TANGRAM 算法文本

本算法文本包含五个部分，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

1. 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4×32的矩形比特阵列描述，256 比特的加密状态用一个 4×64 的矩形比特阵列描述。

2. 术语

2.1 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。

2.2 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 4×4 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 $i\,(i=0,1,2,3)$ 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。

3. 符号和缩略语

符号

m ：一个明文分组

c ：一个密文分组

R_i ：第 i 轮的轮函数

K_i ：第 i 轮的轮子密钥

$RC[i]$ ：第 i 轮的轮常数

$a \| b$ ：比特串 a 和 b 的串联

$<<< x$ ：左循环移动 x 位

$>>> x$ ：右循环移动 x 位

$x \oplus y$ ：将 x 和 y 按位模 2 加（或称异或）

$x \& y$ ：将 x 和 y 按位与

$x \mid y$ ：将 x 和 y 按位或

$\sim x$ ：将 x 按位取反

F_2 ：:二元有限域，即元素为 $\{0,1\}$

F_2^n ： n -维向量集合，向量元素在 F_2 上

$x \bullet y$ ： x 和 y 的内积

$wt(x)$ ： x 的汉明重量

$\#Set$ ：集合 Set 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)

4. 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 R_i 表示在轮子密钥 K_i 作用下的轮变换，m 、c 分别表示明文和密文，TANGRAM 128/128 的算法结构如图 4.1 所示。

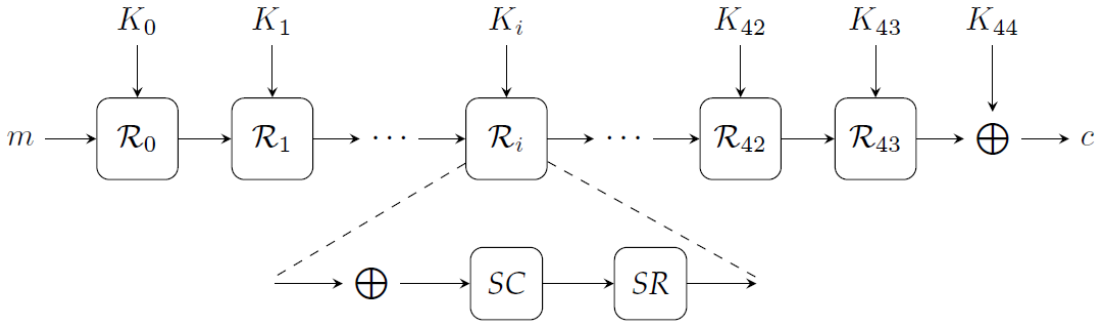


图4.1　　TANGRAM 128/128 的整体加密过程

5. TANGRAM 轮函数

5.1 轮函数结构

5.1.1 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 $W = w_{127} \| \cdots \| w_1 \| w_0$ 表示一个密码状态，我们用一个 4×32 的矩形比特阵列表示 W，如图 5.1 (a)所示，将前 32 个比特 $w_{31} \| \cdots \| w_1 \| w_0$ 放在第 0 行，接下来的 32 个比特 $w_{63} \| \cdots \| w_{33} \| w_{32}$ 放在第 1 行，依次类推。类似地，128 比特的轮子密钥也用一个 4×32 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。

$$\begin{bmatrix} w_{31} & w_{30} & \cdots & w_2 & w_1 & w_0 \\ w_{63} & w_{62} & \cdots & w_{34} & w_{33} & w_{32} \\ w_{95} & w_{94} & \cdots & w_{66} & w_{65} & w_{64} \\ w_{127} & w_{126} & \cdots & w_{98} & w_{97} & w_{96} \end{bmatrix}$$

(a)密码状态

$$\begin{bmatrix} w_{0,31} & w_{0,30} & \cdots & w_{0,2} & w_{0,1} & w_{0,0} \\ w_{1,31} & w_{1,30} & \cdots & w_{1,2} & w_{1,1} & w_{1,0} \\ w_{2,31} & w_{2,30} & \cdots & w_{2,2} & w_{2,1} & w_{2,0} \\ w_{3,31} & w_{3,30} & \cdots & w_{3,2} & w_{3,1} & w_{3,0} \end{bmatrix}$$

(b)密码状态的二维表示

图5.1　　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。

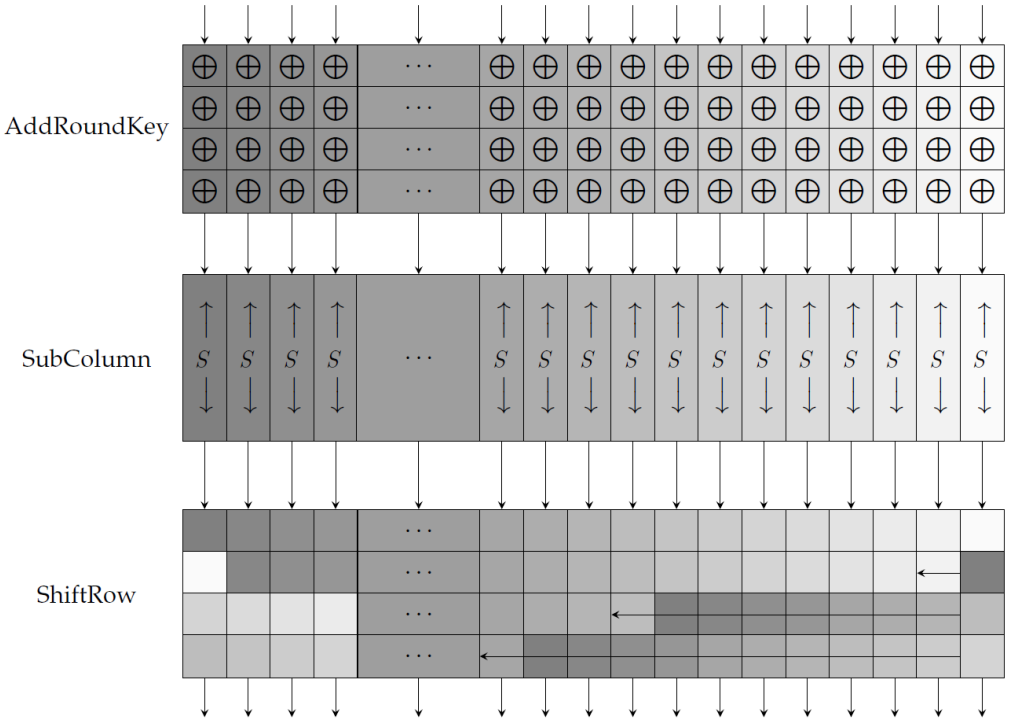


图5.2　　TANGRAM-128 的轮函数示意图

5.1.2 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 $W = w_{255} \| \cdots \| w_1 \| w_0$ 表示一个密码状态，我们用一个 4×64 的矩形比特阵列表示 W，如图 5.3 (a)所示，将前 64 个比特 $w_{63} \| \cdots \| w_1 \| w_0$ 放在第 0 行，接下来的 64 个比特 $w_{127} \| \cdots \| w_{65} \| w_{64}$ 放在第 1 行，依次类推。类似地，256 比特的轮子密钥也用一个 4×64 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。

$$\begin{bmatrix} w_{63} & w_{62} & \cdots & w_2 & w_1 & w_0 \\ w_{127} & w_{126} & \cdots & w_{66} & w_{65} & w_{64} \\ w_{191} & w_{190} & \cdots & w_{130} & w_{129} & w_{128} \\ w_{255} & w_{254} & \cdots & w_{194} & w_{193} & w_{192} \end{bmatrix}$$

(a)密码状态

$$\begin{bmatrix} w_{0,63} & w_{0,62} & \cdots & w_{0,2} & w_{0,1} & w_{0,0} \\ w_{1,63} & w_{1,62} & \cdots & w_{1,2} & w_{1,1} & w_{1,0} \\ w_{2,63} & w_{2,62} & \cdots & w_{2,2} & w_{2,1} & w_{2,0} \\ w_{3,63} & w_{3,62} & \cdots & w_{3,2} & w_{3,1} & w_{3,0} \end{bmatrix}$$

(b)密码状态的二维表示

图5.3　　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。

5.2 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 $Col(j)=a_{3,j}\|a_{2,j}\|a_{1,j}\|a_{0,j}(0\le j\le 31)$ 表示 S 盒的输入，$S(Col(j))=b_{3,j}\|b_{2,j}\|b_{1,j}\|b_{0,j}$ 表示 S 盒的输出。

$$\begin{pmatrix} a_{0,31}\\ a_{1,31}\\ a_{2,31}\\ a_{3,31}\end{pmatrix} \cdots \begin{pmatrix} a_{0,1}\\ a_{1,1}\\ a_{2,1}\\ a_{3,1}\end{pmatrix} \begin{pmatrix} a_{0,0}\\ a_{1,0}\\ a_{2,0}\\ a_{3,0}\end{pmatrix}$$
$$\downarrow S \quad \cdots \quad \downarrow S \quad \downarrow S$$
$$\begin{pmatrix} b_{0,31}\\ b_{1,31}\\ b_{2,31}\\ b_{3,31}\end{pmatrix} \cdots \begin{pmatrix} b_{0,1}\\ b_{1,1}\\ b_{2,1}\\ b_{3,1}\end{pmatrix} \begin{pmatrix} b_{0,0}\\ b_{1,0}\\ b_{2,0}\\ b_{3,0}\end{pmatrix}$$

图5.4　　列替换 SubColumn

TANGRAM 的 S 盒是一个 4 比特到 4 比特的双射 S： $F_2^4\to F_2^4$，S 的描述如表 5.1 所示（十六进制）。

表5.1　　RECTANGLE 的 S 盒真值表

x	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
$S(x)$	8	0	3	E	7	D	C	2	6	F	5	9	A	1	B	4

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 $<<<x$ 表示左循环移动 x 位。

$$\begin{aligned}
(\ a_{0,31}\ \cdots\ a_{0,1}\ \ a_{0,0}\ ) &\xrightarrow{\lll 0} (\ a_{0,31}\ \cdots\ a_{0,1}\ \ a_{0,0}\ )\\
(\ a_{1,31}\ \cdots\ a_{1,1}\ \ a_{1,0}\ ) &\xrightarrow{\lll 1} (\ a_{1,30}\ \cdots\ a_{1,0}\ \ a_{1,31}\ )\\
(\ a_{2,31}\ \cdots\ a_{2,1}\ \ a_{2,0}\ ) &\xrightarrow{\lll 8} (\ a_{2,23}\ \cdots\ a_{2,25}\ \ a_{2,24}\ )\\
(\ a_{3,31}\ \cdots\ a_{3,1}\ \ a_{3,0}\ ) &\xrightarrow{\lll 11} (\ a_{3,20}\ \cdots\ a_{3,22}\ \ a_{3,21}\ )
\end{aligned}$$

图5.5　　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 $<<<x$ 表示左循环移动 x 位。

$$\begin{array}{lll}
\left(\begin{array}{cccc} a_{0,63} & \cdots & a_{0,1} & a_{0,0} \end{array}\right) & \xrightarrow{\lll 0} & \left(\begin{array}{cccc} a_{0,63} & \cdots & a_{0,1} & a_{0,0} \end{array}\right) \\
\left(\begin{array}{cccc} a_{1,63} & \cdots & a_{1,1} & a_{1,0} \end{array}\right) & \xrightarrow{\lll 1} & \left(\begin{array}{cccc} a_{1,62} & \cdots & a_{1,0} & a_{1,63} \end{array}\right) \\
\left(\begin{array}{cccc} a_{2,63} & \cdots & a_{2,1} & a_{2,0} \end{array}\right) & \xrightarrow{\lll 8} & \left(\begin{array}{cccc} a_{2,55} & \cdots & a_{2,57} & a_{2,56} \end{array}\right) \\
\left(\begin{array}{cccc} a_{3,63} & \cdots & a_{3,1} & a_{3,0} \end{array}\right) & \xrightarrow{\lll 41} & \left(\begin{array}{cccc} a_{3,22} & \cdots & a_{3,24} & a_{3,23} \end{array}\right)
\end{array}$$

图5.6　　TANGRAM-256 行移位 ShiftRow_256

6. TANGRAM 算法描述

6.1 密钥扩展算法

6.1.1 TANGRAM 128/128 密钥扩展算法

将 128 比特的种子密钥 $V = v_{127} \| \cdots \| v_1 \| v_0$ 用一个 4×32 的矩形比特阵列表示，如图 6.1 所示。

$$\begin{bmatrix} v_{31} & \cdots & v_2 & v_1 & v_0 \\ v_{63} & \cdots & v_{34} & v_{33} & v_{32} \\ v_{95} & \cdots & v_{66} & v_{65} & v_{64} \\ v_{127} & \cdots & v_{98} & v_{97} & v_{96} \end{bmatrix} \begin{bmatrix} k_{0,31} & \cdots & k_{0,2} & k_{0,1} & k_{0,0} \\ k_{1,31} & \cdots & k_{1,2} & k_{1,1} & k_{1,0} \\ k_{2,31} & \cdots & k_{2,2} & k_{2,1} & k_{2,0} \\ k_{3,31} & \cdots & k_{3,2} & k_{3,1} & k_{3,0} \end{bmatrix}$$

图6.1　　TANGRAM 128/128 的 128 比特密钥状态及其二维表示

令 $Row_i = k_{i,31} \| \cdots \| k_{i,1} \| k_{i,0}$ 表示第 i 行，$0 \le i \le 3$，Row_i 可以看作一个 32 比特的字。在第 $r(r = 0,1,\cdots,43)$ 轮，先提取 128 比特的子密钥，轮子密钥 $K_r = Row_3 \| Row_2 \| Row_1 \| Row_0$；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：

$$k'_{3,j} \| k'_{2,j} \| k'_{1,j} \| k'_{0,j} := S(k_{3,j} \| k_{2,j} \| k_{1,j} \| k_{0,j}),\quad j = 0,1,\cdots,31$$

2. 进行一轮的 4 分支广义 Feistel 变换，如下：

$$\begin{aligned} Row'_0 &:= (Row_0 <<< 7) \oplus Row_1 \\ Row'_1 &:= Row_2 \\ Row'_2 &:= (Row_2 <<< 17) \oplus Row_3 \\ Row'_3 &:= Row_0 \end{aligned}$$

3. 对密钥状态的第一行的 6 个比特 $(k_{0,5} \| k_{0,4} \| k_{0,3} \| k_{0,2} \| k_{0,1} \| k_{0,0})$，异或 6 比特的轮

常数 $RC[r](r=0,1,\cdots,43)$。

将上面三个步骤的复合变换记为 $\text{Update}_{RC[r]}[X]$，其中 X 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 K_{44}。

轮常数 $RC[r](r=0,1,\cdots,43)$ 见表 6.1。

6.1.2 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，$K=K_L\|K_R$，其中 K_L 和 K_R 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：

$K_0=K_L$，

$K_1=K_R$，

$K_{r+2}=K_r\oplus\text{Update}_{RC[r]}(K_{r+1}),\qquad r=0,1,\cdots,48$。

轮常数 $RC[r](r=0,1,\cdots,49)$ 由一个 6 比特的线性反馈移位寄存器生成。用 $(rs_5,rs_4,rs_3,rs_2,rs_1,rs_0)$ 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 rs_0 更新为 $rs_5\oplus rs_4$，初始值 $RC[0]:=0x1$。

表6.1　　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			

表 6.1 列出了 TANGRAM 128/128 和 TANGRAM 128/256 所用到的 49 个轮常数。

6.1.3 TANGRAM 256/256 密钥扩展算法

将 256 比特的种子密钥 $V=v_{255}\|\cdots\|v_1\|v_0$ 用一个 4×64 的矩形比特阵列表示，如图 6.3 所示。

$$\begin{bmatrix} v_{63} & \cdots & v_2 & v_1 & v_0 \\ v_{127} & \cdots & v_{66} & v_{65} & v_{64} \\ v_{191} & \cdots & v_{130} & v_{129} & v_{128} \\ v_{255} & \cdots & v_{194} & v_{193} & v_{192} \end{bmatrix} \begin{bmatrix} k_{0,63} & \cdots & k_{0,2} & k_{0,1} & k_{0,0} \\ k_{1,63} & \cdots & k_{1,2} & k_{1,1} & k_{1,0} \\ k_{2,63} & \cdots & k_{2,2} & k_{2,1} & k_{2,0} \\ k_{3,63} & \cdots & k_{3,2} & k_{3,1} & k_{3,0} \end{bmatrix}$$

图6.2　　TANGRAM256/256 的 256 比特密钥状态及其二维表示

令 $Row_i = k_{i,63} \| \cdots \| k_{i,1} \| k_{i,0}$ 表示第 i 行，$0 \le i \le 3$，Row_i 可以看作一个 64 比特的字。在第 $r(r = 0,1,\cdots,81)$ 轮，256 比特的轮子密钥 $K_r = Row_3 \| Row_2 \| Row_1 \| Row_0$。提取出轮子密钥 K_r 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：

$$k'_{3,j} \| k'_{2,j} \| k'_{1,j} \| k'_{0,j} := S(k_{3,j} \| k_{2,j} \| k_{1,j} \| k_{0,j}), \quad j = 0,1,\cdots,63$$

2．进行一轮的 4 分支广义 Feistel 变换，如下：

$$Row'_0 := (Row_0 <<< 7) \oplus Row_1$$
$$Row'_1 := Row_2$$
$$Row'_2 := (Row_2 <<< 17) \oplus Row_3$$
$$Row'_3 := Row_0$$

3．对密钥状态的第一行的 7 个比特 $(k_{0,6} \| k_{0,5} \| k_{0,4} \| k_{0,3} \| k_{0,2} \| k_{0,1} \| k_{0,0})$，异或 7 比特的轮常数 $RC[r](r = 0,1,\cdots,81)$。

最后，将更新的密钥状态的值赋给 K_{82}。

轮常数 $RC[r](r = 0,1,\cdots,81)$ 由一个 7 比特的线性反馈移位寄存器生成。用 $(rs_6, rs_5, rs_4, rs_3, rs_2, rs_1, rs_0)$ 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 rs_0 更新为 $rs_6 \oplus rs_5$，初始值为 $RC[0] := 0x1$。

表 6.2 列出了所用到的 82 个轮常数。

表6.2　　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									

6.2 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：

算法 1 TANGRAM 128/128 的加密算法

```
GenerateRoundKeys
for i = 0 to 43 do
    AddRoundKey (STATE, K_i)
    SubColumn (STATE)
    ShiftRow (STATE)
end for
AddRoundKey (STATE, K_44)
```

6.2.2TANGRAM128/256 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：

算法 2 TANGRAM 128/256 的加密算法

```
GenerateRoundKeys
for i = 0 to 49 do
    AddRoundKey (STATE, K_i)
    SubColumn (STATE)
    ShiftRow (STATE)
end for
AddRoundKey (STATE, K_50)
```

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：

算法 3 TANGRAM 256/256 的加密算法

1: GenerateRoundKeys
2: **for** $i = 0$ **to** 81 **do**
3: AddRoundKey $(STATE, K_i)$
4: SubColumn $(STATE)$
5: ShiftRow $(STATE)$
6: **end for**
7: AddRoundKey $(STATE, K_{82})$

6.3 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 6.2 所示。

表6.1 TANGRAM 的 S 盒真值表

x	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
$S^{-1}(x)$	1	D	7	2	F	A	8	4	0	B	C	E	6	5	3	9

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：

算法 4 TANGRAM 128/128 的解密算法

1: GenerateRoundKeys
2: **for** $i = 44$ **to** 1 **do**
3: AddRoundKey $(STATE, K_i)$
4: InverseShiftRow $(STATE)$
5: InverseSubColumn $(STATE)$
6: **end for**
7: AddRoundKey $(STATE, K_0)$

6.3.2 TANGRAM 128/256 解密算法

TANGRAM 128/256 的解密算法如算法 5 所示：

算法 5 TANGRAM 128/256的解密算法

1: GenerateRoundKeys
2: **for** $i = 50$ **to** 1 **do**
3: AddRoundKey $(STATE, K_i)$
4: InverseShiftRow $(STATE)$
5: InverseSubColumn $(STATE)$
6: **end for**
7: AddRoundKey $(STATE, K_0)$

6.3.3 TANGRAM 256/256 解密算法

TANGRAM 256/256 的解密算法如算法 6 所示：

算法 6 TANGRAM 256/256 的解密算法

1: GenerateRoundKeys
2: **for** $i = 82$ **to** 1 **do**
3: AddRoundKey $(STATE, K_i)$
4: InverseShiftRow $(STATE)$
5: InverseSubColumn $(STATE)$
6: **end for**
7: AddRoundKey $(STATE, K_0)$

7. RECTANGLE 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。

7.1 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 W，将它表示为 4 个 32 比特子块的连接，即 $W = A_3 \| A_2 \| A_1 \| A_0$，其中，$A_i$ 是 W 的第 i 行，$A_i = a_{i,31} \| \cdots \| a_{i,1} \| a_{i,0}$，$i = 0,1,2,3$。

TANGRAM-256 对于图 5.2 (b)所示的一个密码状态 W，将它表示为 4 个 64 比特子块的连接，即 $W = A_3 \| A_2 \| A_1 \| A_0$，其中，$A_i$ 是 W 的第 i 行，$A_i = a_{i,31} \| \cdots \| a_{i,1} \| a_{i,0}$，$i = 0,1,2,3$。

7.2 加密轮函数

轮密钥加 AddRoundKey: 设轮函数的输入为 $I_3 \| I_2 \| I_1 \| I_0$，轮子密钥为 $SK_3 \| SK_2 \| SK_1 \| SK_0$，两者异或后的结果记为 $A_3 \| A_2 \| A_1 \| A_0$，则有：

$$
\begin{aligned}
A_0 &:= I_0 \oplus SK_0, \\
A_1 &:= I_1 \oplus SK_1, \\
A_2 &:= I_2 \oplus SK_2, \\
A_3 &:= I_3 \oplus SK_3.
\end{aligned}
$$

列替换 SubColumn:用 $A_3 \| A_2 \| A_1 \| A_0$ 和 $B_3 \| B_2 \| B_1 \| B_0$ 分别表示 SubColumn 的输入和输出，则有：

$$
\begin{aligned}
T_1 &:= A_0 \oplus A_2, \\
T_2 &:= A_0 \,\&\, A_1, \\
T_3 &:= A_3 \oplus T_2, \\
B_2 &:= A_2 \oplus T_3, \\
T_5 &:= A_1 \oplus A_2, \\
T_6 &:= T_1 \,\&\, T_3, \\
B_0 &:= T_5 \oplus T_6, \\
T_8 &:= A_1 \mid A_3, \\
T_9 &:= T_1 \oplus T_8, \\
B_3 &:= \sim T_9, \\
T_{11} &:= T_5 \,\&\, T_9, \\
B_1 &:= T_3 \oplus T_{11}.
\end{aligned}
$$

其中，"&，|，~，⊕"分别表示按位与、或、非和异或。T_i 是 32 比特字(对应 TANGRAM-128)

或 64 比特字(对应 TANGRAM-256)的临时变量，$i = 1,2,3,5,6,8,9,11$。

行移位 ShiftRow：用 $B_3 \| B_2 \| B_1 \| B_0$ 和 $C_3 \| C_2 \| C_1 \| C_0$ 分别表示 ShiftRow 的输入和输出，则有：

(TANGRAM-128)

$$C_1 \coloneqq B_1 <<< 1,$$
$$C_2 \coloneqq B_2 <<< 8,$$
$$C_3 \coloneqq B_3 <<< 11.$$

(TANGRAM-256)

$$C_1 \coloneqq B_1 <<< 1,$$
$$C_2 \coloneqq B_2 <<< 8,$$
$$C_3 \coloneqq B_3 <<< 41.$$

其中，$A <<< x$ 表示对 32 比特字(对应 TANGRAM-128)或 64 比特字(对应 TANGRAM-256) A 左循环移动 x 位。

7.3 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 $B_3 \| B_2 \| B_1 \| B_0$ 和 $A_3 \| A_2 \| A_1 \| A_0$ 分别表示 InverseSubColumn 的输入和输出，则有：

$$T_1 \coloneqq \sim A_3,$$
$$T_2 \coloneqq A_0 \,\&\, T_1,$$
$$T_3 \coloneqq A_1 \oplus T_2,$$
$$B_2 \coloneqq A_2 \oplus T_3,$$
$$T_5 \coloneqq A_0 \oplus A_2,$$
$$T_6 \coloneqq T_1 \,\&\, T_3,$$
$$B_1 \coloneqq T_5 \oplus T_6,$$
$$T_8 \coloneqq A_0 \oplus T_1,$$
$$T_9 \coloneqq T_3 \,\&\, T_5,$$
$$B_0 \coloneqq T_8 \oplus T_9,$$
$$T_{11} \coloneqq B_1 \,\&\, B_0,$$
$$B_3 \coloneqq T_3 \oplus T_{11}.$$

其中，T_i 是 32 比特字(对应 TANGRAM-128)或 64 比特字(对应 TANGRAM-256)的临时变量，$i = 1,2,3,5,6,8,9,11$。

行移位的逆变换 InverseShiftRow：用 $C_3 \| C_2 \| C_1 \| C_0$ 和 $B_3 \| B_2 \| B_1 \| B_0$ 分别表示 InverseShiftRow 的输入和输出，则有：

(TANGRAM-128)

$$B_1 \coloneqq C_1 >>> 1,$$
$$B_2 \coloneqq C_2 >>> 8,$$
$$B_3 \coloneqq C_3 >>> 11.$$

(TANGRAM-256)

$$B_1 := C_1 >>> 1,$$
$$B_2 := C_2 >>> 8,$$
$$B_3 := C_3 >>> 41.$$

其中，$A >>> x$ 表示对 32 比特字(对应 TANGRAM-128)或 64 比特字(对应 TANGRAM-256) A 右循环移动 x 位。

二．TANGRAM 设计原理

1. 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 n 、密钥长度为 k 的分组密码，如果不存在一种攻击方法有比 2^n 少得多的数据复杂度并且比 2^k 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 R ，如果最多可以攻击到 $R-m$ 轮的缩减轮密码，就称该分组密码具有 m 轮的绝对安全冗余，或者具有 m/n 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。

2. 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 4×4 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 4×32 或 4×64 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。

3. TANGRAM 的 S 盒设计

3.1 相关定义和定理

定义 3.1 令 S 表示一个 4×4 的 S 盒，对任意 $\Delta I,\Delta O\in F_2^4$，定义 $ND_S(\Delta I,\Delta O)$ 为：

$$ND_S(\Delta I,\Delta O):=\#\left\{x\in F_2^4 \mid S(x)\oplus S(x\oplus\Delta I)=\Delta O\right\}.$$

定义 3.2 (差分均匀度) 令 S 表示一个 4×4 的 S 盒，定义 S 的差分均匀度 $Diff(S)$ 为：

$$Diff(S):=\max_{\Delta I\neq0,\Delta O} ND_S(\Delta I,\Delta O).$$

定义 3.3 令 S 表示一个 4×4 的 S 盒，对任意 $\Gamma I,\Gamma O\in F_2^4$，定义 $\mathrm{Im}b_s(\Gamma I,\Gamma O)$ 为：

$$\mathrm{Im}b_s(\Gamma I,\Gamma O):=\left|\#\left\{x\in F_2^4 \mid \Gamma I\cdot x=\Gamma O\cdot S(x)\right\}-8\right|$$

其中“$\cdot$”表示 F_2^4 上的内积。

定义 3.4 (线性度) 令 S 表示一个 4×4 的 S 盒，定义 S 的线性度 $Lin(S)$ 为：

$$Lin(S):=\max_{\Gamma I,\Gamma O\neq0} 2\,\mathrm{Im}b_s(\Gamma I,\Gamma O)$$

对于任意 4×4 的双射 S 盒，均有 $Diff(S)\geq4$ 并且 $Lin(S)\geq8$，使得这两个值都达到最小值的 S 盒被称为最优 S 盒。

定义 3.5 (最优 S 盒[14]) 令 S 表示一个 4×4 的 S 盒，称 S 是最优 S 盒，若它满足以下三个条件：

1. S 是双射的；
2. $Diff(S)=4$；
3. $Lin(S)=8$.

定义 3.6 ($SetD1_S$ 和 $CarD1_S$) 令 $\Delta I\in F_2^4$ 表示一个非零输入差分，$\Delta O\in F_2^4$ 表示一个非零输出差分。令 $wt(x)$ 表示 x 的比特汉明重量。定义 $SetD1_S$ 为：

$$SetD1_S:=\left\{(\Delta I,\Delta O)\in F_2^4\times F_2^4 \mid wt(\Delta I)=wt(\Delta O)=1 \text{ and } ND_S(\Delta I,\Delta O)\neq0\right\}.$$

令 $CarD1_S$ 表示集合 $SetD1_S$ 的势。

定义 3.7 ($SetL1_S$ 和 $CarL1_S$)令 $\Gamma I \in F_2^4$ 表示一个非零输入选取模式（也称为输入掩码），$\Gamma O \in F_2^4$ 表示一个非零输出选取模式（也称为输出掩码）。定义 $SetL1_S$ 为：

$$SetL1_S := \left\{(\Gamma I, \Gamma O) \in F_2^4 \times F_2^4 \mid wt(\Gamma I) = wt(\Gamma O) = 1 \text{ and } \mathrm{Im}\, b_s(\Gamma I, \Gamma O) \neq 0\right\}.$$

令 $CarL1_S$ 表示集合 $SetL1_S$ 的势。

定义 3.8 (仿射等价[14])两个 4×4 的 S 盒 S 和 S' 被称为仿射等价，如果存在双射的线性映射 A， B 以及常数 $a,b \in F_2^4$，满足

$$S'(x) = B(S(A(x)+a))+b.$$

若两个 S 盒 S 和 S' 仿射等价，则有 $Diff(S) = Diff(S')$ 和 $Lin(S) = Lin(S')$ 。也就是说，S 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 S 和 S' 是两个仿射等价的 S 盒。若 S 是最优 S 盒，则 S' 也是最优 S 盒。

定义 3.9 (置换异或等价 permutation-then-XOR equivalent[14])两个 4×4 的 S 盒 S 和 S' 被称为置换异或等价，如果存在 F_2 上的 4×4 置换矩阵 P_0， P_1 以及常量 $a,b \in F_2^4$，满足

$$S'(x) = P_1(S(P_0(x)+a))+b.$$

我们简称这种等价为 PE 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 S 表示一个 4×4 的 S 盒。对于任意的 $b \in F_2^4$，如下定义 S 的分量布尔函数 S_b：

$$S_b : F_2^4 \to F_2, \tag{3.1}$$

$$S_b(x) := x \mapsto b \cdot S(x) \tag{3.2}$$

3.2 S 盒的选取

4×4 的 S 盒通常比 8×8 的 S 盒的硬件实现面积要小很多，比如，一个 4×4 的 S 盒通常只需 20 ~ 40 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. S 是双射的，即对任意 $x \neq x'$，有 $S(x) \neq S(x')$；
2. 对任意非零输入差分 $\Delta I \in F_2^4$ 和非零输出差分 $\Delta O \in F_2^4$， $ND_S(\Delta I, \Delta O) \leq 4$；
3. $CarD1_S = 2$；
4. 对任意非零输入选取模式 $\Gamma I \in F_2^4$ 和非零输出选取模式 $\Gamma O \in F_2^4$，$\mathrm{Im}\, b_s(\Gamma I, \Gamma O) \leq 4$；
5. $CarL1_S = 2$；
6. S 没有不动点，即对任意 $x \in F_2^4$，有 $S(x) \neq x$。

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 $1\sim5$，那么，任意一个与之 PE 等价 (见定义 3.9)的 S 盒也满足准则 $1\sim5$。

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 $1\sim5$ 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 0 的 S 盒的输出，第二个数代表输入为 1 的 S 盒的输出，依次类推。

表3.1　　满足准则 $1\sim5$ 的 4 个 PE 类的代表元

PE_0	6, 0, 8, 15, 12, 3, 7, 13, 11, 14, 1, 4, 5, 9, 10, 2
PE_1	3, 2, 8, 13, 15, 5, 6, 10, 9, 14, 4, 7, 0, 12, 11, 1
PE_2	6, 8, 15, 4, 12, 7, 9, 3, 11, 1, 0, 14, 5, 10, 2, 13
PE_3	8, 1, 6, 12, 5, 15, 10, 3, 7, 11, 13, 2, 0, 14, 9, 4

如果一个 S 盒满足准则 $1\sim5$，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 $1\sim5$，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 $4\times4!\times4!=2304$ 个 S 盒，用 $\{S_i : i = 0,1,\cdots,2303\}$ 来表示这 2304 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 $i(i = 0,1,2,3)$ 个输出比特是第二轮中某个 S 盒的第 i 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 1776 个 S 盒被删除，只剩下 528 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 528 个 S 盒做进一步筛选。

算法 1 对 2304 个候选 S 盒的初步筛选

INPUT: The set of 2304 S-boxes $\mathcal{S}=\{S_i\}$, $i=0,1,\cdots,2303$.
OUTPUT: Discard a part of the S-boxes which can result in a differential (or linear) trial with a single active S-box in each round.
for $i=0$ **to** 2303 **do**
    Let $(\Delta I_j, \Delta O_j)$ denote the two pairs which belong to the set $SetD1_{S_i}$, $j=1,2$.
    if $(\Delta I_1=\Delta O_1)$ or $(\Delta I_2=\Delta O_2)$ or $(\Delta I_2=\Delta O_1$ *and* $\Delta I_1=\Delta O_2)$ **then**
        Discard the S-box S_i from $\mathcal{S}$ and Continue.
    end if
    Let $(\Gamma I_j, \Gamma O_j)$ denote the two peirs which belong to the set $SetL1_{S_i}$, $j=1,2$.
    if $(\Gamma I_1=\Gamma O_1)$ or $(\Gamma I_2=\Gamma O_2)$ or $(\Gamma I_2=\Gamma O_1$ *and* $\Gamma I_1=\Gamma O_2)$ **then**
        Discard the S-box S_i from $\mathcal{S}$.
    end if
end for
Output $\mathcal{S}$

令 $\Pr ob_{D25}$ 表示最优 25 轮差分迹的概率，Cor^2_{L25} 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 528 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：$\Pr ob_{D25}<2^{-128}$ 和 $Cor^2_{L25}<2^{-128}$。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 $\Pr ob_{D15}$ 和 Cor^2_{L15} 的值，这 36 个 S 盒被划分成 2 组，表 3.2 给出了 $\Pr ob_{D25}$ 和 Cor^2_{L25} 的值以及对应 S 盒的个数。

表3.2　　32 个候选 S 盒分成 2 个组

Group	1	2
$Prob_{D25}$	130	130
Cor^2_{L25}	136	134
Number of Sboxes	28	8

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 28 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 $16\times16=256$ 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，$x_3\|x_2\|x_1\|x_0$ 表示输入的 4 个比特，$y_3\|y_2\|y_1\|y_0$ 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。

表3.3　　TANGRAM 的 S 盒的分量布尔函数

y_0	=	$x_1 + x_0x_1 + x_2 + x_0x_1x_2 + x_0x_3 + x_2x_3$	dep=4;deg=3;term=6
y_1	=	$x_1 + x_2 + x_0x_2 + x_3 + x_2x_3 + x_1x_2x_3$	dep=4;deg=3;term=6
y_2	=	$x_0x_1 + x_2 + x_3$	dep=4;deg=2;term=3
y_3	=	$1 + x_0 + x_1 + x_2 + x_3 + x_1x_3$	dep=4;deg=2;term=6
$y_0 + y_1$	=	$x_0x_1 + x_0x_2 + x_0x_1x_2 + x_3 + x_0x_3 + x_1x_2x_3$	dep=4;deg=3;term=6
$y_0 + y_2$	=	$x_1 + x_0x_1x_2 + x_3 + x_0x_3 + x_2x_3$	dep=4;deg=3;term=5
$y_0 + y_3$	=	$1 + x_0 + x_0x_1 + x_0x_1x_2 + x_3 + x_0x_3 + x_1x_3 + x_2x_3$	dep=4;deg=3;term=8
$y_1 + y_2$	=	$x_1 + x_0x_1 + x_0x_2 + x_2x_3 + x_1x_2x_3$	dep=4;deg=3;term=5
$y_1 + y_3$	=	$1 + x_0 + x_0x_2 + x_1x_3 + x_2x_3 + x_1x_2x_3$	dep=4;deg=3;term=6
$y_2 + y_3$	=	$1 + x_0 + x_1 + x_0x_1 + x_1x_3$	dep=3;deg=2;term=5
$y_0 + y_1 + y_2$	=	$x_2 + x_0x_2 + x_0x_1x_2 + x_0x_3 + x_1x_2x_3$	dep=4;deg=3;term=5
$y_0 + y_1 + y_3$	=	$1 + x_0 + x_1 + x_0x_1 + x_2 + x_0x_2 + x_0x_1x_2 + x_0x_3 + x_1x_3 + x_1x_2x_3$	dep=4;deg=3;term=10
$y_0 + y_2 + y_3$	=	$1 + x_0 + x_2 + x_0x_1x_2 + x_0x_3 + x_1x_3 + x_2x_3$	dep=4;deg=3;term=7
$y_1 + y_2 + y_3$	=	$1 + x_0 + x_0x_1 + x_2 + x_0x_2 + x_3 + x_1x_3 + x_2x_3 + x_1x_2x_3$	dep=4;deg=3;term=9
$y_0 + y_1 + y_2 + y_3$	=	$1 + x_0 + x_1 + x_0x_2 + x_0x_1x_2 + x_3 + x_0x_3 + x_1x_3 + x_1x_2x_3$	dep=4;deg=3;term=9

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。

表3.4　　TANGRAM 的 S 盒的差分分布表(表中元素表示 $ND_S(\Delta I, \Delta O)$)

ΔI \ ΔO	0	1	2	4	8	3	5	6	9	A	C	7	B	D	E	F
0	16	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·
1	·	·	·	·	2	·	·	·	2	2	2	·	2	2	2	2
2	·	2	·	·	·	2	2	2	·	·	·	·	4	·	2	2
4	·	·	·	·	·	·	·	·	·	·	4	·	·	4	4	4
8	·	·	·	·	·	·	·	4	·	·	2	4	·	2	2	2
3	·	2	·	·	·	2	2	2	·	4	·	·	·	·	2	2
5	·	4	4	·	·	·	4	·	·	·	·	4	·	·	·	·
6	·	·	2	4	2	2	·	·	·	·	·	·	2	2	·	2
9	·	·	·	·	2	·	·	4	2	2	·	4	2	·	·	·
A	·	2	·	·	·	2	2	2	4	·	2	·	·	2	·	·
C	·	4	4	·	2	·	·	·	2	2	·	·	2	·	·	·
7	·	·	2	4	·	2	·	·	2	2	2	·	·	·	2	·
B	·	2	·	·	4	2	2	2	·	·	2	·	·	2	·	·
D	·	·	·	·	2	·	4	·	2	2	·	4	2	·	·	·
E	·	·	2	4	·	2	·	·	2	2	·	·	·	2	·	2
F	·	·	2	4	2	2	·	·	·	·	2	·	2	·	2	·

表3.5　　TANGRAM 的 S 盒的线性逼近表(表中元素表示 $\mathrm{Im}\, b_s(\Gamma I, \Gamma O)$)

ΓI \ ΓO	0	1	2	4	8	3	5	6	9	A	C	7	B	D	E	F
0	8	·	·	·	·	·	·	·	·	·	·	·	·	·	·	·
1	·	·	2	·	·	2	·	2	4	2	4	2	2	·	2	2
2	·	·	·	·	·	·	4	4	·	·	·	·	·	4	4	·
4	·	·	·	·	·	·	·	4	·	·	·	4	·	·	4	4
8	·	2	·	·	·	2	2	·	2	4	4	2	2	2	·	2
3	·	4	2	·	·	2	·	2	·	2	4	2	2	·	2	2
5	·	·	2	·	4	2	·	2	·	2	·	2	2	4	2	2
6	·	4	·	·	·	4	·	·	4	·	·	·	4	·	·	·
9	·	2	2	·	·	4	2	2	2	2	·	4	·	2	2	·
A	·	2	4	·	·	2	2	·	2	·	4	2	2	2	·	2
C	·	2	·	4	·	2	2	·	2	4	·	2	2	2	·	2
7	·	·	2	·	4	2	4	2	·	2	·	2	2	·	2	2
B	·	2	2	·	·	·	2	2	2	2	·	·	4	2	2	4
D	·	2	2	4	4	·	2	2	2	2	·	·	·	2	2	·
E	·	2	4	4	·	2	2	·	2	·	·	2	2	2	·	2
F	·	2	2	4	4	·	2	2	2	2	·	·	·	2	2	·

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。

表3.6　　TANGRAM 的逆 S 盒的分量布尔函数

y_0	=	$1+x_0x_1+x_0x_2+x_1x_2+x_3+x_0x_3+x_0x_2x_3$	dep=4;deg=3;term=7
y_1	=	$x_1+x_2+x_0x_3+x_1x_3$	dep=4;deg=2;term=4
y_2	=	$x_0+x_1+x_2+x_0x_3$	dep=4;deg=2;term=4
y_3	=	$x_0+x_0x_1+x_2+x_0x_2+x_1x_3+x_2x_3+x_1x_2x_3$	dep=4;deg=3;term=7
y_0+y_1	=	$1+x_1+x_0x_1+x_2+x_0x_2+x_1x_2+x_3+x_1x_3+x_0x_2x_3$	dep=4;deg=3;term=9
y_0+y_2	=	$1+x_0+x_1+x_0x_1+x_2+x_0x_2+x_1x_2+x_3+x_0x_2x_3$	dep=4;deg=3;term=9
y_0+y_3	=	$1+x_0+x_2+x_1x_2+x_3+x_0x_3+x_1x_3+x_2x_3+x_0x_2x_3+x_1x_2x_3$	dep=4;deg=3;term=10
y_1+y_2	=	$x_0+x_1x_3$	dep=3;deg=2;term=2
y_1+y_3	=	$x_0+x_1+x_0x_1+x_0x_2+x_0x_3+x_2x_3+x_1x_2x_3$	dep=4;deg=3;term=7
y_2+y_3	=	$x_1+x_0x_1+x_0x_2+x_0x_3+x_1x_3+x_2x_3+x_1x_2x_3$	dep=4;deg=3;term=7
$y_0+y_1+y_2$	=	$1+x_0+x_0x_1+x_0x_2+x_1x_2+x_3+x_0x_3+x_1x_3+x_0x_2x_3$	dep=4;deg=3;term=9
$y_0+y_1+y_3$	=	$1+x_0+x_1+x_1x_2+x_3+x_2x_3+x_0x_2x_3+x_1x_2x_3$	dep=4;deg=3;term=8
$y_0+y_2+y_3$	=	$1+x_1+x_1x_2+x_3+x_1x_3+x_2x_3+x_0x_2x_3+x_1x_2x_3$	dep=4;deg=3;term=8
$y_1+y_2+y_3$	=	$x_0x_1+x_2+x_0x_2+x_2x_3+x_1x_2x_3$	dep=4;deg=3;term=5
$y_0+y_1+y_2+y_3$	=	$1+x_2+x_1x_2+x_3+x_0x_3+x_2x_3+x_0x_2x_3+x_1x_2x_3$	dep=4;deg=3;term=8

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 $c_i(i=0,1,2,3)$ 表示对第 i 行左循环移位的参数，这些参数的选取准则如下：

1. $c_0 = 0$，$c_1 = 1$，$c_2 = 8$，$c_2 < c_3$；
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 $c_1 = 1$ 且 $c_2 = 8$。

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 (c_1, c_2, c_3) 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 $(c_1, c_2, c_3) = (1,8,11)$ 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 (c_1, c_2, c_3) 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 $(c_1, c_2, c_3) = (1,8,41)$ 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，我们估计最多能攻击到 30 轮（占全部轮数的 68%）。因此，TANGRAM128/128 的绝对安全冗余为 14 轮，相对安全冗余为 32%。

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，确保其相对安全冗余不低于 32%。

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 2^{64}，2^{64} 和 2^{128}。

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 n 比特的 R 轮分组密码，需要找到一个 $R-w$ 轮的、并且概率大于 2^{1-n}（对于 TANGRAM-128，这个概率值是 2^{-127}；对于

TANGRAM-256，这个概率值是 2^{-255} ）的差分传播， w 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 $R-w$ 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

#R	Prob.	#R	Prob.	#R	Prob.	#R	Prob.
1	2	8	32	15	76	22	114
2	4	9	40	16	82	23	119
3	7	10	49	17	87	24	124
4	10	11	55	18	92	25	130
5	14	12	60	19	98	26	135
6	18	13	66	20	103		
7	25	14	71	21	108		

表 3.2　256 比特分组长度下 TANGRAM 最优差分迹的概率

#R	Prob.	#R	Prob.	#R	Prob.	#R	Prob.
1	2	14	71	27	140	40	210
2	4	15	76	28	146	41	215
3	7	16	82	29	151	42	220
4	10	17	87	30	156	43	226
5	14	18	92	31	162	44	231
6	18	19	98	32	167	45	236
7	25	20	103	33	172	46	242
8	32	21	108	34	178	47	247
9	40	22	114	35	183	48	252
10	49	23	119	36	188	49	258
11	55	24	124	37	194	50	263
12	60	25	130	38	199		
13	66	26	135	39	204		

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 2^{-130}，小于边界值 2^{-127}；256 比特分组长度下 TANGRAM 的 49 轮最优差分迹的概率为 2^{-258}，小于边界值 2^{-255}。

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。

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 p，定义它的偏差 ε 为 $(p-1/2)$，相关系数 C 为 2ε。利用线性密码分析方法[15,16]攻击一个分组长度为 n 比特的 R 轮分组密码，需要找到一个 $R-u$ 轮的、并且相关系数的绝对值大于 $2^{-\frac{n}{2}}$（对于 128 比特分组长度下的 TANGRAM，这个值是 2^{-64}；对于 256 比特分组长度下的 TANGRAM，这个值是 2^{-128}）的线性传播，u 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 $R-u$ 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：

$$E(C_t^2)=\sum_i (C_i)^2 \tag{4.1}$$

其中，C_t 是线性传播的相关系数，C_i 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。

表4.1　　128 比特分组长度下 TANGRAM 最优线性迹的相关势

#R	Cor. Pot.	#R	Cor. Pot.	#R	Cor. Pot.	#R	Cor. Pot.
1	2	8	34	15	76	22	118
2	4	9	40	16	82	23	124
3	8	10	46	17	88	24	130
4	12	11	52	18	94	25	136
5	16	12	58	19	100		
6	20	13	64	20	106		
7	26	14	70	21	112		

表4.2　256 比特分组长度下 TANGRAM 最优线性迹的相关势

#R	Cor. Pot.	#R	Cor. Pot.	#R	Cor. Pot.
1	2	11	52	21	112
2	4	12	58	22	118
3	8	13	64	23	124
4	12	14	70	24	130
5	16	15	76	25	136
6	20	16	82	26	142
7	26	17	88	27	148
8	34	18	94	28	154
9	40	19	100	29	160
10	46	20	106	30	

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 1 ~ 25 轮的最优线性迹和 256 比特分组长度下 TANGRAM 的第 1 ~ 29 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 2^{-130}，小于边界值 2^{-128}；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 2^{-262}，小于边界值 2^{-256}。

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 2^{124} 个数据的明文集，将该集合加密 12 轮后所有输出的异或在 $w_{3,20}$ 和 $w_{3,9}$ 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 $w_{3,61}$ ）为任意常数，遍历剩下 255 比特得到一组包含 2^{255} 个数据的明文集，将该集合加密 16 轮后所有输出的异或在 $w_{3,61}$ 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

1 在 x64 处理器平台上的软件实现

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。

1.2 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。

1.3 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

(a)TANGRAM S 盒比特切片实现指令序列

1	$t1 = a \wedge c$	7	$e = t5 \wedge t6$
2	$t2 = a \,\&\, b$	8	$t8 = b \mid d$
3	$t3 = d \wedge t2$	9	$t9 = t1 \wedge t8$
4	$g = c \wedge t3$	10	$h = \sim t9$
5	$t5 = b \wedge c$	11	$t11 = t5 \,\&\, t9$
6	$t6 = t1 \,\&\, t3$	12	$f = t3 \wedge t11$

(b)TANGRAM 逆 S 盒比特切片实现指令序列

1	$t1 = \sim d$	7	$f = t5 \wedge t6$
2	$t2 = a \,\&\, t1$	8	$t8 = a \wedge t1$
3	$t3 = b \wedge t2$	9	$t9 = t3 \,\&\, t5$
4	$g = c \wedge t3$	10	$e = t8 \wedge t9$
5	$t5 = a \wedge c$	11	$t11 = f \,\&\, e$
6	$t6 = t1 \,\&\, t3$	12	$h = t3 \wedge t11$

注：其中，S 盒 (或逆 S 盒) 的输入为 $d\|c\|b\|a$，S 盒 (或逆 S 盒) 的输出为 $h\|g\|f\|e$(均为小尾端表示)。

$\wedge$ 表示按位" 异或"，$\mid$ 表示按位" 或"，$\&$ 表示按位" 与"，$\sim$ 表示按位" 取反"。

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

算法	ENC.(bytes)	DEC.(bytes)	KS.(bytes)
TANGRAM 128/128	5916	6216	3995
TANGRAM 128/256	6712	7065	5628
TANGRAM 256/256	11516	12765	8200

	ENC.(bytes)	DEC.(bytes)	KS.80(bytes)	KS.128(bytes)
RECTANGLE	3801	3823	3977	3767

注：其中，ENC. 代表加密函数，DEC. 代表解密函数，KS. 代表密钥扩展函数。

1.5 测试方案及测试结果

1.5.1 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 foo，将连续 x 次执行 foo 记为 $x \cdot foo$。计程序块 P 执行所需的时钟周期数为 $cycles(P)$。

最小时钟周期测试方法　测试多次运行 foo 所需的最小时钟周期数。其中，foo 为加密

函数、解密函数或密钥扩展。具体方法：循环执行 *loops* 次 $1\cdot foo$ 和 $(x+1)\cdot foo$ 。得到 $t_1=\min_{loops} cycles(1\cdot foo)$ 和 $t_2=\min_{loops} cycles((x+1)\cdot foo)$ 。返回 $(t_2-t_1+0.5x)/x$ 。

时钟周期的均值和标准差的测试　测试多次运行 $x\cdot foo$ 所需的时钟周期数的均值和标准差。其中， *foo* 为加密函数、解密函数或密钥扩展。具体方法：对执行 $x\cdot foo$ 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 av_1 和采样 1 标准差 sig_1（至多为 $0.05\times av_1$）。
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 $[av_1-sig_1, av_1+sig_1]$ 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 av_2 和标准差 sig_2。
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 sig_2 小于均值 av_2 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。

1.5.2 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

算法	KS.Average(cycles)	KS.Minimal(cycles)
TANGRAM 128/128	351.5(1.1%)	348.5
TANGRAM 128/256	450.3(1.8%)	447.5
TANGRAM 256/256	625.3(0.5%)	622.0
RECTANGLE-80	279.1(0.4%)	277.5
RECTANGLE-128	252.8(0.4%)	254.0

注：表中“KS.Average”列中，括号中的百分数为统计测试的标准差。

- 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

(a) TANGRAM 加解密短消息的最小速度

TANGRAM 128/128		
消息长度	ENC.Minimal (cycles/byte)	DEC.Minimal (cycles/byte)
256B	24.7	28.1
512B	23.7	27.1
768B	23.4	26.8
1KB	23.2	26.7
2KB	23.0	26.4
TANGRAM 128/256		
消息长度	ENC.Minimal (cycles/byte)	DEC.Minimal (cycles/byte)
256B	28.2	32.2
512B	27.0	31.0
768B	26.6	30.6
1KB	26.4	30.4
2KB	26.1	30.1
TANGRAM 256/256		
消息长度	ENC.Minimal (cycles/byte)	DEC.Minimal (cycles/byte)
256B	24.9	29.1
512B	23.0	27.1
768B	22.4	26.6
1KB	22.1	26.3
2KB	21.6	25.8

(b) RECTANGLE 加解密短消息的最小速度

RECTANGLE-80		
消息长度	ENC.Minimal (cycles/byte)	DEC.Minimal (cycles/byte)
256B	31.6	33.8
512B	30.9	33.1
768B	30.8	32.9
1KB	30.6	32.8
2KB	30.4	32.6
RECTANGLE-128		
消息长度	ENC.Minimal (cycles/byte)	DEC.Minimal (cycles/byte)
256B	31.5	33.7
512B	30.9	33.0
768B	30.7	32.8
1KB	30.6	32.7
2KB	30.4	32.5

- 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

TANGRAM 128/128		
消息长度	ENC.Minimal (cycles/byte)	DEC.Minimal (cycles/byte)
1MB	23.0	26.3
2MB	22.9	26.3
3MB	22.9	26.4
4MB	22.9	26.3
5MB	22.9	26.3
6MB	22.9	26.3
7MB	22.9	26.3
8MB	23.1	26.3
9MB	22.9	26.2

TANGRAM 128/256		
消息长度	ENC.Minimal (cycles/byte)	DEC.Minimal (cycles/byte)
1MB	25.9	29.8
2MB	25.8	29.8
3MB	25.8	29.9
4MB	25.8	29.8
5MB	25.8	29.8
6MB	25.8	29.8
7MB	25.8	29.8
8MB	25.8	29.8
9MB	25.8	29.8

TANGRAM 256/256		
消息长度	ENC.Minimal (cycles/byte)	DEC.Minimal (cycles/byte)
1MB	20.8	25.5
2MB	20.8	25.5
3MB	20.8	25.6
4MB	20.8	25.5
5MB	20.8	25.5
6MB	20.8	25.5
7MB	20.8	25.5
8MB	20.8	25.5
9MB	21.1	25.6

1.6 代码功能结构说明

1.6.1 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。

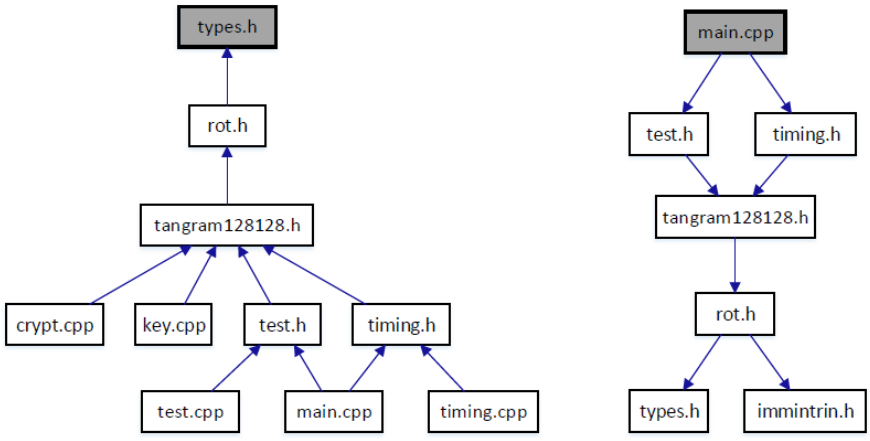
这三个软件代码包分别提供以下功能：

- 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

图1.1　　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。

1.6.3　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。

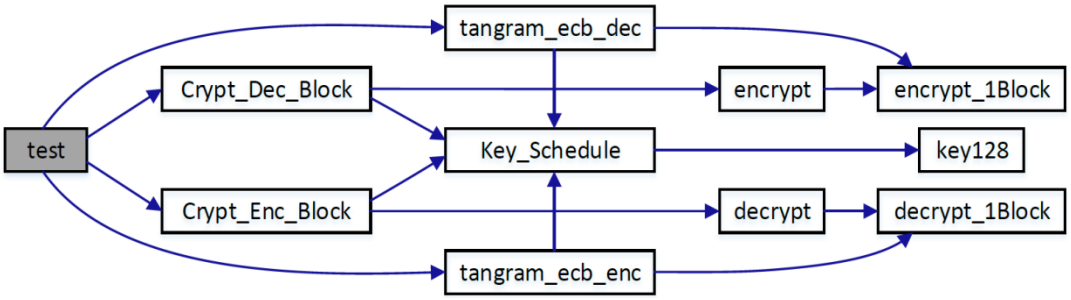
enc_cycles 用以测试加密算法加密指定长度短消息的最小时钟周期数。

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。

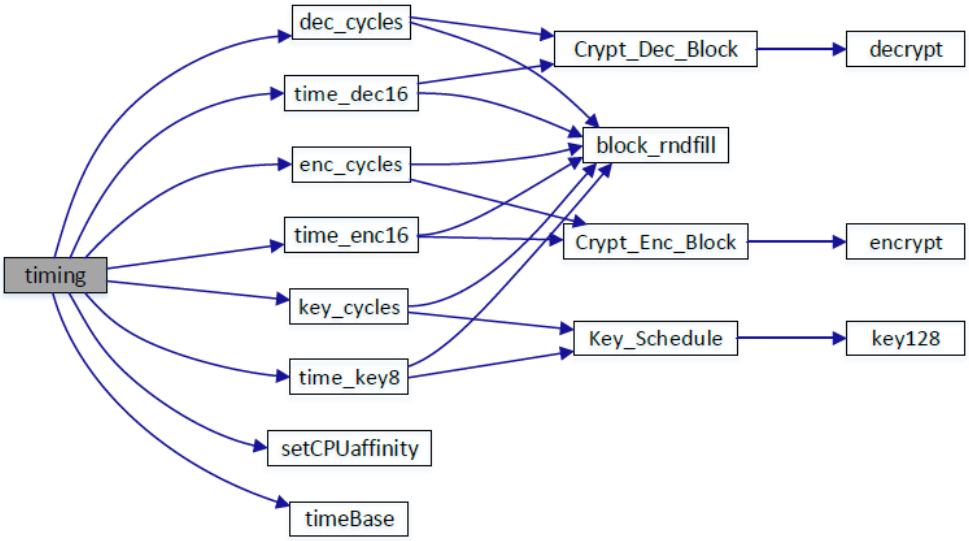
timing 执行速度测试的函数入口，通过调用这个函数可以一次性的进行：测试密钥扩展的最小时钟周期数开销 (cycles)、测试密钥扩展的平均时钟周期数和统计偏差 (cycles)、测试加解密短消息最大速率(cycles/byte)、测试 ECB 模式加解密长消息最大速率(cycles/byte)、测试加解密短消息平均速率及统计的标准差(cycles/byte)。

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



a) 正确性测试的函数调用图



b) 时间测试的函数调用图

图1.2 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);
```

2 硬件实现的性能分析

为了评估 TANGRAM 密码算法的硬件开销，我们选取了最接近算法本身的 Round-based 实现风格，即每个系统时钟周期内完成一次轮函数。我们使用 VerilogHDL 硬件描述语言对 TANGRAM 的硬件结构进行建模描述。使用 Synopsys DesignCompiler G-2012.06-SP5 对硬件代码进行综合，综合的目标工艺库 NANGATE 45 open cell library(PDKv1 3 v2010 12)。图 2.1 给出了 TANGRAM128/128 的硬件结构图，其它版本的硬件结构与其类似。

算 法	硬件实现代价（等效门电路）	所需的时钟数
TANGRAM128/128 加解密	3641.98	46
TANGRAM128/256 加解密	4879.95	52
TANGRAM256/256 加解密	7235.84	84

上面的表格中总结了各个版本 TANGRAM 的硬件面积和完成一次加/解密所需要的时钟数。可以看出最轻量级的 TRANGRAM128/128 仅需 3.6 千个等效门电路（等效门电路是以同工艺下 NAND 门的面积为单位），而 TRANGRAM256/256 的硬件实现代价是最大的，需要 7.2 千个等效门电路。因为采用 Round-based 的实现风格，所需时钟数为轮函数执行的轮数加上读取输入和产生输出的首末二轮。从硬件设计的角度来说，TANGRAM 的加密算法和解密算法的轮函数的硬件实现开销的差别仅仅取决于所用 S 盒的硬件面积的不同。经过综合实验，TANGRAM 的加密轮函数和解密轮函数的 S 盒的硬件开销一致，均为 19.298 个等效门电路。

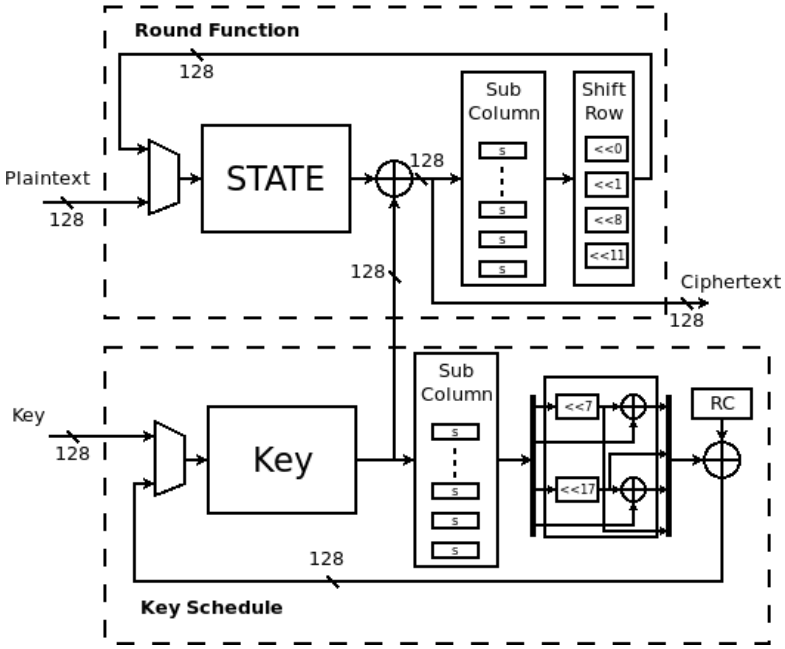


图 2.1　TANGRAM128/128 的硬件结构图

五. TANGRAM 的优缺点声明

TANGRAM 的主要优点如下。

1．设计简洁

TANGRAM 的整体结构为 SP 网络。

TANGRAM-128： 128 比特的加密状态用一个 4×32 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。

TANGRAM-256： 256 比特的加密状态用一个 4×64 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。

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，我们估计最多能攻击到 30 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

对于 RECTANGLE，我们在多个不同平台上做了实现，结果表明 RECTANGLE 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。

由于 TANGRAM 是 RECTANGLE 在分组长度（或密钥长度）上的扩展，两者在整体结构和主要模块上均具有很大的相似性，因此我们相信 TANGRAM 也能够在硬件和软件实现上均有很好的表现：硬件方面，TANGRAM 也具有很好的吞吐量和实现面积之间的性价比；软件方面，TANGRAM 在 8 位 AVR、16 位 MSP 和 32 位 ARM 三种微处理器平台上、以及高端处理器平台，均具有非常好的实现性能。

我们在 X64 平台上的同一测试机上，分别对 TANGRAM 和 RECTANGLE 做了实现和测试，TANGRAM128/128、TANGRAM128/256、TANGRAM256/256 的加密速度分别是 23、26.1 和 21.6 cycles/byte，而 RECTANGLE 的加密速度为 30.4cycles/byte。可以看到，TANGRAM128/128 的加密速度比 RECTANGLE 提高了 24%，而 TANGRAM256/256

的加密速度比 RECTANGLE 提高了大约 29%。

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 4×4 的 S 盒的并置，P 层是一个比特置换。SP 网络很常见，被 AES 等很多分组密码采用；4×4 的 S 盒也很常见，在 Serpent、NOEKEON、PRESENT 和 RECTANGLE 等很多分组密码中使用；DES、PRESENT、SPONGENT 和 RECTANGLE 等密码算法都采用了比特置换作为扩散层；另外，TANGRAM 的扩散层是对密码状态中的 4 行做行移位操作，这与 AES 的扩散层也有类似的地方。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。因此，可以说，TANGRAM 密码模块的选取，没有晦涩难懂之处，都是基于我们对其密码性质充分理解的基础之上。

6．对模块的仔细细选

在设计 TANGRAM 的过程中，类似于 RECTANGLE 的设计，我们细致地挑选目标 S 盒，详见[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，循环移位操作无论在硬件还是软件上都可以容易、高效地实现；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 and Application of Cryptographic Techniques Lofthus, Norway, May 23--27, 1993 Proceedings, 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 DES implementation 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 Cryptographic Techniques Prague, Czech Republic, May 2--6, 1999 Proceedings, chap. Cryptanalysis of Skipjack} Reduced to 31 Rounds Using Impossible Differentials, 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 encryption standard. 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 Cryptographic Techniques Bruges, Belgium, May 14--18, 2000 Proceedings, chap. Advanced Slide 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: 4th International Workshop, FSE'97 Haifa, Israel, January 20--22 1997 Proceedings, 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 encryption standard. 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. Integral Cryptanalysis, 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 International Workshop, WAIFI 2007, Madrid, Spain, June 21-22, 2007. Proceedings, chap. On the 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 International Cryptology Conference Santa Barbara, California, USA August 21--25, 1994 Proceedings, chap. The First Experimental Cryptanalysis of the Data Encryption 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 and Application of Cryptographic Techniques Lofthus, Norway, May 23--27, 1993 Proceedings, 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, Revised Selected Papers, chap. A New Classification of 4-bit Optimal S-Boxes and Its 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