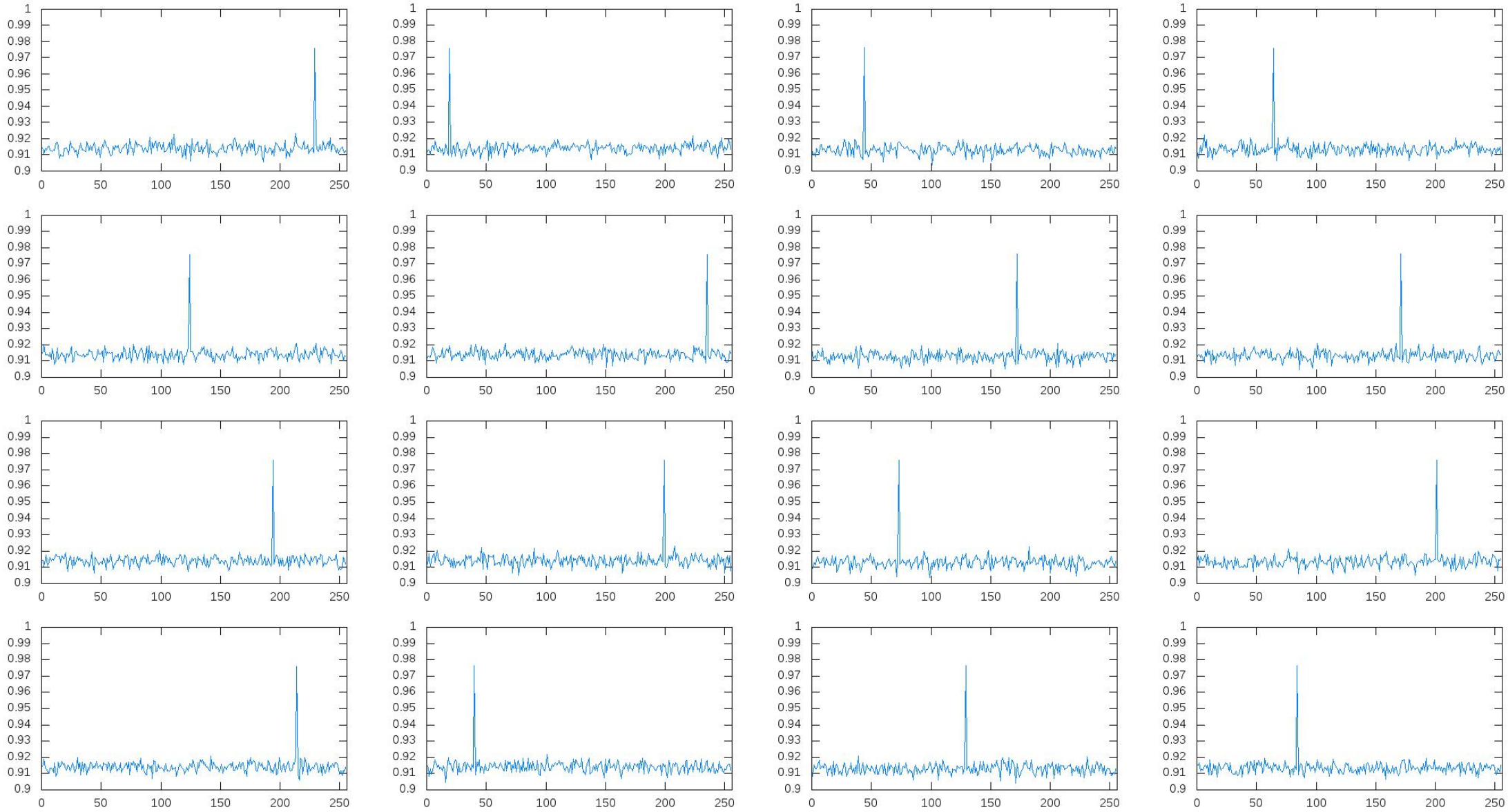
分析

- 正确的密钥推荐率为 1；
- 错误密钥的推荐率为 0.9943；

时间复杂度： $N \times (\mathrm{AES} + 2^2 \times 2^4 \times 2^2 \times 2^4)$



x86 架构下样本量取 N=16000 的推荐率（每列根据同一个表得出）



arm 架构下样本量取 N=50000 的推荐率（每列根据同一个表得出）

- 演示

./last_round_all_b

gnuplot key_byte.gnu

谢谢

参考文献：

[1] M. Lipp, D. Gruss, R. Spreitzer, C. Maurice and S. Mangard. ARMageddon: Cache Attacks on Mobile Devices. USENIX 2016

[2]D. Osvik, A. Shamir and E. Tromer. Cache Attacks and Countermeasures: The Case of AES. CT-RSA 2006

[3] E. Savas and C. Yilmaz. A generic Method for the Analysis of a Class of Cache Attacks: A Case Study for AES. Comput. J. 58, 10 (2015)

[4] G. Irazoqui, M. S. Inci, T. Eisenbarth, B. Sunar. Wait a Minute! A fast, Cross-VM Attack on AES. RAID 2014

[5] Ashokkumar C. , Ravi Prakash Giri , Bernard Menezes: Highly Efficient Algorithms for AES Key Retrieval in Cache Access Attacks. IEEE European Symposium on Security and Privacy. 2016