Ballet 分组密码算法

设计文档

崔婷婷 王美琴 樊燕红 胡凯 付勇 黄鲁宁

2019 年 10 月 9 日

算法设计文档要求

(1) 算法设计文档应至少包含算法完整描述、设计原理、安全性分析、性能分析、优缺点分析等。

(2) 算法设计文档应给出不同计算平台（如8/32/64位软实现平台、FPGA硬件实现、ASIC硬件实现）和不同应用场景（如低功耗、低延时、硬件资源实现受限环境等）等的算法实现性能和实现代价论述。

(3) 算法设计文档应给出算法特色的论述，如是否具有白盒实现优势、轻量级实现优势、防侧信道分析优势等。

1 算法描述

在描述算法之前，如下定义本文档中使用的表示符号：

1) $n/k/r$：分组长度/密钥长度/算法总轮数；
2) ⋘和⋙：左循环移位和右循环移位；
3) ⊞和⊟：模加和模减操作；
4) ⊕：异或操作；
5) $x = x_{n-1}x_{n-2}\cdots x_1x_0$：$x_{n-1}$为$x$最高位，$x_0$为$x$最低位；
6) Ballet-n/k：分组长度为n，密钥长度为k的 Ballet 算法版本；
7) Ballet-n：分组长度为n的 Ballet 所有可能版本；

1.1 算法参数

本文档中给出了一个新的分组密码算法——Ballet 算法。该算法共三个版本：Ballet-128/128、Ballet-128/256 和 Ballet-256/256，每个版本的参数如表 1 所示。

版本	分组长度（n）	密钥长度（k）	总轮数（r）
Ballet-128/128	128	128	46
Ballet-128/256	128	256	48
Ballet-256/256	256	256	74

表 1：Ballet 算法的三个版本的参数设定

1.2 算法结构

Ballet 算法采用 ARX 结构，轮函数中仅包含循环移位、模加和异或操作，结构简单易实现。其每个版本均使用相同的轮函数结构，见图 1。为了加解密的相似性，最后一个轮函数运算时省略最后一个线性置换操作。

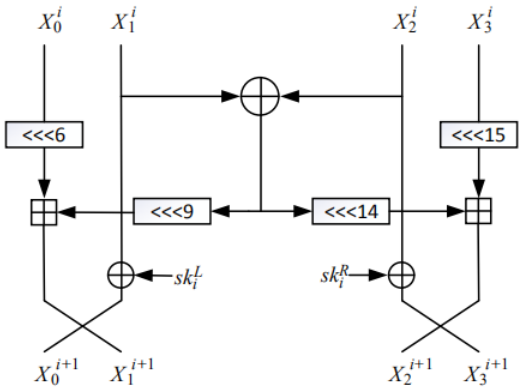


图 1： Ballet 算法的轮函数（加密过程，当作为算法的最后一轮时，省略最后一个线性置换操作）

假设 Ballet 算法的第i个轮函数（$i = 0,1,\cdots,r-1$）的输入为$(X_0^i||X_1^i||X_2^i||X_3^i)$，使用的子密钥为$(sk_i^L||sk_i^R)$，其中$X_*^i, sk_i^* \in \mathbb{F}_2^{n/4}$。则加密过程如下：

For $0 \le i < r-1$:
$X_0^{i+1} = X_1^i \oplus sk_i^L$,
$X_1^{i+1} = (X_0^i \lll 6) \boxplus [(X_1^i \oplus X_2^i) \lll 9]$,
$X_2^{i+1} = (X_3^i \lll 15) \boxplus [(X_1^i \oplus X_2^i) \lll 14]$,
$X_3^{i+1} = X_2^i \oplus sk_i^R$.

$X_0^{i+1} = (X_0^i \lll 6) \boxplus [(X_1^i \oplus X_2^i) \lll 9]$,
$X_1^{i+1} = X_1^i \oplus sk_i^L$,
$X_2^{i+1} = X_2^i \oplus sk_i^R$,
$X_3^{i+1} = (X_3^i \lll 15) \boxplus [(X_1^i \oplus X_2^i) \lll 14]$.

表 2：Ballet 算法的加密过程

作为加密过程的逆过程，Ballet 算法的解密过程如下：

For $0 \le i < r-1$:
$X_0^{i+1} = X_1^i \oplus sk_i^L$,
$X_1^{i+1} = [X_0^i \boxminus (X_1^i \oplus X_2^i \oplus sk_i^L \oplus sk_i^R) \lll 9] \ggg 6$,
$X_2^{i+1} = [X_3^i \boxminus (X_1^i \oplus X_2^i \oplus sk_i^L \oplus sk_i^R) \lll 14] \ggg 15$,
$X_3^{i+1} = X_2^i \oplus sk_i^R$.

$X_0^{i+1} = [X_0^i \boxminus (X_1^i \oplus X_2^i \oplus sk_i^L \oplus sk_i^R) \lll 9] \ggg 6$,
$X_1^{i+1} = X_1^i \oplus sk_i^L$,
$X_2^{i+1} = X_2^i \oplus sk_i^R$,
$X_3^{i+1} = [X_3^i \boxminus (X_1^i \oplus X_2^i \oplus sk_i^L \oplus sk_i^R) \lll 14] \ggg 15$.

表 3：Ballet 算法的解密过程

1.3 密钥生成算法

Ballet 算法按照分组长度和密钥长度的关系可分为 Ballet-n/n和 Ballet-$n/2n$两种。针对 Ballet-n/n版本，密钥生成算法如下：

假设主密钥$K = k_0||k_1$;
For $0 \le i < r$:
$sk_i^L||sk_i^R = k_0$;
输出$sk_i^L||sk_i^R$;

$k_{temp} = k_1$;
$k_1 = k_0 \oplus (k_1 \lll 3) \oplus (k_1 \lll 5) \oplus i$;
$k_0 = k_{temp}$.

表 4：Ballet-n/n版本中的密钥生成算法

针对 Ballet-$n/2n$版本，密钥生成算法如下：

假设主密钥$K = k_0||k_1||t_0||t_1$;

For $0 \le i < r$:
$sk_i^L||sk_i^R = k_0$;
输出$sk_i^L||sk_i^R$;

$t_{temp} = t_1$;
$t_1 = t_0 \oplus (t_1 \lll 7) \oplus (t_1 \lll 17)$;
$t_0 = t_{temp}$;

$k_{temp} = k_1$;
$k_1 = k_0 \oplus (k_1 \lll 3) \oplus (k_1 \lll 5)$;
$k_0 = k_{temp}$.

$k_1 = k_1 \oplus t_1 \oplus i$.

表 5: Ballet-$n/2n$版本中的密钥生成算法

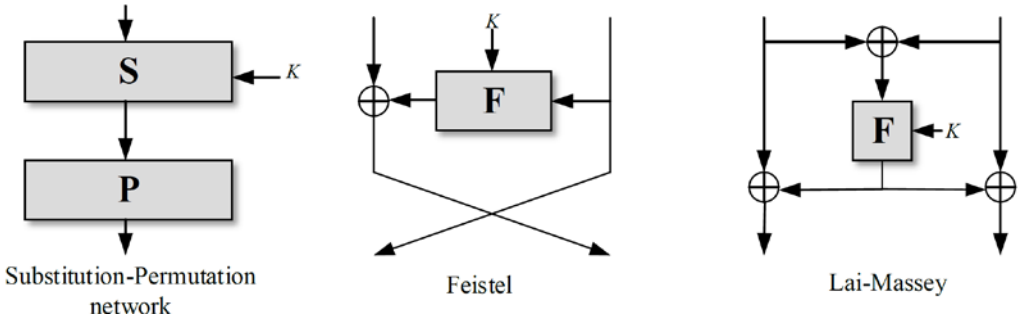
2 设计原理

在本算法的设计时，我们综合考虑算法的安全性、软件实现速度和硬件实现代价。大致设计原理为：提出简化版的 Lai-Massey 结构，尽可能加快混淆扩散速度以保障算法的安全性；采用模加操作代替 S 盒操作作为算法的非线性组件，加快算法的软件实现速度，并提高防御侧信道攻击的能力；减少轮函数中的操作个数，以模加操作配合循环移位操作的结构尽可能轻量化算法实现。

创新点一：算法结构简化 Lai-Massey Vs Feistel, SPN

Feistel、SPN 和 Lai-Massey 结构是算法设计中常用的三种成熟结构，如下图所示。然而在实际的算法设计中，大多采用前两种结构，基于 Lai-Massey 结构设

计的算法较少。但是值得注意的是，Lai-Massey 结构具有良好的混淆扩散能力，并且算法结构对称，有利于组件的并行计算。本算法结构在 Lai-Massey 结构的基础上进行简化，去除 F 函数而增设模加操作，在降低硬件实现代价的基础上依然保持良好的混淆扩散速度。



创新点二：算法结构 4 分支 Vs 2 分支

本算法结构将输入均分为 4 支进行运算，具有两大优势。第一，可减小每个组件大小（运算比特数为分组的 1/4），增加组件个数，从而增加算法设计的灵活性，并有利于轮函数中组件的并行运算；第二，算法结构为 4 分支使得算法中每个组件的运算为分组大小的 1/4,从而有利于 128 比特分组版本在 32 位平台的软件实现和 256 比特分组版本在 64 位平台下的软件实现。

创新点三：模加+循环移位 Vs S 盒+线性矩阵

模加操作的第x比特输出和所有的低x比特输入有关，因此其混淆扩散程度并不均匀，输入的低位比特混淆扩散较快，高位比特混淆扩散较慢。但佐以适当的循环移位操作，选择适当的循环移位参数，短轮数下即可实现快速的混淆扩散。模加操作和循环移位操作配合的方式，不仅有利于算法的混淆扩散速度，也可以降低组件个数，提升软件实现速度，降低硬件实现代价。

3 安全性分析

本设计文档给出 Ballet 算法的差分分析、线性分析、不可能差分分析、零相关分析和积分分析的结果。鉴于算法结构本身特点，其它分析方法（如中间相遇攻击和代数攻击等）对算法安全性的影响较小，设计文档中不再给出分析结果。

3.1 差分分析

为了评估 Ballet 算法抵抗差分分析的能力，我们基于 SAT 自动化搜索方法，对短轮数下的最优差分路线进行搜索，Ballet-128 和 Ballet-256 的 9 轮差分路线概率上界均为2^{-43}，因此 Ballet-128 的有效差分路线（概率$> 2^{-128}$）不超过$3 \times 9 - 1 = 26$轮，Ballet-256 的有效差分路线（概率$> 2^{-256}$）不超过$6 \times 9 - 1 = 53$轮。全轮 Ballet 算法能够抵抗差分攻击。

3.2 线性分析

为了评估 Ballet 算法抵抗线性分析的能力，我们同样利用基于 SAT 的自动化搜索方法对短轮数下的 Ballet 算法进行分析，Ballet-128 的 8 轮和 9 轮的最优线性路线的相关度分别为2^{-19}和2^{-23}，因此26轮（8 轮+9 轮+9 轮）的线性路线的相关度不超过$2^{-19} \times 2^{-23} \times 2^{-23} = 2^{-65} < 2^{-64}$，从而均为无效线性路线。换言之，Ballet-128 的有效线性路线不超过 25 轮。Ballet-256 的 8 轮最优线性路线的相关度为2^{-19}，9 轮的线性路线上界为2^{-23}，因此 52 轮的线性路线的相关度不超过$2^{-130} < 2^{-128}$，即 Ballet-256 的有效线性路线不超过 51 轮。全轮 Ballet 算法能够抵抗线性分析。

3.3 不可能差分分析

不可能差分分析是利用概率为 0 的差分路线进行的分析方法。我们利用基于 MILP 的自动化搜索方法对 Ballet 的不可能差分路线进行搜索。鉴于 Ballet 算法为 ARX 结构，我们仅搜索输入和输出差分汉明重量均为 1 的情况。

通过搜索发现，Ballet-128 在上述搜索空间中最长不可能差分路线为 7 轮路线，其中一条路线如下：

$$0^6 10^{25} || 0^{32} || 0^{32} || \, 0^{32} \nrightarrow 0^{32} || 0^{32} || 10^{31} || \, 0^{32}.$$

其中0^*表示$*$比特 0。Ballet-256 在上述搜索空间中的最长不可能差分路线也为 7 轮，其中一条路线如下：

$$0^6 10^{57} || 0^{64} || 0^{64} || \, 0^{64} \nrightarrow 0^{64} || 0^{64} || 10^{63} || \, 0^{64}.$$

全轮 Ballet 算法能够抵抗不可能差分分析。

3.4 零相关线性分析

零相关线性分析是利用相关度为 0 的线性路线进行的分析方法。我们利用基于 MILP 的自动化搜索方法对 Ballet 的零相关线性路线进行搜索。由于 Ballet 算法为 ARX 结构，我们仅搜索输入和输出掩码的汉明重量均为 1 的情况。

通过搜索发现，Ballet-128 在上述搜索空间中最长零相关线性路线为 6 轮路线，其中一条路线如下：

$$0^{32}||10^{31}||0^{32}||\ 0^{32} \nrightarrow 0^{31}1||0^{32}||0^{32}||\ 0^{32}.$$

Ballet-256 在上述搜索空间中的最长零相关线性路线也为 6 轮，其中一条路线如下：

$$0^{64}||10^{63}||0^{64}||\ 0^{64} \nrightarrow 0^{63}1||0^{64}||0^{64}||\ 0^{64}.$$

全轮 Ballet 算法能够抵抗零相关分析。

3.5 积分分析

积分分析是对分组密码算法有效的分析手段之一，我们使用比特级的 Division Property 对 Ballet 算法抵抗积分分析的能力进行评估。基于 SAT 的自动化积分路线搜索方法，选用2^{n-1}个明文（设置最高比特为常数，遍历剩余 127 比特），观察数据在密文上是否存在平衡比特（2^{n-1}个密文在该比特上的异或和为 0）。搜索结果发现 Ballet-128 和 Ballet-256 在该搜索范围内均不存在 7 轮及以上积分路线，全轮 Ballet 算法能够抵抗积分分析。

3.6 相关密钥类分析

相关密钥类攻击中，最有效的攻击手段有滑动攻击、相关密钥差分攻击和相关密钥不可能差分攻击等。由于我们在密钥生成算法中加入了不同的轮常数，因此能够有效的抵抗滑动攻击。针对相关密钥差分攻击，我们对短轮数下的相关密钥差分路线进行了搜索，未找到高概率的迭代路线，也未找到弱密钥的存在。鉴于算法的足够安全冗余度和 ARX 算法分析的困难度，我们认为 Ballet 算法能够抵抗相关密钥差分攻击。对于相关密钥不可能差分攻击，鉴于 7 轮的单密钥不可能差分路线和密钥生成算法的扩散速度，全轮算法能够抵抗相关密钥不可能差分

攻击。

4 性能分析

4.1 软件性能分析

本节给出 Ballet 算法在 64bit Windows 环境和 32bit ARM 环境下的实现结果。测试环境分别为：Intel(R) Core(TM) i7-8700T CPU 2.40 GHz 主频、64 位 Windows10 操作系统、16GB 内存，X86-64 环境和基于 STM32 F103 的开发板(主频 72Mhz)：stm32f1zet6 核心板 6、512K 片内 Flash、64K 片内 RAM。

在进行性能测试时，采用 256 字节数据作为输入，每次测试变换密钥，取10^5次 CBC 模式运算测试结果的平均值为加/解密速率（单位：Mbps）的结果，其中加/解密执行时间包含密钥扩展算法运算时间。测试结果如下表所示。由于 CBC 模式下的解密过程可以并行实现，我们在表中给出的解密速率是基于多路实现结果。

加密速率=加密数据大小/加密执行时间（单位：Mbps）

解密速率=解密数据大小/解密执行时间（单位：Mbps）

算法软件实现类别		加密速率	解密速率	ROM 占用
64bit Windows 环境下的算法速率优化 C 实现	128/128	2038 Mbps	7899 Mbps	-
	128/256	2289 Mbps	7080 Mbps	-
	256/256	2246 Mbps	4425 Mbps	-
32bit ARM 环境下的算法速率优化 C 实现	128/128	10.6 Mbps	9.57 Mbps	2.6k
	128/256	9.75 Mbps	9.43 Mbps	2.9k
	256/256	4.34 Mbps	4.21 Mbps	3.5k

表 6 软件实现结果

4.2 硬件性能分析

本节给出 Ballet 算法的 Verilog 硬件仿真实现结果。采用的仿真工具为 ModelSim SE 10.1a，综合工具为 Synopsys Design Vision（F-2011.09-SP1 Version），以及工艺库为 HJTC 0.11um。

算法 Verilog 硬件仿真实现采用 32 个分组数据进行加/解密。算法实现的加/解密运算用时包含密钥扩展运算用时（单位：微秒）。算法电路面积规模为包含加密、解密、密钥扩展运算的算法整体实现的含互连线面积（单位：平方微米）。具体方法如下：

（1） 仿真验证

使用ModelSim对算法实现正确性进行仿真验证，并记录32个分组数据运算（含密钥扩展运算）占用的时钟周期数。

（2） 性能自测试

在给定的时钟约束条件上进行前端综合，得到并记录以下测试数据：

1) 关键路径时长

在Synopsys Design Complier中进行关键路径时长测试。根据算法设计提供的自测数据，设置相应的时钟约束条件，通过综合软件得到关键路径时长。

2) 速率

在关键路径时长合理的条件下，通过以下公式计算加解密速率：

加密速率=加密数据长度/（时钟约束条件*加密占用的时钟钟周期数）（单位：Mbps）

解密速率=解密数据长度/（时钟约束条件*解密占用的时钟周期数）（单位：Mbps）

3) 面积

在可满足的最小时钟约束条件下，在 HJTC 0.11um 工艺库基础上进行综合，记录算法实现面积（包含加密、解密、密钥扩展运算的算法整体实现的含互连线面积，单位：平方微米）。

4) 吞面比

加密吞面比=加密速率（单位：Mbps）/算法电路面积（单位：平方微米）

解密吞面比=解密速率（单位：Mbps）/算法电路面积（单位：平方微米）

实现结果如下表所示。

	自测试结果		
	128/128	128/256	256/256
运算周期	3054	3184	4874
时钟约束	3.31 ns	3.3 ns	3.59 ns
加密速率	822.78 Mbps	791.59 Mbps	950.79 Mbps
解密速率	798.36 Mbps	768.08 Mbps	922.35 Mbps
面积	45967.11 um^2	64610.57 um^2	96026.89 um^2
面积	4739.37 GE	6661.57 GE	9900.70 GE
加密吞面比	0.017899	0.012252	0.009901
解密吞面比	0.017368	0.011888	0.009605

表 7 Verilog 硬件仿真实现结果

5 优缺点分析

Ballet 算法在设计时充分考虑了算法的安全性、软件实现速度、硬件实现代价以及抵抗侧信道攻击的能力，具体优点和缺点如下。

5.1 优点一：简单、灵活、可延展性强

Ballet 算法的轮函数中仅包含 2 个模加操作和 3 个异或操作（长度均为分组长度的 1/4），无 S 盒和复杂线性层，简洁明了，代码量少，内存消耗小，易于实现，且所有版本均使用相同的轮函数，算法灵活性和延展性强。

5.2 优点二：软件实现快

Ballet 算法是在 Lai-Massey 结构的基础进行简化设计而成，轮函数为 4 分支的近似对称结构。同时，轮函数中仅包含模加操作（2 个）、异或操作（3 个）和循环移位操作（4 个），利于软件实现，尤其是 128 比特分组版本算法在 32 位平台上和 256 比特分组版本算法在 64 位平台上的实现。除此，Ballet 算法在采用单路（1-way）实现方式时依然具有较大的优势。

5.3 优点三：易于轻量化实现

Ballet 算法的轮函数中消耗硬件面积的操作仅包含 2 个模加和 3 个异或操作（长度均为分组长度的 1/4），硬件面积较小，易于轻量化实现。

5.4 优点四：安全

通过对 Ballet 算法抵抗差分分析、线性分析等现有攻击方法的分析，可得 Ballet 能够抵抗现有攻击方法，并具有充分的安全冗余。

5.5 优点五：防护侧信道攻击代价小

现有的侧信道攻击大多针对算法中的 S 盒进行。Ballet 算法采用 ARX 结构，非线性操作为模加操作而非 S 盒，因此防护侧信道攻击的代价小。

5.6 缺点一：安全性评估能力 Vs 轮数

对ARX类算法安全性的合理评估是一个难点，现多通过短轮数下的安全界叠加来评估长轮数下的安全性，从而导致评估的安全界较松。在此前提下，为保证算法充分的抵抗现有攻击的能力，我们将算法的轮数设定较高，从而一定程度上影响了软硬件的表现。在之后的研究工作中，我们将进一步对ARX算法抗差分分析和线性分析的安全界进行研究，提出更优的评估方法。