ANT 系列分组密码算法

设计文档

王美琴 陈师尧 樊燕红 付勇 黄鲁宁

山东大学网络空间安全学院

2019 年 10 月

V2.0 (第二轮提交版本)

1 算法描述

本章将给出 ANT 系列分组密码算法的详细说明。ANT 系列算法采用了经典的 Feistel 结构，其所有分组可用 ANT-$2n/m$表示。不同分组的相应参数在表 1 中给出。

表 1: ANT 系列算法相应参数

分组比特长度 $(2n)$	密钥比特长度 (m)	字比特长度 (n)	轮数(R)	密码算法名称
128	128	64	46	ANT-128/128
	256	64	48	ANT-128/256
256	256	128	74	ANT-256/256

本文中的一些常用表示如下：

- $x = x_{n-1}||x_{n-2}||\cdots||x_0$ 代表 n比特的变量x（从高位x_{n-1}到低位x_0）。
- $x \ll a$ 表示变量x左移位a比特。
- $x \lll b$ 表示变量x左循环移位b比特。
- $x \oplus y$ 表示变量x和变量y进行异或操作。
- $x \odot y$ 表示变量x和变量y进行与操作。

1.1 轮函数

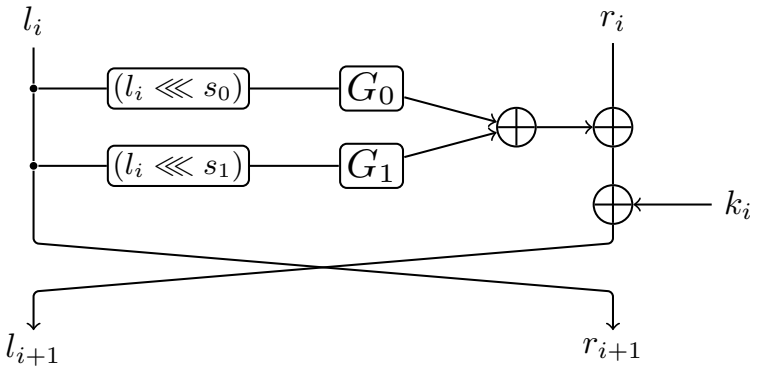

图 1: ANT 系列分组密码算法轮函数

ANT 系列分组密码算法轮函数见图 1，可以定义公式如下：

$$(l_{i+1}, r_{i+1}) = R_{sk_i}(l_i, r_i) = (G_0(l_i \lll s_0) \oplus G_1(l_i \lll s_1) \oplus r_i \oplus sk_i, l_i),$$

注意为保证加解密一致，最后一轮左支经过轮函数后不进行交换。循环移位参数 (s_0, s_1) 在 ANT 所有分组中设定为(3,16)。G_0和G_1为非线性函数，包含两层，两层中间是一层比特级的置换。在具体介绍非线性函数G_0和G_1前，先引入表示如下：

令 $x^0 = x^0_{n-1}||x^0_{n-2}||\cdots||x^0_0$ 表示 G_0 或 G_1 的 n 比特输入，$y^0 = y^0_{n-1}||y^0_{n-2}||\cdots||y^0_0$ 表示 G_0 或 G_1 第一层输出的 n 比特。类似的，$x^1 = x^1_{n-1}||x^1_{n-2}||\cdots||x^1_0$和$y^1 = y^1_{n-1}||y^1_{n-2}||\cdots||y^1_0$分别表示 G_0或G_1第二层的输入和G_0或G_1的输出。PERM 表示G_0或G_1中采用的比特级置换。如下给出非线性函数G_0或G_1的具体描述，并在附录 2 中给出 ANT-128 的G_0和G_1示例图。

对于G_0：

$$y^1 = G_0(x^0),$$

其中先经过第一层

$$y^0_j = \begin{cases} \left(x^0_{j+3} \odot x^0_{j+2}\right) \oplus x^0_j, & j\ mod\ 4 = 0, \\ x^0_j, & others. \end{cases} \quad 对\ 0 \le j < n.$$

再经过比特级的置换

$$x^1_{PERM(j)} = y^0_j \ 对\ 0 \le j < n.$$

最后经过第二层

$$y^1_j = \begin{cases} \left(x^1_{j+3} \odot x^1_{j+2}\right) \oplus x^1_j, & j\ mod\ 4 = 0, \\ x^1_j, & others. \end{cases} \quad 对\ 0 \le j < n.$$

类似的，对于G_1：

$$y^1 = G_1(x^0),$$

其中先经过第一层

$$y^0_j = \begin{cases} \left(x^0_{j+2} \odot x^0_{j+1}\right) \oplus x^0_j, & j\ mod\ 4 = 1, \\ x^0_j, & others. \end{cases} \quad 对\ 0 \le j < n.$$

再经过比特级的置换

$$x^1_{PERM(j)} = y^0_j \ 对\ 0 \le j < n.$$

最后经过第二层

$$y^1_j = \begin{cases} \left(x^1_{j+2} \odot x^1_{j+1}\right) \oplus x^1_j, & j\ mod\ 4 = 1, \\ x^1_j, & others. \end{cases} \quad 对\ 0 \le j < n.$$

对于不同 ANT 分组的非线性函数G_0和G_1，其所采用的比特级置换 PERM，现给出表达式如下（PERM 置换的表格形式见附录 3），

对 128 分组：

$$PERM(j) = \begin{cases} (j+58) mod\ 64, & j\ mod\ 4 = 3, \\ (j+54)\ mod\ 64, & j\ mod\ 4 = 2, \\ (j+30)\ mod\ 64, & j\ mod\ 4 = 1, \\ (j+2)\ mod\ 64, & j\ mod\ 4 = 0. \end{cases}$$

对 256 分组：

$$PERM(j) = \begin{cases} (j+122)\ mod\ 128, & j\ mod\ 4 = 3, \\ (j+118)\ mod\ 128, & j\ mod\ 4 = 2, \\ (j+62)\ mod\ 128, & j\ mod\ 4 = 1, \\ (j+2)\ mod\ 128, & j\ mod\ 4 = 0. \end{cases}$$

1.2 密钥生成算法

ANT 系列分组密码算法可以采用$2n$比特或$4n$比特长度主密钥K, 可采用如下两种基于 LFSR（线性反馈移位寄存器）的密钥扩展方案。

对于 ANT-$2n/2n$：$2n$比特的主密钥$K = k_{2n-1}||k_{2n-2}||\cdots||k_0$可以分成 2 个字(高$n$比特和低$n$比特)，

$$K_1 = k_{2n-1}||k_{2n-2}||\cdots||k_n,\ K_0 = k_{n-1}||k_{n-2}||\cdots||k_0.$$

$K_1||K_0$则作为图 2 中 LFSR 的初始状态。每次先取出当前低位寄存器K_i中的n比特则作为当前轮的轮密钥sk_i，然后进行 LSFR 的状态更新，$(i+1)$作为轮常数（i为当前轮数且 $0 \le i < R$），K_{i+1}经过图 3 中的A操作迭代变换 3 次。注意不同分组下，A操作的输入均分成了 8 个$(n/8)$比特的变量，$X_7||X_6||\cdots||X_0$。如下给出更新操作表达式：

$$K_{i+2} = A^3(k_{i+1}) \oplus k_i \oplus (i+1)$$

对于不同分组，移位参数(t_0, t_1)在表 2 中给出。

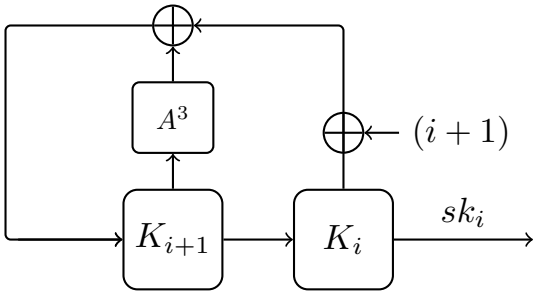

图 2: ANT-$2n/2n$密钥生成算法

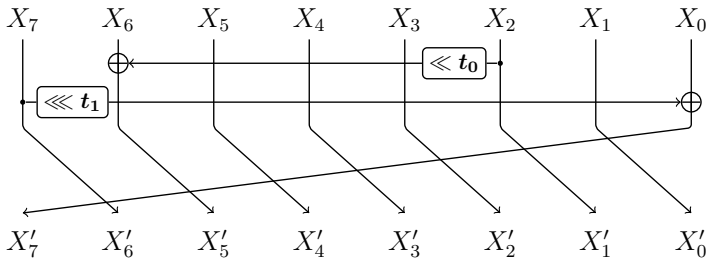

图 3: A操作

表 2: 不同分组对应的移位参数(t_0, t_1)

t_0	t_1	密码算法名称
7	1	ANT-128/128
7	1	ANT-128/256
15	1	ANT-256/256

对于 ANT-$2n/4n$：类似地，$4n$比特长度主密钥$K = k_{4n-1}||k_{4n-2}||\cdots||k_0$可以分成 4 个字，

$$K_3 = k_{4n-1}||k_{4n-2}||\cdots||k_{3n},\ K_2 = k_{3n-1}||k_{3n-2}||\cdots||k_{2n},$$

$$K_1 = k_{2n-1}||k_{2n-2}||\cdots||k_n,\ K_0 = k_{n-1}||k_{n-2}||\cdots||k_0.$$

$K_3||K_2||K_1||K_0$则作为图 4 中 LFSR 的初始状态。同样，当前低位寄存器K_i中的n比特则作为当前轮的轮密钥sk_i，然后开始当轮密钥状态更新，$(i+1)$作为轮常数，K_{i+3}经图 3 中的A操作迭代变换 3 次，其中的移位参数(t_0, t_1)同样在表 2 中给出。如下给出更新操作表达式：

$$K_{i+4} = A^3(k_{i+3}) \oplus k_{i+1} \oplus K_i \oplus (i+1)$$

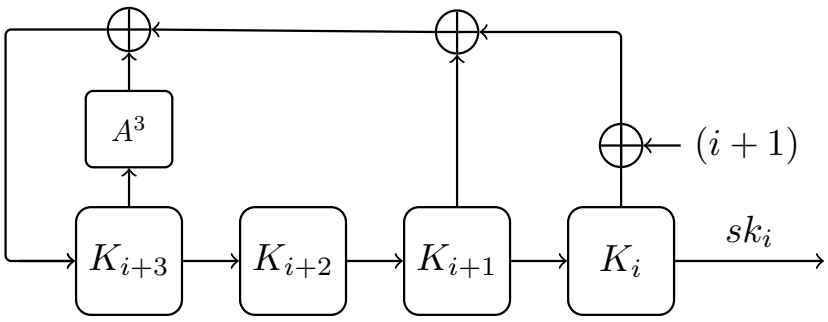

图 4: ANT-$2n/4n$密钥生成算法

2 设计原理

- **创新点一，Feistel 结合 Expand-then-compress**：Feistel 结构是一种经典的对称密码算法结构。与 SPN 结构相比，Feistel 结构具有加解密一致的特性，有利于实现。尤其是对于硬件实现，加解密可以共用同一个轮函数操作，大大减小了硬件面积。但 Feistel 结构的特性也导致其相比于 SPN 结构扩散的更慢（每轮只更新状态的一半）。因此，ANT 系列分组密码算法在设计时，结合了 Expand-then-compress 的设计思想，将算法的一支先进行扩展，再进入非线性函数G_0和G_1，最后通过异或操作进行压缩。这种扩展再压缩的方式，一方面可以增加扩散速度，另一方面，可以并行处理以减小轮函数的时延。
- **创新点二，比特级非线性函数G_0/G_1**：非线性函数G_0和G_1均为 2 层，并且均仅采用比特级的与操作、异或操作和置换操作，其中与操作作为唯一的非线性操作。相对于传统的 S 盒，这样可以极大地减小硬件面积，同时侧信道防护所需要的代价很小。异或操作的位置和置换$PERM$的选取主要基于两点，首先是为了保证每个比特都能经过非线性操作，其次是为了平衡压缩操作之后各个输出比特的代数度。另外，置换$PERM$还考虑了 bit-slice 软件多路实现，即根据比特位置下标模 4，分为了 4 个类进行置换，同时为了进一步增加扩散效果，每个类之内的比特再进行循环移位。

  值得注意的是创新点一和二相互结合，使 ANT 算法取得了较好的扩散性。ANT-128 算法的全扩散轮数为 5 轮，ANT-256 算法 的全扩散轮数为 7 轮（考虑到 ANT 算法的轻量级设计和 Feistel 结构，已经达到较快的扩散速度）。
- **创新点三，密钥生成算法**：密钥生成算法在设计时主要考虑的是硬件代价小、扩散性好以及能够匹配轮函数的时延。因此，我们采用了线性的密钥生成算法，通过 LFSR 的形式来更新密钥状态。采用 Feistel 结构的 A 操作迭代 3 次，用时延换取较大面积才能达到的扩散效果。而在保证较好的扩散性的同时，我们令变量经过t_0移位操作后仅剩下一个比特，进一步减小硬件代价。

3 安全性分析

3.1 差分分析和线性分析

差分分析(DC)和线性分析(LC)均是经典的密码分析方法。为了展示 ANT 系列分组密码算法对差分攻击和线性攻击的抵抗能力，在表 3 中给出了不同分组最优的差分特征和线性特征的搜索情况（利用 SAT/SMT 技术进行搜索）。由此可以得到，ANT-128 不存在超过 27 轮的有效差分或线性特征（由表 3 可得 28 轮的差分特征的下界为$49 + 42 + 42 = 133$），ANT-256 不存在超过 47 轮的有效差分或线性特征（$45 \times 6 = 270$）。

表 3: ANT 不同分组最优的差分特征和线性特征（概率以$-\log_2 p$形式给出且线性特征使用相关度）

轮数	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
差分									
ANT-128	2	4	8	12	20	28	36	42	49
ANT-256	2	4	8	12	20	28	45	-	-
线性									
ANT-128	1	2	4	6	10	14	20	26	≥ 28
ANT-256	1	2	4	6	10	14	23	-	-

- ：表示没有搜索出相应的界

我们还对 ANT 算法的 differential 和 linear hull 效应进行了测试，发现 ANT 系列算法 differential 和 linear hull 效应不显著。根据搜索得到 ANT-128 的 6 轮最优差分路线（概率为2^{-20}）和线性路线（相关性为2^{-10}），首先利用 SAT 技术进行路线堆积，发现堆积效应不显著。然后，使用2^{24}的数据量对路线进行验证，实验结果与理论堆积结果相符。因此，ANT 系列分组密码算法的轮数足以抵抗差分攻击和线性攻击。

3.2 不可能差分分析和零相关线性分析

不可能差分分析（IDC）和零相关线性分析(ZC)分别是对差分分析和线性分

析方法的拓展。同样利用 SAT/SMT 技术，对不同分组 ANT 的不可能差分和零相关线性路线进行搜索，结果如下表 4:

表 4: ANT 算法不同分组的 IDC 和 ZC 路线搜索到的最长轮数

IDC 路线轮数	ZC 路线轮数	密码算法名称
9	10	ANT-128
13	13	ANT-256

因此，不可能差分和零相关线性攻击对 ANT 系列分组密码算法不够成威胁。如下给出一条 ANT-128 的 9 轮 IDC 路线示例：

$$(l_0, r_0) = (0x0000000000000000, 0x8000000000000000)$$

$$\nRightarrow$$

$$(l_9, r_9) = (0x4000000000000000, 0x0000000000000000)$$

3.3 积分分析

积分攻击利用特定的明文集合在密文处的非随机的代数性质来区分随机置换和目标密码算法。本文利用 SAT/SMT 技术，基于比特级的可分性对不同分组的 ANT 算法进行积分路线搜索。对于 ANT-128，可以搜索得到不存在超过 16 轮的积分路线；对于 ANT-256，可以搜索得到不存在超过 21 轮的积分路线。因此，ANT 系列分组密码算法的轮数足以抵抗积分攻击。

4 性能分析

4.1 软件性能分析

我们对 ANT 系列分组密码算法进行了 64-bit Windows 下的 CBC 单路加/解密速度测试，CBC 多路解密速度测试和 32-bit ARM 下的 CBC 单路加/解密速度测试。

测试过程采用 256 字节短数据进行一次 CBC 模式下的加密，包含了密钥生成算法的时间。如此进行多次并且每次加密均运行一次完整的密钥生成算法，最后取速率的平均值，具体测试结果见表 5。

表 5：ANT 系列分组算法速率优化 C 实现

算法软件实现类别		CBC 加/解密速度 (Mbps)	CBC 多路解密速度 (Mbps)
64-bit Windows	128/128	597	6065
	128/256	581	6137
	256/256	587	2635
32-bit ARM	128/128	2.2	-
	128/256	2.1	-
	256/256	2.2	-

64-bit PC 端测试硬件环境：Windows 7, i7-6700, 8G DDR4 2400M

64-bit PC 端测试软件环境：VS 2017, X64 release, avx2

32-bit ARM 测试硬件环境：STM32F103ZET6, 主频 72Mhz

32-bit ARM 测试软件环境：EWARM8.2

4.2 硬件性能分析

我们利用 Verilog HDL 语言对 ANT 系列分组密码算法进行了基于单轮和 2 轮的实现，采用 32 个分组数据进行加/解密，且至少调用 1 次密钥扩展算法。硬件仿真数据如下表 6。

表 6 :基于单轮实现的硬件指标（基于 HJTC 110nm 标准元件库）

	自测试结果		
	128/128	128/256	256/256
运算周期 (Cycles)	2991	3121	4811
时钟约束 (ns)	2.14	2.07	2.14
加密速率 (Mbps)	1300.28	1288.24	1616.57
解密速率 (Mbps)	1260.05	1248.42	1566.94
面积 (um^2)	31234.66	49160.10	63201.84
加密吞面比 (Mbps/um^2)	0.0416	0.0262	0.0256
解密吞面比 (Mbps/um^2)	0.0403	0.0254	0.0248

5 优缺点声明

ANT 系列分组密码算法的设计初衷是：相比于国际上传统的分组密码算法，给出一款适合轻量级实现，便于侧信道防护且代价小，同时适合 bit-slice 软件多路并行实现的分组密码算法。因此，本算法采用了经典的 Feistel 结构并结合 Expand-then-compress 的设计理念。在关键的轮函数上，采用了比特级的设计（与操作，异或操作，置换操作，循环移位操作），与传统利用 S 盒作为非线性层的算法相比，硬件面积小且侧信道防护代价小。一些组件如比特级置换的选取，也考虑了软件多路 bit-slice 的实现性能。然而 ANT 系列分组算法比特级轮函数的设计理念在一定程度上影响其单路的软件性能。密钥生成算法中为了提升扩散效果，导致其操作步骤较多，也在一定程度上影响 ANT 算法的软件性能。

6 算法特色

轻量级实现优势： ANT 系列算法各个组件在设计时都考虑了轻量级实现的代价（面积和时延），整体采用 Feistel 结构，因而硬件实现中具有加解密轮函数部分可以复用的优势；轮函数利用比特级的与操作，来代替传统的 S 盒作为非线性组件；采用线性的密钥生成算法，并结合 Feistel 迭代操作，在匹配轮函数时延的同时，用时延换取较大面积才能达到的扩散效果。这些设计理念都极大减小了算法的硬件面积，使 ANT 系列分组密码算法具有轻量级实现的优势。如 ANT-128/128，在推荐的工艺库（HJTC 110nm）下，对 32 个分组数据加/解密的基于单轮实现进行仿真，仅为 3220 个 GE。

侧信道防护优势：随着侧信道攻击技术日趋成熟，侧信道攻击已经成为一种重要的密码算法攻击手段。该方法攻击成本低、成功率高，主要利用密码芯片执行密码算法过程所泄露出的物理信息来提取密钥信息。对于 ANT 系列分组密码算法来说，其仅有与操作，循环移位操作，异或操作和比特级的置换操作，在进行侧信道防护时，容易进行门限实现并且实现代价很小。因此，ANT 系列分组密码算法还具有侧信道防护的优势。

7 第二轮修改说明

根据竞赛第一轮的算法安全性要求，同时权衡轮数与算法安全性和性能之间的关系。本团队在保证安全性的前提下，对 ANT 系列算法进行了如下修改：

- ANT 系列分组密码算法为保证加解密完全一致，所有分组最后一轮不交换。
- ANT-128/128 由 58 轮缩减为 46 轮。
- ANT-128/256 由 70 轮缩减为 48 轮。
- ANT-256/256 由 100 轮缩减为 74 轮。

修改依据：

- 最后一轮不交换，在软硬件实现时无需任何额外处理，加解密轮函数部分完全一致。
- 对 ANT-256 进行了进一步搜索，得到 8 轮的最优差分路线概率为2^{-45}，8 轮的最优线性路线相关性为2^{-23}。
- 综合竞赛的安全性要求和预留的冗余度，我们根据单密钥下各个版本所对应的最长攻击轮数（有效区分器轮数上界+密钥恢复轮数上界）和全扩散轮数，对 ANT 系列算法的轮数进行了更新。

附录

附录 1 测试向量

ANT-128/128（46 轮）测试向量

主密钥$K = (K_1||K_0)$: (0x1f1e1d1c1b1a1918, 0x1716151413121110)

明文$Pt = (l_0, r_0)$: (0x0f0e0d0c0b0a0908, 0x0706050403020100)

密文$Ct = (l_{46}, r_{46})$: (0x19fbd4bfff604b75, 0x23df8c6ea0835539)

ANT-128/256（48 轮）测试向量

主密钥$K = (K_3||K_2||K_1||K_0)$:

(0x0000000000000000, 0x1f1e1d1c1b1a1918)

()

明文$Pt = (l_0, r_0)$:

(0x0f0e0d0c0b0a0908, 0x0706050403020100)

密文$Ct = (l_{48}, r_{48})$:

(0x41b7fc3e25a5e9a1, 0x2d04b790d9bc3713)

ANT-256/256（74 轮）测试向量

主密钥$K = (K_1||K_0)$:

(0x0000000000000000, 0x1f1e1d1c1b1a1918)

(0x0000000000000000, 0x1716151413121110)

明文$Pt = (l_0, r_0)$:

(0x1f1e1d1c1b1a1918, 0x1716151413121110)

(0x0f0e0d0c0b0a0908, 0x0706050403020100)

密文$Ct = (l_{74}, r_{74})$:

(0x3710ef6b982fde25, 0x10ad55c4f6c88936)

(0xfa91b143554589fb, 0x393aa8417fe7d481)

附录 2 ANT-128 中G_0和G_1示图

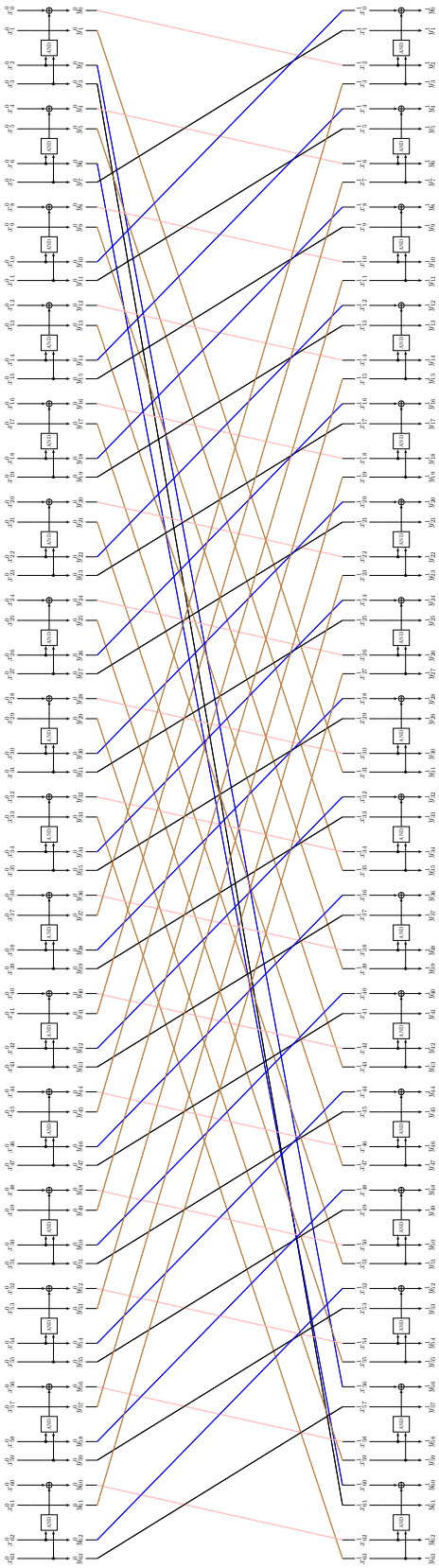

图 5：ANT-128 中的G_0

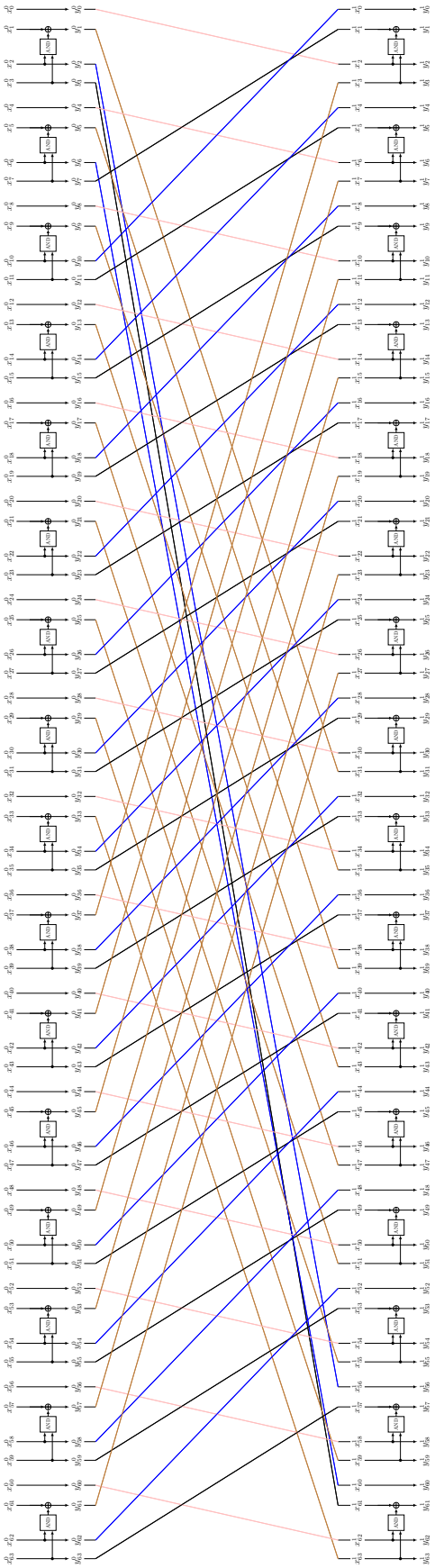

图 6：ANT-128 中的G_1

附录 3 PERM 置换表

表 7: ANT-128 中的 PERM

j	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
$PERM(j)$	2	31	56	61	6	35	60	1	10	39	0	5	14	43	4	9
j	**16**	**17**	**18**	**19**	**20**	**21**	**22**	**23**	**24**	**25**	**26**	**27**	**28**	**29**	**30**	**31**
$PERM(j)$	18	47	8	13	22	51	12	17	26	55	16	21	30	59	20	25
j	**32**	**33**	**34**	**35**	**36**	**37**	**38**	**39**	**40**	**41**	**42**	**43**	**44**	**45**	**46**	**47**
$PERM(j)$	34	63	24	29	38	3	28	33	42	7	32	37	46	11	36	41
j	**48**	**49**	**50**	**51**	**52**	**53**	**54**	**55**	**56**	**57**	**58**	**59**	**60**	**61**	**62**	**63**
$PERM(j)$	50	15	40	45	54	19	44	49	58	23	48	53	62	27	52	57

表 8: ANT-256 中的 PERM

j	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
$PERM(j)$	2	63	120	125	6	67	124	1	10	71	0	5	14	75	4	9
j	**16**	**17**	**18**	**19**	**20**	**21**	**22**	**23**	**24**	**25**	**26**	**27**	**28**	**29**	**30**	**31**
$PERM(j)$	18	79	8	13	22	83	12	17	26	87	16	21	30	91	20	25
j	**32**	**33**	**34**	**35**	**36**	**37**	**38**	**39**	**40**	**41**	**42**	**43**	**44**	**45**	**46**	**47**
$PERM(j)$	34	95	24	29	38	99	28	33	42	103	32	37	46	107	36	41
j	**48**	**49**	**50**	**51**	**52**	**53**	**54**	**55**	**56**	**57**	**58**	**59**	**60**	**61**	**62**	**63**
$PERM(j)$	50	111	40	45	54	115	44	49	58	119	48	53	62	123	52	57
j	**64**	**65**	**66**	**67**	**68**	**69**	**70**	**71**	**72**	**73**	**74**	**75**	**76**	**77**	**78**	**79**
$PERM(j)$	66	127	56	61	70	3	60	65	74	7	64	69	78	11	68	73
j	**80**	**81**	**82**	**83**	**84**	**85**	**86**	**87**	**88**	**89**	**90**	**91**	**92**	**93**	**94**	**95**
$PERM(j)$	82	15	72	77	86	19	76	81	90	23	80	85	94	27	84	89
j	**96**	**97**	**98**	**99**	**100**	**101**	**102**	**103**	**104**	**105**	**106**	**107**	**108**	**109**	**110**	**111**
$PERM(j)$	98	31	88	93	102	35	92	97	106	39	96	101	110	43	100	105
j	**112**	**113**	**114**	**115**	**116**	**117**	**118**	**119**	**120**	**121**	**122**	**123**	**124**	**125**	**126**	**127**
$PERM(j)$	114	47	104	109	118	51	108	113	122	55	112	117	126	59	116	121