TANGRAM算法设计文档

本算法设计文档包含五个部分，它们是：

第一部分 TANGRAM算法描述

第二部分 TANGRAM设计原理

第三部分 TANGRAM的安全性分析

第四部分 TANGRAM的软硬件性能分析

第五部分 TANGRAM的优缺点声明

# 一．TANGRAM算法描述

## TANGRAM算法概述

TANGRAM，就是七巧板。TANGRAM算法的主要设计思想是采用比特切片方法来设计适合多个软硬件平台的系列分组密码，以灵活地适用于多种不同的应用场景。这就像七巧板一样，一套七巧板可以拼出多种不同的图形，这是TANGRAM分组密码得名的原因。

TANGRAM是一族迭代型分组密码算法，它包含三个成员，其分组长度/密钥长度分别为128/128、128/256和256/256比特，分别记作TANGRAM128/128、TANGRAM128/256和TANGRAM256/256。为方便起见，按照分组长度的不同，我们将分组长度为128和256比特的TANGRAM分组密码分别记作TANGRAM-128和TANGRAM-256。也就是说，TANGRAM-128包含TANGRAM128/128和TANGRAM128/256两个分组密码，TANGRAM-256指TANGRAM256/256。

TANGRAM的整体结构为SP网络（Substitution-Permutation network），128比特的加密状态用一个的矩形比特阵列描述，256比特的加密状态用一个的矩形比特阵列描述。

## 术语

### SP网络

SP网络是一种广泛使用的分组密码整体结构，AES、Serpent、Noekeon、PRESENT等分组密码都采用了SP网络。SP中的S是指Substitution（替换），P是指Permutation（置换，或线性变换）。SP网络的结构非常清晰，S一般被称为混淆层，主要起混淆的作用，通常由若干个并行的S盒组成；P一般被称为扩散层，主要起扩散的作用，通常是一个置换或线性变换。直观地看，先经过混淆层，再经过扩散层，就很靠近Shannon提出的混淆原则和扩散原则。TANGRAM的整体结构采用了SP网络。

### 比特切片方法

比特切片方法由Eli Biham于1997年提出[5]，目标是提高DES的软件实现性能。在90年代末期，此方法也被用来加速DES的密钥穷搜攻击。之后，比特切片方法被广泛应用于对称密码算法的设计和实现中，包括AES规划的候选算法Serpent、NESSIE项目的候选算法NOEKEON、ESTREAM计划的候选算法Trivium、SHA-3竞赛的候选算法Keccak (即SHA-3)和JH，等等。TANGRAM的主要设计思想就是采用比特切片方法来设计适合多个软硬件平台、满足本次密码算法设计竞赛要求的分组密码。

比特切片方法的基本思想是：密码算法的软件实现模拟硬件实现的过程，软件实现的一个逻辑指令对应多个硬件逻辑门操作。以TANGRAM-128为例，TANGRAM-128的S层由32个相同的的S盒并置组成，在软件实现时，如果采用查表实现，需要做32次查表操作。如果使用比特切片方法，首先将TANGRAM-128的S盒表示为基于比特的逻辑指令序列（即S盒的每一个输出比特都由4个输入比特按照若干逻辑指令计算得到）；考虑32个S盒的第() 个输出比特，它们可以通过相同的逻辑指令序列计算而得，这样，将32个S盒的相同位置的输入比特放在一个32位的机器字当中，通过一串作用在32位机器字上的逻辑运算指令，就可以一下子实现32个S盒了。在常见的台式机上，仅需要12个基本的逻辑指令就能完成TANGRAM-128的S层操作。可见，相比于32次查表操作，12次基本逻辑指令的操作要高效很多。

需要区分的是，Serpent、NOEKEON、RECTANGLE、TANGRAM的算法设计本身就基于比特切片方法，无论是对一个分块加密还是对多个分块并行加密，都可以采用比特切片方法来实现；相比之下，对于DES和PRESENT，它们不是基于比特切片方法设计的，采用比特切片实现必须要对多个分块并行加密，对一个分块的加密，无法去谈比特切片实现。而且，对DES和PRESENT进行比特切片实现时，还需要将明文转换成比特切片格式并且将密文转换成正常格式，这需要额外的1 ~ 2 cycles/byte。

除了可以提升密码算法的软件实现性能，比特切片方法的另一个优点是：相比于查表实现，比特切片实现可以抵抗缓存时间攻击，因此防护侧信道攻击的代价更低。

## 符号和缩略语

符号

 : 一个明文分组

 : 一个密文分组

 : 第轮的轮函数

 : 第轮的轮子密钥

 : 第轮的轮常数

 : 比特串和的串联

 : 左循环移动位

 : 右循环移动位

 : 将和按位模2加（或称异或）

 : 将和按位与

 : 将和按位或

 : 将按位取反

 : :二元有限域，即元素为

 : -维向量集合，向量元素在上

 : 和的内积

 : 的汉明重量

 : 集合的元素个数

缩略语

AES : 高级加密标准 (Advance Encryption Standard)

DES : 数据加密标准 (Data Encryption Standard)

DPA : 差分能量攻击 (Differential Power Analysis)

SP : 替换-置换 (Substitution-Permutation)

S层 : 替换层 (Substitution layer)，也称混淆层

P层 : 置换层 (Permutation layer)，也称扩散层

PE : 置换异或等价 (Permutation-then-XOR Equivalent)

SHA : 安全杂凑算法 (Secure Hash Algorithm)

NIST : 美国国家标准技术研究所 (National Institute for Standards and Technology)

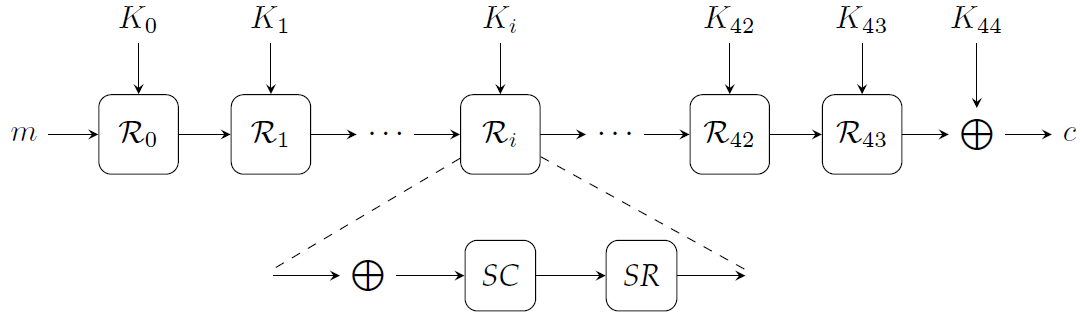
ECB : 电码本 (Electronic Code Book)

CBC : 密文块链接 (Cipher Block Chaining)

## TANGRAM算法结构

TANGRAM的整体结构是SP网络，TANGRAM128/128总轮数为44轮，TANGRAM128/256总轮数为50轮，TANGRAM256总轮数为82轮。在加密算法的最后再增加一个子密钥异或操作。

每一轮变换包含三个步骤：轮子密钥加 AddRoundKey（ARK），列替换 SubColumn（SC），行移位 ShiftRow（SR）。以TANGRAM128/128为例，用表示在轮子密钥作用下的轮变换，、分别表示明文和密文，TANGRAM 128/128的算法结构如图4.1所示。



1. TANGRAM 128/128的整体加密过程

## TANGRAM轮函数

### 轮函数结构

#### TANGRAM-128的轮函数结构

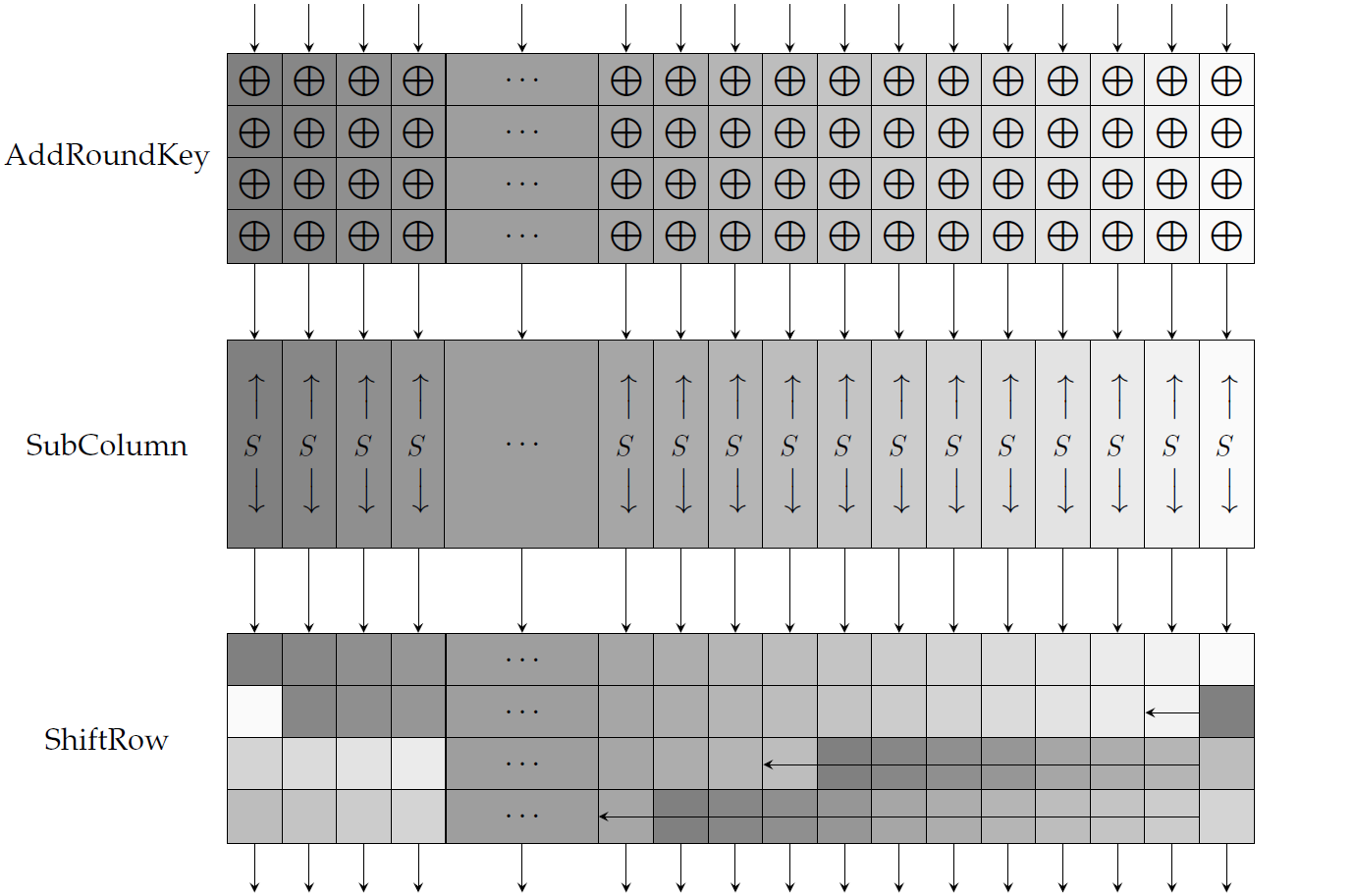
将128比特的明文、或128比特的中间状态、或128比特的密文统称为密码状态。令表示一个密码状态，我们用一个的矩形比特阵列表示，如图5.1 (a)所示，将前32个比特放在第0行，接下来的32个比特放在第1行，依次类推。类似地，128比特的轮子密钥也用一个的矩形比特阵列来表示。为了描述方便，以下用二维形式表示TANGRAM-128的密码状态，如图5.1 (b)所示。

(a)密码状态 (b)密码状态的二维表示

1. TANGRAM-128的密码状态

TANGRAM-128的轮函数由以下三个步骤组成：轮子密钥加AddRoundKey，列替换SubColumn，行移位 ShiftRow。图5.2图示了TANGRAM-128的轮函数结构。



1. TANGRAM-128的轮函数示意图

#### TANGRAM-256的轮函数结构

将256比特的明文、或256比特的中间状态、或256比特的密文统称为密码状态。令表示一个密码状态，我们用一个的矩形比特阵列表示，如图5.3 (a)所示，将前64个比特放在第0行，接下来的64个比特放在第1行，依次类推。类似地, 256比特的轮子密钥也用一个的矩形比特阵列来表示。为了描述方便，以下用二维形式表示TANGRAM-256的密码状态，如图5.2 (b)所示。

(a)密码状态 (b)密码状态的二维表示

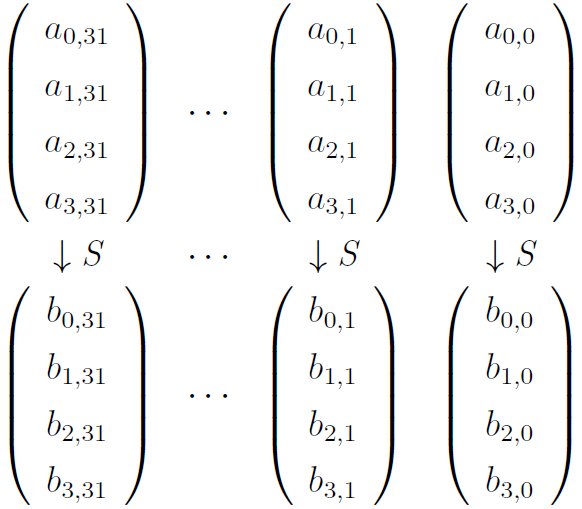
1. TANGRAM-256的密码状态

TANGRAM-256的轮函数由以下三个步骤组成：轮子密钥加AddRoundKey，列替换SubColumn，行移位 ShiftRow。

### 轮函数变换

**轮密钥加AddRoundKey** : 将128比特(或256比特)的轮子密钥逐比特异或到128比特(或256比特)的密码状态。

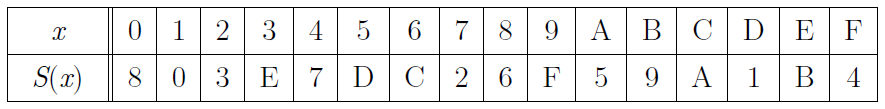
**列替换SubColumn**: 对每一列的4个比特做S盒替换。以TANGRAM-128为例，如图5.4所示，令表示S盒的输入，表示S盒的输出。



1. 列替换SubColumn

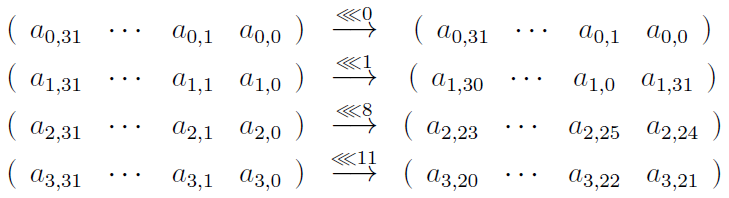
TANGRAM的S盒是一个4比特到4比特的双射: ，的描述如表5.1所示（十六进制）。

1. TANGRAM的S盒真值表



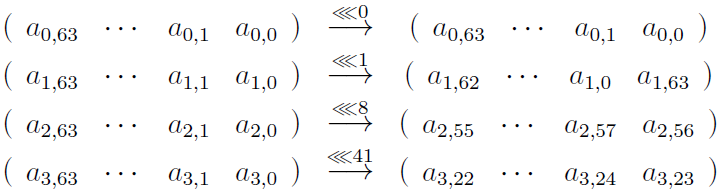
**行移位ShiftRow** : TANGRAM-128与TANGRAM-256采用不同的移位参数，这里分别用ShiftRow\_128和ShiftRow\_256来表示，以示区分。

**TANGRAM-128**对每一行的32个比特做左循环移位。第0行保持不动，第1行左循环移动1位，第2行左循环移动8位，第3行左循环移动11位。如图5.5所示，其中表示左循环移动位。



1. TANGRAM-128行移位ShiftRow\_128

**TANGRAM-256**对每一行的64个比特做左循环移位。第0行保持不动，第1行左循环移动1位，第2行左循环移动8位，第3行左循环移动41位。如图5.6所示，其中表示左循环移动位。



1. TANGRAM-256行移位ShiftRow\_256

## TANGRAM算法描述

### 6.1密钥扩展算法

#### TANGRAM 128/128密钥扩展算法

将128比特的种子密钥用一个的矩形比特阵列表示，如图6.1所示。

1. TANGRAM 128/128的128比特密钥状态及其二维表示

令表示第行，，可以看作一个32比特的字。在第轮，先提取128比特的子密钥，轮子密钥；然后，128比特的密钥状态做以下更新：

1. 对密钥状态进行S盒操作（与加密算法中的S盒相同），即：

，

1. 进行一轮的4分支广义Feistel变换，如下：



1. 对密钥状态的第一行的6个比特，异或6比特的轮常数。

将上面三个步骤的复合变换记为，其中为128比特。在第43轮的密钥状态更新之后，将更新的密钥状态的值赋给。

轮常数见表6.1。

#### TANGRAM 128/256密钥扩展算法

将256比特的种子密钥分成左右两半部分，，其中和都是128比特。利用Feistel网络，TANGRAM128/256所需的51个128比特的子密钥如下生成：

，

，

。

轮常数由一个6比特的线性反馈移位寄存器生成。用表示反馈寄存器的状态，在每一次更新时，将状态左移动1位，并将更新为，初始值。表6.1列出了TANGRAM 128/128和TANGRAM 128/256所用到的49个轮常数。

1. TANGRAM 128/128和TANGRAM 128/256的密钥扩展算法中使用的轮常数

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| i | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
| RC[i] | 01 | 02 | 04 | 08 | 10 | 21 | 03 | 06 | 0C | 18 | 31 | 22 | 05 |
| i | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 |
| RC[i] | 0A | 14 | 29 | 13 | 27 | 0F | 1E | 3D | 3A | 34 | 28 | 11 | 23 |
| i | 26 | 27 | 28 | 29 | 30 | 31 | 32 | 33 | 34 | 35 | 36 | 37 | 38 |
| RC[i] | 07 | 0E | 1C | 39 | 32 | 24 | 09 | 12 | 25 | 0B | 16 | 2D | 1B |
| i | 39 | 40 | 41 | 42 | 43 | 44 | 45 | 46 | 47 | 48 |  |  |  |
| RC[i] | 37 | 2E | 1D | 3B | 36 | 2C | 19 | 33 | 26 | 0D |  |  |  |

#### TANGRAM 256/256密钥扩展算法

将256比特的种子密钥用一个的矩形比特阵列表示，如图6.2所示。

1. TANGRAM256/256的256比特密钥状态及其二维表示

令表示第行，，可以看作一个64比特的字。在第轮，256比特的轮子密钥。提取出轮子密钥之后，密钥状态做以下更新：

1. 对密钥状态进行S盒操作（与加密算法中的S盒相同），即：

，

1. 进行一轮的4分支广义Feistel变换，如下：



1. 对密钥状态的第一行的7个比特，异或7比特的轮常数。

最后，将更新的密钥状态的值赋给。

轮常数由一个7比特的线性反馈移位寄存器生成。用表示反馈寄存器的状态，在每一次更新时，将状态左移动1位，并将更新为，初始值为。表6.2列出了所用到的82个轮常数。

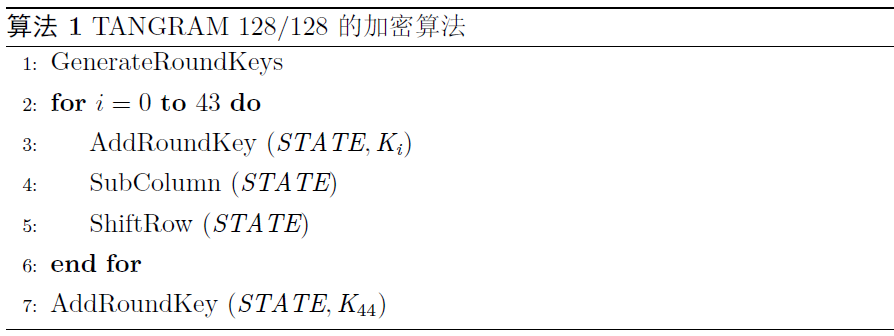
1. TANGRAM 256/256的密钥扩展算法中使用的轮常数

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| i | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
| RC[i] | 01 | 02 | 04 | 08 | 10 | 20 | 41 | 03 | 06 | 0C | 18 | 30 | 61 |
| i | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 |
| RC[i] | 42 | 05 | 0A | 14 | 28 | 51 | 23 | 47 | 0F | 1E | 3C | 79 | 72 |
| i | 26 | 27 | 28 | 29 | 30 | 31 | 32 | 33 | 34 | 35 | 36 | 37 | 38 |
| RC[i] | 64 | 48 | 11 | 22 | 45 | 0B | 16 | 2C | 59 | 33 | 67 | 4E | 1D |
| i | 39 | 40 | 41 | 42 | 43 | 44 | 45 | 46 | 47 | 48 | 49 | 50 | 51 |
| RC[i] | 3A | 75 | 6A | 54 | 29 | 53 | 27 | 4F | 1F | 3E | 7D | 7A | 74 |
| i | 52 | 53 | 54 | 55 | 56 | 57 | 58 | 59 | 60 | 61 | 62 | 63 | 64 |
| RC[i] | 68 | 50 | 21 | 43 | 07 | 0E | 1C | 38 | 71 | 62 | 44 | 9 | 12 |
| i | 65 | 66 | 67 | 68 | 69 | 70 | 71 | 72 | 73 | 74 | 75 | 76 | 77 |
| RC[i] | 24 | 49 | 13 | 26 | 4D | 1B | 36 | 6D | 5A | 35 | 6B | 56 | 2D |
| i | 78 | 79 | 80 | 81 |  |  |  |  |  |  |  |  |  |
| RC[i] | 5B | 37 | 6F | 5E |  |  |  |  |  |  |  |  |  |

### 加密算法

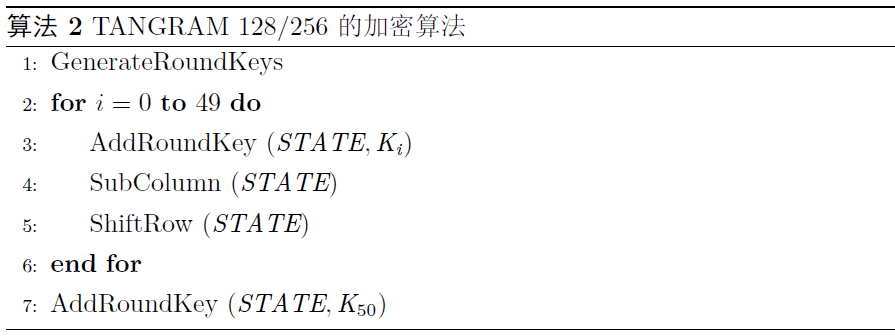
#### 6.2.1 TANGRAM128/128加密算法

TANGRAM128/128的加密算法将轮函数在轮子密钥的作用下迭代44轮，最后再增加一个轮子密钥加，如算法1所示：



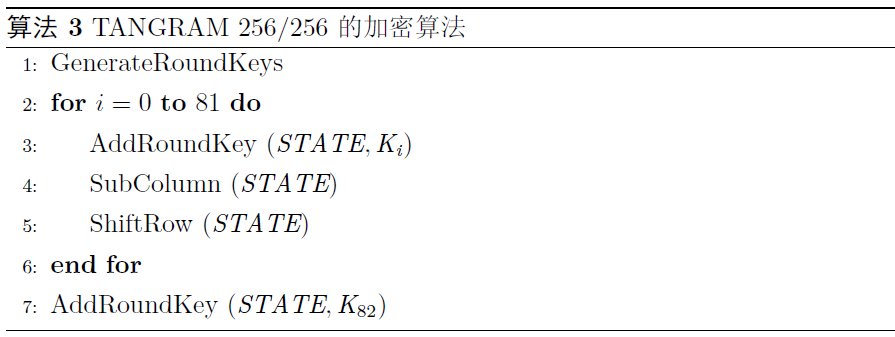
#### 6.2.2 TANGRAM128/256加密算法

TANGRAM128/256的加密算法将轮函数在轮子密钥的作用下迭代50轮，最后再增加一个轮子密钥加，如算法2所示：



#### 6.2.3 TANGRAM 256/256加密算法

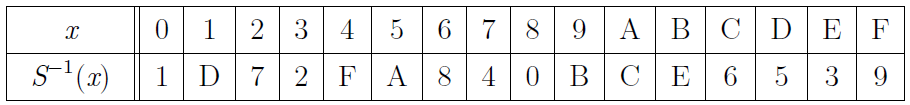
TANGRAM 256/256的加密算法将轮函数在轮子密钥的作用下迭代82轮，最后再增加一个轮子密钥加，如算法3所示：



### 解密算法

**列替换的逆变换InverseSubColumn**: 对密码状态每一列的4个比特做逆S盒替换。TANGRAM逆S盒的描述如表6.3所示。

表6.3 TANGRAM的S盒真值表



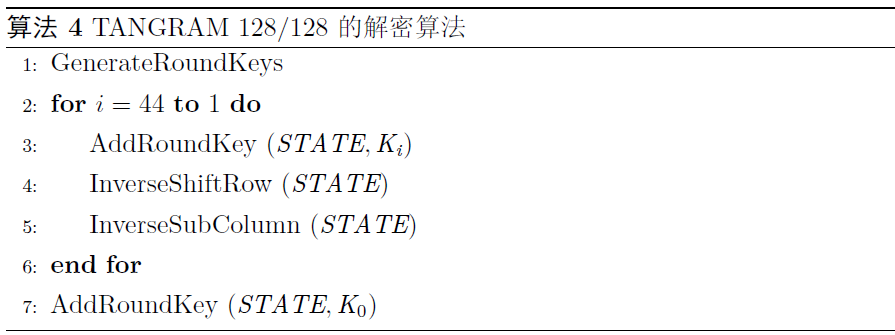
**行移位的逆变换InverseShiftRow**:

**TANGRAM-128**对密码状态每一行的32个比特做右循环移位。第0行保持不动，第1行右循环移动1位，第2行右循环移动8位，第3行右循环移动11位。

**TANGRAM-256**对密码状态每一行的64个比特做右循环移位。第0行保持不动，第1行右循环移动1位，第2行右循环移动8位，第3行右循环移动41位。

#### 6.3.1 TANGRAM128/128解密算法

TANGRAM128/128的解密算法如算法4所示：



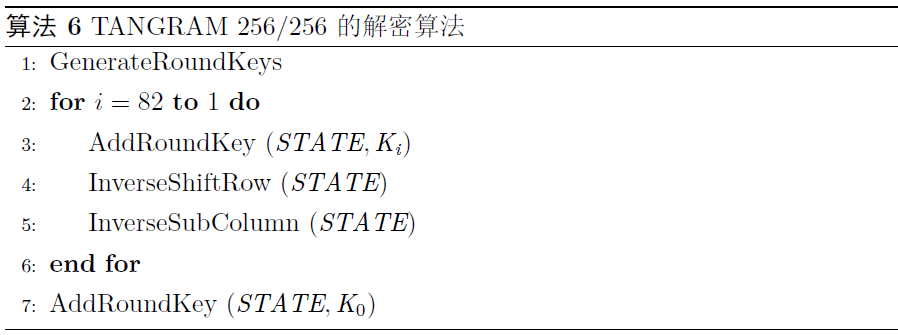
#### 6.3.2 TANGRAM 128/256解密算法

TANGRAM 128/256的解密算法如算法5所示：



#### 6.3.3 TANGRAM 256/256解密算法

TANGRAM 256/256的解密算法如算法6所示：



## TANGRAM算法的第二种描述

下面给出TANGRAM加密轮函数和解密轮函数的第二种描述，该描述对应TANGRAM的比特切片实现。TANGRAM-128的加/解密算法以32比特字为单位，TANGRAM-256的加/解密算法以64比特字为单位进行操作。

### 符号规定

为方便描述，我们对TANGRAM-128和TANGRAM-256的密码状态进行统一的定义，在TANGRAM-128和TANGRAM-256中，这些符号分别代表不同长度的比特子块。

**TANGRAM-128**对于图5.1 (b)所示的一个密码状态，将它表示为4个32比特子块的连接，即，其中，是的第行，，。

**TANGRAM-256**对于图5.2 (b)所示的一个密码状态，将它表示为4个64比特子块的连接，即，其中，是的第行，，。

### 加密轮函数

**轮密钥加AddRoundKey**: 设轮函数的输入为，轮子密钥为，两者异或后的结果记为，则有：



**列替换SubColumn**:用和分别表示SubColumn的输入和输出，则有：



其中，"，，，"分别表示按位与、或、非和异或。是32比特字(对应TANGRAM-128)或64比特字(对应TANGRAM-256)的临时变量，。

**行移位ShiftRow** : 用和分别表示ShiftRow的输入和输出。

TANGRAM-128：



**TANGRAM-256**



其中，表示对32比特字(对应TANGRAM-128)或64比特字(对应TANGRAM-256)左循环移动位。

### 解密轮函数

**轮密钥加AddRoundKey** :与加密轮函数中的AddRoundKey相同。

**列替换的逆变换InverseSubColumn**:用和分别表示InverseSubColumn的输入和输出，则有：



其中，是32比特字(对应TANGRAM-128)或64比特字(对应TANGRAM-256)的临时变量，。

**行移位的逆变换InverseShiftRow** : 用和分别表示InverseShiftRow的输入和输出。

TANGRAM-128：



TANGRAM-256：



其中，表示对32比特字(对应TANGRAM-128)或64比特字(对应TANGRAM-256)右循环移动位。

# 二．TANGRAM设计原理

## TANGRAM的整体设计原则

安全性: 密码算法的第一要素是安全性。对于一个分组长度为、密钥长度为的分组密码，如果不存在一种攻击方法有比少得多的数据复杂度并且比少得多的时间复杂度，就称此分组密码是安全的。通常用一个分组密码的安全冗余来评价其安全强度。设分组密码的总轮数为，如果最多可以攻击到轮的缩减轮密码，就称该分组密码具有轮的绝对安全冗余，或者具有的相对安全冗余。安全冗余反映了一个分组密码抵抗现有的安全性分析方法的安全强度。

实现性能: 密码算法的第二要素是实现性能，即在特定平台上进行加密运算或者解密运算所需要的资源。在硬件实现中，通常用实现面积和吞吐量来衡量；在软件实现中，通常用加密速度、代码量和内存占用来衡量。设计一个密码算法的最大挑战就是：保证足够安全性的同时，尽可能具有最优的实现性能。处理器字长和指令集的差异，会让一个密码算法的性能在一些平台上表现优秀、而在另外一些平台上表现很差。考虑到一个分组密码具有多种多样的应用场合，我们在设计TANGRAM时，兼顾硬件和软件实现，尽可能地使TANGRAM在多个平台上都具有不错表现。

简单性: 一个简单的密码算法，由于其描述和运算的简单性，会利于正确无误的实现，也会吸引更多人对其安全性和实现性能的关注，进而对该密码算法建立更强的信任感。因此，简单性是我们设计TANGRAM的一项原则。

## TANGRAM轮函数的设计

TANGRAM的设计基于比特切片方法，整体结构为SP网络。考虑一个分组长度为128比特或256比特的SP型分组密码，设S层由32个或64个的S盒并置构成。因而在比特切片的实现中，子块的长度为32比特或64比特。将128比特或256比特的密码状态用一个或的比特阵列表示，S层对每一列独立地进行相同的S盒替换，那么，接下来的P变换应该让输出的每一列依赖于其它一些列，以提供必要的扩散性。我们选择了对每一行的32比特字或64比特字进行循环移位操作：循环移位在硬件上可以用简单的拉线实现；能够达到每一列依赖于其它一些列的目的；比特切片实现时，循环移位在软件上可以很容易的高效实现。至此，我们得到了TANGRAM轮函数的基本框架。

得益于我们对S盒的细致选取（详见下面的第3节），TANGRAM达到了很好的安全性和实现性能的平衡；P置换的设计也非常重要（详见第4节），它仅由三个32比特字或64比特字上的循环移位组成，不仅能提供必要的扩散性，也兼顾了硬件和软件的实现性能。TANGRAM的S盒层和P置换层组合在一起，所形成的密码算法整体具有很好的抵抗差分/线性攻击，这也在很大程度上提高了TANGRAM的安全性和实现性能之间的性价比。

## TANGRAM的S盒设计

### 3.1 相关定义和定理

**定义3.1**令表示一个的盒，对任意，定义为：



**定义3.2 (差分均匀度)** 令表示一个的盒，定义的差分均匀度为：



**定义3.3**令表示一个的盒，对任意，定义为：



其中“”表示上的内积。

**定义3.4 (线性度)** 令表示一个的盒，定义的线性度为：



对于任意的双射盒，均有并且，使得这两个值都达到最小值的盒被称为最优盒。

**定义3.5 (最优S盒[14])** 令表示一个的盒，称是最优盒，若它满足以下三个条件：

1. 是双射的；
2. ；
3. .

**定义3.6** (和) 令表示一个非零输入差分，表示一个非零输出差分。令表示的比特汉明重量。定义为：



令表示集合的势。

**定义3.7** (和) 令表示一个非零输入选取模式（也称为输入掩码），表示一个非零输出选取模式（也称为输出掩码）。定义为：



令表示集合的势。

**定义3.8 (仿射等价[14])**两个的盒和被称为仿射等价，如果存在双射的线性映射，以及常数，满足



若两个盒和仿射等价，则有和。也就是说，盒的差分均匀度和线性度在仿射变换下是不变量。

**定理3.1 ([14])** 令和是两个仿射等价的S盒。若是最优盒，则也是最优盒。

**定义3.9 (置换异或等价permutation-then-XOR equivalent[14])**两个的盒和被称为置换异或等价，如果存在上的置换矩阵，以及常量，满足



我们简称这种等价为等价(***PE equivalence***)。

若两个S盒PE等价，则它们一定仿射等价。

**定义3.10 (分量布尔函数)**令表示一个的盒。对于任意的，如下定义的分量布尔函数：

|  |  |
| --- | --- |
|  | (3.1) |
|  | (3.2) |

### 3.2 S盒的选取

的S盒通常比的S盒的硬件实现面积要小很多，比如，一个的S盒通常只需门就可以实现，而AES的S盒的最小实现面积也大于200门。因此，现有的大多数轻量级分组密码都采用了4比特S盒。TANGRAM也采用了4比特S盒。Serpent每隔8轮依次使用8个不同的S盒，每一轮使用一个相同的S盒，然而，使用不同的S盒并不能大幅度提高密码算法的安全性([12]中有类似观点)，反而会增加密码算法的硬件实现面积以及软件实现代码量。因此，我们决定TANGRAM的每一个轮函数都使用16个相同的4比特S盒。

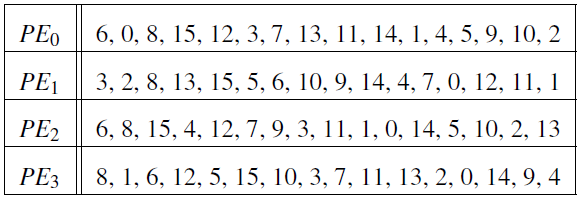
TANGRAM的S盒设计准则如下：

1. 是双射的，即对任意，有；
2. 对任意非零输入差分和非零输出差分，；
3. ；
4. 对任意非零输入选取模式和非零输出选取模式，；
5. ；
6. 没有不动点，即对任意，有。

一个S盒若满足准则1、2和4，那么它是最优的S盒，并且任意一个与之仿射等价的S盒也满足准则1、2和4 (见定义3.8和定理3.1)。一个S盒若满足准则，那么，任意一个与之PE等价 (见定义3.9)的S盒也满足准则。

根据定理3.1，所有的最优S盒可以通过仿射等价关系划分等价类。实际上，所有的最优4比特S盒只存在16种不同的仿射等价类[14]。利用[14]中给出的16个仿射等价类的代表元，我们设计了一个算法将所有满足准则的4比特S盒划分成PE等价类，结果显示满足条件的只有4个PE类。表3.1给出了这4个PE类的代表元，其中，每一行中，第一个数代表输入为的S盒的输出，第二个数代表输入为的S盒的输出，依次类推。

1. 满足准则的4个PE类的代表元



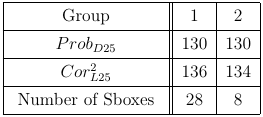
如果一个S盒满足准则，那么给它的输入变量或者输出变量异或一个常数，所得到的新S盒仍然满足准则，并且也不会改变相应密码算法的最优差分迹的概率和最优线性迹的相关系数(差分迹及其概率、线性迹及其相关系数，详见第三部分的安全性分析)。因此，我们只需考虑表3.1中的4个代表元所生成的个S盒，用来表示这个S盒。

首先，利用算法1，能够删除掉一部分会导致每轮只有一个差分（或者线性）活跃S盒的候选S盒。考虑两轮TANGRAM，可以容易地验证，第一轮中任意一个S盒的第个输出比特是第二轮中某个S盒的第个输入比特。因此，如果算法3中的第5行中的条件成立，那么一定存在每轮只有一个活跃S盒的任意轮数的差分迹；类似地，如果第9行中的条件成立，那么一定存在每轮只有一个活跃S盒的任意轮数的线性迹。利用算法3，有个S盒被删除，只剩下个候选S盒。接下来，我们通过考查相应密码算法抵抗差分和线性密码分析的安全性来对这个S盒做进一步筛选。



令表示最优25轮差分迹的概率，表示最优25轮线性迹的相关势(相关势的定义见第三部分的安全性分析)。固定P置换ShiftRow，对于个候选S盒中的每一个，检验相应的128比特分组长度下的密码算法是否满足以下两个不等式：和。实验结果表明，528个S盒中，有124个S盒满足前者不等式；而在这124个S盒中，又有36个满足后者不等式。根据和的值，这36个S盒被划分成2组，表3.2给出了和的值以及对应S盒的个数。

1. 32个候选S盒分成2个组



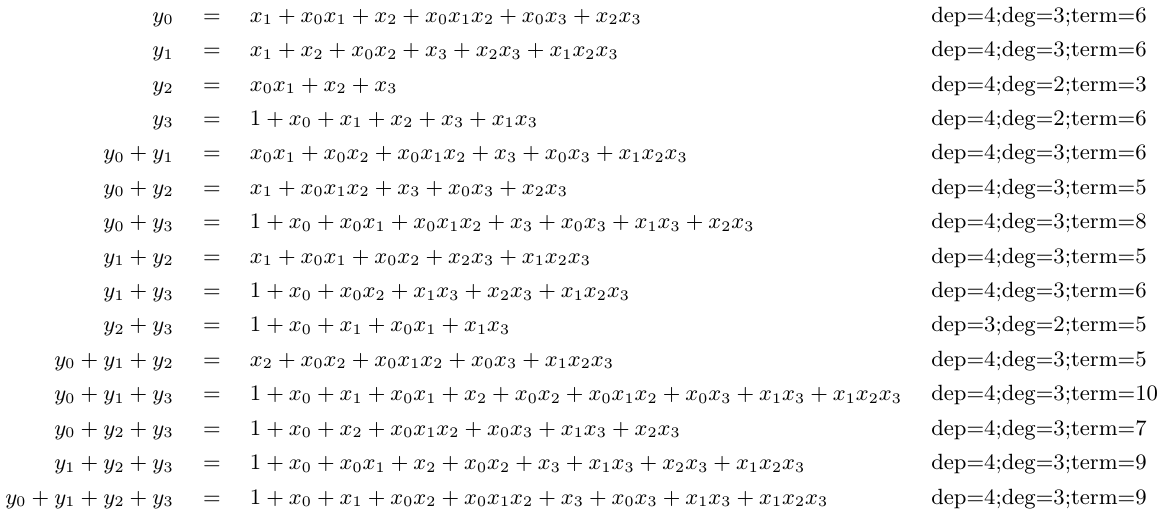
在这两组S盒中，由于第1组的最优25轮线性迹相关势更高，我们选择了第1组的28个S盒进行256比特版本密码算法的测试。固定P置换ShiftRow\_256，对于个候选S盒中的每一个，我们搜索其在相应的256比特分组长度下的密码算法的最优差分迹与最优线性迹，结果表明49轮的最优差分迹概率和最优线性迹相关势均小于2-256。

我们选取了第1组S盒中的第1个S盒。通过在这个S盒变换的前后各加一个常数，可以得到个PE等价的S盒。在这256个S盒中，我们选择了一个没有不动点、并且软件实现速度和硬件面积需求均有较好表现的S盒作为TANGRAM的S盒。

### 3.3 TANGRAM的S盒的密码特性

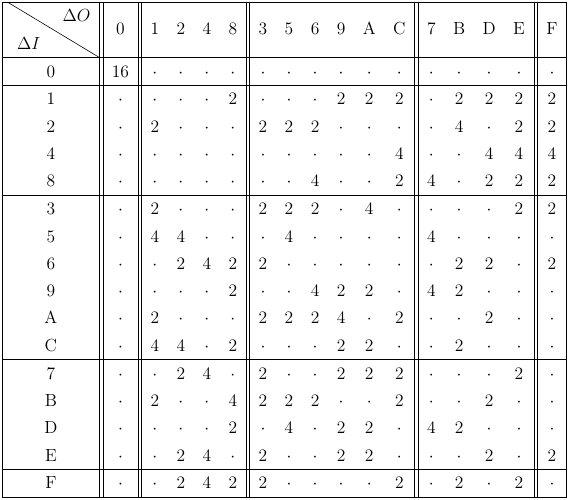
**S盒的分量布尔函数**TANGRAM的S盒作为向量布尔函数，其分量布尔函数(见定义3.10)的代数正规型（ANF）如表3.3所示。其中，表示输入的4个比特，表示输出的4个比特；dep表示该分量布尔函数所依赖的输入比特的个数；deg表示代数次数；term表示项数。

1. TANGRAM的S盒的分量布尔函数

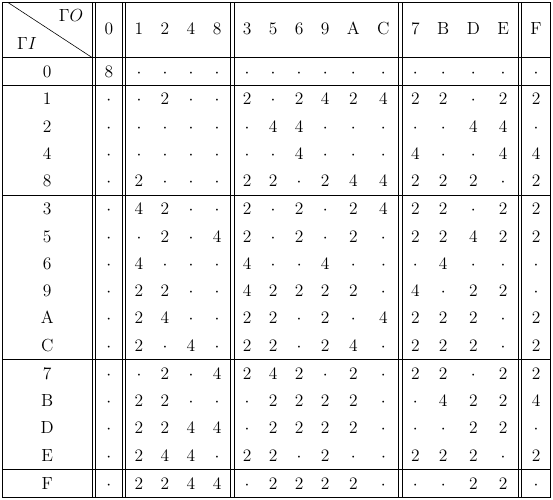


**S盒的差分分布表和线性逼近表**TANGRAM的S盒的差分分布表见表3.4，线性逼近表见表3.5。

1. TANGRAM的S盒的差分分布表(表中元素表示)

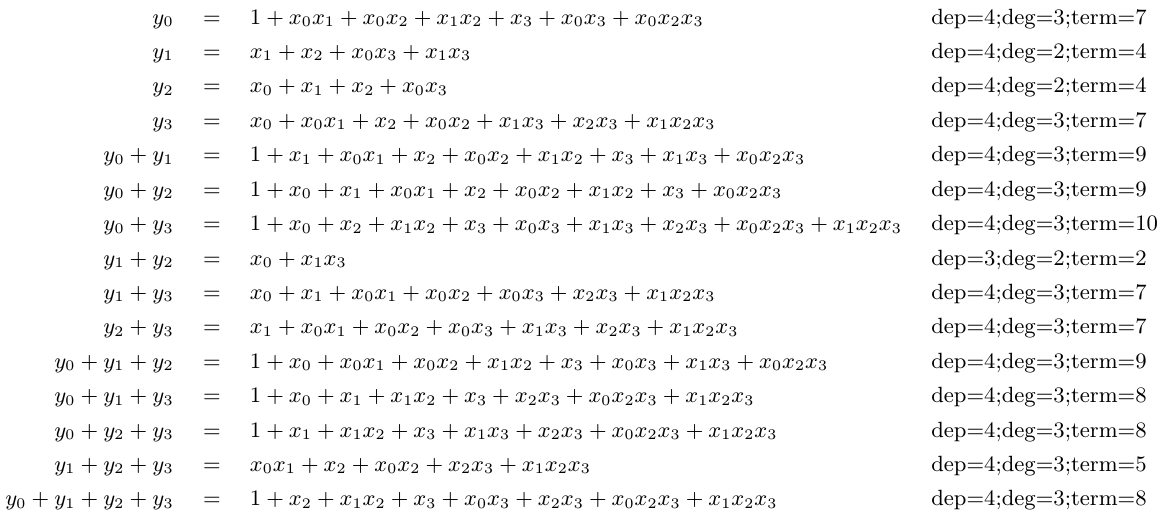


1. TANGRAM的S盒的线性逼近表(表中元素表示)



**逆S盒的向量布尔函数**TANGRAM的逆S盒作为向量布尔函数，其分量布尔函数的代数正规型如表3.6所示。

1. TANGRAM的逆S盒的分量布尔函数



## 4. TANGRAM的扩散层设计

扩散层的设计旨在提供扩散性，使输出的每一列依赖于输入的一些列。我们所选的设计是对每一行的32个比特或64个比特做左循环移位，令表示对第行左循环移位的参数，这些参数的选取准则如下：

1. , , ，；
2. 达到全扩散所需的轮数最少。

注：考虑到在一些低端处理器上，循环左移的参数选为1和8时更为高效，因此我们选取了且。

对于TANGRAM-128，我们的实验结果表明，共有5组候选参数满足以上准则，它们分别是：（1,8,11），（1,8,13），（1,8,19），（1,8, 21），（1,8, 29），其中每个三元组表示的取值。对于5组候选参数中的每一组，经过6轮的轮函数变换之后，128个输入比特中的每个比特都会影响到128个输出比特中的每个比特。从这5种候选参数中，我们选取了作为ShiftRow的循环移位参数。

对于TANGRAM-256，我们的实验结果表明，共有13组候选参数满足以上准则，它们分别是：（1,8, 9），（1,8,11），（1,8,13），（1,8,18），（1,8,19），（1,8, 21），（1,8, 25），（1,8, 35），（1,8, 38），（1,8, 41），（1,8, 42），（1,8, 45），（1,8, 57），其中每个三元组表示的取值。对于13组候选参数中的每一组，经过8轮的轮函数变换之后，256个输入比特中的每个比特都会影响到256个输出比特中的每个比特。从这13种候选参数中，我们选取了作为ShiftRow的循环移位参数。

## 5. TANGRAM的密钥扩展算法设计

TANGRAM的密钥扩展算法设计准则如下：

1. 共用加密算法中的S盒，减少实现代价；
2. 扩散层使用了广义Feistel结构，增加其扩散性；
3. 使用轮常数消除对称性。

TANGRAM的密钥扩展算法满足以上设计准则。通过一些初步分析，我们相信TANGRAM能够抵抗与密钥扩展算法有关的攻击。

# 三．TANGRAM的安全性分析

## 1．TANGRAM安全性概述和声明

TANGRAM128/128的总轮数为44轮，根据我们的安全性分析和评估结果，我们最多能攻击到28轮（占全部轮数的63.6%）。因此，TANGRAM128/128的绝对安全冗余为16轮，相对安全冗余为36.4%。

6轮TANGRAM-128达到全扩散，因此我们在44轮的基础上又增加了6轮，将50轮作为TANGRAM128/256的总轮数。根据我们对28轮TANGRAM128/128的差分攻击结果，我们估计，对TANGRAM128/256最多能攻击到30轮（占全部轮数的60%）。因此，TANGRAM128/128的绝对安全冗余为20轮，相对安全冗余为40%。

TANGRAM256/256的总轮数为82轮，我们估计最多能攻击到56轮（占全部轮数的68%）。因此，TANGRAM256/256的绝对安全冗余为26轮，相对安全冗余为32%。

我们认为：以当前分组密码分析方法的发展现状作为参考，我们为TANGRAM的3个版本预留的冗余都足够保证其抵抗数学类攻击的安全性。

因此，我们声明：TANGRAM128/128、TANGRAM128/256、TANGRAM256/256的安全强度分别是128、256和256比特，在同一个密钥下这三个版本可以加密的消息分块的数量上限分别为264，264和2128。

## 2．TANGRAM的可证明安全

在论述CTR、CBC等分组密码工作模式的合理性时，首先假设底层分组密码是伪随机的，然后在此假设下证明工作模式的伪随机性。如果一个工作模式存在这样的证明，就称它是可证明安全的。

然而，对于一个分组密码算法，现在还没有方法可以从数学上严谨地证明其安全性。如果一个分组密码能够抵抗现有的所有安全性分析方法，我们就称它是安全的。评估一个分组密码抵抗某一种安全性分析方法的安全强度，通常用这种分析方法能够攻击到的轮数及其攻击复杂度来衡量。因此，当一个分组密码能够提供它抵抗某一种安全性分析方法的足够证据时，我们就称它具有抵抗该分析方法的“可证明安全性”。

我们评估了TANGRAM抵抗差分、线性、不可能差分、积分、相关密钥等分析方法的安全性。在此评估结果的基础上，我们认为，TANGRAM具有抵抗这些已知安全性分析方法的“可证明安全性”。

## 3．TANGRAM抵抗差分密码分析的安全性

利用差分密码分析方法[7,8]攻击一个分组长度为比特的轮分组密码，需要找到一个轮的、并且概率大于（对于TANGRAM-128，这个概率值是；对于TANGRAM-256，这个概率值是）的差分传播，的取值和具体分组密码有关。差分迹也称为差分特征。一个差分传播是若干差分迹的集合，集合中的差分迹具有相同的输入差分和相同的轮变换后的输出差分，差分传播的概率是集合中所有差分迹的概率之和。

M.Matsui给出了搜索DES的最优差分迹/线性迹的分支限界算法，基于此，我们提出了一个优化算法[3]。利用该优化算法，我们搜索了128比特分组长度下TANGRAM的第1-26轮的最优差分迹和256比特分组长度下TANGRAM的第1-50轮的最优差分迹，结果如表3.1和表3.2所示。

表3.1 128比特分组长度下TANGRAM最优差分迹的概率

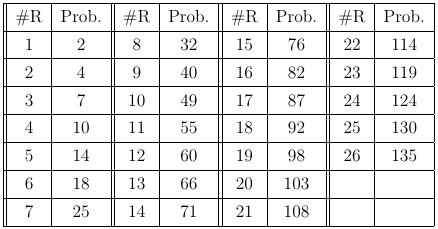
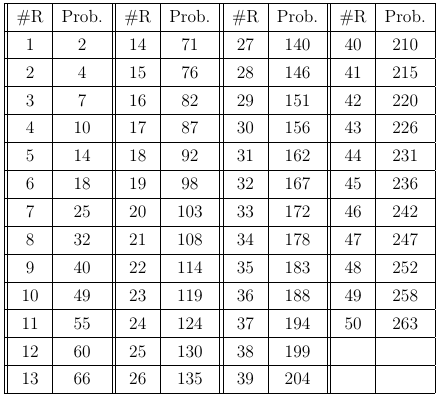


表3.2 256比特分组长度下TANGRAM最优差分迹的概率



从表3.1可以看到，128比特分组长度下TANGRAM的25轮最优差分迹的概率为，小于边界值；256比特分组长度下TANGRAM的49轮最优差分迹的概率为，小于边界值。

TANGRAM扩散层的设计非常简单，因此需要考虑TANGRAM差分/线性迹的聚集效应。我们曾对RECTANGLE差分迹的聚集效应进行了深入研究和实验验证，结果表明：RECTANGLE差分迹的聚集效应非常有限，即使利用聚集效应，也不能构造出轮数更长的差分区分器。

RECTANGLE的分组长度为64比特，而TANGRAM的分组长度为128或256比特，因此受限于计算机的处理能力，对TANGRAM差分/线性迹聚集的实验在实际中几乎不可行。

然而，由于TANGRAM和RECTANGLE在设计上的很大相似性，可以推断：TANGRAM同样具有非常有限的差分迹聚集效应。而我们为TANGRAM预留了足够的安全冗余，因此我们相信三个版本的TANGRAM分组密码均足以抵抗差分密码分析。

### 3.1对缩减轮TANGRAM的差分攻击

本小节利用TANGRAM-128的一个24轮差分迹，给出一个对28轮TANGRAM128/128的差分攻击。该24轮差分迹的概率为，表3.3给出了这个差分迹。为了清晰起见，下表单独列出该差分迹的输入差分和输出差分：

|  |  |  |
| --- | --- | --- |
|  | 第0轮的输入差分 | 第23轮的输出差分 |
| 十六进制 | 0x00000000000000000000000500000005 | 0x60000000000000008004000040000002 |
| 二进制 | 00000000000000000000000100000001  00000000000000000000000000000000  00000000000000000000000100000001  00000000000000000000000000000000 | 00000000000000000000000000000000  00000000000000000000000000000001  00000000000000000001000010000000  00000000000000001000000000000000 |

表3.3 TANGRAM-128的一条24轮差分迹（概率）

|  |  |  |  |
| --- | --- | --- | --- |
| 第i轮 | 轮概率（-log2） | 轮输入差分 | S盒的输出差分 |
| 0 | 4 | 0x00000000000000000000000500000005 | 0x00000000000000000000000100000005 |
| 1 | 5 | 0x00000000000000000000000500000001 | 0x00000000000000000000000100000008 |
| 2 | 5 | 0x00000000000000000000800100000000 | 0x00000000000000000000600800000000 |
| 3 | 5 | 0x000000000000c0000002000000000000 | 0x00000000000020000001000000000000 |
| 4 | 6 | 0x00000000000200000001000000000000 | 0x00000000000100000008000000000000 |
| 5 | 5 | 0x00000000800100000000000000000000 | 0x00000000600800000000000000000000 |
| 6 | 5 | 0xc0000002000000000000000000000000 | 0x20000001000000000000000000000000 |
| 7 | 6 | 0x00000001000000000000000000000002 | 0x00000008000000000000000000000001 |
| 8 | 5 | 0x00000000000000000000000000008001 | 0x00000000000000000000000000006008 |
| 9 | 5 | 0x00000000000000000000c00000020000 | 0x00000000000000000000200000010000 |
| 10 | 6 | 0x00000000000000000002000000010000 | 0x00000000000000000001000000080000 |
| 11 | 5 | 0x00000000000000008001000000000000 | 0x00000000000000006008000000000000 |
| 12 | 5 | 0x00000000c00000020000000000000000 | 0x00000000200000010000000000000000 |
| 13 | 6 | 0x00000002000000010000000000000000 | 0x00000001000000080000000000000000 |
| 14 | 5 | 0x00008001000000000000000000000000 | 0x00006008000000000000000000000000 |
| 15 | 5 | 0x0002000000000000000000000000c000 | 0x00010000000000000000000000002000 |
| 16 | 6 | 0x00010000000000000000000000020000 | 0x00080000000000000000000000010000 |
| 17 | 5 | 0x00000000000000000000000080010000 | 0x00000000000000000000000060080000 |
| 18 | 5 | 0x0000000000000000c000000200000000 | 0x00000000000000002000000100000000 |
| 19 | 6 | 0x00000000000000020000000100000000 | 0x00000000000000010000000800000000 |
| 20 | 5 | 0x00000000000080010000000000000000 | 0x00000000000060080000000000000000 |
| 21 | 5 | 0x0000c000000200000000000000000000 | 0x00002000000500000000000000000000 |
| 22 | 5 | 0x00060000000100000000000000000000 | 0x00040000000800000000000000000000 |
| 23 | 4 | 0x80000000000000000000000000040000 | 0x600000000000000000000000000c0000 |

利用上述24轮差分迹，可以攻击到28轮TANGRAM128/128。首先给出一些记号：

* 对于轮TANGRAM128/128，在第轮中使用轮子密钥，最后和子密钥异或。
* 用，和分别表示第轮的输入、*SubColumn*变换后的输出、以及轮输出。
* 一个密码状态有32列，用表示第列，。

将上述24轮差分迹放置于第2轮到第25轮，在前面增加两轮，在后面增加两轮以及最后的子密钥异或。攻击者需要猜测的一些比特。为了得到第2轮的输入差分，需要猜测的40个比特和的12个比特，然后对明文做部分加密；为了得到第25轮的输出差分，需要猜测的52个比特和的16个比特，然后对密文做部分解密。对于正确对，第27轮有19个不活跃的S盒，从而对应的密文差分在76个比特位置上为0。攻击步骤如下：

1. 考虑一个包含有个明文的集合，它们在第0,7,8,13,16,21,23,24,29和31列遍历所有可能的取值，而在其它22列取一个特定的常数，称这个集合为一个结构。一个结构可以生成大约个明文对。选取个结构，则有个明文，个明文对。
2. 对每一个结构：
   1. 将所有的密文放到一个以76个非活跃比特为索引的哈希表中。仅考虑在同一个表项的密文，这些密文所形成的密文对在76个非活跃比特上的差分都为0。经过这一步过滤后，剩余的数据对的个数大约为。
   2. 猜测的部分比特，并做部分加密：
      1. 分别猜测的4个比特，分别对明文进行部分加密，计算上的差分，检验差分在3个比特位置是否为0，如果不满足，则丢弃相应的数据对。经过这一步过滤，大约剩余个数据对。
      2. 分别猜测的4个比特，对剩余的数据对进行部分加密，计算上的差分，检验差分在2个比特位置是否为0，如果不满足，则丢弃相应的数据对。经过这一步过滤，大约剩余个数据对。
   3. 猜测的部分比特，并做部分加密：
      1. 猜测的4个比特，对剩余数据对做部分加密， 计算的差分，检验它是否等于0x5。如果不满足，则丢弃相应的数据对。经过这一步过滤，大约剩余个数据对。
      2. 分别猜测的4个比特，对剩余数据对做部分加密， 计算的差分，检验它是否分别等于0x1和0x4。如果不满足，则丢弃相应的数据对。经过这一步过滤，大约剩余个数据对。
   4. 猜测的部分比特，并做部分解密：
      1. 分别猜测的4个比特，对密文进行部分解密，计算的差分，检验差分在3个比特位置是否为0。经过这一步过滤，大约剩余个数据对。
      2. 分别猜测的4个比特，对密文进行部分解密，计算的差分，检验差分在2个比特位置是否为0。经过这一步过滤，大约剩余个数据对。
   5. 猜测的部分比特，并做部分解密：
      1. 分别猜测的4个比特，对密文进行部分解密，计算上的差分，检验差分是否分别为0x2，0x4和0x4。如果不满足，否则丢弃相应的数据对。经过这一步过滤，大约剩余个数据对.
      2. 猜测的4个比特，对密文进行部分解密，计算上的差分，检验差分是否为0x8。如果满足，则对所猜测的密钥值对应的计数器增加1。否则丢弃相应的数据对。经过这一步过滤，大约剩余个数据对。
   6. 在计数器中，如果剩余的数据对的个数大于2，则将对应的密钥猜测作为正确密钥的候选值。
   7. 在所有正确密钥的候选值中，通过穷搜种子密钥的其它比特，得到唯一正确的种子密钥。

攻击复杂度：首先估计步骤2.a到步骤2.e的时间复杂度。步骤2.a需要的存储访问；步骤2.b.i需要大约的一轮加密，步骤2.b.ii需要大约的一轮加密；步骤2.c需要大约的一轮加密；步骤2.d.1需要大约的一轮加密，步骤2.d.2需要大约的一轮加密；步骤Step 2.e需要大约的一轮加密。步骤2.a到步骤2.e的总共时间复杂度大约为次一轮加密，或者次28轮加密。

令，攻击的数据复杂度为个明文，存储复杂度为个密钥计数器。对于错误的密钥猜测，期望剩余的数据对个数为；对于正确的密钥猜测，期望剩余的数据对个数为。由Poisson分布，，，因此对于128比特密钥的版本，步骤2.g的时间复杂度为，这是占主要复杂度的步骤。因此，攻击的总体时间复杂度大约为次28轮加密。由Poisson分布，，，因此攻击的成功率大约为。

对TANGRAM128/128，在我们所考虑的安全性分析方法中，上面对28轮的差分攻击，是能攻击到的最长轮数。根据这个攻击，我们估计，对TANGRAM128/256，最多能攻击到30轮。

## 4．TANGRAM抵抗线性密码分析的安全性

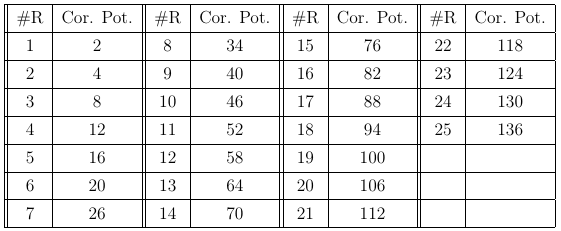
令一个线性迹（也称为线性逼近，或者线性特征）成立的概率为，定义它的偏差为，相关系数为。利用线性密码分析方法[15,16]攻击一个分组长度为比特的轮分组密码，需要找到一个轮的、并且相关系数的绝对值大于（对于128比特分组长度下的TANGRAM，这个值是；对于256比特分组长度下的TANGRAM，这个值是）的线性传播，的取值和具体分组密码有关。一个线性传播是若干线性迹的集合，集合中的线性迹具有相同的输入选取模式和相同的轮变换后的输出选取模式，线性传播的相关系数是集合中所有线性迹的相关系数之和。线性迹的相关系数带有正负号，是正号还是负号，这和子密钥的取值有关。鉴于此，通常我们利用下面的定理来评估。

**定理4.1 ([12])**将相关系数的平方称为相关势。一个线性传播的平均相关势是构成它的所有线性迹的相关势之和，即：

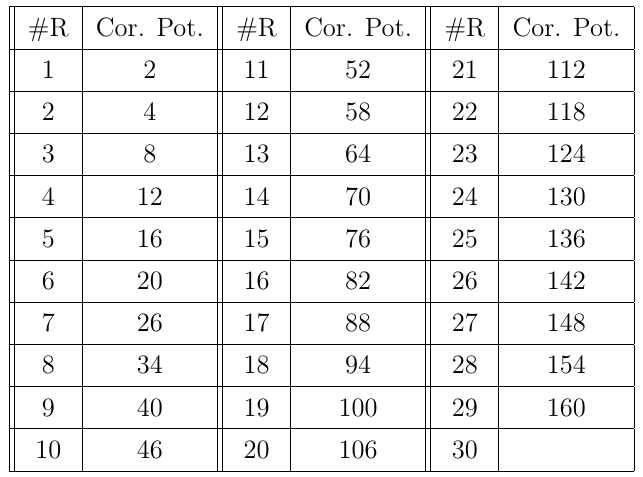
|  |  |
| --- | --- |
|  | (4.1) |

其中，是线性传播的相关系数，是其中一个线性迹的相关系数。平均是指在所有密钥空间上取平均值。

1. 128比特分组长度下TANGRAM最优线性迹的相关势



1. 256比特分组长度下TANGRAM最优线性迹的相关势



修改最优差分迹的搜索算法，我们搜索了128比特分组长度下TANGRAM的第轮的最优线性迹和256比特分组长度下TANGRAM的第轮的最优线性迹，结果如表4.4和表4.5所示。从表4.4可以看到，24轮的最优线性迹的相关势为，小于边界值；由于计算能力所限，256比特分组长度的最优线形性迹只跑到29轮，但根据表4.4所呈现的规律：从第8轮开始，每增加一轮，则线性相关势的重量增加了6，因此，可以推断46轮的最优线性迹的相关势为，小于边界值。

类似于差分分析，我们需要进一步考虑线性迹的聚集效应。我们曾对RECTANGLE线形迹的聚集效应进行了深入研究和实验验证，结果表明：RECTANGLE线形迹的聚集效应也非常有限。同样，可以推断：TANGRAM也具有非常有限的线性迹聚集效应。考虑到我们为TANGRAM预留的安全冗余，我们相信三个版本的TANGRAM分组密码均足以抵抗线性密码分析。

## 5．TANGRAM抵抗不可能差分密码分析的安全性

不可能差分密码分析[6]利用概率为0的差分传播，这种差分传播一般通过中间相遇方法构造。根据不可能差分密码分析的研究现状，一个分组密码的最长不可能差分区分器的轮数接近于全扩散轮数的两倍。

比如，RECTANGLE的全扩散轮数是4轮，我们能找到的它的最长不可能差分区分器的轮数是8轮。

由于TANGRAM-128和TANGRAM-256的全扩散轮数分别是6轮和8轮，因此我们预估它们的最长不可能差分区分器的轮数大约为12轮和16轮，这与相应TANGRAM版本的的总轮数相差很大。因此TANGRAM的三个版本均具有足够的抵抗不可能差分密码分析的安全性。

## 6．TANGRAM抵抗积分密码分析的安全性

积分密码分析(也称为平方攻击，饱和攻击)[11,13]关注的是多个数据之和的传播性质，积分区分器的概率也为1。可分性[17] 是学术界近几年提出的一种一般化积分性质，利用可分性可以构造较传统方法更长更精确的积分区分器。我们将采取基于混合整数线性规划的技术[18] 搜索TANGRAM系列算法的积分区分器。

基于可分性的积分区分器与密码算法的轮子密钥无关，因此针对TANGRAM算法的积分分析只需搜索状态长度为128比特和256比特两个版本的积分区分器。搜索结果显示：TANGRAM-128存在12轮积分区分器，并且在混合整数线性规划的搜索框架下不存在13轮积分区分器；TANGRAM-256存在16轮积分区分器，但是受限于计算资源，我们无法确定TANGRAM-256是否存在17轮积分区分器。虽然搜索算法无法返回TANGRAM- 256的17轮结果，但是基于可分性在其它分组密码上的应用，我们相信TANGRAM-256的积分区分器应该不超过20轮。

**TANGRAM-128的12轮积分区分器**：固定TANGRAM-128最左边S盒的4比特为任意常数，遍历剩下的124比特得到一组包含2124个数据的明文集，将该集合加密12轮后所有输出的异或在和处取值恒为零。

**TANGRAM-256的16轮积分区分器**：固定TANGRAM-256最左边S盒的高位比特（即）为任意常数，遍历剩下255比特得到一组包含2255个数据的明文集，将该集合加密16轮后所有输出的异或在处取值恒为零。

同样，我们构造出的可分积分区分器的轮数和相应TANGRAM版本的总轮数相差很大。因此TANGRAM的三个版本均具有足够的抵抗积分密码分析的安全性。

## 7．抗相关密钥密码分析

在利用密钥扩展算法的攻击中，最有效的是滑动攻击[9,10]和相关密钥攻击[4]。对于TANGRAM，在密钥扩展算法中嵌入不同的轮常量可以阻止滑动攻击；类似于加密算法中的S盒层的使用，给密钥扩展算法提供足够的混淆性；一轮广义Feistel网络可以提供适合的扩散性。基于上述考虑，我们相信，TANGRAM不存在弱密钥、等价密钥，对密钥的选择没有限制，其三个版本均具有足够的抵抗密钥扩展类攻击的安全性。

## 8．抗代数攻击

代数攻击应用于分组密码的成功例子非常罕见。TANGRAM的S盒没有特殊的代数结构。因此，我们相信代数攻击对TANGRAM的安全性没有任何威胁。

## 9．TANGRAM安全余量分析

对于一个密码算法，很难预测未来密码分析方法的发展趋势。鉴于此，我们更加强调密码算法针对已知密码分析方法的抵抗能力及其安全冗余。我们对TANGRAM的三个版本均做了安全性分析和评估，并为每个版本预留了足够的安全冗余。

尤其值得一提的是，TANGRAM基于RECTANGLE而设计，它们在整体结构、S盒选取、扩散层设计等重要方面都具有很强的相似性。TANGRAM可以看作RECTANGLE在分组长度和密钥长度上的扩展。我们对RECTANGLE已经有了全面、深入的安全性分析结果，这些结果能够增强我们对TANGRAM安全性的信心。

我们希望，在对TANGRAM设计和安全性分析充分理解的基础上，同时参考已有的对DES、AES、PRESENT和RECTANGLE等分组密码的安全性分析结果，能够使人们建立起对TANGRAM安全性的信心。

# 四、TANGRAM的软硬件性能分析

## 在x64平台的基本软件实现

### 1.1 TANGRAM算法资源需求分析

按照TANGRAM加密/解密轮函数的第二种描述，下面给出TANGRAM的计算资源需求。

TANGRAM-128加密解密算法资源需求

在没有任何说明的情况下，XOR、AND、OR、NOT、Rotation分别表示32位字上的异或、与、或、非和循环移位。

表1.1给出了TANGRAM-128加密/解密模块的资源需求。由表1.1，可以容易地计算出TANGRAM-128加密/解密算法的资源需求，如表1.2所示。

表1.1 TANGRAM-128加密/解密模块的资源需求

|  |  |  |  |
| --- | --- | --- | --- |
| **加密模块** | AddRoundKey | SubColumn | ShiftRow |
| 所需计算 | 4个XOR | 7个XOR，3个AND，  1个OR，1个NOT | 3个Rotation |
| **解密模块** | AddRoundKey | SubColumn | ShiftRow |
| 所需计算 | 4个XOR | 7个XOR，4AND，  1个NOT | 3个Rotation |

表1.2 TANGRAM-128加密/解密算法的资源需求

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **TANGRAM 128/128** | XOR | AND | OR | NOT | Rotation |
| 加密算法 | 488个 | 132个 | 44个 | 44个 | 132个 |
| 解密算法 | 488个 | 176个 | 0个 | 44个 | 132个 |
| **TANGRAM 128/256** | XOR | AND | OR | NOT | Rotation |
| 加密算法 | 554个 | 150个 | 80个 | 50个 | 150个 |
| 解密算法 | 554个 | 200个 | 0个 | 50个 | 150个 |

TANGRAM-256加密解密算法资源需求

在没有任何说明的情况下，XOR、AND、OR、NOT、Rotation分别表示64位字上的异或、与、或、非和循环移位。

表1.3给出了TANGRAM-256加密/解密模块的资源需求。由表1.3，可以容易地计算出TANGRAM-256加密/解密算法的资源需求，如表1.4所示。

表1.3 TANGRAM-256加密/解密模块的资源需求

|  |  |  |  |
| --- | --- | --- | --- |
| **加密模块** | AddRoundKey | SubColumn | ShiftRow |
| 所需计算 | 4个XOR | 7个XOR，3个AND，  1个OR，1个NOT | 3个Rotation |
| **解密模块** | AddRoundKey | SubColumn | ShiftRow |
| 所需计算 | 4个XOR | 7个XOR，4AND，  1个NOT | 3个Rotation |

表1.4 TANGRAM-256加密/解密算法的资源需求

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **TANGRAM 256/256** | XOR | AND | OR | NOT | Rotation |
| 加密算法 | 884个 | 240个 | 80个 | 80个 | 240个 |
| 解密算法 | 884个 | 320个 | 0个 | 80个 | 240个 |

密钥扩展算法资源需求

**TANGRAM-128**加密/解密算法使用32比特字上的运算，它的密钥扩展算法使用32比特字；**TANGRAM-256**加密/解密算法使用64比特字上的运算，它的密钥扩展算法使用64比特字。表1.5给出了密钥扩展算法一轮操作的计算需求。

表1.5 TANGRAM密钥扩展算法执行一轮操作的资源需求

|  |  |  |  |
| --- | --- | --- | --- |
| 密钥扩展类型 | 并行S盒操作 | 广义Feistel网络 | 轮常数异或  （密钥异或） |
| TANGRAM 128/128 | 32比特字：  7个XOR，3个AND，  1个OR，1个NOT | 32比特字：  2个Rotation，  2个XOR | 6比特字：1个XOR |
| TANGRAM 128/256 | 32比特字：  7个XOR，3个AND，  1个OR，1个NOT | 32比特字：  2个Rotation，  2个XOR | 6比特字：1个XOR  32比特字：1个XOR |
| TANGRAM 256/256 | 64比特字：  7个XOR，3个AND，  1个OR，1个NOT | 64比特字：  2个Rotation，  2个XOR | 6比特字：1个XOR |

由本节结果可以看出，TANGRAM的运算仅使用了以下5种操作：XOR, AND, OR, NOT, Rotation。虽然在不同的平台上，各密码模块会有不同的优化实现方式；但由于这五种操作在大多数平台上都是常见的、易于实现的基本运算，这使得TANGRAM无论在硬件平台、还是低端微处理器和高端处理器平台，都具有很好的表现。此外，TANGRAM的解密算法和加密算法所需的计算资源基本一样。

### 测试环境说明

测试的硬件环境是一台个人计算机。该电脑具有3.40 GHz的Intel(R) Core(TM) i7-6700 CPU，8.00 GB的内存。软件环境是64-位Windows 7操作系统，使用MicrosoftVisualStudio 2010 ProfessionalEdition，Intel C/C++ Compiler。

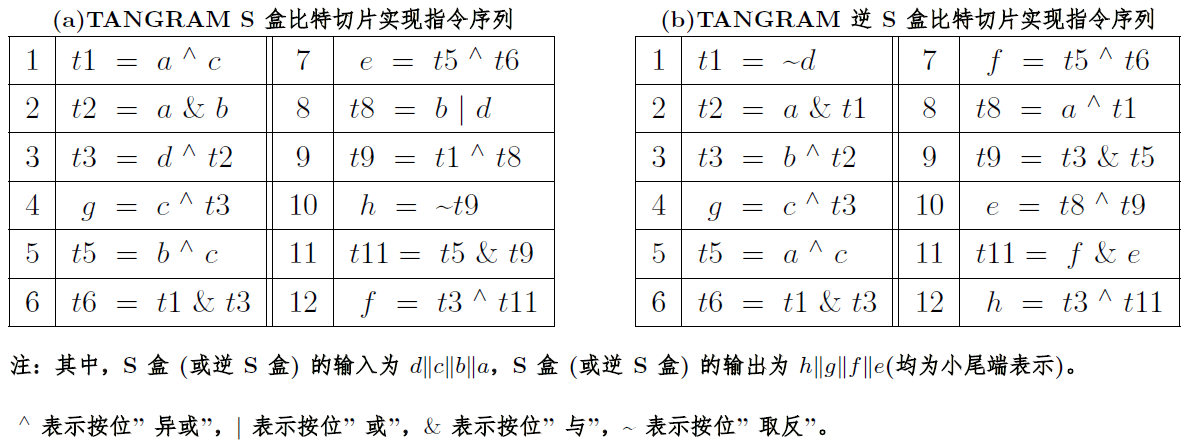
在测试的过程中，为了减少测试的随机性，我们在测试之前关闭了超线程 (Intel Hyper-Threading)、关闭CPU超频加速 (Intel Turbo Boost)、将进程与CPU绑定、关闭所有其它不必要的进程等。

### 软件实现方案

由于TANGRAM的设计基于比特切片思想，其软件实现通过比特切片实现而不是使用查找表（look up table, LUT）。基本操作模块都使用逻辑运算指令来实现，通过对每一个基本模块（列替换、行移位等）最优化实现它们所需要的指令个数，使得代码量尽可能小的同时吞吐量尽可能大。实现中一次处理一个分组，但每个分组块的处理可以做到细粒度的并行。

**列替换SubColumn的实现**在64-位处理器上的软件实现中，混淆层的32个（或64个）相同的S盒可以使用一串逻辑运算指令执行。通过Brain Gladman公布于其个人网页上的一个C语言程序[1]的简单扩展版本，我们可以生成表1.6(a)和表1.6(b)中含12项的指令序列，来分别实现TANGRAM的S盒和逆S盒。

表1.6 TANGRAM的S盒和逆S盒的比特切片实现指令序列



**行移位ShiftRow的实现**扩散层ShiftRow只需要3个32位字（或64位字）上的循环移位，x64处理器的指令集一般包含32位字（和64位字）上的循环左移和右移任意比特的指令。使用这样的循环移位指令，整个扩散层只需要3个指令。

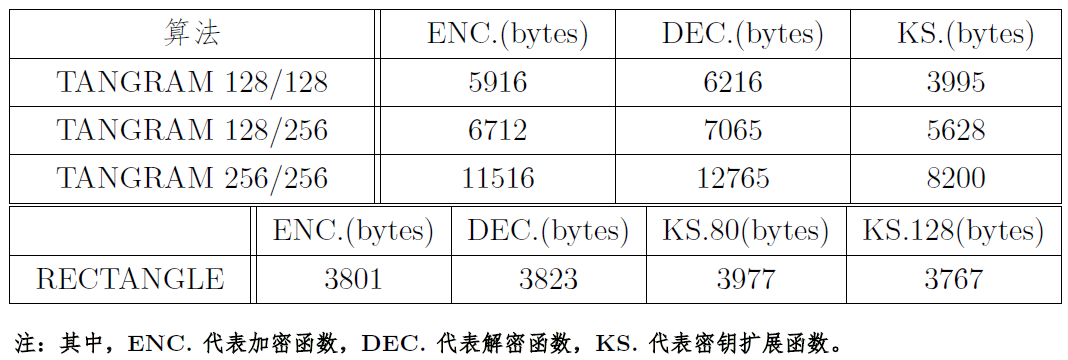
**轮密钥加AddRoundKey的实现**轮密钥加操作需要轮子密钥的读取和4个XOR操作。

由于我们在对算法进行软件测试时，使用的是MicrosoftVisualStudio 2010 ProfessionalEdition，我们需要包含<intrin.h>头文件，使用相应的固有函数。在实现代码中，使用 Intel C/C++ Compiler。通过Intel vTune Amplifier 性能调优工具，查看生成的汇编代码，我们发现以上的三种操作部件通过使用Intel C/C++ Compiler编译可以全部在寄存器中完成（除了轮子密钥xor的指令）。因而，除了输入、输出和轮子密钥的读取，就不再需要额外的内存操作了，核心加密和解密程序基本上只使用6种逻辑运算指令：and、or、not、xor、rol、mov。加密和解密程序的内存需求可以降到极致。

### 1.4 代码量分析

考虑到 x64 平台的内存比较大，指令缓存充裕，我们的实现都是全轮展开而不是通过循环实现。通过使用 Intel vTune Amplifier 工具，我们对各种实现进行了性能分析。通过查看编译C代码后得到的汇编代码，找到运行时函数代码所在的内存首地址和终止地址，将函数终止地址减去起始地址，我们可以得到运行时一个函数的代码在内存中所占用的字节数。表1.7列出了各函数的实现代码量。作为对比，我们在相同的环境下也测试了RECTANGLE的代码量。

表1.7 TANGRAM和RECTANGLE在x64平台上实现的代码量分析



### 1.5 测试方案及测试结果

#### 速度测试方案

我们对运行所需CPU时钟周期数进行多次采样，取CPU时钟周期数的最小值、平均值和标准差。具体测试方法是参照Brain Gladman对AES测试软件执行时间时采用的方法[2]：

将待测函数计为，将连续次执行记为。计程序块执行所需的时钟周期数为。

**最小时钟周期测试方法**测试多次运行所需的最小时钟周期数。其中，为加密函数、解密函数或密钥扩展。具体方法：循环执行次和。得到和。返回。

**时钟周期的均值和标准差的测试**测试多次运行所需的时钟周期数的均值和标准差。其中，为加密函数、解密函数或密钥扩展。具体方法：对执行所需的时钟周期个数进行初始采样1和正式采样2。

1. 首先执行初始采样1，得到采样1均值和采样1标准差（至多为）。
2. 随后执行正式采样2，在正式采样2过程中，只有一次采样的时间在范围内，才记为一次有效采样。得到正式采样2中有效采样的均值和标准差。
3. 采样2结束后，只有有效采样次数占采样2总采样次数的比例达到 90% 以上，并且采样2标准差小于均值的 10% 时，才认为时间测试成功，返回 TRUE。若测试不成功，重新采样测量。
4. 若连续测试10次都没有成功则将标准差的可接受限度放低 5%，再次重新采样测量，当标准差的可接受限度降到了均值的 30% 时，仍没有测试成功。此时返回FALSE，测试失败。

对处理长消息的时钟周期数均值和标准差的测试比较耗时，因此只包含对处理长消息的最小时钟周期数的测试。

#### 2 测试结果

代码空间和存储空间的占用情况

**（代码空间）**通过使用 Intel vTune Amplifier 工具，我们对各种实现进行了性能分析。在代码空间方面，运行时加解密和密钥扩展的代码空间如表1.2所示。

**（存储空间）**通过Intel vTune Amplifier 性能调优工具查看汇编代码，我们发现一般除了输入、输出和轮子密钥的读取和写入，就不再需要额外的内存操作了，核心加密和解密程序基本上只使用逻辑运算指令和寄存器间的赋值，加密和解密程序的内存需求降到了极致。如果不考虑待加解密的数据所占的存储空间，整个算法核心代码除了轮扩展密钥需要存储空间：

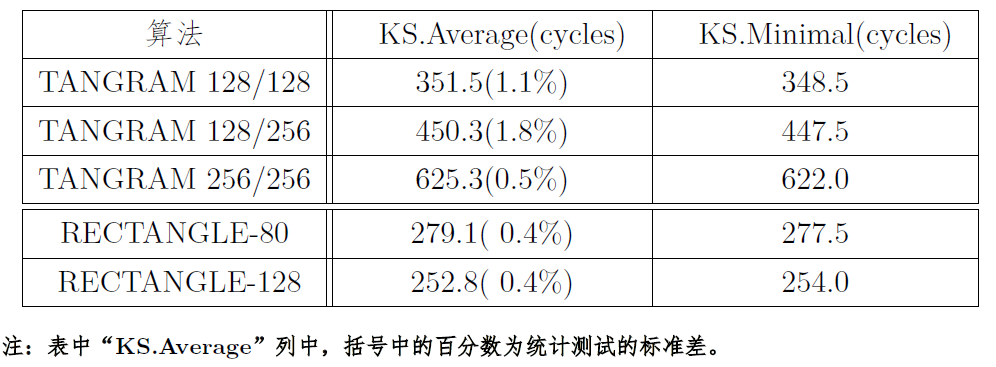
* TANGRAM 128/128的密钥需要45\*16=720字节存储空间
* TANGRAM 128/256的密钥需要51\*16=816字节存储空间
* TANGRAM 256/256的密钥需要83\*32=2656字节存储空间

以及函数的接口所需的内存栈空间外，可以认为加密、解密和密钥扩展部件不再需要额外的存储空间。

加解密性能测试结果

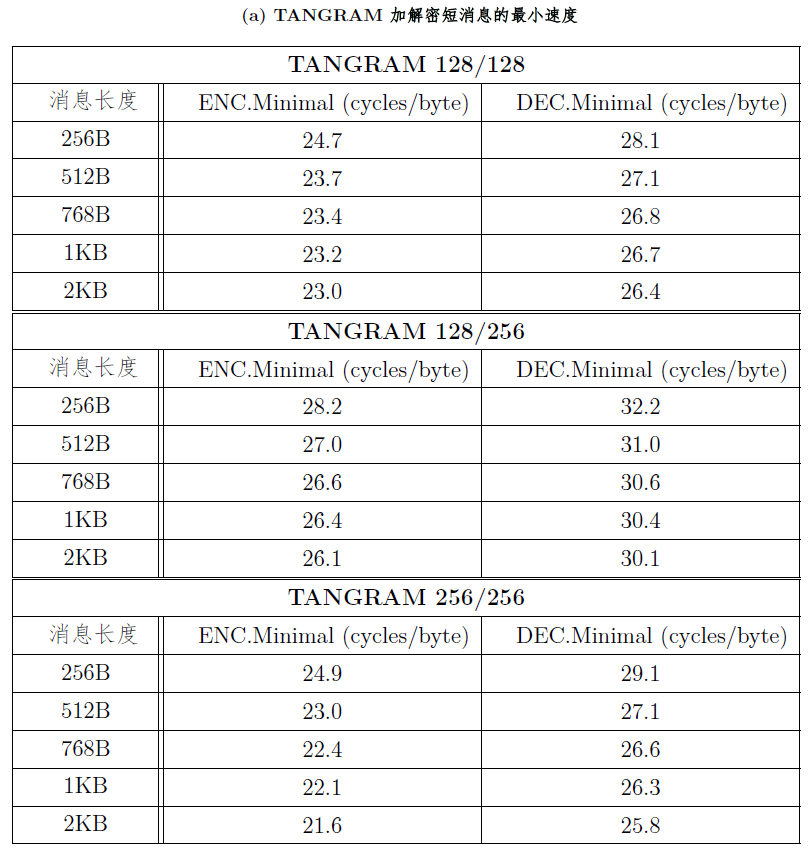
表1.8给出了TANGRAM三种算法在x64平台上的密钥扩展的性能测试结果。作为对比，我们在相同的环境下也测试了RECTANGLE的密钥扩展性能。

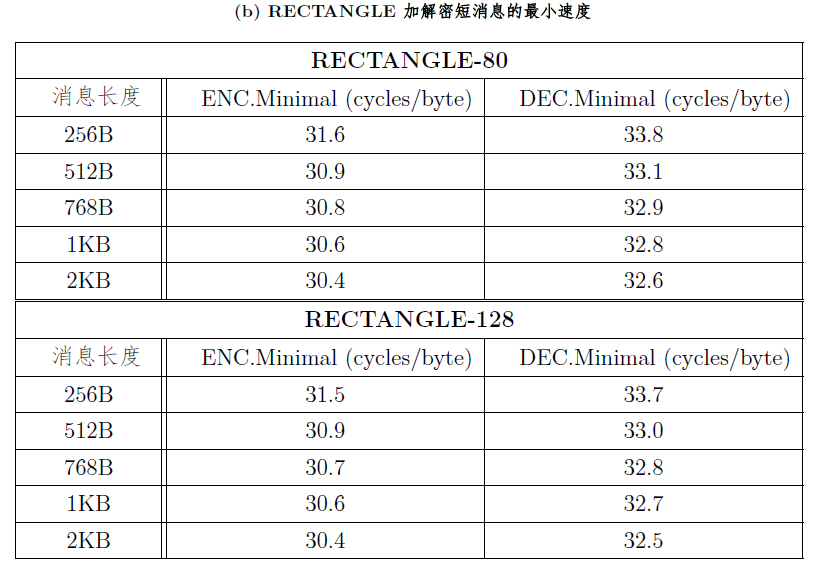
表1.8 TANGRAM和RECTANGLE在x64位平台上实现的密钥扩展性能测试结果



* 表1.9给出了TANGRAM三种算法在x64平台上加解密短消息的性能测试结果，结果以最小速度(cycles/byte)表示。作为对比，我们在相同的环境下也测试RECTANGLE加解密短消息的最小速度(cycles/byte)。

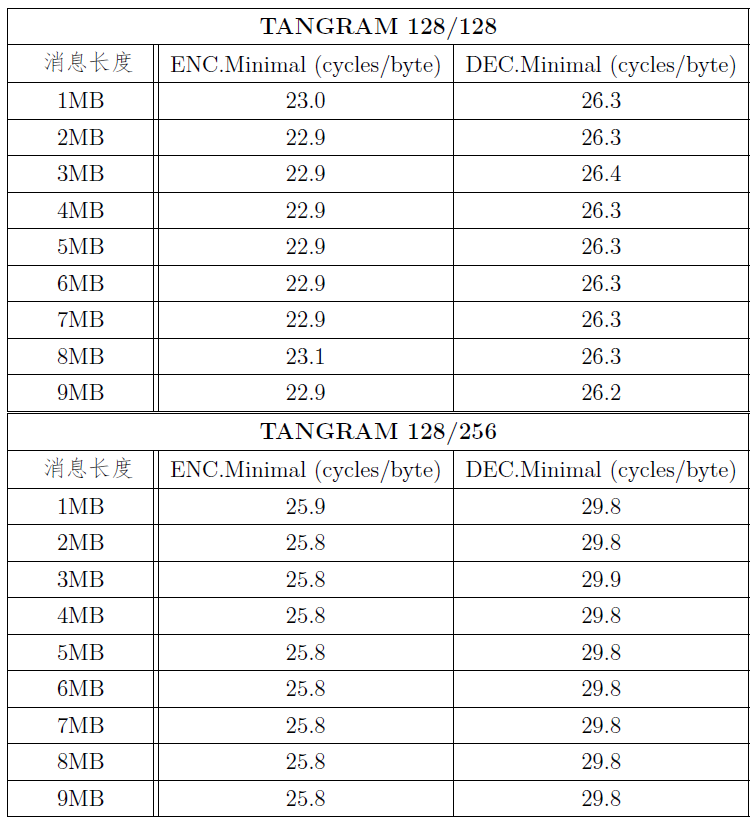
表1.9 TANGRAM和RECTANGLE在x64平台上加解密短消息性能测试结果

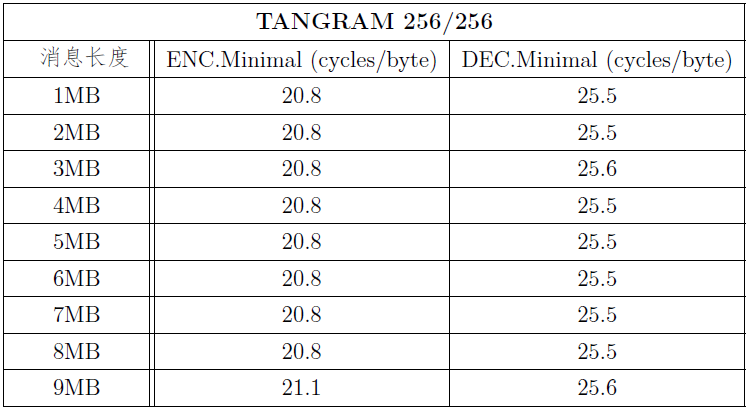




* 表1.10给出了TANGRAM三种算法在x64平台上用ECB模式加解密长消息的性能测试结果，结果以最小速度 (cycles/byte)表示。

表1.10 TANGRAM在x64位平台上ECB模式加解密长消息性能测试结果





### 代码功能结构说明

#### 代码功能说明

TANGRAM包含三个算法，因此我们提交三个软件代码包：

* TANGRAM 128128\_x64
* TANGRAM 128256\_x64
* TANGRAM 256256\_x64

分别提供TANGRAM128/128、TANGRAM128/256和TANGRAM256/256 在x64平台上加解密、正确性测试和速度测试的功能。

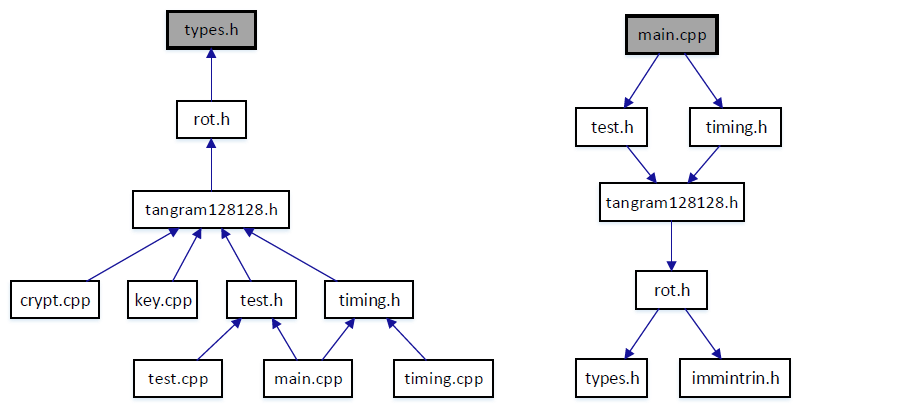
这三个软件代码包分别提供以下功能：

* 加密函数，对给定长度的数据块进行加密；
* 解密函数，对给定长度的数据块进行解密；
* 轮加密函数，对给定长度的数据块进行特定轮的加密；
* 密钥扩展，将128比特密钥（对应TANGRAM128/128）或256比特密钥（对应TANGRAM128/256和TANGRAM256/256）扩展为轮子密钥；
* 测试向量生成代码，加密若干分组之后的输出；
* 速度的测试代码，测试密钥扩展、加密和解密短消息、ECB模式加密和解密长消息的速度；
* 使用通用指令集，每次加密一个分组。

三个算法对应的软件包中，代码结构相同，只在数据的分组长度和密钥扩展函数有区别。下面以TANGRAM128128\_x64软件包为例，说明代码结构和主要函数结构。

#### 1.6.2 代码结构说明

这里列出了源代码文件列表，并附带简要说明，图1.1刻画出了文件的包含关系:



(a)所有文档自底向上依赖关系(b)主函数自上向下对各头文件的依赖关系

1. 代码文档包含关系图

types.h包含数据类型的宏定义等。

rot.h包含密码算法所使用的循环移位操作的定义。

Tangram128128.h包含密码算法的参数宏定义、加密、解密和密钥扩展等函数声明。

crypt.cpp包含密码算法的加密、解密函数定义。

key.cpp包含密码算法的密钥扩展相关的函数定义。

test.h包含对密码算法的加密、解密、密钥扩展算法的正确性测试和测试向量生成的函数声明。

test.cpp包含对密码算法的加密、解密、密钥扩展算法的正确性测试和测试向量生成的函数定义。

timing.h包含对密码算法的加密、解密、密钥扩展算法的速度测试的函数声明。

timing.cpp包含对密码算法的加密、解密、密钥扩展算法的速度测试函数定义。

main.cpp是主函数的定义，调用测试正确性的函数和测试速度的函数进行测试。

#### 主要函数结构说明

**主要函数功能说明**这里列出了主要函数，并附带简要说明：

encrypt 对给定个数的数据块使用通用指令集进行加密。

decrypt 对给定个数的数据块使用通用指令集进行解密。

encrypt\_variousRound 对给定个数的数据块使用通用指令集进行加密指定轮数。

decrypt\_variousRound 对给定个数的数据块使用通用指令集进行解密指定轮数。

Crypt\_Enc\_Block对给定长度的数据块进行加密，不包含密钥扩展。

Crypt\_Dnc\_Block对给定长度的数据块进行解密，不包含密钥扩展。

Crypt\_Enc\_Block\_Round对给定长度的数据块进行特定轮的加密，不包含密钥扩展。

Crypt\_Dec\_Block\_Round对给定长度的数据块进行特定轮的解密，不包含密钥扩展。

tangram\_ecb\_enc对给定长度的数据块进行ECB模式加密，包含密钥扩展。

tangram\_ecb\_dec对给定长度的数据块进行ECB模式解密，包含密钥扩展。

key128TANGRAM128/128的密钥扩展。

Key\_Schedule TANGRAM 128/128的密钥扩展接口函数，调用key128生成密钥。

**test**测试使用TANGRAM 128/128加密若干分组、解密被加密的密文。将测试结果以二进制形式打印到文件。

time\_key8 用以测试加密算法执行密钥扩展的平均执行速度。测试执行 8 次密钥扩展所需的时钟周期均值和标准差。

time\_enc16 用以测试加密算法加密指定长度消息的平均执行速度。测试执行 16 次加密算法加密指定长度消息所需的时钟周期均值和标准差。

time\_dec16 用以测试解密算法解密指定长度消息的平均执行速度。测试执行 16 次解密算法解密指定长度消息所需的时钟周期均值和标准差。

key\_cycles 用以测试执行密钥扩展算法所需的最小时钟周期数。

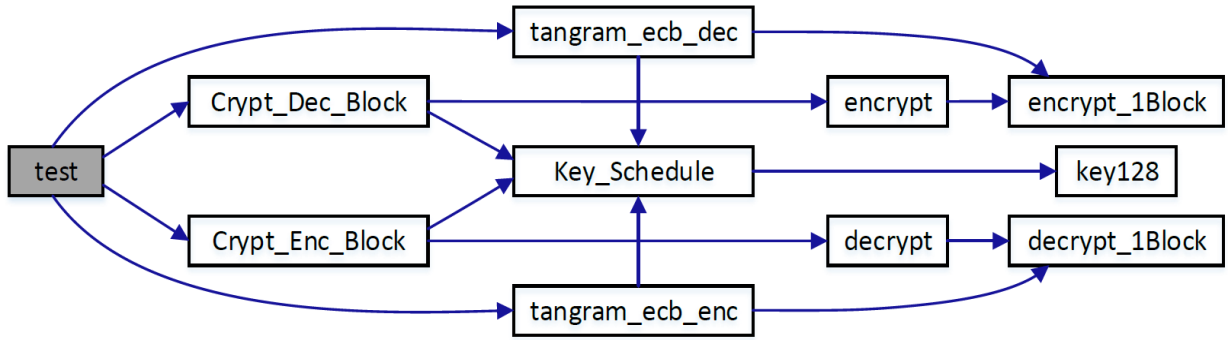
enc\_cycles 用以测试加密算法加密指定长度短消息的最小时钟周期数。

dec\_cycles 用以测试解密算法解密指定长度短消息的最小时钟周期数。

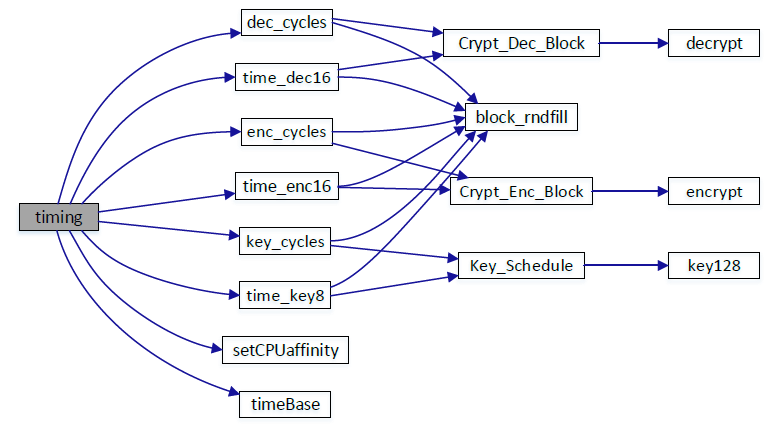
**timing**执行速度测试的函数入口，通过调用这个函数可以一次性的进行：测试密钥扩展的最小时钟周期数开销 (cycles)、测试密钥扩展的平均时钟周期数和统计偏差 (cycles)、测试加解密短消息最大速率(cycles/byte)、测试ECB模式加解密长消息最大速率(cycles/byte)、测试加解密短消息平均速率及统计的标准差(cycles/byte)。

main 调用测试正确性的函数test()和测试速度的函数timing()进行测试。

**主要函数结构说明**如图1.2所示



1. 正确性测试的函数调用图



1. 时间测试的函数调用图
2. 主要函数结构关系图

### 1.7 编程接口说明

* TANGRAM128/128算法的编程接口位于文件：

TANGRAM128128\_x64/codes/tangram128128.h；

* TANGRAM128/256算法的编程接口位于文件：

TANGRAM128256\_x64/codes/tangram128256.h；

* TANGRAM256/256算法的编程接口位于文件：

TANGRAM256256\_x64/codes/tangram256256.h。

三个算法的编程接口分别包含以下五个：

* 加密函数接口。如果函数执行正确则返回0，否则返回1。

int Crypt\_Enc\_Block(

unsigned char \*input,

int in\_len,

unsigned char \*output,

int \*out\_len,

unsigned char \*key ,

int keylen);

* 解密函数接口。如果函数执行正确则返回0，否则返回1。

int Crypt\_Dec\_Block(

unsigned char \*input,

int in\_len,

unsigned char \*output,

int \*out\_len,

unsigned char \*key,

int keylen);

* 分轮加密函数接口。函数执行第1轮至第CryptRound轮加密，输出第CryptRound轮结果。如果函数执行正确则返回0，否则返回1。

int Crypt\_Enc\_Block\_Round(

unsigned char \*input,

int in\_len,

unsigned char \*output,

int \*out\_len,

unsigned char \*key,

int keylen,

int CryptRound);

* 分轮解密函数接口。函数执行第1轮至第CryptRound轮解密，输出第CryptRound轮结果。如果函数执行正确则返回0，否则返回1。

int Crypt\_Dec\_Block\_Round(

unsigned char \*input,

int in\_len,

unsigned char \*output,

int \*out\_len,

unsigned char \*key,

int keylen,

int CryptRound);

* 密钥扩展算法函数接口。Direction：0加密，1解密。如果函数执行正确则返回0，否则返回1。

int Key\_Schedule(

unsigned char \*Seedkey,

int KeyLen,

unsigned char Direction,

unsigned char \*Subkey);

## 在x64平台利用SSE/AVX指令集的优化软件实现

### 2.1 测试环境说明

测试的硬件环境是一台个人计算机。该电脑具有3.40 GHz的Intel(R) Core(TM) i7-6700 CPU，16.0 GB的内存。软件环境是64位Windows 10操作系统，使用MicrosoftVisualStudio 2010 ProfessionalEdition，Intel C/C++编译器。

### 2.2 测试结果

利用SSE指令集，对于TANGRAM-128，每次完成4个分组块的并行运算；对于TANGRAM-256，每次完成2个分组块的并行运算。利用AVX指令集，对于TANGRAM-128，每次完成8个分组块的并行运算；对于TANGRAM-256，每次完成4个分组块的并行运算。首先需要对分组块进行重组，具体地，以TANGRAM-128的SSE指令集实现为例： 将4个分组的第一个32比特字放在同一个128比特的寄存器中，然后将4个分组的第二个32比特字放在另一个128比特寄存器中，依次类推，最后，4个分组的0、1、2、3个32比特字分别放在、、、这四个128比特寄存器中。当处理个分组时，若不是4的倍数，则128比特寄存器的相应位置置为0。由于TANGRAM的设计思路就是基于比特切片技术，以上数据格式转换的开销非常低 (使用(v)punpck系列指令实现)。在数据格式转换好之后，随后的实现就与处理一个分组的情形大致相似了，只不过需要使用SSE或AVX指令类型进行操作。

下面的表格给出了我们的测试结果，从中可以看出，利用SSE和AVX指令集，TANGRAM的软件实现速度得到了数倍提升。

ECB模式速度测试结果

消息长度：256字节

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 算法软件实现类别 | | | 加密速率 | | 解密速率 | |
| (cpb) | (Mbps) | (cpb) | (Mbps) |
| 64bit Windows 环境下的算法速率优化C实现 | 基本  实现 | 128/128 | 26.7 | 1044.5 | 31.1 | 896.0 |
| 128/256 | 28.9 | 962.1 | 33.4 | 834.7 |
| 256/256 | 26.7 | 1044.5 | 26.7 | 1044.5 |
|  | SSE指令集优化实现 | 128/128 | 8.5 | 3276.8 | 9.3 | 2994.9 |
| 128/256 | 10.5 | 2652.6 | 11.5 | 2422.0 |
| 256/256 | 18.4 | 1513.7 | 21.2 | 1313.8 |
| AVX指令集优化实现 | 128/128 | 5.9 | 4720.8 | 6.1 | 4566.0 |
| 128/256 | 6.9 | 4036.6 | 7.2 | 3868.4 |
| 256/256 | 11.7 | 2380.6 | 11.6 | 2401.1 |

ECB模式速度测试结果

消息长度：2KB

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 算法软件实现类别 | | | 加密速率 | | 解密速率 | |
| (cpb) | (Mbps) | (cpb) | (Mbps) |
| 64bit Windows 环境下的算法速率优化C实现 | 基本  实现 | 128/128 | 23.4 | 1191.8 | 27.8 | 1001.6 |
| 128/256 | 26.4 | 1055.0 | 32.0 | 870.5 |
| 256/256 | 22.0 | 1267.2 | 27.0 | 1032.0 |
| SSE指令集优化实现 | 128/128 | 7.3 | 3815.5 | 8.1 | 3438.6 |
| 128/256 | 8.3 | 3355.8 | 9.4 | 2963.1 |
| 256/256 | 15.3 | 1820.4 | 17.2 | 1619.3 |
| AVX指令集优化实现 | 128/128 | 4.7 | 5926.1 | 5.1 | 5461.3 |
| 128/256 | 5.3 | 5255.2 | 5.8 | 4802.2 |
| 256/256 | 9.1 | 3060.7 | 9.4 | 2963.1 |

CBC模式速度测试结果

消息长度：256字节

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 算法软件实现类别 | | | 加密速率 | | 解密速率 | |
| (cpb) | (Mbps) | (cpb) | (Mbps) |
| 64bit Windows 环境下的算法速率优化C实现 | 基本  实现 | 128/128 | 26.7 | 1044.5 | 31.1 | 896.0 |
| 128/256 | 30.4 | 917.0 | 34.9 | 797.2 |
| 256/256 | 24.5 | 1135.5 | 31.1 | 895.9 |

CBC模式速度测试结果

消息长度：2KB

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 算法软件实现类别 | | | 加密速率 | | 解密速率 | |
| (cpb) | (Mbps) | (cpb) | (Mbps) |
| 64bit Windows 环境下的算法速率优化C实现 | 基本  实现 | 128/128 | 24.4 | 1142.2 | 28.1 | 990.8 |
| 128/256 | 27.8 | 1002.9 | 32.4 | 860.0 |
| 256/256 | 22.0 | 1267.2 | 27.0 | 1032.0 |

CTR模式速度测试结果

消息长度：256字节

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 算法软件实现类别 | | | 加密速率 | | 解密速率 | |
| (cpb) | (Mbps) | (cpb) | (Mbps) |
| 64bit Windows 环境下的算法速率优化C实现 | SSE指令集优化实现 | 128/128 | 8.6 | 3238.7 | 8.6 | 3238.7 |
| 128/256 | 10.2 | 2730.7 | 10.2 | 2730.7 |
| 256/256 | 20.6 | 1352.1 | 20.7 | 1345.5 |
| AVX指令集优化实现 | 128/128 | 6.0 | 4642.1 | 5.7 | 4886.5 |
| 128/256 | 6.8 | 4096.0 | 6.8 | 4096.0 |
| 256/256 | 11.4 | 2443.2 | 11.7 | 2380.6 |

CTR模式速度测试结果

消息长度：2KB

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 算法软件实现类别 | | | 加密速率 | | 解密速率 | |
| (cpb) | (Mbps) | (cpb) | (Mbps) |
| 64bit Windows 环境下的算法速率优化C实现 | SSE指令集优化实现 | 128/128 | 7.2 | 3868.4 | 7.2 | 3868.4 |
| 128/256 | 8.4 | 3315.8 | 8.3 | 3355.8 |
| 256/256 | 16.7 | 1667.8 | 16.7 | 1667.8 |
| AVX指令集优化实现 | 128/128 | 4.7 | 5926.1 | 4.6 | 6055.0 |
| 128/256 | 5.2 | 5356.3 | 5.2 | 5356.3 |
| 256/256 | 8.9 | 3129.5 | 8.9 | 3129.5 |

## 在32位ARM Cortex-M3平台的软件实现

实现所使用的是纯C代码，并使用Keil5 + Jlink进行编译调试，性能测试采用Keil5软件记录时间，通过比较执行代码片段前后的时间差得到运行时间，具体测试结果如下面两个表所示。我们给出了两种不同速度单位下的测试结果，第1个表格的速度单位是Mbps，第2个表格中的结果由设备频率72MHz换算为Cycles/Byte（表示加密/解密一个字节平均需要多少时钟周期）。

|  |  |  |  |
| --- | --- | --- | --- |
|  | 32bit ARM环境下的算法速率优化C实现（单位：Mbps） | | |
| TANGRAM 128/128 | TANGRAM 128/256 | TANGRAM 256/256 |
| 加密速率 | 2.13Mbps | 1.77 Mbps | 1.01Mbps |
| 解密速率 | 2.22Mbps | 1.83 Mbps | 0.99 Mbps |

|  |  |  |  |
| --- | --- | --- | --- |
| Cycles/Byte | 32bit ARM环境下的算法速率优化C实现（单位：cycles/byte） | | |
| TANGRAM128/128 | TANGRAM128/256 | TANGRAM256/256 |
| 加密速率 | 270.4 Cycles/Byte | 325.4 Cycles/Byte | 570.3 Cycles/Byte |
| 解密速率 | 259.5 Cycles/Byte | 314.8 Cycles/Byte | 581.8 Cycles/Byte |

ARM平台拥有丰富的指令集，使用如多内存加载、循环移位、逻辑运算、字节交换等指令可以非常方便地实现TANGRAM算法，我们分析评估，在使用**汇编语言的情况下，TANGRAM的性能至少可以提升约30%-50%。后续提供汇编优化版本。**

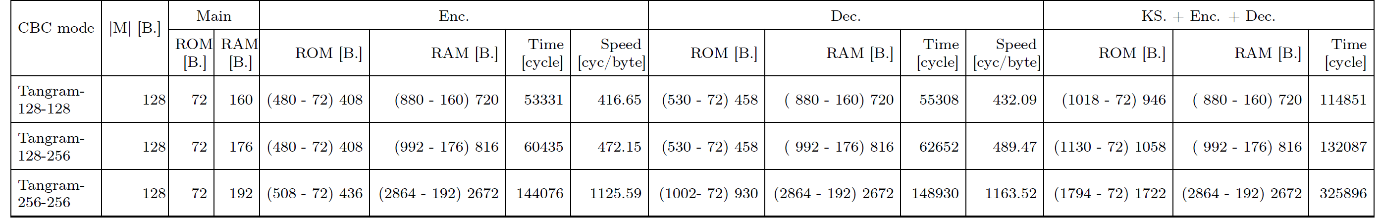
此外，Neon是适用于ARM Cortex-A系列处理器的一种128位SIMD(Single Instruction, Multiple Data,单指令、多数据)扩展结构。类似于Intel处理器中的SSE指令，利用NEON指令集可以更大幅度地提升Tangram在ARM平台的加解密速度。

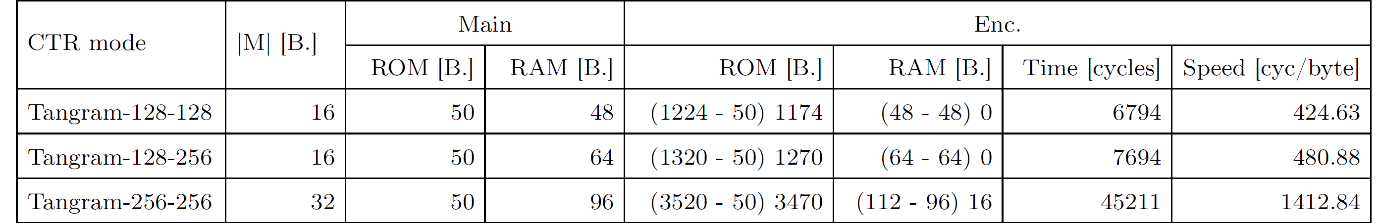
## 在8位AVR平台的软件实现

为了评估TANGRAM在8位微控制器上的性能，我们进行了针对Atmel AVR 8位微处理器的软件实现。具体的目标设备是ATmega128，其上配有128Kbytes的闪存（flash, 即只读内存ROM），4Kbytes的随机存储内存（RAM），以及32个8比特通用寄存器。实现所考虑的应用场景包括：

1. 使用CBC模式加密和解密128字节消息，消息、种子密钥、加密模式所需的初始向量都是使用RAM存放。此外，各轮的轮子密钥需要通过调用密钥扩展算法生成，并存放在RAM中。这种应用场景对应于传感器网络和IoT设备之间的安全通信。
2. 使用CTR模式加密一个消息块儿的消息（128比特或256比特），各轮轮子密钥已经扩展好并固定写到程序闪存中，无需存储主密钥也无需调用密钥扩展算法。加密的消息和计数器的值存于RAM中。这种应用场景对应于IoT认证所需执行的挑战-握手认证协议。

实现所使用的是汇编代码，并使用Atmel Studio 7.0 中所带有的AVRmacro assembler进行编译。我们关注以下三个方面的性能表现：ROM开销（以字节Bytes记），RAM开销（以字节Bytes记），执行时间（以cycles记），以及执行速度（以cycles/byte记）。这几方面的实现结果也是在Atmel Studio 7.0平台上完成仿真测试。以下两张表格汇总了我们的测试结果。





由测试结果可以看出，对于128比特分组长度的TANGRAM，当考虑第一种应用场景（CBC模式加密解密128字节数据），其可以在不损失运行时间的情况下花费极小的代码量（ROM）；对于所考虑的第二种应用场景（CTR模式执行认证握手协议处理较短数据），其只需要可忽略的随机存储内存（RAM）。实际上，第一种应用场景中几乎全部的RAM是用来存储扩展出来的轮子密钥；类似地，在第二种应用场景下很大一部分的ROM需求是用来存储扩展出来的轮子密钥，而加密解密部分所需的代码量和随机存储内存（ROM和RAM）都非常少。对于256比特分组长度的TANGRAM，由于状态大到无法全部存放于寄存器上进行更新，我们需要将一半的状态存储于RAM中,这不仅需要额外的RAM开销，而且需要不断地从RAM中装载和暂存一半的状态字节，这导致执行时间也有了不容忽视的额外开销，因而相较于128比特分组长度的版本，256比特分组长度的版本不是很适合在资源受限的低端微处理器上实现。

## 硬件实现

为了评估TANGRAM的硬件实现性能，我们针对两类硬件实现平台进行评估，即专门应用的集成电路（ASIC）和可编程逻辑阵列（FPGA）。 我们选取了两个常见的工艺库来评估ASIC硬件实现：UMC130nm和 学术界常用的开源工艺库Nangate45nm。 Xilinx公司的7系xc7s100fgga676-1FPGA平台被用来评估TANGRAM在FPGA上的实现结果和性能。

按照《全国密码算法设计竞赛分组密码算法第二轮算法材料提交说明》（以下简称第二轮材料说明）要求，我们准备了可用于ECB模式的TANGRAM三个版本的硬件实现。除此之外，为了说明TANGRAM算法硬件实现的灵活性，我们同时提供了基于CTR模式的纯加密模块的轻量级实现和高速实现。本设计的所有实现代码请见TANGRAM的实现源代码文件夹。

我们采用了多种优化方式来实现TANGRAM的ECB模式。如图1（需要更新）所示，其中加解密算法共享多个基本模块单元，包括轮子密钥加、轮状态、密钥存储等。对于TANGRAM128/256版本，因为密钥扩展算法采用了Feistel网络，所以可以省去密钥逆向扩展模块。为了节约轮子密钥的存储空间，我们采取了在线生成轮密钥的方式。对解密操作，对于同一次ECB操作中的所有分组数据，我们只在第一组数据到来之后，正向运算出最末轮的轮密钥，并存储在额外的存储模块（Inv Key Storage）中，之后的解密操作通过在线执行密钥逆向扩展模块来在线生成。

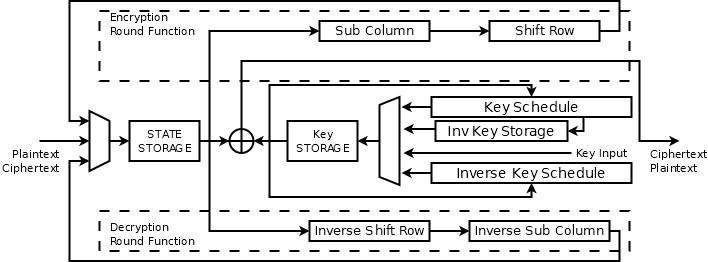


图1. TANGRAM的ECB模式的硬件结构

TANGRAM的ECB模式的硬件实现在UMC130nm和Nangate45nm下进行测试，结果如表1和表2所示。其中运算周期涵盖全部32组数据所需的系统时钟数（结果由Modelsim的功能仿真所验证）。我们可以看到解密比加密多花了一组加密操作的时钟周期，这是由于我们需要生成最后一轮的子密钥用于反推，但是这个操作只需要执行一次。关键路径的时长是通过DesignCompiler软件分析综合网表后，通过分析时钟约束，来选取最长延迟下（Maximum Delay）下的关键路径所得。加解密速度根据第二轮材料说明中给出的计算方法得到。表中的面积是Design Compiler 中所给出的Total area的面积。但是这个面积数据的单位是由工艺库给出，有时候是平方微米，有的时候是工艺库自定的面积单位。经过查验对应工艺库的PDK说明文档，本设计所采用的工艺库的面积单元为平方微米。等效门GE是通过比较实现面积和该工艺下NAND2门所需面积的比值来计算，在UMC130nm工艺库下，NAND2所需要的面积是4 um2，而Nangate 45nm下NAND2面积是0.798 um2。我们可以看到，不同工艺节点对整体面积的影响非常大，而等效门数基本上保持相似。

表1. TANGRAM的ECB模式在UMC 130nm 工艺库下的自测结果

|  |  |  |  |
| --- | --- | --- | --- |
|  | 在UMC 130nm 工艺库下ECB模式（全32组数据）自测结果 | | |
| 128/128 | 128/256 | 256/256 |
| 运算周期 | 加密：1440  解密：1485 | 加密：1632  解密：1683 | 加密：2656  解密：2739 |
| 时钟约束 | 6.83 ns | 7.73 ns | 9.03 ns |
| 加密速率 | 416Mbps | 324.7 Mbps | 341 Mbps |
| 解密速率 | 404 Mbps | 314.8Mbps | 331.2 Mbps |
| 面积 | 26607um2  (约为6652 GE) | 31656um2  (约为7914 GE) | 52605um2  (约为13152 GE) |
| 加密吞面比 | 0.01563 Mbps/ um2 | 0.01026Mbps/ um2 | 0.00648Mbps/ um2 |
| 解密吞面比 | 0.01518 Mbps/ um2 | 0.00994Mbps/ um2 | 0.00630Mbps/ um2 |

表2. TANGRAM的ECB模式在Nangate45nm 工艺库下的自测结果

|  |  |  |  |
| --- | --- | --- | --- |
|  | 在Nangate 45nm开源工艺库下ECB模式（全32组数据）自测结果 | | |
| 128/128 | 128/256 | 256/256 |
| 运算周期 | 加密：1440  解密：1485 | 加密：1632  解密：1683 | 加密：2656  解密：2739 |
| 时钟约束 | 1.26 ns | 1.51 ns | 1.74 ns |
| 加密速率 | 2257 Mbps | 1662 Mbps | 1773 Mbps |
| 解密速率 | 2189 Mbps | 1612 Mbps | 1719 Mbps |
| 面积 | 5025um2  (约为6297 GE) | 6249um2  (约为7831 GE) | 9975um2  (约为12500GE) |
| 加密吞面比 | 0.449 Mbps/ um2 | 0.266 Mbps/ um2 | 0.178 Mbps/ um2 |
| 解密吞面比 | 0.436 Mbps/ um2 | 0.258 Mbps/ um2 | 0.172 Mbps/ um2 |

表3给出了TANGRAM的ECB模式在主流FPGA上的实现结果。通过设定时钟约束，然后综合加后端的布局布线，对于三个版本的TANGRAM的ECB模式均能满足对应时钟约束的要求。表3同时给出了在对应时钟约束下，TANGRAM的ECB模式能够达到的加解密吞吐要求。

表3. TANGRAM的ECB模式在Xilinxxc7s100fgga676-1FPGA平台下的自测结果

|  |  |  |  |
| --- | --- | --- | --- |
|  | 在Xilinxxc7s100fgga676-1FPGA平台ECB模式的性能自测 | | |
| 128/128 | 128/256 | 256/256 |
| 所需查找表 LUTs | 863 | 854 | 1526 |
| 所需寄存器FFs | 397 | 653 | 783 |
| 所需BRAM和DSP | 0 | 0 | 0 |
| 时钟约束 | 5 ns | 4ns | 5 ns |
| 加密速率 | 586.89 Mbps | 627.45 Mbps | 924.08 Mbps |
| 解密速率 | 551.65 Mbps | 608 Mbps | 953.11 Mbps |

考虑到不同应用需求，我们以128/128版本的TANGRAM为例，针对不同的应用场景进行硬件实现的优化。实际应用中，为了节省面积和高速并行化处理，计数器模式CTR被广泛采用。采用CTR模式，硬件上只需要实现加密模块，而不需要考虑解密模块，能够有效的减少硬件开销。同时CTR模式可以多个分组块并行计算，非常有利于高速处理大数据量，这一特性使得利用GPU和硬件实现来进行加速变得更加有效。因此，接下来对于不同应用场景的分析，我们聚焦于TANGRAM的CTR模式实现。

考虑到IoT设备的广泛使用，低面积实现往往是设计者对于轻量级应用的首要考虑因素。我们这里实现了Round-based TANGRAM。因为上述说明，我们这里采用CTR模式，仅保留其加密部分。表4中给出了Round-based TANGRAM的实现结果，并与其它模式进行了对比。我们需要强调的是，由于TANGRAM的高度灵活性，可以采用类似RECTANGLE的优化实现中串行实现的做法来进一步优化所需面积，通过高度复用AddRoundKey模块和仅仅使用1个S盒来达到面积最优的串行实现。由于竞赛时间所限，TANGRAM的各个版本的串行实现会在将来给出。

另一个应用场景是服务器和数据中心。这一类平台所需要处理的数据量往往非常巨大。为了满足这一类应用场景，我们给出了完全展开的全流水的版本。这一版本中，我们将所有轮的运算都实例化，并在每轮之间加入寄存器来达到流水线实现的目的。因为大数据运行中，往往连续的数据都会采用同一主密钥，所以我们也同时并行化实现所有轮的密钥扩展算法，但是由于在一组运算的过程中，主密钥不会更新，所以我们没有在不同密钥扩展轮之间插入寄存器。需要注意的是，针对这一模式来计算关键路径的时候，我们只需要关注轮运算算法实现中的关键路径即可。表4给出了全流水版本TANGRAM的硬件性能。此处的最大系统频率和吞吐量是基于综合工具预测的，实际ASIC实现中，对于极高工作频率和吞吐来说，瓶颈往往不是关键路径，而是其它因素，如散热和噪音等。这里将数据列出只是为了表明对于极端应用场景下，得益于TANGRAM算法的高灵活性，TANGRAM往往不会是包含其的加密系统中的速度瓶颈。

这里我们希望说明，高吞吐量实现和低延迟实现的需求往往并不一致。高吞吐量实现要求单位时间内能够并行处理大的数据总量。低延迟实现对数据加密开始到结束之间的延迟要求较高，典型的应用有存储器加密。一般通过完全展开轮函数，并且在每一轮之间不插入寄存器这一方法来达到低延迟实现，这种实现也被叫做单一时钟周期实现（One-clock-cycle implementation）。

表4.多种应用场景下的TANGRAM128/128实现对比 (Nangate 45nm)

|  |  |  |  |
| --- | --- | --- | --- |
|  | ECB  Roundbased | CTR  Roundbased | CTR  全流水 |
| 面积 | 5025 um2  (约为6297 GE) | 2914.6um2  (约为3653 GE) | 81147um2  (约为101688GE) |
| 最大系统频率 | 793.6 MHz | 1.81 GHz | 1.72 GHz |
| 吞吐量 | 2.257 Gbps | 5.148 Gbps | 220.16 Gbps |
| 功耗 | 747uW | 632uW | 400860uW |
| 吞吐量/面积 | 0.449 Mbps/um2 | 1.766 Mbps/um2 | 2.713 Mbps/um2 |
| 每比特能耗 | 3.3097\*10-13 J/bit | 1.2277\*10-13 J/bit | 18.21\*10-13J/bit |

根据上面多个平台多个版本的TANGRAM硬件实现结果，我们认为TANGRAM算法是一个高度灵活的、非常硬件实现友好型的分组密码。得益于其规整和统一的轮函数和密钥扩展算法，一方面针对低面积的硬件实现，可以通过高度复用相关运算模块；另一方面，针对高吞吐量和低延迟的相关应用，可以轻松实例化所有轮函数并行化来实现。

# 五. TANGRAM的优缺点声明

TANGRAM的主要优点和创新点如下：

## 1．设计简洁、易于扩展

TANGRAM的整体结构为SP网络，设计非常简洁：对于TANGRAM-128，128比特的加密状态用一个的比特矩阵描述，S层对每一列的4个比特做S盒替换，P层对每一行的32个比特做左循环移位；对于TANGRAM-256，256比特的加密状态用一个的比特矩阵描述，S层对每一列的4个比特做S盒替换，P层对每一行的64个比特做左循环移位。

TANGRAM的3个版本基于同一框架和设计方法，具有很好的可扩展性。利用这种设计思路，对于任意正整数t，可以类似地设计出分组长度为4t比特的分组密码或者固定置换，以满足不同安全需求和应用场景，比如资源非常有限的轻量级分组密码、认证加密方案、密码杂凑函数等。

在TANGRAM的S盒变换之前和之后，分别异或一个4比特的常数，所得到的16\*16 = 256个S盒中，有64个S盒不存在不动点。将TANGRAM的S盒替换为这64个S盒中的任意一个，对其设计准则和安全强度均没有影响，对软硬件实现的影响也很小。

## 2．安全性分析深入

我们评估了TANGRAM抵抗差分、线性、积分、不可能差分以及相关密钥等分析方法的安全性。TANGRAM128/128的总轮数为44轮，根据评估结果，我们最多能攻击到28轮；6轮TANGRAM-128达到全扩散，因此我们在44轮的基础上又增加了6轮，将50轮作为TANGRAM128/256的总轮数；TANGRAM256/256的总轮数为82轮，我们估计最多能攻击到56轮。

我们认为：以当前分组密码分析方法的发展现状作为参考，我们为TANGRAM的3个版本预留的冗余都足够保证其抵抗数学类攻击的安全性。

## 3．硬件和软件实现均表现很好

TANGRAM的主要设计思想是采用比特切片方法来设计适合多个软硬件平台的分组密码。关于比特切片方法及其优势，详见第一部分的2.2小节。

我们在多个不同平台上实现了TANGRAM，结果表明TANGRAM的硬件和软件实现均表现很好，因此可以灵活地适用于多种应用场景。详见第四部分“TANGRAM的软硬件性能分析”。

## 4．防护侧信道代价低

得益于简洁的设计和比特切片方法，TANGRAM防护侧信道攻击的代价很低。以Threshold防护为例，TANGRAM的S盒是其中唯一复杂的部分，根据Begul博士等人发表于CHES’2012的论文，TANGRAM的S盒属于第266类，仅需要3个shares，并且在每一个share中，一阶防护仅需一对G和F函数；此外，TANGRAM基于比特切片方法设计，相比于查表实现，比特切片的软件实现能够抵抗时间缓存攻击。

## 5．设计准则公开透明

TANGRAM的主要设计思想是：采用比特切片方法设计适合多个软硬件平台的分组密码。TANGRAM采用了SP网络，S层是32（或64）个的S盒的并置，P层是一个比特置换。第二部分我们已经详细给出了TANGRAM的整体设计思想和各个模块的选取准则。可以说，TANGRAM的设计，没有晦涩之处，都是基于我们对其模块的密码性质以及整体密码算法的安全性的充分理解和把握的基础之上。

## 6．对模块的仔细选择

在设计TANGRAM的过程中，我们细致地挑选目标S盒，对它的两个密码指标做了特殊限制，以提升它抵抗差分/线性密码分析的性价比，详见第二部分中的TANGRAM的S盒设计以及我们的论文[19]；P置换的设计也非常重要，TANGRAM的P置换仅包含3个32（或64）比特字上的循环移位，无论在硬件还是软件上都可以容易、高效地实现，这3个循环移位参数需要仔细选择，以提升算法的整体安全强度，详见第二部分中的TANGRAM的扩散层设计；将TANGRAM的S盒和P置换组合在一起，所构成的整体密码算法具有很好的抵抗差分/线性密码分析的能力。上述因素综合在一起，TANGRAM才得以在安全性和实现性能之间达到了很好的平衡。

TANGRAM的局限性是：对于新设计的密码算法，需要对它进行透彻的安全性分析和评估，我们将继续此方面的工作。

参考文献：

[1] Serpent S-Boxes as boolean functions. URL http://www.gladman.me.uk/

[2] AES and combined encryption/authentication modes (2014). URL <https://github.com/BrianGladman/AES>

[3] Bao, Z., Zhang, W., Lin, D.: Information Security and Cryptology: 10th International Conference, Inscrypt 2014, Beijing, China, December 13-15, 2014, Revised Selected Papers, chap. Speeding Up the Search Algorithm for the Best Differential and Best Linear Trails, pp. 259--285. Springer International Publishing, Cham (2015). DOI 10.1007/978-3-319-16745-9\_15. URL <http://dx.doi.org/10.1007/978-3-319-16745-9_15>

[4] Biham, E.: Advances in Cryptology --- EUROCRYPT '93: Workshop on the Theory andApplication of Cryptographic Techniques Lofthus, Norway, May 23--27, 1993Proceedings, chap. New Types of Cryptanalytic Attacks Using Related Keys, pp.398--409. Springer Berlin Heidelberg, Berlin, Heidelberg (1994). DOI 10.1007/3-540-48285-7\_34. URL <http://dx.doi.org/10.1007/3-540-48285-7_34>

[5] Biham, E.: Fast Software Encryption: 4th International Workshop, FSE'97 Haifa,Israel, January 20--22 1997 Proceedings, chap. A fast new DESimplementation in software, pp. 260--272. Springer Berlin Heidelberg, Berlin, Heidelberg (1997). DOI 10.1007/BFb0052352. URL <http://dx.doi.org/10.1007/BFb0052352>

[6] Biham, E., Biryukov, A., Shamir, A.: Advances in Cryptology --- EUROCRYPT '99:International Conference on the Theory and Application of CryptographicTechniques Prague, Czech Republic, May 2--6, 1999 Proceedings, chap.Cryptanalysis of Skipjack} Reduced to 31 Rounds Using ImpossibleDifferentials, pp. 12--23. Springer Berlin Heidelberg, Berlin, Heidelberg (1999). DOI 10.1007/3-540-48910-X\_2. URL http://dx.doi.org/10.1007/3-540-48910-X\_2

[7] Biham, E., Shamir, A.: Differential cryptanalysis of DES-like cryptosystems. Journal of Cryptology 4(1), 3-72. DOI 10.1007/BF00630563. URL http://dx.doi.org/10.1007/BF00630563

[8] Biham, E., Shamir, A.: Differential cryptanalysis of the data encryptionstandard. Springer Science & Business Media (2012)

[9] Biryukov, A., Wagner, D.: Fast Software Encryption: 6th International Workshop,FSE'99 Rome, Italy, March 24--26, 1999 Proceedings, chap. Slide Attacks, pp.245--259. Springer Berlin Heidelberg, Berlin, Heidelberg (1999). DOI 10.1007/3-540-48519-8\_18. URL http://dx.doi.org/10.1007/3-540-48519-8\_18

[10] Biryukov, A., Wagner, D.: Advances in Cryptology --- EUROCRYPT 2000:International Conference on the Theory and Application of CryptographicTechniques Bruges, Belgium, May 14--18, 2000 Proceedings, chap. AdvancedSlide Attacks, pp. 589--606. Springer Berlin Heidelberg, Berlin, Heidelberg (2000). DOI 10.1007/3-540-45539-6\_41. URL <http://dx.doi.org/10.1007/3-540-45539-6_41>

[11] Daemen, J., Knudsen, L., Rijmen, V.: Fast Software Encryption: 4thInternational Workshop, FSE'97 Haifa, Israel, January 20--22 1997Proceedings, chap. The block cipher Square, pp. 149--165. Springer Berlin Heidelberg, Berlin, Heidelberg (1997). DOI 10.1007/BFb0052343. URL http://dx.doi.org/10.1007/BFb0052343

[12] Daemen, J., Rijmen, V.: The design of Rijndael: AES-the advanced encryptionstandard. Springer Science & Business Media (2013)

[13] Knudsen, L., Wagner, D.: Fast Software Encryption: 9th International Workshop,FSE 2002 Leuven, Belgium, February 4--6, 2002 Revised Papers, chap. IntegralCryptanalysis, pp. 112--127. Springer Berlin Heidelberg, Berlin, Heidelberg (2002). DOI 10.1007/3-540-45661-9\_9. URL http://dx.doi.org/10.1007/3-540-45661-9\_9

[14] Leander, G., Poschmann, A.: Arithmetic of Finite Fields: First InternationalWorkshop, WAIFI 2007, Madrid, Spain, June 21-22, 2007. Proceedings, chap. Onthe Classification of 4 Bit S-Boxes, pp. 159--176. Springer Berlin Heidelberg, Berlin, Heidelberg (2007). DOI 10.1007/978-3-540-73074-3\_13. URL <http://dx.doi.org/10.1007/978-3-540-73074-3_13>

[15] Matsui, M.: Advances in Cryptology --- CRYPTO '94: 14th Annual InternationalCryptology Conference Santa Barbara, California, USA August 21--25, 1994Proceedings, chap. The First Experimental Cryptanalysis of the DataEncryption Standard, pp. 1--11. Springer Berlin Heidelberg, Berlin, Heidelberg (1994). DOI 10.1007/3-540-48658-5\_1. URL http://dx.doi.org/10.1007/3-540-48658-5\_1

[16] Matsui, M.: Advances in Cryptology --- EUROCRYPT'93: Workshop on the Theory andApplication of Cryptographic Techniques Lofthus, Norway, May 23--27, 1993Proceedings, chap. Linear Cryptanalysis Method for DES Cipher, pp.386--397. Springer Berlin Heidelberg, Berlin, Heidelberg (1994). DOI 10.1007/3-540-48285-7\_33. URL http://d

[17]Todo, Y. (2015, April). Structural evaluation by generalized integral property. In Annual International Conference on the Theory and Applications of Cryptographic Techniques (pp. 287-314). Springer, Berlin, Heidelberg.

[18] Xiang, Z., Zhang, W., Bao, Z., & Lin, D. (2016, December). Applying MILP method to searching integral distinguishers based on division property for 6 lightweight block ciphers. In International Conference on the Theory and Application of Cryptology and Information Security (pp. 648-678). Springer, Berlin, Heidelberg.

[19] Zhang, W., Bao, Z., Rijmen, V., Liu, M.: Fast Software Encryption: 22nd International Workshop, FSE 2015, Istanbul, Turkey, March 8-11, 2015, RevisedSelected Papers, chap. A New Classification of 4-bit Optimal S-Boxes andIts Application to PRESENT, RECTANGLE and SPONGENT, pp. 494--515. Springer Berlin Heidelberg, Berlin, Heidelberg (2015). DOI 10.1007/978-3-662-48116-5\_24. URL http://dx.doi.org/10.1007/978-3-662-48116-5\_24